High-Performance Spatial Data Compression for Scientific Applications

Ronald Kriemann, Hatem Ltaief, Minh Bau Luong,
Francisco E. Hernández Pérez, Hong G. Im and David Keyes

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CCRC
Clean Combustion
Research Center

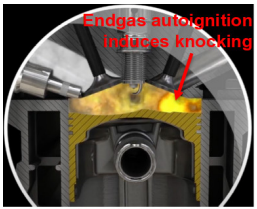
Euro-Par 2022

MAX PLANCK INSTITUTE
FOR MATHEMATICS IN THE SCIENCES

Combustion Application

Challenges in Downsizing and Boosting Engines

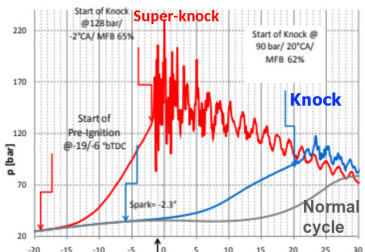
Modern engines operate at *higher load and elevated pressure* for higher efficiency, better fuel economy and lower emissions.



https://youtu.be/qMZ7dFZvhhI

This also leads to higher knock propensity, and even *super-knock* (extremely high pressure).

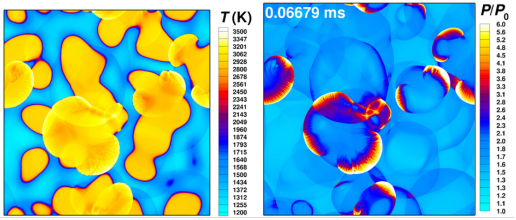
Z. Wang et al. 2014 IJER



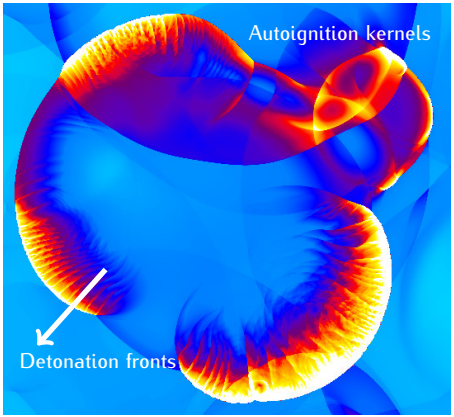
Combustion Application

Development of detonation waves

Complex interplay between auto-ignition fronts and acoustic pressure resonance.



Isocontours of temperature and pressure at the onset of detonation.



For the 2-D case, the full dataset has a size of ~100TB with 10 millions of grid points[1].

Need for *compression*!

[1] M.B. Luong, S. Desai, F.E. Hernández Pérez, R. Sankaran, B. Johansson, H.G. Im, A statistical analysis of developing knock intensity in a mixture with temperature inhomogeneities, Proc. Combust. Inst. 37 (2020).

Combustion Application

Lossy compression possible as computation permits reduced accuracy ($\varepsilon \leq 10^{-4}$).

Compression candidates

ZFP[1]

- block-wise orthogonal transforms (blocks of size 4^d)
- \+ very fast
- \- limited compression rate

SZ[2]

- curve fitting algorithm
- \+ good compression rate for general data

TTHRESH[3]

- Tucker decomposition (HOSVD) plus bit-plane truncation
- \- very slow due to global HOSVD

MGARD[4]

- multi-grid technique plus lossless compression
- \- not parallel on shared memory machines

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

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

[3] Ballester-Ripoll, R., et.al.: "TTHRESH: Tensor Compression for Multidimensional Visual Data". IEEE Trans. on Vis. and Comp. Graphics 26(9), 2891–2903 (2020).

[4] Whitney, B., et.al.: "Multilevel techniques for compression and reduction of scientific data – the univariate case". Comp.Vis.Sci. 19, 65–76 (2018)

HLRcompress

Low-Rank Approximation

Given $D \in \mathbb{C}^{n \times n}$ and a user-defined $\varepsilon > 0$ one looks for $U, V \in \mathbb{C}^{n \times k}$ with $k \ll n$ such that

$$\frac{\left\| D - U \cdot V^H \right\|_2}{\left\| D \right\|_2} \leq \varepsilon$$

Low-Rank Approximation

Given $D \in \mathbb{C}^{n \times n}$ and a user-defined $\varepsilon > 0$ one looks for $U, V \in \mathbb{C}^{n \times k}$ with $k \ll n$ such that

$$\frac{\left\| D - U \cdot V^H \right\|_2}{\left\| D \right\|_2} \leq \varepsilon$$

Singular Value Decomposition

Computes best approximation.

Runtime complexity: $\mathcal{O}\left(n^3\right)$.

Rank Revealing QR

Randomized SVD

Cross Approximation

function SVD(**in**: D, ε, **out**: U, V)
  $[U_s, S_s, V_s] :=$ `svd(` D `)`;
  $k :=$ `rank(`S_s, ε`)`;
  $U := U_s(:, 1:k) \cdot S_s(1:k, 1:k)$;
  $V := V_s(:, 1:k)$;

Low-Rank Approximation

Given $D \in \mathbb{C}^{n \times n}$ and a user-defined $\varepsilon > 0$ one looks for $U, V \in \mathbb{C}^{n \times k}$ with $k \ll n$ such that

$$\frac{\left\| D - U \cdot V^H \right\|_2}{\left\| D \right\|_2} \leq \varepsilon$$

Singular Value Decomposition

Rank Revealing QR

Randomized SVD

Cross Approximation

Based on reordering the remaining columns during QR.

Rank and error control by $R(i : k, i : k)$.

Runtime complexity: $\mathcal{O}\left(n^3\right)$.

function RRQR(**in**: D, ε, **out**: U, V)
  $[Q, R, P] =$ `qrp`(D);
  for $i = 1, \ldots, n$ **do**
    $S(i) := \left\| R(i : n, i : n) \right\|_F$;
  $k :=$ `rank`(S, ε);
  $U := Q(:, 1 : k)$;
  $V := P \cdot R(1 : k, :)^H$;

Low-Rank Approximation

Given $D \in \mathbb{C}^{n \times n}$ and a user-defined $\varepsilon > 0$ one looks for $U, V \in \mathbb{C}^{n \times k}$ with $k \ll n$ such that

$$\frac{\left\| D - U \cdot V^H \right\|_2}{\left\| D \right\|_2} \leq \varepsilon$$

Singular Value Decomposition

Rank Revealing QR

Randomized SVD

Cross Approximation

Approximate column basis and convert matrix into new basis.

No guaranteed error control.

Runtime complexity: $\mathcal{O}\left(k \cdot n^2\right)$.

function ColumnBasis(**in**: D, ε, **out**: Q)
  $Q := 0$; $U_0 := U$; $n_b := 4$;
  for $i = 1, \ldots, n/n_b$ **do**
    $T_i :=$ `randnorm`(n, n_b);
    $Q_i :=$ `qr`$((I - QQ^H)D \cdot T_i)$;
    $Q_i := (I - QQ^H)Q_i$;
    $Q := [Q, Q_i]$;
    if $\left\| D - QQ^H D \right\| \leq \varepsilon \cdot \left\| D \right\|$ **then**
      break;

Low-Rank Approximation

Given $D \in \mathbb{C}^{n \times n}$ and a user-defined $\varepsilon > 0$ one looks for $U, V \in \mathbb{C}^{n \times k}$ with $k \ll n$ such that

$$\frac{\left\| D - U \cdot V^H \right\|_2}{\left\| D \right\|_2} \leq \varepsilon$$

Singular Value Decomposition

Rank Revealing QR

Randomized SVD

Approximate column basis and convert matrix into new basis.

No guaranteed error control.

Runtime complexity: $\mathcal{O}\left(k \cdot n^2\right)$.

function RandSVD(**in**: D, ε, **out**: U, V)
  $B := \texttt{ColumnBasis}(D, \varepsilon)$;
  $[Q, R] := \texttt{qr}(D^H \cdot B)$;
  $[U_s, S_s, V_s] := \texttt{svd}(R)$;
  $k := \texttt{rank}(S_s, \varepsilon)$;
  $U := B \cdot V_s(:, 1:k) S(1:k, 1:k)$;
  $V := D^H \cdot B \cdot U_s(:, 1:k)$;

Cross Approximation

Low-Rank Approximation

Given $D \in \mathbb{C}^{n \times n}$ and a user-defined $\varepsilon > 0$ one looks for $U, V \in \mathbb{C}^{n \times k}$ with $k \ll n$ such that

$$\frac{\left\| D - U \cdot V^H \right\|_2}{\left\| D \right\|_2} \leq \varepsilon$$

Singular Value Decomposition

Rank Revealing QR

Randomized SVD

Cross Approximation

Successively selects pairs of rows/columns for rank-1 updates. ACA may fail.

Runtime complexity: $\mathcal{O}\left(k^2 \cdot n\right)$.

function ACA(**in**: D, ε, **out**: U, V)
  $c_1 = 1$;
  for $i = 1, \ldots$ **do**
    $u_i := \texttt{column}(D, c_i) - U \cdot V(c_i, :)'$;
    $r_i := \texttt{maxidx}(w_i)$; $u_i := u_i / u_i(r_i)$;
    $v_i := \texttt{row}(D, r_i)' - V \cdot U(r_i, :)'$;
    $U := [U, u_i]$; $V := [V, v_i]$;
    if $\left\| u_i \cdot v_i' \right\|_F \leq \varepsilon \left\| U \cdot V^H \right\|_F$ **then**
      break;
    $c_{i+1} := \texttt{maxidx}(v_i)$;

Low-Rank Approximation

Given $D \in \mathbb{C}^{n \times n}$ and a user-defined $\varepsilon > 0$ one looks for $U, V \in \mathbb{C}^{n \times k}$ with $k \ll n$ such that

$$\frac{\left\| D - U \cdot V^H \right\|_2}{\left\| D \right\|_2} \leq \varepsilon$$

Singular Value Decomposition

Rank Revealing QR

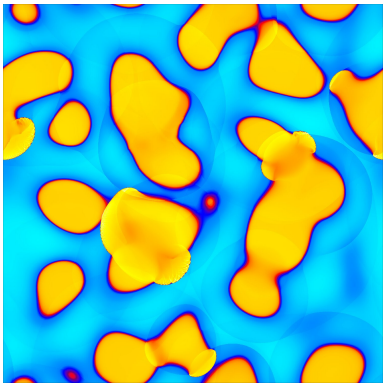
Randomized SVD

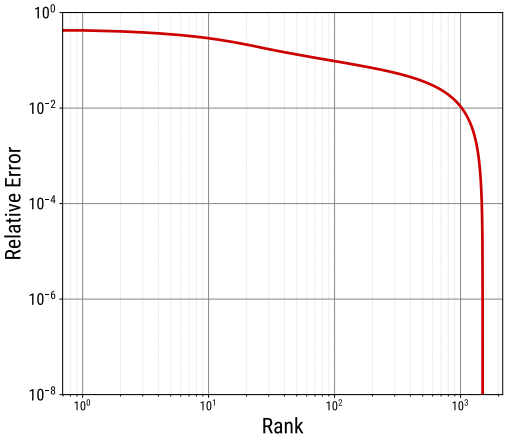
Cross Approximation

All algorithms also usable for *recompression* of low-rank matrices in $\mathcal{O}\left(k^2 \cdot n\right)$.

Global vs. Local Approximation

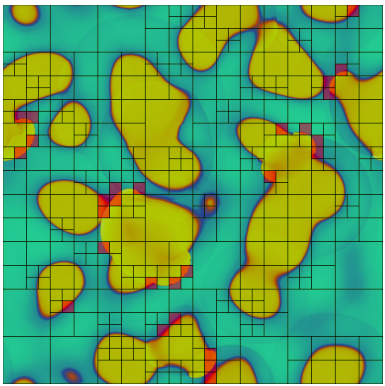
Normally, D is *not globally* low-rank approximable, i.e. $k \approx n$.

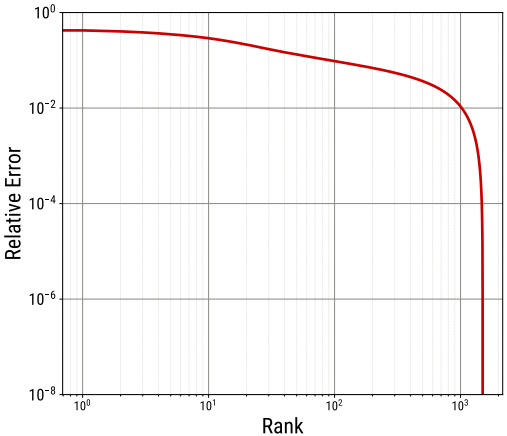




Global vs. Local Approximation

Normally, D is *not globally* low-rank approximable, i.e. $k \approx n$.

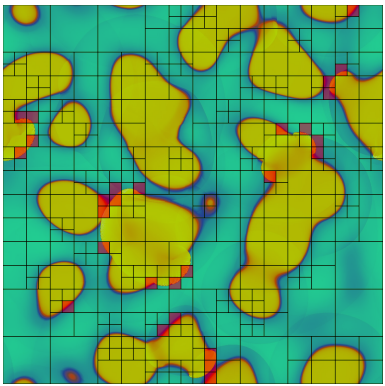


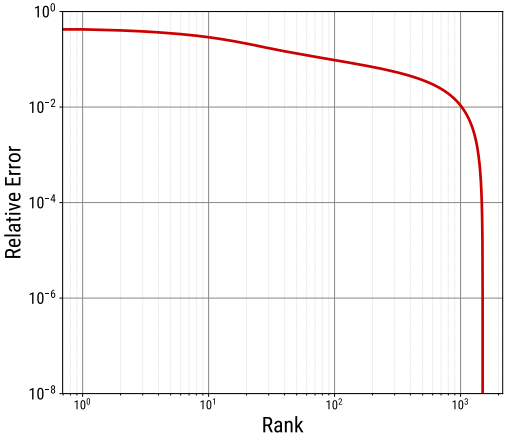


But D is *locally* low-rank approximable!

Global vs. Local Approximation

Normally, D is *not globally* low-rank approximable, i.e. $k \approx n$.



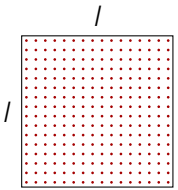


But D is *locally* low-rank approximable!

Problem: identify local, low-rank approximable blocks in D.

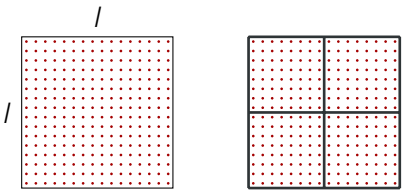
Initial Partitioning

Let $D = (d_{ij})_{i,j=0}^{n-1}$, $d_{ij} \in \mathbb{C}$ be $N = n^2$ datapoints with $i, j \in I := \{0, \cdots, n-1\}$ at positions $(i \cdot h, j \cdot h), h > 0$. The (index) set $I \times I$ can recursively be split until blocks $(\tau, \sigma) \subseteq I \times I$ with $\min(\#\tau, \#\sigma) > n_{\text{tile}}, n_{\text{tile}} > 0$, are constructed:



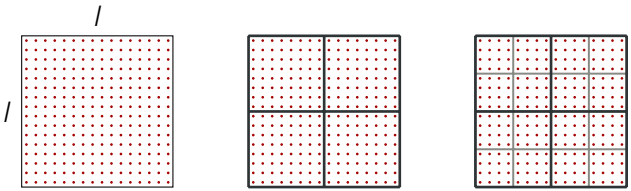
Initial Partitioning

Let $D = (d_{ij})_{i,j=0}^{n-1}$, $d_{ij} \in \mathbb{C}$ be $N = n^2$ datapoints with $i, j \in I := \{0, \cdots, n-1\}$ at positions $(i \cdot h, j \cdot h), h > 0$. The (index) set $I \times I$ can recursively be split until blocks $(\tau, \sigma) \subseteq I \times I$ with $\min(\#\tau, \#\sigma) > n_{\text{tile}}, n_{\text{tile}} > 0$, are constructed:



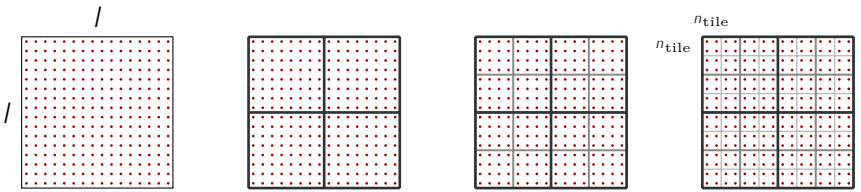
Initial Partitioning

Let $D = (d_{ij})_{i,j=0}^{n-1}$, $d_{ij} \in \mathbb{C}$ be $N = n^2$ datapoints with $i, j \in I := \{0, \cdots, n-1\}$ at positions $(i \cdot h, j \cdot h), h > 0$. The (index) set $I \times I$ can recursively be split until blocks $(\tau, \sigma) \subseteq I \times I$ with $\min(\#\tau, \#\sigma) > n_{\text{tile}}, n_{\text{tile}} > 0$, are constructed:



Initial Partitioning

Let $D = (d_{ij})_{i,j=0}^{n-1}$, $d_{ij} \in \mathbb{C}$ be $N = n^2$ datapoints with $i, j \in I := \{0, \cdots, n-1\}$ at positions $(i \cdot h, j \cdot h)$, $h > 0$. The (index) set $I \times I$ can recursively be split until blocks $(\tau, \sigma) \subseteq I \times I$ with $\min(\#\tau, \#\sigma) > n_{\text{tile}}$, $n_{\text{tile}} > 0$, are constructed:



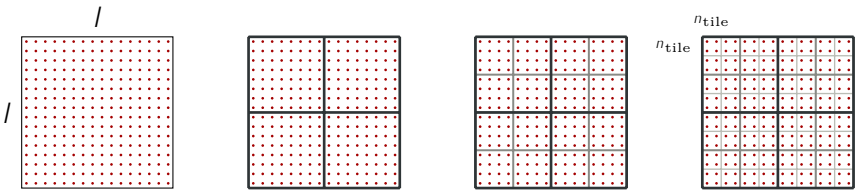
Such constructed blocks contain *spatially neighboured* datapoints.

The construction yields an hierarchical tree $\mathcal{T}$ with root (I, I) and non-leaf blocks $(\tau, \sigma) \in \mathcal{T}$ with sub-nodes

$$\begin{pmatrix} (\tau_0, \sigma_0) & (\tau_0, \sigma_1) \\ (\tau_1, \sigma_0) & (\tau_1, \sigma_1) \end{pmatrix}$$

Initial Partitioning

Let $D = (d_{ij})_{i,j=0}^{n-1}$, $d_{ij} \in \mathbb{C}$ be $N = n^2$ datapoints with $i, j \in I := \{0, \cdots, n-1\}$ at positions $(i \cdot h, j \cdot h)$, $h > 0$. The (index) set $I \times I$ can recursively be split until blocks $(\tau, \sigma) \subseteq I \times I$ with $\min(\#\tau, \#\sigma) > n_{\text{tile}}$, $n_{\text{tile}} > 0$, are constructed:



Such constructed blocks contain *spatially neighboured* datapoints.

The construction yields an hierarchical tree $\mathcal{T}$ with root (I, I) and non-leaf blocks $(\tau, \sigma) \in \mathcal{T}$ with sub-nodes

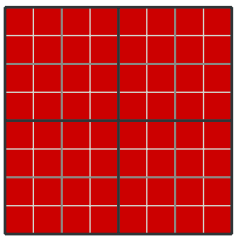
$$\begin{pmatrix} (\tau_0, \sigma_0) & (\tau_0, \sigma_1) \\ (\tau_1, \sigma_0) & (\tau_1, \sigma_1) \end{pmatrix}$$

Remark

For non-tensor grids, construction can be easily performed by, e.g., kd-tree algorithm or space filling curves.

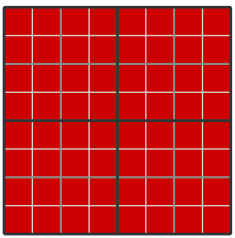
Approximation Phase

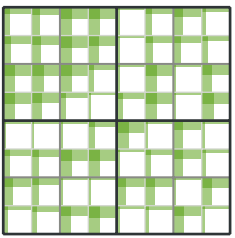
For each leaf $(\tau, \sigma) \in \mathcal{T}$ one computes a low-rank approximation $U_{\tau,\sigma} V_{\tau,\sigma}^H$ of the corresponding data block $D_{\tau,\sigma} := D|_{\tau,\sigma}$ with a block-local accuracy $\varepsilon_{\tau,\sigma}$.



Approximation Phase

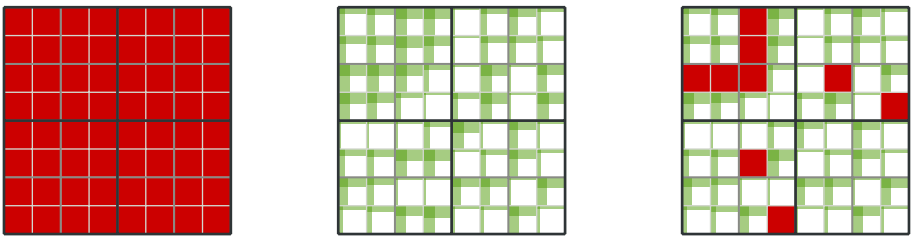
For each leaf $(\tau, \sigma) \in \mathcal{T}$ one computes a low-rank approximation $U_{\tau,\sigma} V_{\tau,\sigma}^H$ of the corresponding data block $D_{\tau,\sigma} := D|_{\tau,\sigma}$ with a block-local accuracy $\varepsilon_{\tau,\sigma}$.





Approximation Phase

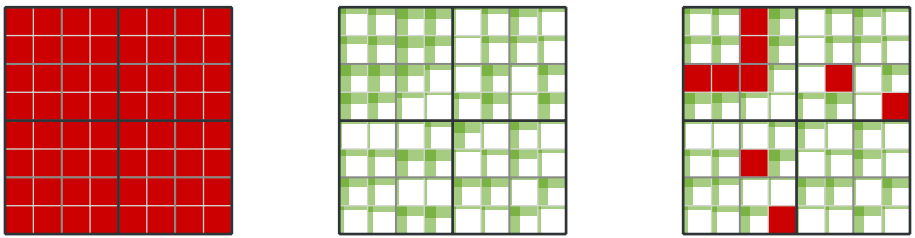
For each leaf $(\tau, \sigma) \in \mathcal{T}$ one computes a low-rank approximation $U_{\tau,\sigma} V_{\tau,\sigma}^H$ of the corresponding data block $D_{\tau,\sigma} := D|_{\tau,\sigma}$ with a block-local accuracy $\varepsilon_{\tau,\sigma}$.



If the low-rank representation uses less memory ($k_{\tau,\sigma} < n_{\text{tile}}/2$) it is kept.

Approximation Phase

For each leaf $(\tau, \sigma) \in \mathcal{T}$ one computes a low-rank approximation $U_{\tau,\sigma} V_{\tau,\sigma}^H$ of the corresponding data block $D_{\tau,\sigma} := D|_{\tau,\sigma}$ with a block-local accuracy $\varepsilon_{\tau,\sigma}$.



If the low-rank representation uses less memory ($k_{\tau,\sigma} < n_{\text{tile}}/2$) it is kept.

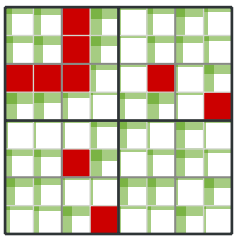
Remark

All *approximations are independent and can be performed* in parallel. *Also, since by construction the blocks are of similar size,* batch functions *may be used, e.g., on GPUs.*

Merging Phase

If for a non-leaf $(\tau, \sigma) \in \mathcal{T}$ *all* sub-blocks are in low-rank format, the sub-blocks may be *merged* and *recompressed* while maintaining the approximation error.

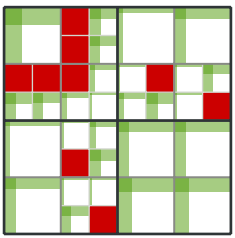
If the recompression results in *less* memory, the coarser low-rank block is kept.



Merging Phase

If for a non-leaf $(\tau, \sigma) \in \mathcal{T}$ *all* sub-blocks are in low-rank format, the sub-blocks may be *merged* and *recompressed* while maintaining the approximation error.

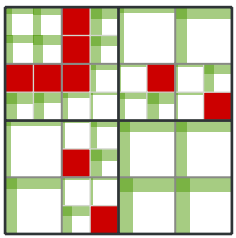
If the recompression results in *less* memory, the coarser low-rank block is kept.



Merging Phase

If for a non-leaf $(\tau, \sigma) \in \mathcal{T}$ *all* sub-blocks are in low-rank format, the sub-blocks may be *merged* and *recompressed* while maintaining the approximation error.

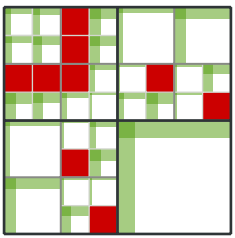
If the recompression results in *less* memory, the coarser low-rank block is kept.



Merging Phase

If for a non-leaf $(\tau, \sigma) \in \mathcal{T}$ *all* sub-blocks are in low-rank format, the sub-blocks may be *merged* and *recompressed* while maintaining the approximation error.

If the recompression results in *less* memory, the coarser low-rank block is kept.

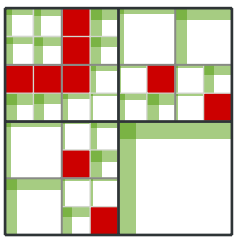


The procedure stops if no sub-block can be merged or if the root (I, I) is reached.

Merging Phase

If for a non-leaf $(\tau, \sigma) \in \mathcal{T}$ *all* sub-blocks are in low-rank format, the sub-blocks may be *merged* and *recompressed* while maintaining the approximation error.

If the recompression results in *less* memory, the coarser low-rank block is kept.



The procedure stops if no sub-block can be merged or if the root (I, I) is reached.

Remark

Each merge/recompression is again independent and can be performed in parallel. However, due to different ranks in the blocks, batch mode may be challenging to efficiently deploy on manycore architectures.

Binary Compression

After a block (τ, σ) can not be merged, it either holds dense data $D_{\tau,\sigma}$ or the low-rank factors $U_{\tau,\sigma}, V_{\tau,\sigma}$.

Depending on the computation, these are stored in *double* or *single* precision.

The approximation error ε typically leaves room for further *compression of the binary representation*.

For this, we employ *ZFP* in fixed accuracy mode to further compress $D_{\tau,\sigma}$ or $U_{\tau,\sigma}, V_{\tau,\sigma}$ while maintaining the approximation error.

Remark

We also tried SZ but the localized compression were less optimal.

Numerical Results

Numerical Results

Hardware

All benchmarks are performed on a *two-socket 64-core AMD Epyc 7702*.

GPU benchmarks on *NVIDIA A100*.

Software

SZ:	v2.1, best compression mode, with OpenMP
ZFP:	v0.5.5, fixed-accuracy mode, with OpenMP
MGARD:	v1.0, using additional lossless compression (`CPU_Lossless`).
HLRcompress:	$n_{\text{tile}} = 32$, with **Intel TBB**

Remark

MGARD only usable with accuracy $\varepsilon \geq 10^{-5}$.

Also: GCC v10 compiler and Intel MKL for BLAS/LAPACK (AVX2 code path).

Logarithmic Kernel

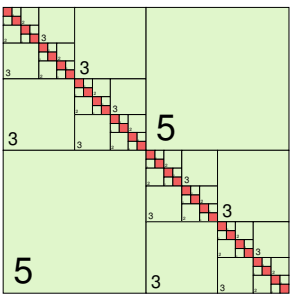
This example uses

$$D_{ij} = log|x_i - x_j|_2$$

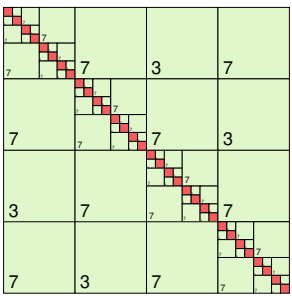
with $x_i = (\sin ih, \cos ih), h = 2\pi/n$.

This data set is *very* smooth resulting in a long merge phase of HLRcompress.

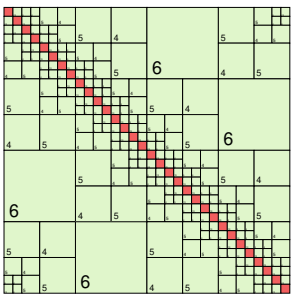
Block Structure



$\varepsilon = 10^{-4}$



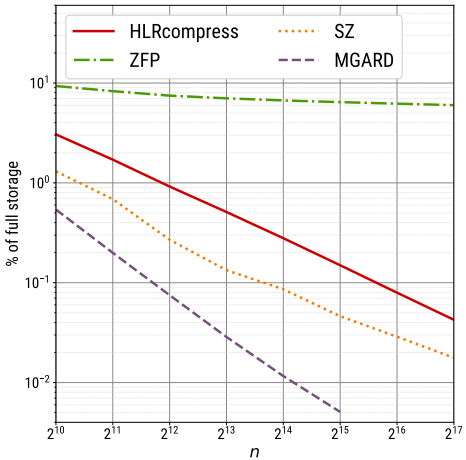
$\varepsilon = 10^{-6}$



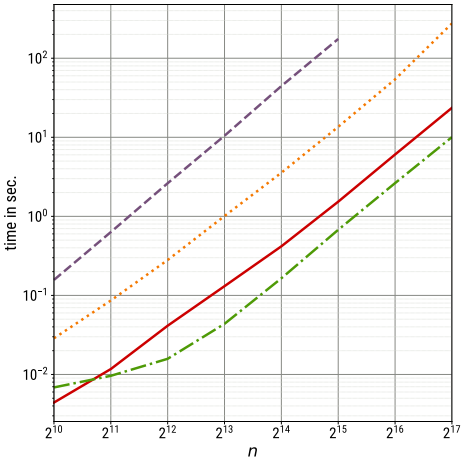
$\varepsilon = 10^{-8}$

Logarithmic Kernel

Compression Rate



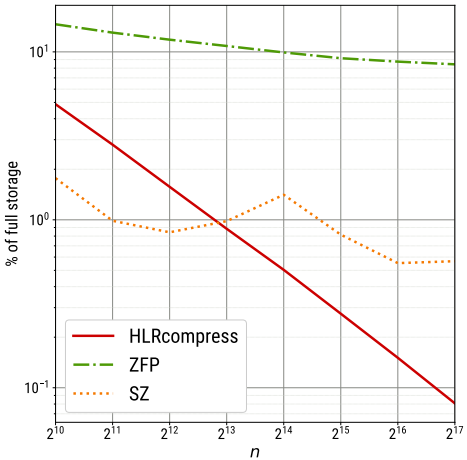
Runtime



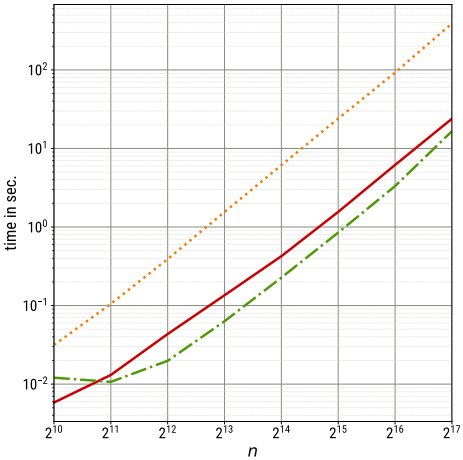
$$\varepsilon = 10^{-4}$$

Logarithmic Kernel

Compression Rate



Runtime



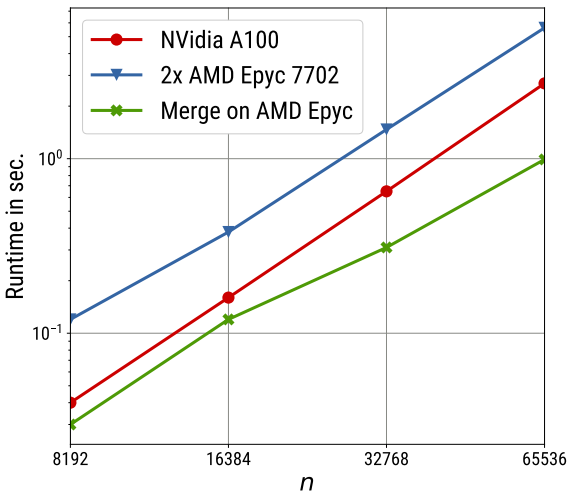
$$\varepsilon = 10^{-6}$$

Logarithmic Kernel

Approximation Phase on GPUs

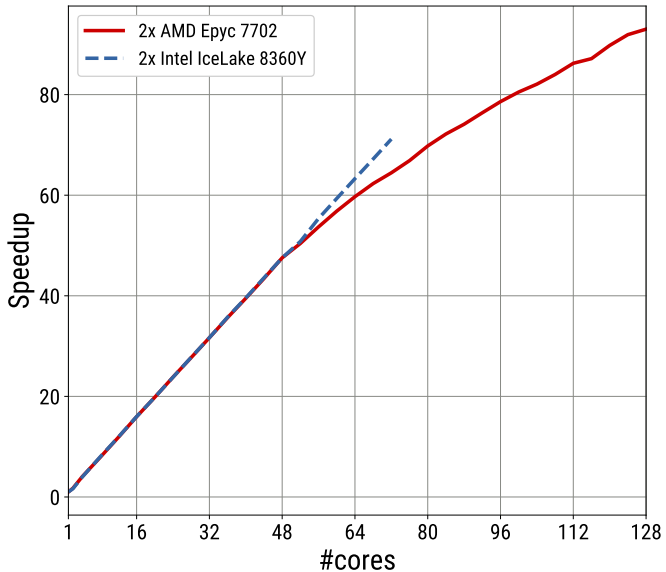
Using `gesvdjBatched` for initial approximation phase.

Not usable for merge phase.



Logarithmic Kernel

Parallel Scaling



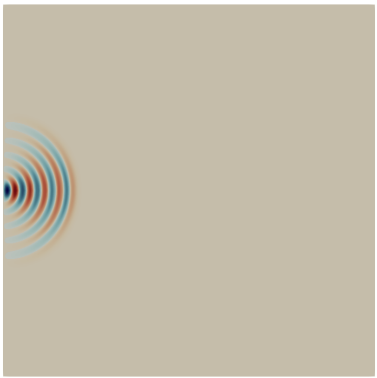
Wave Equation

D is defined by the solutions of

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - \Delta u &= f \quad \text{in } \Omega \times [0, T], \\ u(x,t) &= g \quad \text{on } \partial\Omega \times [0, T], \text{ and} \\ u(x,0) &= 0 \quad \text{in } \Omega, \end{aligned}$$

with $\Omega = [-\pi, \pi]^2$, $T > 0$ and

$$g = \begin{cases} \sin(8\pi t) & \text{for } x = (-\pi, x_1), |x_1| < 0. \\ 0 & \text{otherwise} \end{cases}$$



$t = 1$

The data becomes *more chaotic* with time, degrading low-rank compression rate.

Also, *spatial resolution* has significant influence on low-rank approximation.

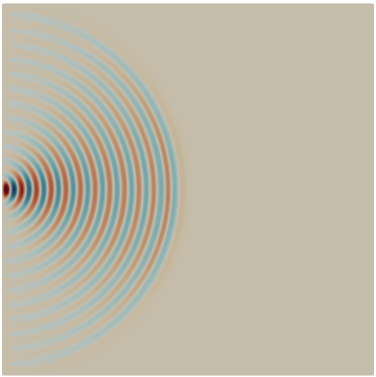
Wave Equation

D is defined by the solutions of

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - \Delta u &= f \quad \text{in } \Omega \times [0, T], \\ u(x,t) &= g \quad \text{on } \partial\Omega \times [0, T], \text{ and} \\ u(x,0) &= 0 \quad \text{in } \Omega, \end{aligned}$$

with $\Omega = [-\pi, \pi]^2$, $T > 0$ and

$$g = \begin{cases} \sin(8\pi t) & \text{for } x = (-\pi, x_1), |x_1| < 0. \\ 0 & \text{otherwise} \end{cases}$$



$t = 3$

The data becomes *more chaotic* with time, degrading low-rank compression rate.

Also, *spatial resolution* has significant influence on low-rank approximation.

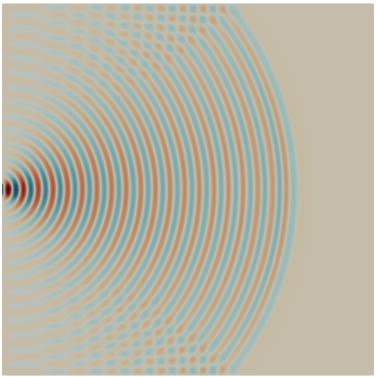
Wave Equation

D is defined by the solutions of

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - \Delta u &= f \quad \text{in } \Omega \times [0, T], \\ u(x,t) &= g \quad \text{on } \partial\Omega \times [0, T], \text{ and} \\ u(x,0) &= 0 \quad \text{in } \Omega, \end{aligned}$$

with $\Omega = [-\pi, \pi]^2$, $T > 0$ and

$$g = \begin{cases} \sin(8\pi t) & \text{for } x = (-\pi, x_1), |x_1| < 0. \\ 0 & \text{otherwise} \end{cases}$$



$t = 5$

The data becomes *more chaotic* with time, degrading low-rank compression rate.

Also, *spatial resolution* has significant influence on low-rank approximation.

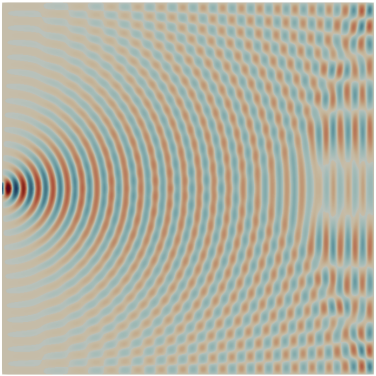
Wave Equation

D is defined by the solutions of

$$\frac{\partial^2 u}{\partial t^2} - \Delta u = f \quad \text{in } \Omega \times [0, T],$$
$$u(x, t) = g \quad \text{on } \partial\Omega \times [0, T], \text{ and}$$
$$u(x, 0) = 0 \quad \text{in } \Omega,$$

with $\Omega = [-\pi, \pi]^2$, $T > 0$ and

$$g = \begin{cases} \sin(8\pi t) & \text{for } x = (-\pi, x_1), |x_1| < 0. \\ 0 & \text{otherwise} \end{cases}$$



$t = 7$

The data becomes *more chaotic* with time, degrading low-rank compression rate.

Also, *spatial resolution* has significant influence on low-rank approximation.

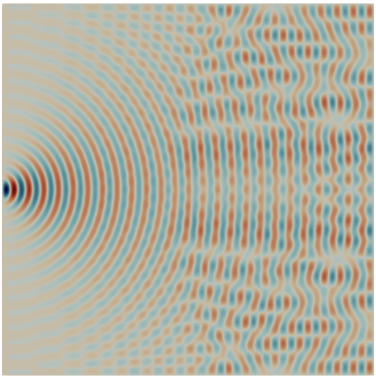
Wave Equation

D is defined by the solutions of

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - \Delta u &= f \quad \text{in } \Omega \times [0, T], \\ u(x, t) &= g \quad \text{on } \partial\Omega \times [0, T], \text{ and} \\ u(x, 0) &= 0 \quad \text{in } \Omega, \end{aligned}$$

with $\Omega = [-\pi, \pi]^2$, $T > 0$ and

$$g = \begin{cases} \sin(8\pi t) & \text{for } x = (-\pi, x_1), |x_1| < 0. \\ 0 & \text{otherwise} \end{cases}$$



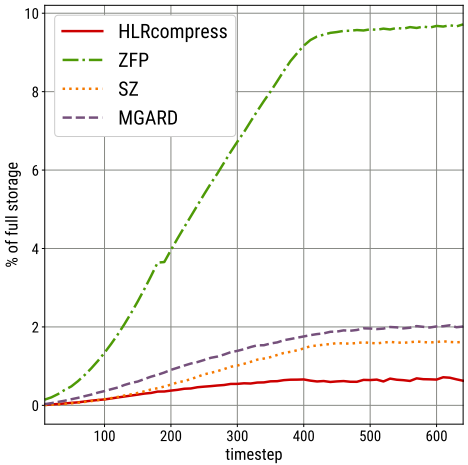
$t = 10$

The data becomes *more chaotic* with time, degrading low-rank compression rate.

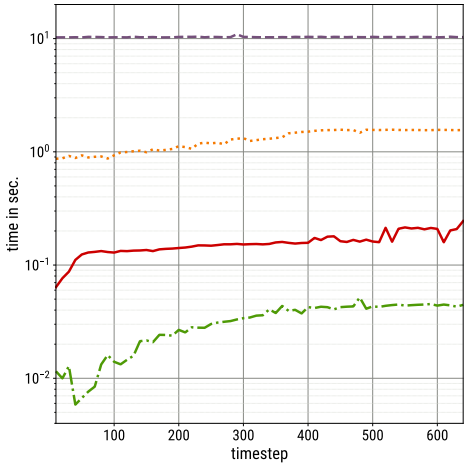
Also, *spatial resolution* has significant influence on low-rank approximation.

Wave Equation

Compression Rate



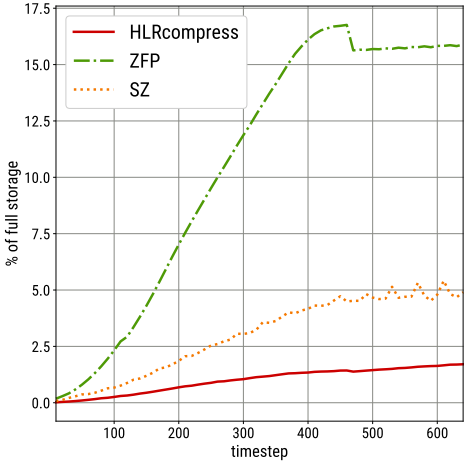
Runtime



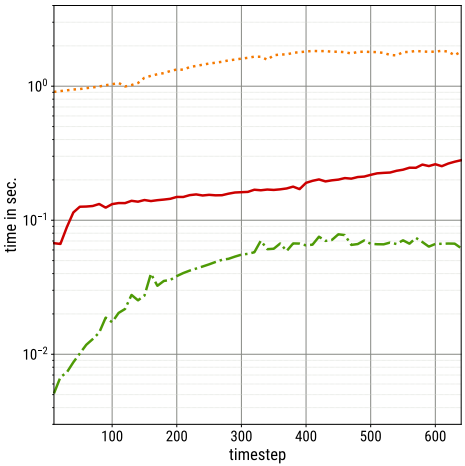
$$\varepsilon = 10^{-4}$$

Wave Equation

Compression Rate



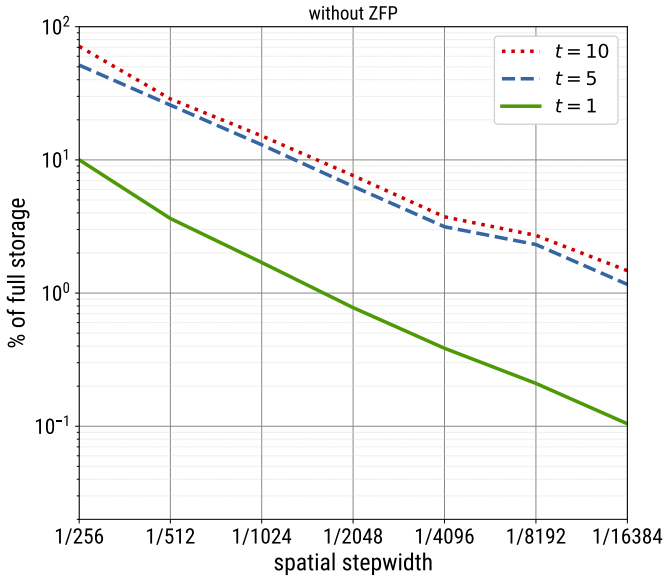
Runtime



$$\varepsilon = 10^{-6}$$

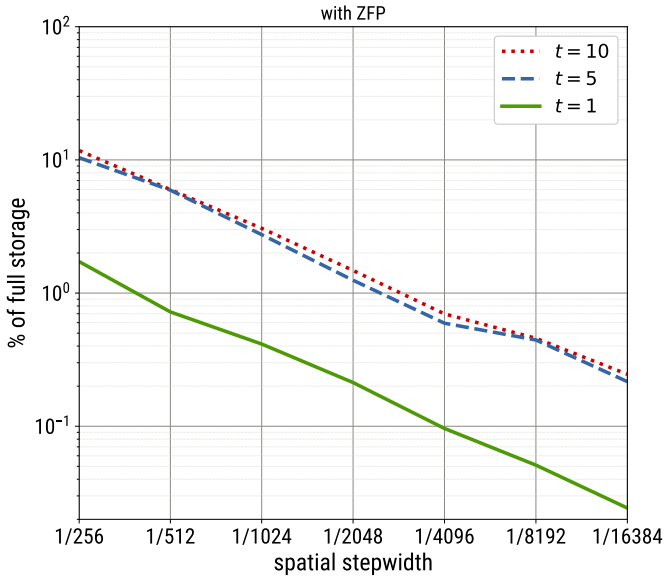
Wave Equation

Dependence on Spatial Resolution



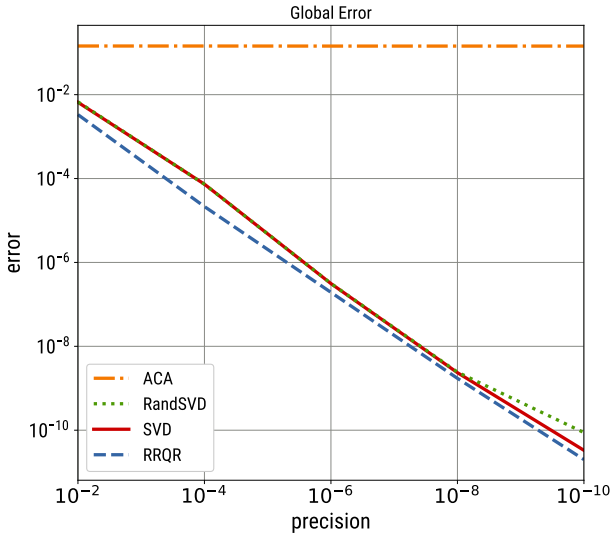
Wave Equation

Dependence on Spatial Resolution



Wave Equation

Comparison of Low-Rank Approximation Algorithms



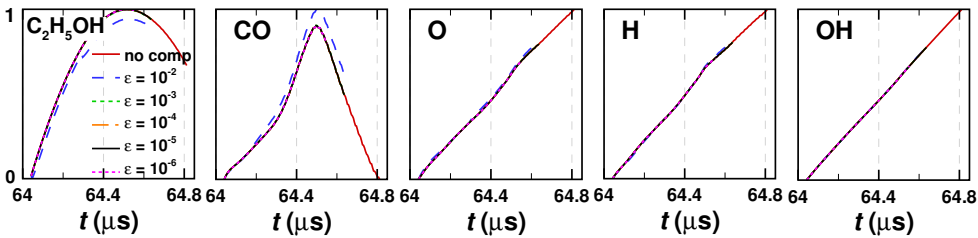
[1] Massei, S. and Robol, L. and Kressner, D.: "Hierarchical adaptive low-rank format with applications to discretized partial differential equations". Num. Lin. Alg. with App. (2022)

Combustion Application

Separate compression of 44 primitive variables:

- temperature, pressure, velocity and
- chemical species.

Dataset size is $20k \times 20k$ per variable.

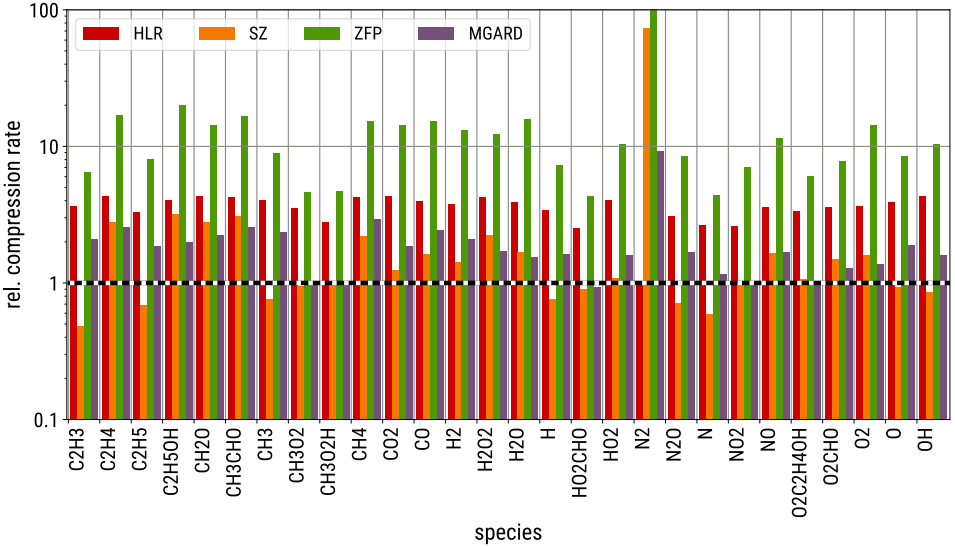


Profile reproduction with $\varepsilon \leq 10^{-3}$, using $\varepsilon = 10^{-4}$ for compression.

HLRcompress: using RandSVD, $n_{\text{tile}} = 256$.

Combustion Application

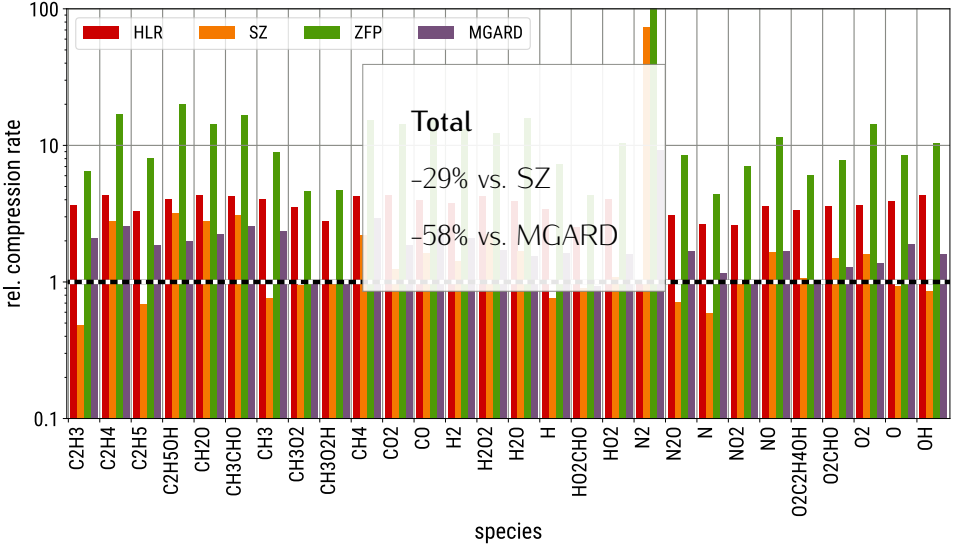
Compression Rate



Baseline: HLRcompress

Combustion Application

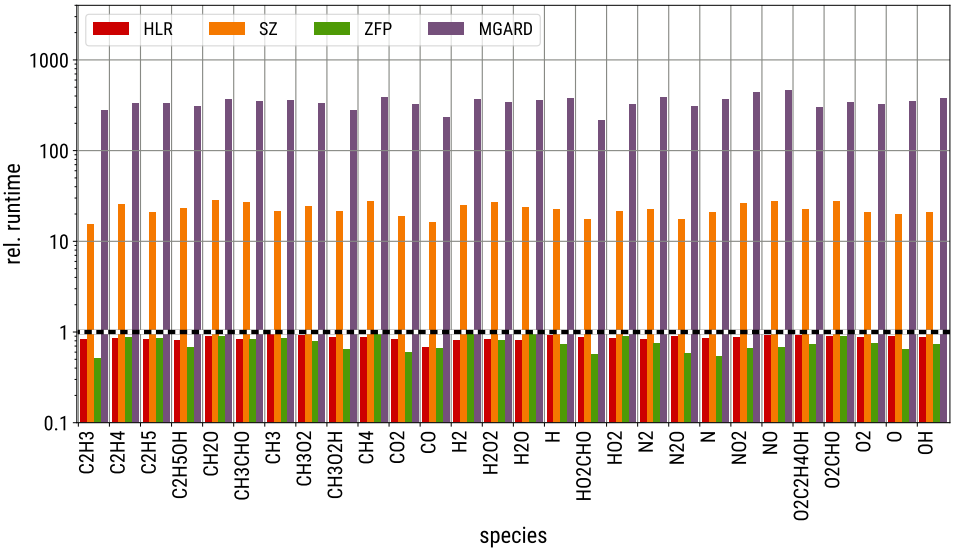
Compression Rate



Baseline: HLRcompress

Combustion Application

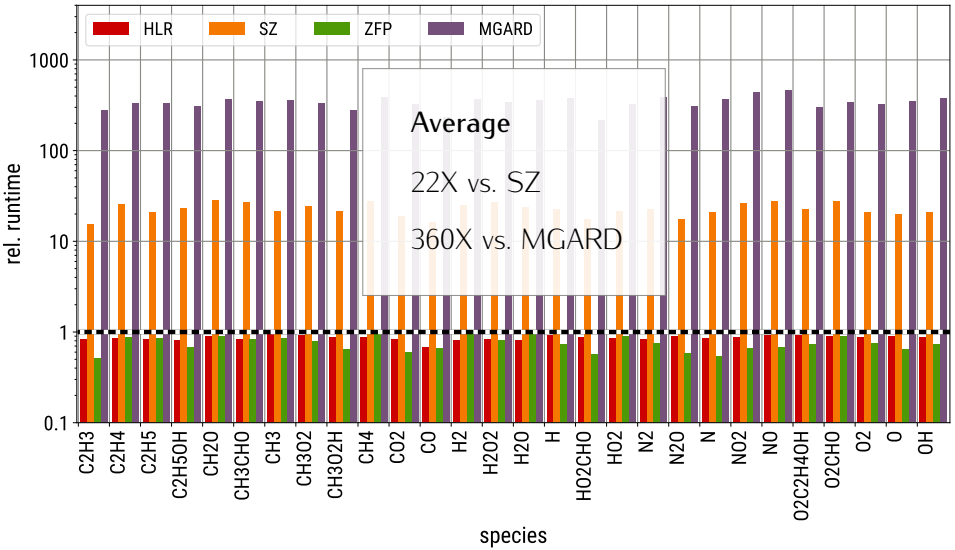
Runtime



Baseline: HLRcompress

Combustion Application

Runtime



Baseline: HLRcompress

Conclusion & Outlook

Conclusion & Outlook

HLRcompress proved to be

- very efficient for 2D simulation data sets,
- fast and highly parallel,
- efficiently ported to GPUs.

Source code available via

https://gitlab.mis.mpg.de/rok/HLRcompress

[1] Ehrlacher, V., Grigori, L., Lombardi, D., Song, H.: "Adaptive Hierarchical Subtensor Partitioning for Tensor Compression", SIAM J. on Sci.Comp. 43, 139–163 (2021)

Conclusion & Outlook

HLRcompress proved to be

- very efficient for 2D simulation data sets,
- fast and highly parallel,
- efficiently ported to GPUs.

Source code available via

`https://gitlab.mis.mpg.de/rok/HLRcompress`

What's next?

- tensor compression for higher dimensional data[1],
- integration of *arithmetic*.

[1] Ehrlacher, V., Grigori, L., Lombardi, D., Song, H.: "Adaptive Hierarchical Subtensor Partitioning for Tensor Compression", SIAM J. on Sci.Comp. 43, 139–163 (2021)

Thank You

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CCRC
Clean Combustion
Research Center

MAX PLANCK INSTITUTE
FOR MATHEMATICS IN THE SCIENCES