Performance of numerical algorithms for low-rank tensor operations in tensor-train / matrix-product-states format

Melven Röhrig-Zöllner

German Aerospace Center (DLR), Institute for Software Technology

DLR

Content

- Introduction
- Tensor-train / matrix-product states for data compression
- Solving linear systems in tensor-train / matrix-product-states format
- Underlying linear algebra operations
- Conclusion

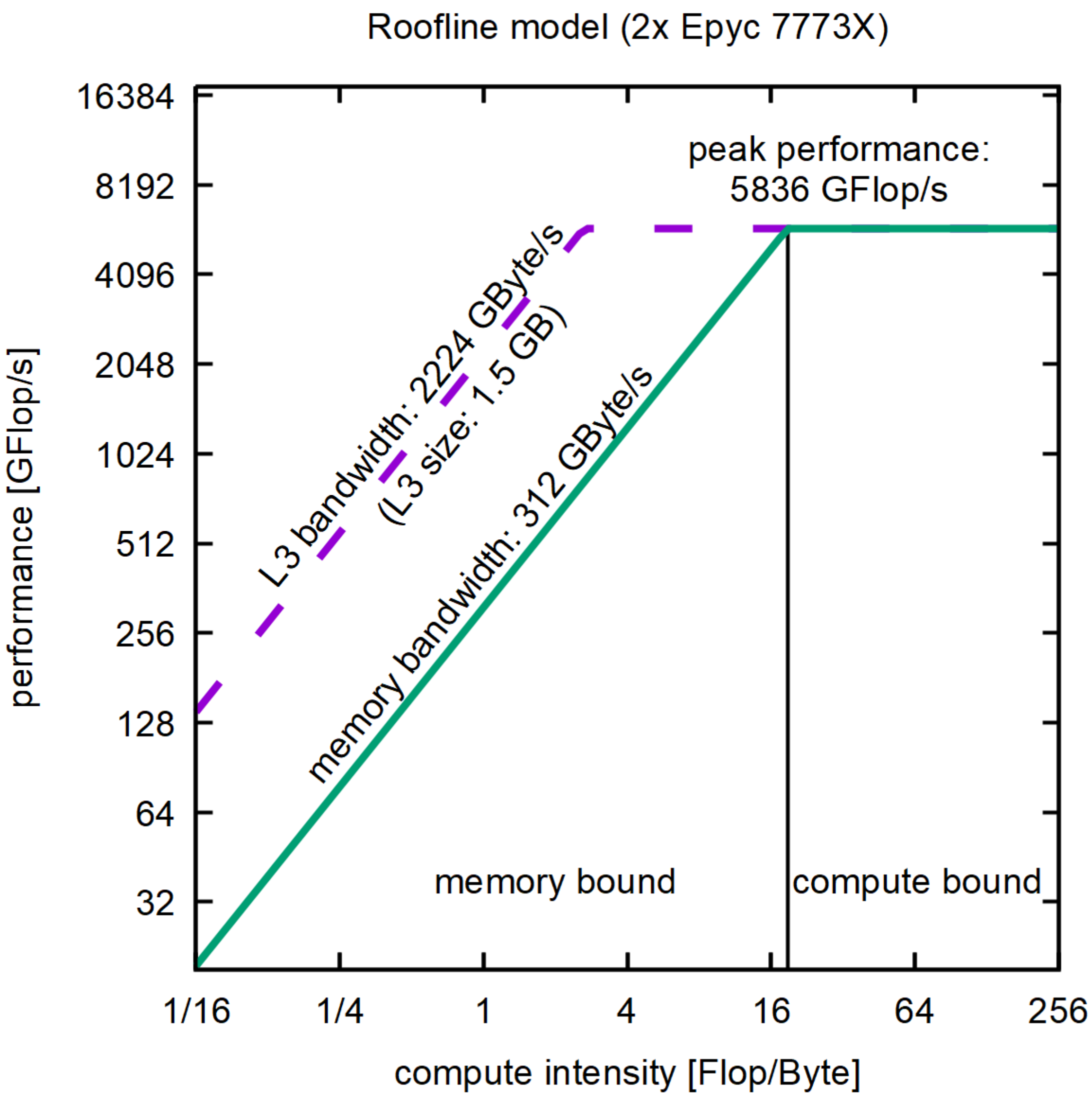
Introduction: performance engineering

- **Levels of parallelism:**
  - single-core: ~100 flop per cycle "on the fly"
  - multi-core: e.g. 128 cores (shared memory)
  - multi-node: cluster of nodes (fast network)

- **Data transfers vs. arithmetic operations:**
  Computer model:
  1. load n bytes from memory
  2. perform k arithmetic operations (Flop)
  3. store m bytes in memory

  → Compute intensity: $I_c = \frac{k}{n+m}$

- **Expected (ideal) runtime:**
  - memory-bound ($I_c < 16$): $t = (n+m)/\text{bandwidth}$
  - compute-bound ($I_c > 16$): $t = k/\text{performance}$



Introduction: linear algebra

Matrix-matrix product (GEMM):

$$\underset{(n \times k)}{C} \leftarrow \underset{(n \times m)}{A} \; \underset{(m \times k)}{B}$$

- $2nmk$ flops, $8(nk + nm + mk)$ data transfers
  - *compute bound* for $n \approx m \approx k \gg 100$
  - *memory bound* for $\min(n, m, k) \lesssim 100$

QR decomposition:

$$QR = A,$$

with $Q^T Q = I$, and R upper triangular.

- $O(nm^2)$ flops, $O(nm)$ data transfers
  - *memory bound* for $m \lesssim 100$

  → tall-skinny QR (TSQR)

Singular Value Decomposition (SVD):

$$A = USV^T,$$

with $U^T U = I$, $V^T V = I$, and $S = \mathrm{diag}(\sigma_1, \ldots, \sigma_m)$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m \geq 0$.

- $O(nm^2)$ flops (usually: runtime $t_{\mathrm{SVD}} \gg t_{\mathrm{QR}} > t_{\mathrm{GEMM}}$)

- Truncated SVD → best rank-r approximation:

$$\|A - \hat{B}\|_F = \min_{\mathrm{rank}(B) \leq r} \|A - B\|_F$$

with $\hat{B} = U \; \mathrm{diag}(\sigma_1, \ldots, \sigma_r, 0, \ldots, 0) \; V^T$

Tensor network notation

Tensor = multi-dimensional array:

$$\boldsymbol{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$$

- Contraction: sum over product of entries
  → generalization of matrix-matrix product
  example (contraction of 3rd dim. of $\boldsymbol{X}$ and 2nd of $\boldsymbol{Y}$)

$$Z_{i_1,i_2,i_4,j_1,j_3} = \sum_{i_3} X_{i_1,i_2,\boldsymbol{i_3},i_4} Y_{j_1,\boldsymbol{i_3},j_3}$$

- In 2d: all contractions with transpose+GEMM:

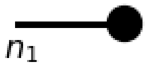
$$AB, A^T B, AB^T, A^T B^T$$

- > 2d: too many combinations!
  → tensor network notation

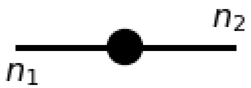
Tensor network notation (from physics!):

Scalar (dot):

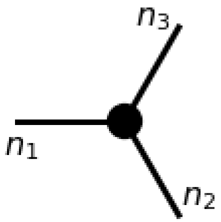
Vector (one leg):



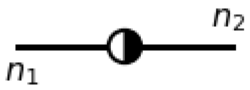
Matrix (two legs):



3d tensor (3 legs):



Matrix with orthogonal columns:

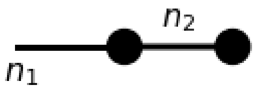


Tensor network notation (cont.)

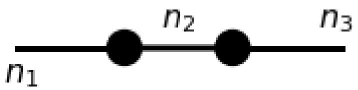
Contractions:

Vector dot product:

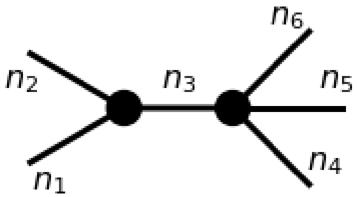
Matrix-vector product:



Matrix-matrix product:

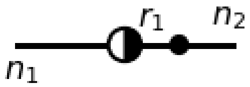


Contraction of a 3d and a 4d tensor:



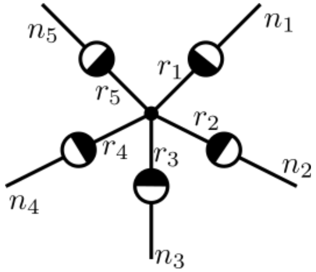
Decompositions:

QR:



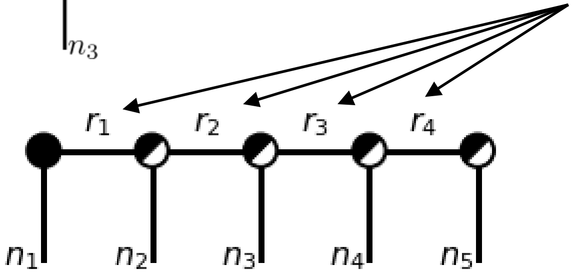
SVD:

Orthogonal Tucker (5d):



ranks / bond-dimensions

Tensor-train (5d) / matrix product states:



Content

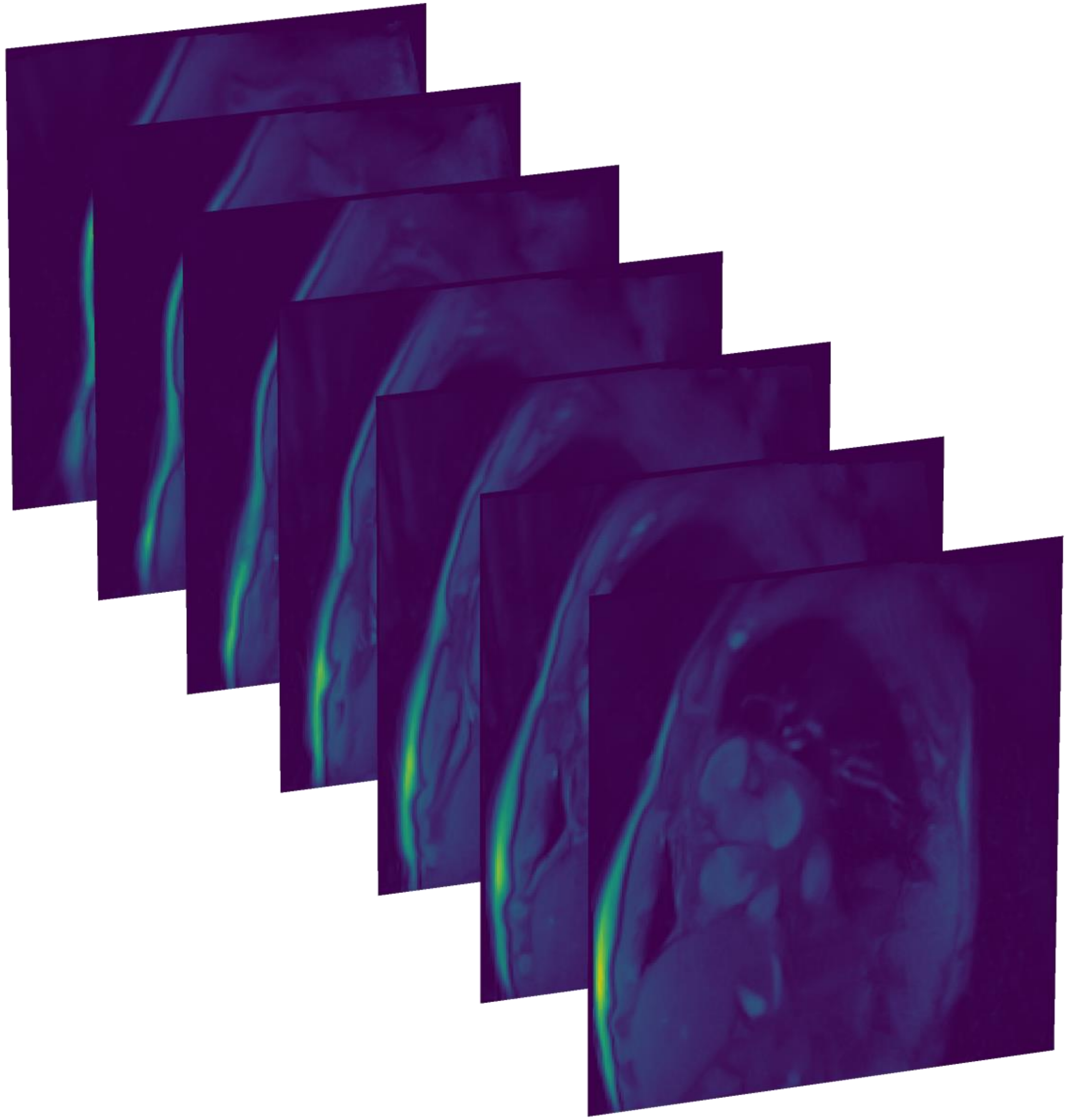
Data compression with TT-SVD: motivation

- Input: large, dense, high-dimensional data

- Example: Cardio MRT videos:
    - Multiple patients
    - Multiple views/cuts
    - Slices in z direction
    - Finer resolution in x/y direction
    - Videos over time

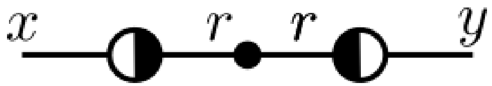
- Data looks very “similar” in all directions

- Goals:
    1. Compress data in tensor-train format
    2. Data analysis with compressed data

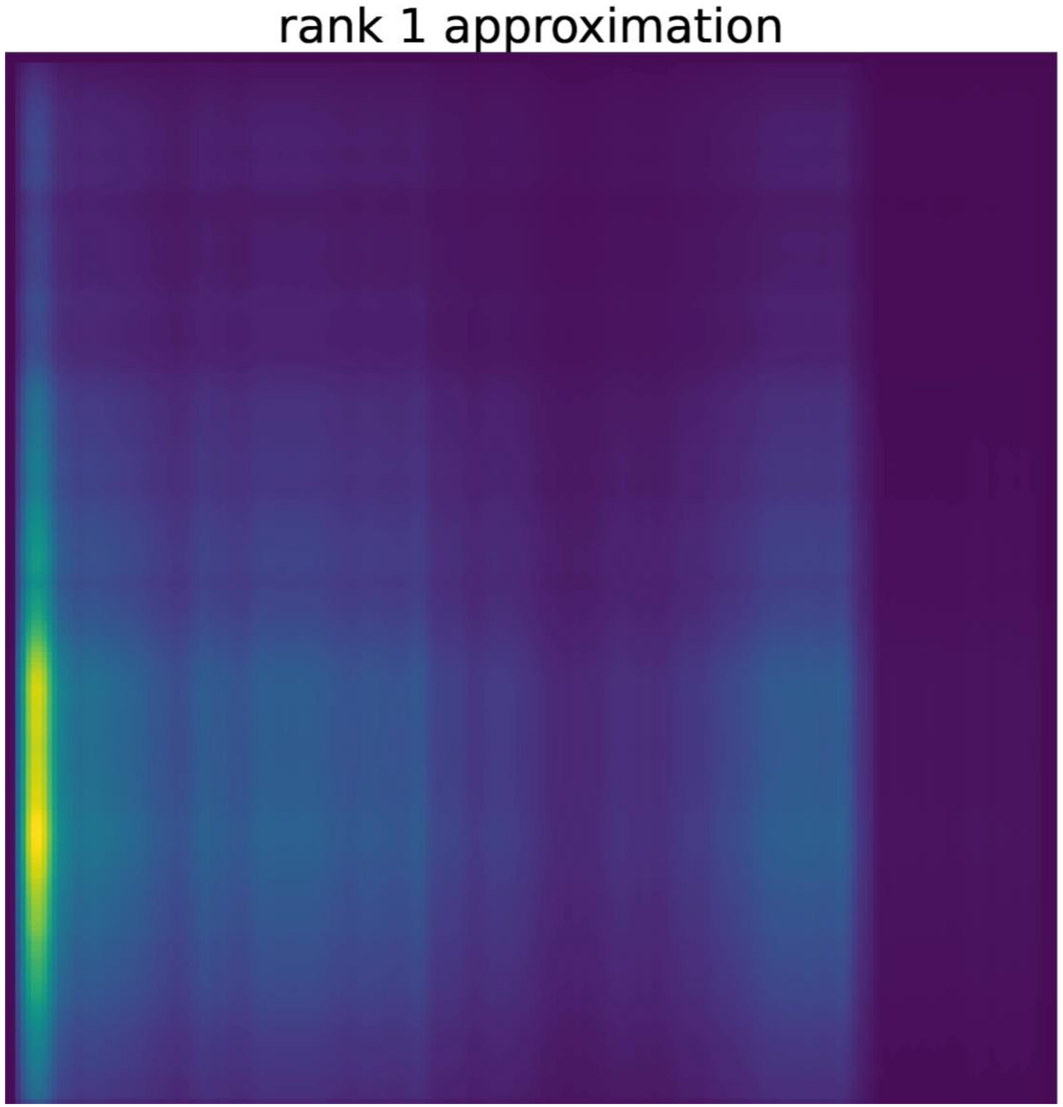


Truncated SVD in 2d

- Interpret picture as matrix
- Approximate matrix by lower rank:

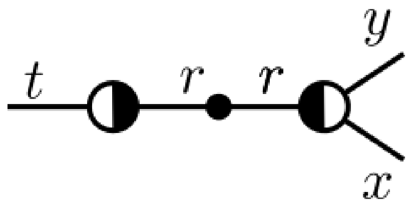


Data size
uncompressed: 160 kB
compressed: $1.6r$ kB



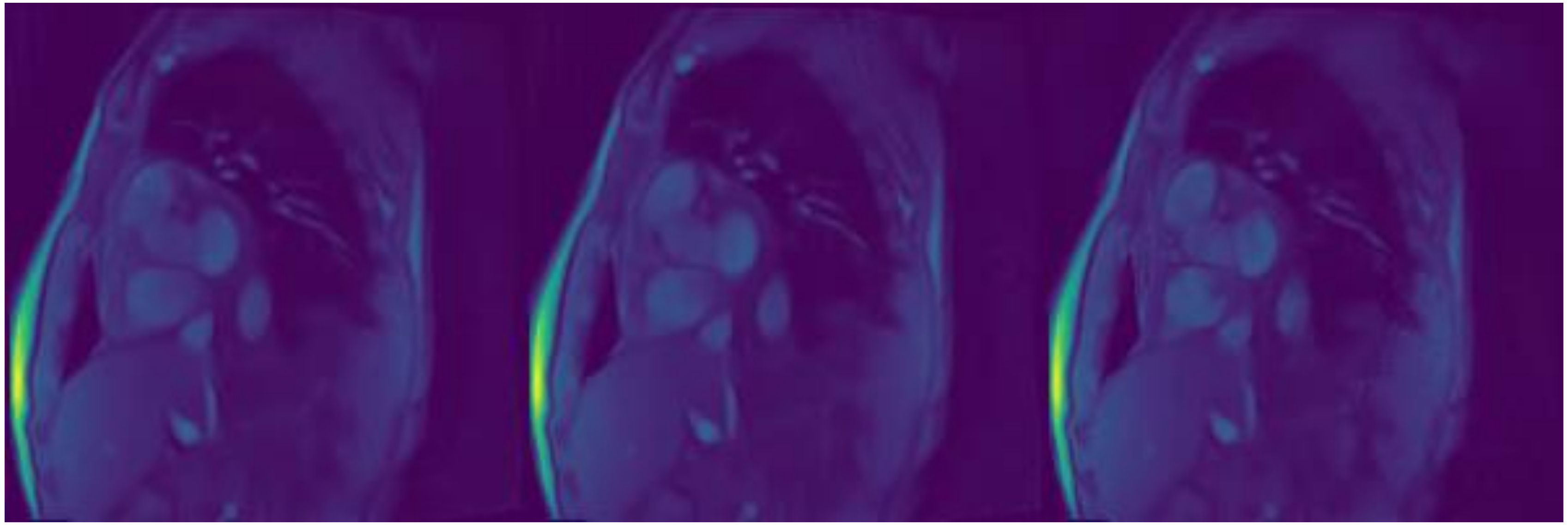
Truncated SVD over time

- We can do the same for a movie over time…
- Interpret movie as matrix (time x pixels)
- Approximate matrix by lower rank:



Data size
uncompressed: 48 MB
compressed: $\sim 161r$ kB

rank 3 | rank 10 | original

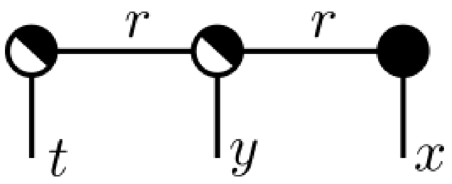


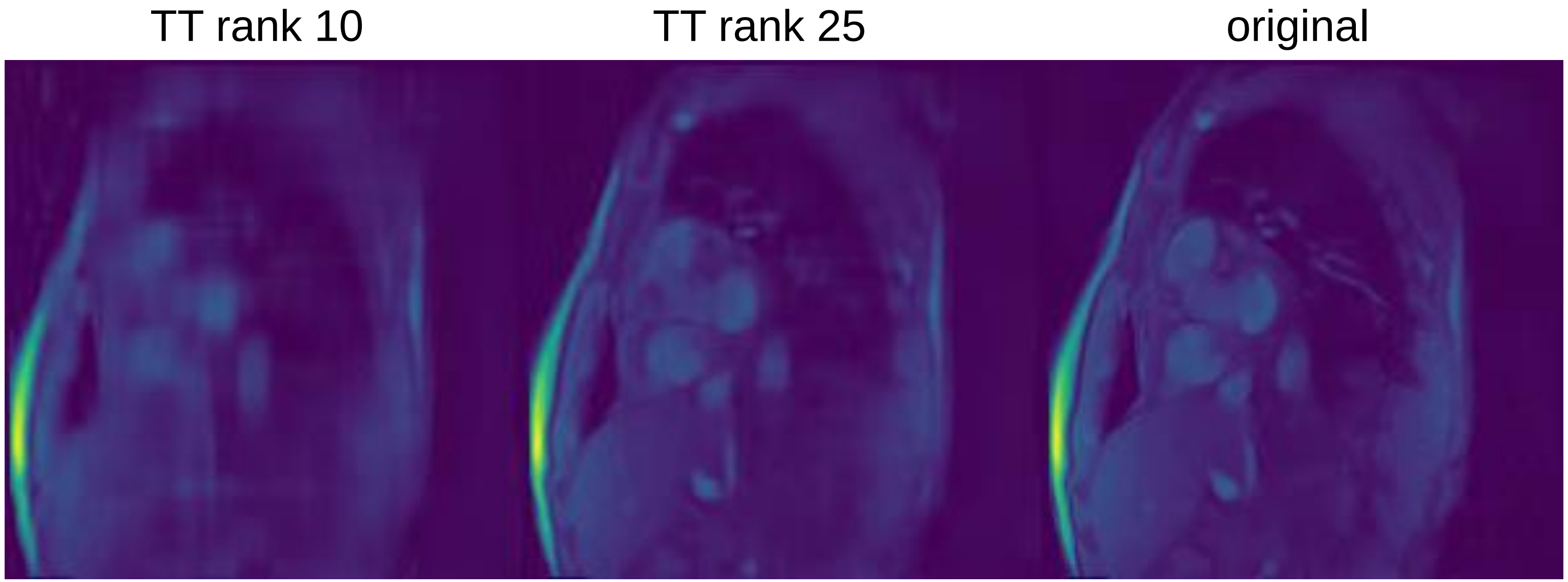
Truncate in 2d and over time → tensor-train / MPS

Data size

uncompressed:	48 MB
TT rank 10:	100 kB
TT rank 25:	550 kB

- We can combine both ideas…
- First truncate over time, then in x/y direction:





Tensor-train decomposition / MPS for data compression

Lossy global compression

ranks / bond-dim. $r_1, \dots, r_{d-1}$:

higher ranks → more accurate
(but bigger)

"considers all data at once"

complete TT needed to
decompress one element

Storage complexity: $O(dnr^2)$

vs. $O(n^d)$ uncompressed

Performance of the tensor-train SVD

Setup: given dense $X \in \mathbb{R}^{n^d}$, desired rank $r_{\max}$

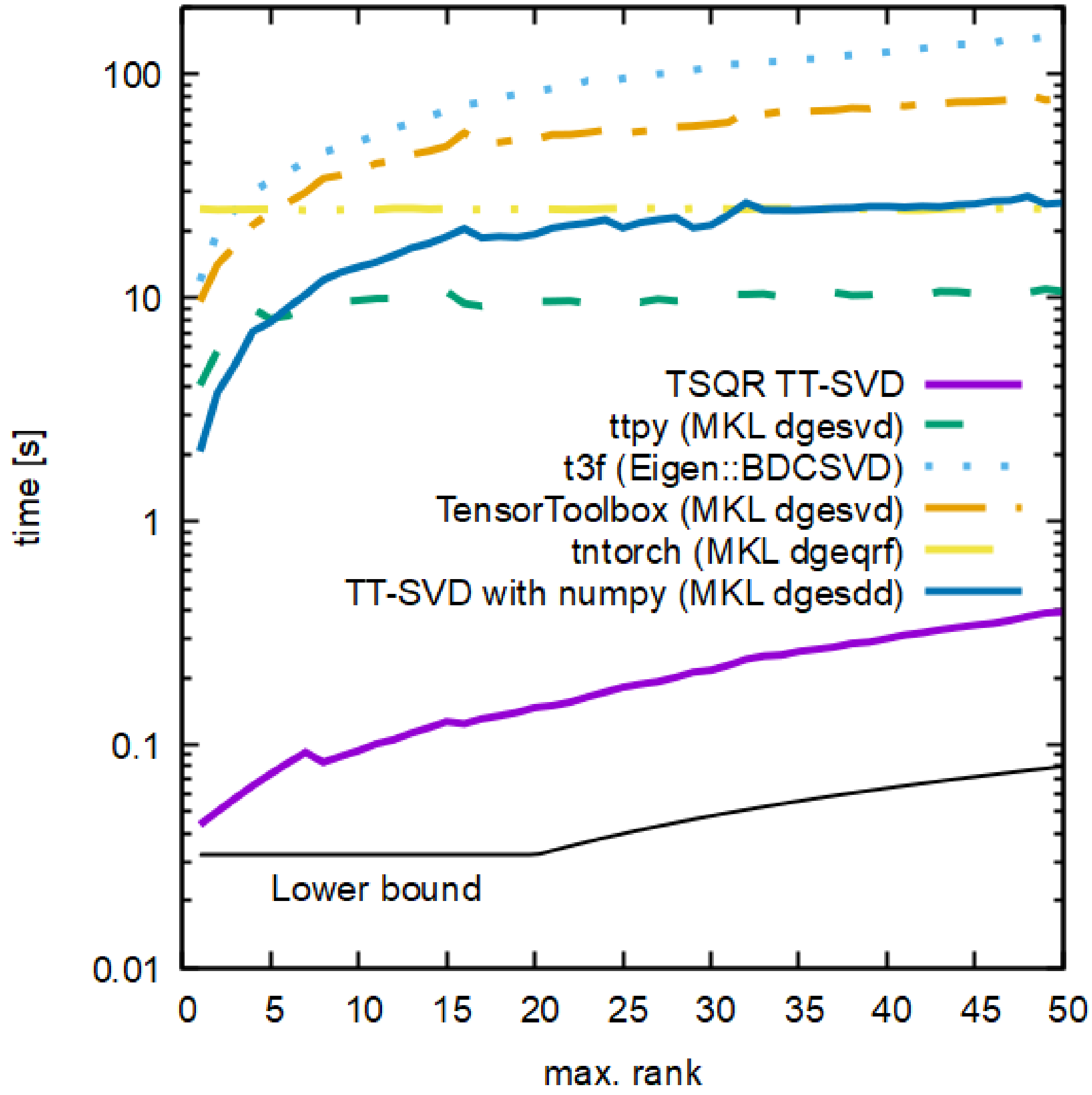
Core idea:

- neat combination of (tall-skinny) matrix operations (TSQR, SVD, fused GEMM+reshape)

Results:

- *Memory bound* for small $r_{\max}$ (transfer volume $\sim 2.2 n^d$ values)
- *Compute bound* for $r_{\max} \gtrsim 100$ (operations $\lesssim 12 n^d r_{\max}$ Flops)
- **Common implementations are too slow** (factor ≥ 50)

Röhrig-Zöllner et.al.: Performance of the low-rank TT-SVD for large dense tensors on modern multi-core CPUs, SISC, 2022 (doi: 10.1137/21m1395545)



TT-SVD of a 2^{27} tensor on a 14-core Intel Skylake Gold 6132

TT-SVD for a real-world data set

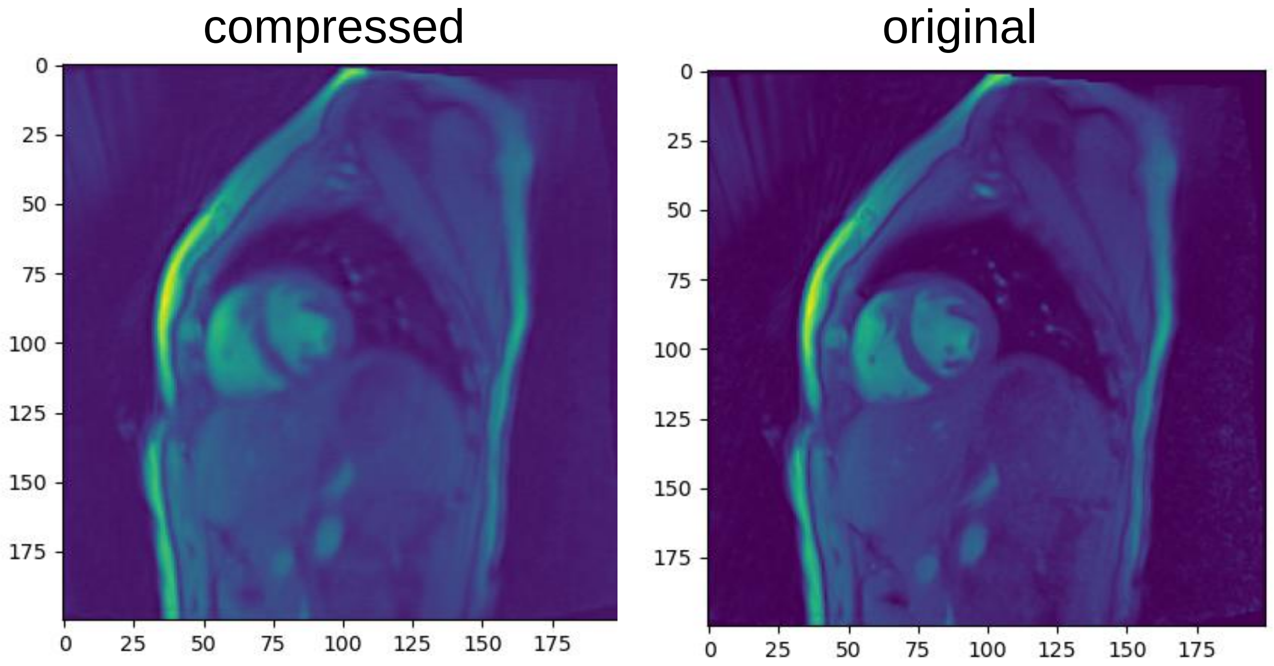
Data size
Uncompressed: 8.7 GB (uint16)
17.3 GB (float)
Compressed: 124 MB
$(\epsilon = 0.002)$

Cardio-MRT data (subset):

$19 \times 19 \times 300 \times 200 \times 200$

patients slices frames x-dir. y-dir.

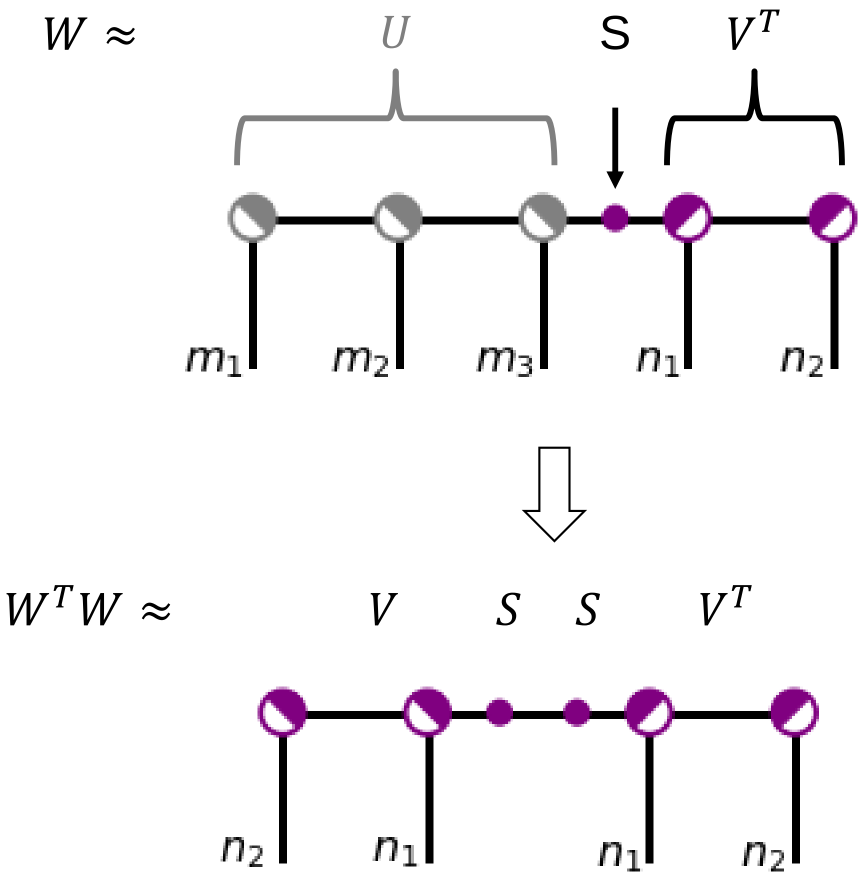
(complete data set: more patients × different views)




```
TensorTrain_float
 with dimensions = [19, 19, 5, 5, 3, 2, 2, 2, 2, 2, 5, 5, 2, 2, 2, 5, 5]
           ranks = [19, 361, 711, 1336, 1565, 1267, 861, 943, 1177, 1382, 693, 187, 100, 50, 25, 5]
```

Example for data analysis step in TT format: extract distances

- Given: vectors $(w_1, \dots, w_n) =: W,\ w_i \in \mathbb{R}^m$
- Calculate the pair-wise distances of all vectors:
  $$d_{ij}^2 = \|w_i - w_j\|_2^2 = \|w_i\|_2^2 - 2\langle w_i, w_j\rangle + \|w_j\|_2^2$$
- Or their cosine similarities: $s_{ij} = \frac{\langle w_i, w_j\rangle}{\|w_i\|_2 \|w_j\|_2}$
- Both require $O(mn^2)$ operations
- Alternative approach:
  - Compress in TT format: $O(mnr)$ operations
  - Rearrange sub-tensors
    ➔ approximation for cosine similarities

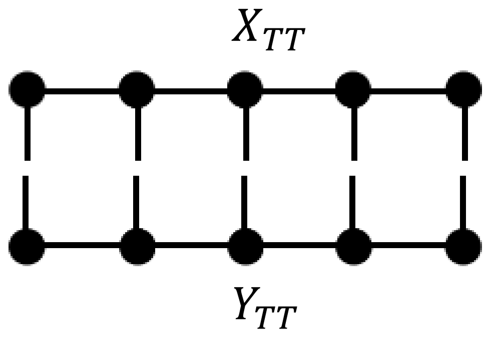


Content

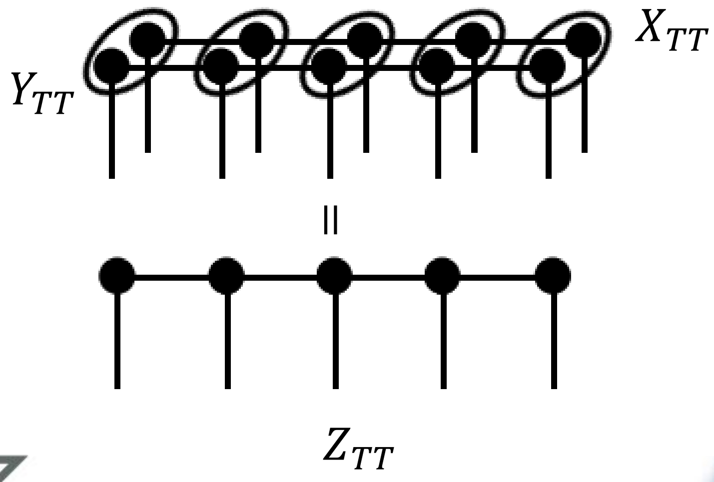
- Introduction
- Tensor-train / matrix-product states for data compression
- Solving linear systems in tensor-train / matrix-product-states format
- Underlying linear algebra operations
- Conclusion

Linear algebra in tensor-train / matrix-product-states format

Scalar product: $\alpha \leftarrow \langle X_{TT}, Y_{TT} \rangle$

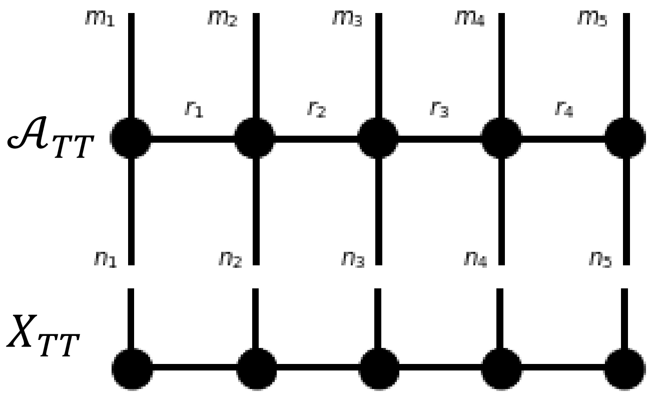


Addition: $Z_{TT} \leftarrow X_{TT} + Y_{TT}$

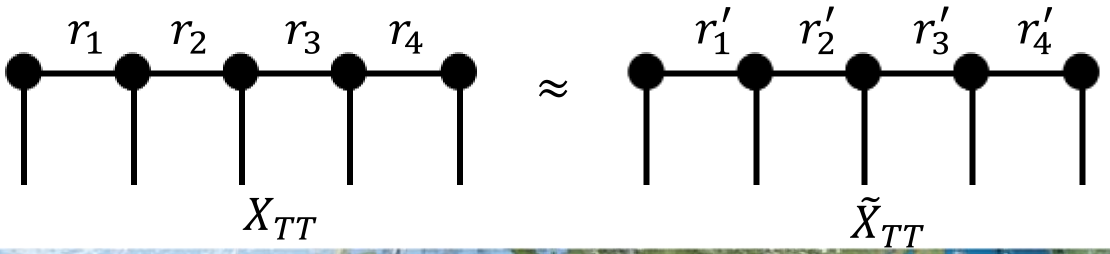


➔ "Normal linear algebra in TT format"
(e.g. linear systems, eigenvalue problems)

Matrix-vector product: $Y_{TT} \leftarrow \mathcal{A}_{TT} X_{TT}$



Truncation: find $\tilde{x}$ with smaller ranks



Linear solvers in TT format: motivation

Solve large-scale linear systems
infeasible to store solution as a “dense” vector
(e.g., operator dimension: $\mathbb{R}^{100^{10} \times 100^{10}}$)

Possible applications:
- High-dim. / parameterized / stochastic PDEs
- Large-scale optimal control problems
- Problems from quantum physics:
  often Hermitian eigenvalue problems
  (closely related to linear systems but not discussed here!)
- Data science: e.g., compressed sensing

Problem definition

Given:
- Low-rank linear operator: $\mathcal{A}_{TT} \in \mathbb{R}^{n^d} \rightarrow \mathbb{R}^{n^d}$
- Low-rank right-hand side: $B_{TT} \in \mathbb{R}^{n^d}$
- Desired tolerance: ϵ_{tol}

Calculate:
- Iterative solution X^*_{TT} with
  $$\|\mathcal{A}_{TT} X^*_{TT} - B_{TT}\|_* \leq \epsilon_{\text{tol}}$$
- (choice of norm $\|\cdot\|_*$ depends on solution method)

Focus here on $n \gg 2$, e.g. dimensions 50^{10}
(performance characteristics differ for 2^N!)

Common algorithms: TT-GMRES, TT-MALS, TT-AMEn

Methods:

- "Global": TT-GMRES
  GMRES applied in TT format with additional truncations
- "Local" – projection onto 2 sub-tensors: TT-MALS
  (like DMRG but formulated for generic linear systems)
- "Local" – projection onto 1 sub-tensor: TT-AMEn
  (subspace "enriched" with projected residuals)

Inner-outer iteration schemes:
Outer "sweeps":
moving subspace correction
→ e.g. fix all but 2 sub-tensors
Inner "iteration":
GMRES/CG solver (or TT-GMRES)

Required FP operations (in my tests):
(simple PDE, $20^6 \leq n^d \leq 100^{14}$, $r < 100$, $\epsilon_{\text{tol}} = 10^{-8}$):

- TT-GMRES about 100x slower than TT-MALS
- TT-MALS about 100x slower than TT-AMen

Explanation: intermediate steps with higher ranks than X^*_{TT}!

TT-GMRES: $O(dnr_{\max}^3 + dn^2 r_{\max}^2)$

TT-MALS: $O(dnrr_{\max}^2 + dn^2 rr_{\max})$

TT-AMEn: $O(dnr^3 + dn^2 r^2)$

Preconditioning

Common practice for **sparse solvers**…

Required properties:

1. $\mathrm{cond}(PA) \ll \mathrm{cond}(A)$
2. cheap $y \leftarrow Px$ (apply precond. to vector)

Different variants:

- left preconditioning: $PAx = Pb$
- right preconditioning: $APy = b$, $x = Py$
- two-sided precond.: $P_L A P_R y = P_L b$, $x = P_L y$

Need a few more constraints for **TT solvers**!

Desired properties ("global" preconditioner):

1. $\mathrm{cond}(\mathcal{P}\mathcal{A}_{TT}) \ll \mathrm{cond}(\mathcal{A}_{TT})$
   (fewer total (inner) iterations)
2. $\mathrm{rank}(\mathcal{P}\mathcal{A}_{TT}) \approx \mathrm{rank}(\mathcal{A}_{TT})$
   (complexity is cubic in the rank)
3. "make the operator more symmetric"
   (better convergence / possibly smaller $r_{\max}$)
4. "preserve problem structure"

Suggestion: simple rank-1 preconditioner

Idea:

- approximate TT-operator with rank 1: $\tilde{\mathcal{A}}_{TT} \approx \mathcal{A}_{TT}$ with $\operatorname{rank}(\tilde{\mathcal{A}}_{TT}) = 1$
- Rank-1 inverse: $\left(\tilde{A}_1 \otimes \tilde{A}_2 \otimes \cdots \otimes \tilde{A}_d\right)^{-1} = \tilde{A}_1^{-1} \otimes \tilde{A}_2^{-1} \otimes \cdots \otimes \tilde{A}_d^{-1}$

Two-sided preconditioned operator (for symm. problems $\mathcal{L}_{TT}^T = \mathcal{R}_{TT}$):

$$\mathcal{L}_{TT}\mathcal{A}_{TT}\mathcal{R}_{TT}$$

using SVDs $\tilde{A}_k = U_k S_k V_k^T$:

$$L_k = S_k^{-1/2} U_k^T, \qquad R_k = V_k S_k^{-1/2}$$

➔ Combines properties 1, 2, 3 but not 4.

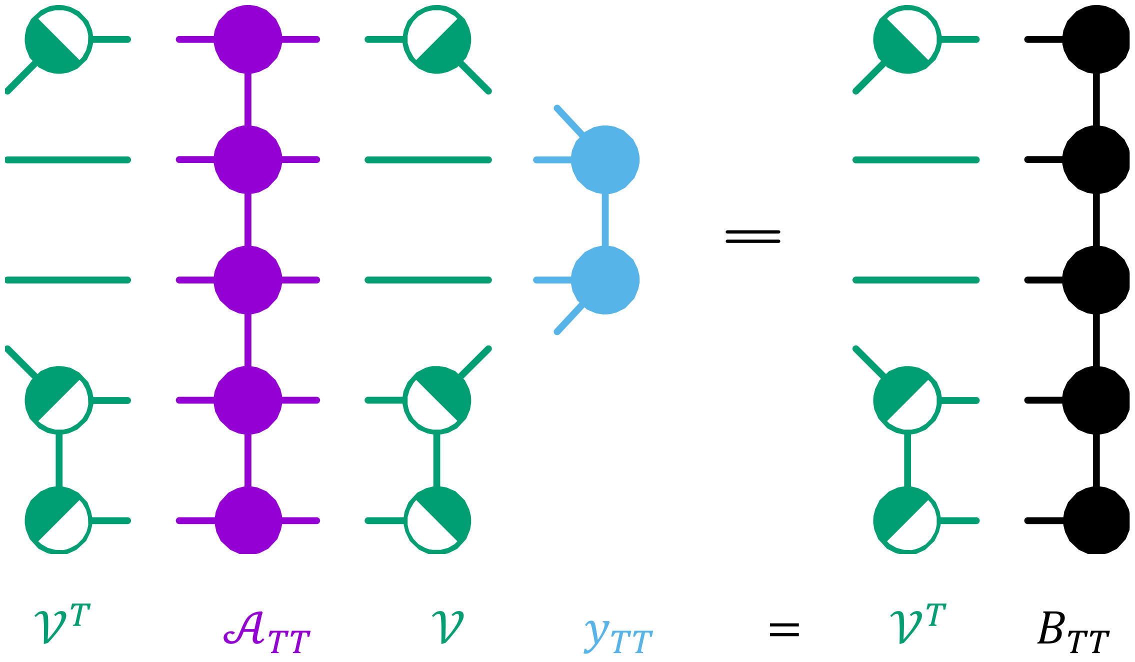
Generic, fast and works well in my test cases.

Replace by problem-specific preconditioner if possible!

TT-MALS projection

Idea:
- Vary only (X_k, X_{k+1})
  (keeping $X_1, \dots X_{k-1}, X_{k+2}, \dots, X_d$ fixed)
- Minimize energy:
  $$J(X_{TT}) \coloneqq 0.5\langle X_{TT}, \mathcal{A}_{TT} X_{TT}\rangle - \langle X_{TT}, B_{TT}\rangle$$
- Sweep over dimensions $(k \leftarrow k \pm 1)$

Resulting "local" problem



Properties:
- $\mathcal{V}^T\mathcal{V} = I$ with $\mathcal{V}y_{TT} = X_{TT}$

For sym. pos. def. operator $\mathcal{A}_{TT}$:
- Minimizes $\|X_{TT} - X_{TT}^*\|_{\mathcal{A}_{TT}}$
- $\text{cond}(\mathcal{V}^T\mathcal{A}_{TT}\mathcal{V}) \leq \text{cond}(\mathcal{A}_{TT})$

Idea for non-symmetric projection

Sym. projection sub-optimal for non-sym. operator!
➔ use $\mathcal{W}^T \mathcal{A}_{TT} \mathcal{V}$ with $\mathcal{W} \neq \mathcal{V}$

Idea:

- Try to build $\mathcal{W}$ to span directions of $\mathcal{A}_{TT}\mathcal{V}$

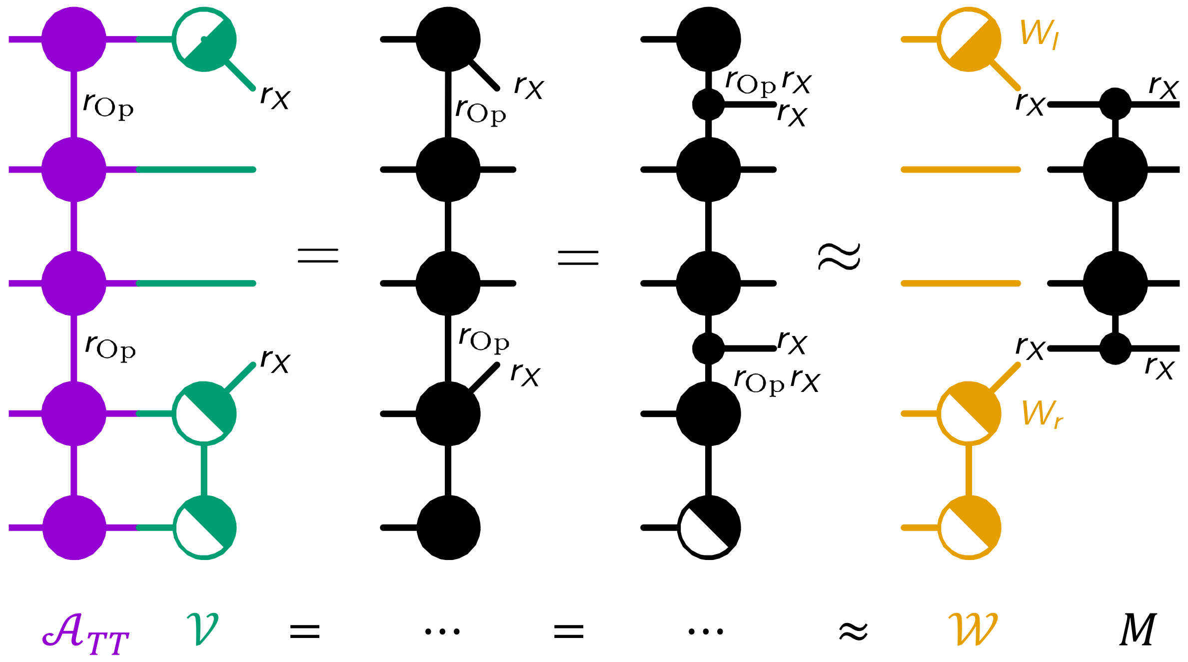
Properties:

- $\mathcal{W}^T\mathcal{W} = I$ with $\mathcal{A}_{TT}\mathcal{V} \approx \mathcal{W}M$
- Solution $\mathcal{W}^T\mathcal{A}_{TT}\mathcal{V}y_{TT} = \mathcal{W}^T B_{TT}$ approximates:

$$\min_{y_{TT}} \|\mathcal{A}_{TT}\mathcal{V}y_{TT} - B_{TT}\|_F$$

- W_l, W_r chosen to make $\mathcal{W}^T\mathcal{A}_{TT}\mathcal{V}$ more normal

Derivation of the “local” operator



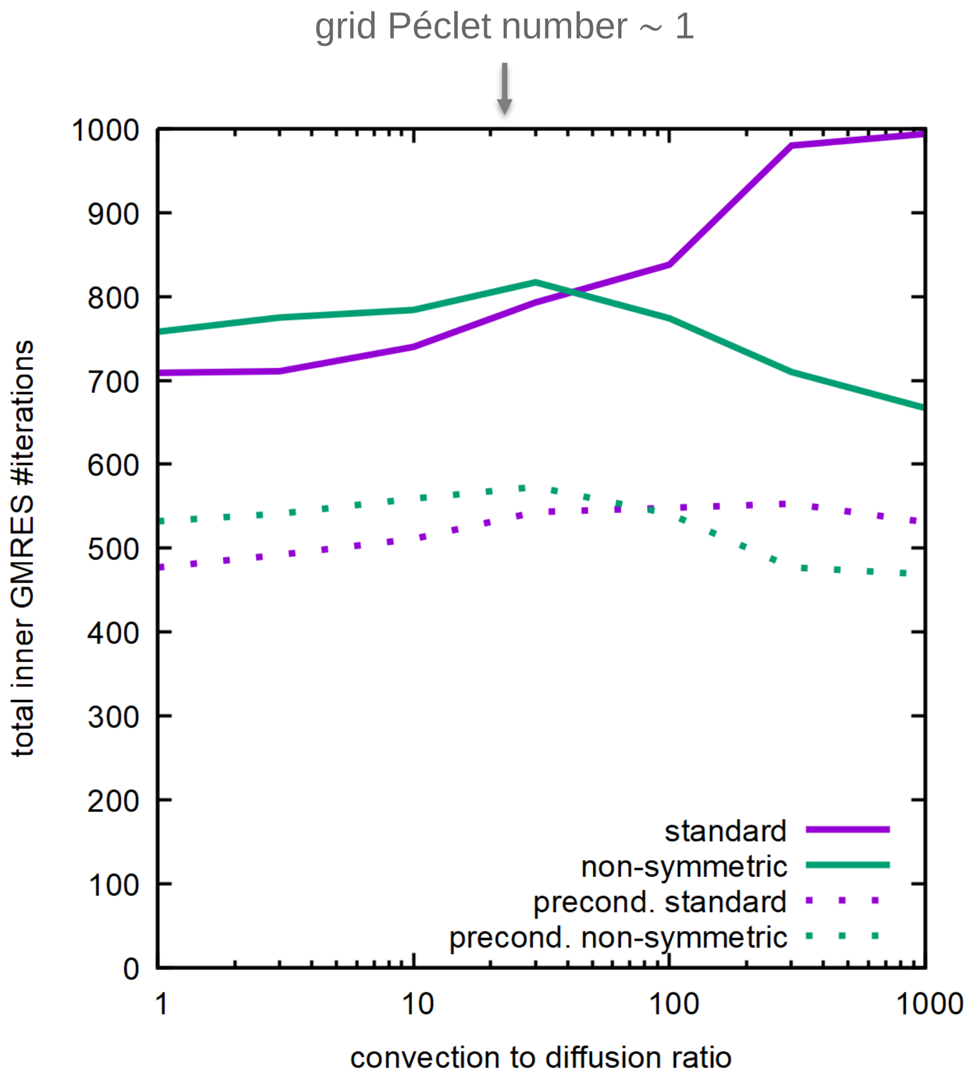
Results with non-symmetric projection

Setup:

- Simple PDE (dimensions 20^{10})
- TT-MALS with inner TT-GMRES
- Varying asymmetry (conv. to diff. ratio)

Observation:

- Alternative projection beneficial for **strongly** non-symmetric problems



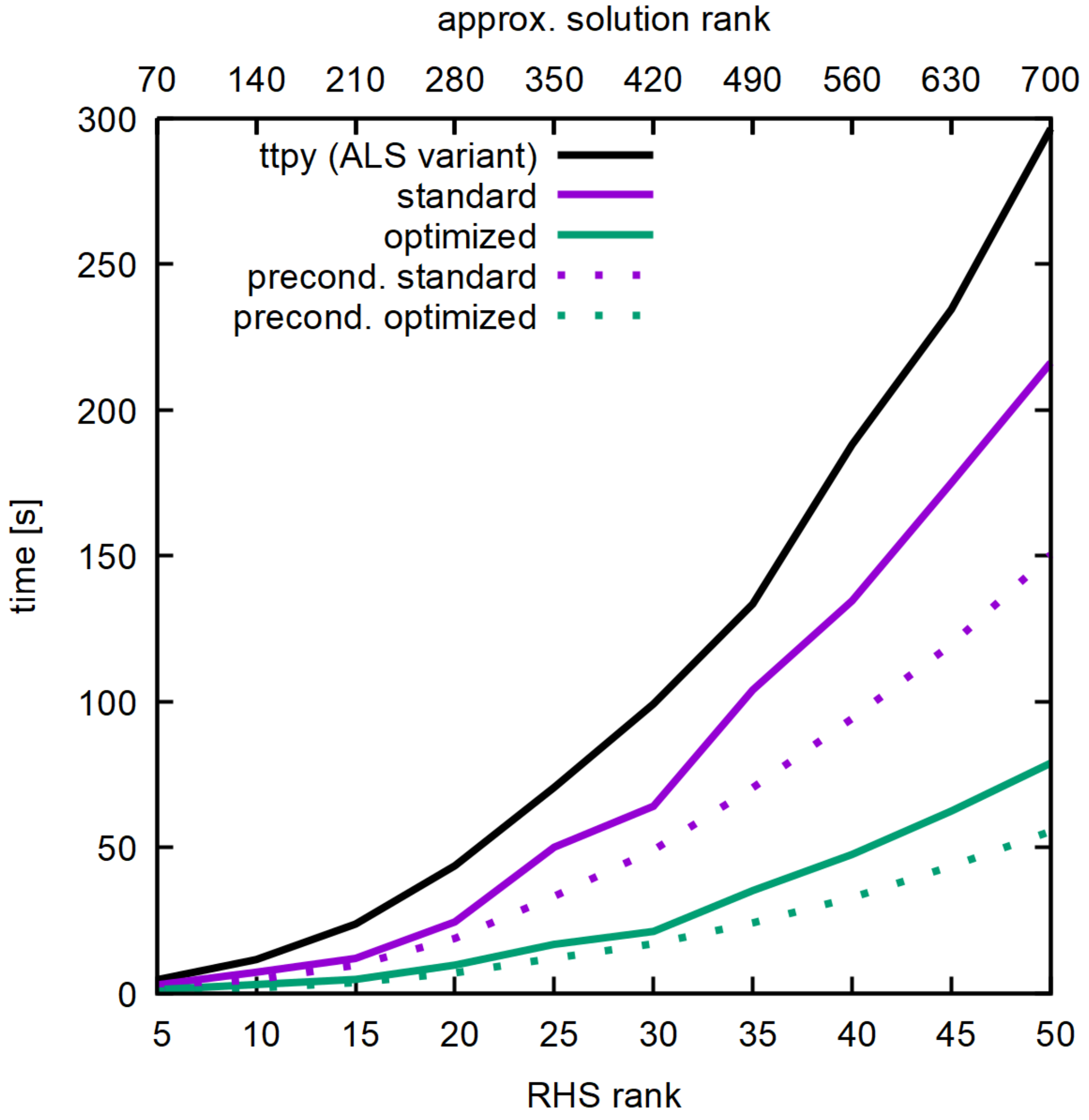
TT-AMEn performance

Setup: simple PDE (conv.-diff. ratio = 10, $\epsilon_{\mathrm{tol}} = 10^{-8}$)

Core ideas:
- Simple preconditioner
- Improved orthogonalization and SVD steps (with TSQR)
- Faster contractions (dim. reordering + padding)

Results:
- *Not 1 dominating part in the algorithm*
  → needs combination of improvements!
- **Significant speedup (factor ~5)**

Remark:
- Tweaked ttpy version with fast underlying BLAS (MKL)



TT-AMEn for a 50^{10} problem on a 64-core AMD EPYC 7773X

Content

Underlying linear algebra operations: motivation

High speedups: 50x (TT-SVD), 5x (TT-AMEn) (on the same hardware!)

Fair comparisons:

- No comparison vs. “unoptimized” code! → all implementations call BLAS/LAPACK
- Use the same (multi-threaded) BLAS/LAPACK library (MKL with workaround for AMD)
- Except for some specialized operations…
  - Q-less tall-skinny QR (“TSQR” that only returns R)
  - Fused tall-skinny GEMM+reshape
  - Fused axpy+dot

So what did I change?

- Improve mapping of high-level operations to “building blocks”
- Exploit specialized operations (less generic/accurate than BLAS/LAPACK for all inputs)
- Improve data layout (BLAS/LAPACK must work with what it gets…)

Building blocks of one TT-SVD "step"

Standard implementation (large SVD for each step):

Given tall-skinny $X \in \mathbb{R}^{n \times k}$, calculate:

$$X = USV^T,$$
$$Q \leftarrow V_{:,1:r},$$
$$B \leftarrow U_{:,1:r} S_{1:r,1:r},$$
$$X' \leftarrow \text{reshape}(B, \dots)$$

Underlying operations:

- SVD $(n \times k)$
- copy $(k \times r)$
- r axpy (n)
- reshape $(n \times r)$

Actual problem: calculate Y, Q with $Q^T Q = I$:

$$\|X - BQ^T\|_F \leq \tau,$$
$$X' \leftarrow \text{reshape}(B, \dots)$$

Optimized implementation:

$$X = QR,$$
$$R = USV^T,$$
$$Q \leftarrow V_{:,1:r},$$
$$X' \leftarrow \text{reshape}(XQ, \dots)$$

Underlying operations:

- Q-less TSQR $(n \times k)$
- SVD $(k \times k)$
- copy $(k \times r)$
- tall-skinny GEMM+reshape $(n \times k \cdot k \times r)$

Linear solver building blocks: QRs and SVDs

Optimizations assume tall-skinny / very rectangular matrices!

Orthogonalization

Given $X = X_1 X_2$, calculate:

$$X_1 = QB,$$
$$X_1' \leftarrow Q,$$
$$X_2' \leftarrow BX_2$$

Standard: pivoted Householder QR

Optimized with Q-less TSQR:

$$X_1 = QR,$$
$$X_1' \leftarrow X_1 R^{-1},$$
$$X_2' \leftarrow RX_2$$

but X_1' inaccurate for $\mathrm{cond}(R) \gg 1$

Truncation

Given $X = X_1 X_2$ with $X_2^T X_2 = I$, calculate:

$$\|X_1 - QB\|_F < \tau,$$
$$X_1' = Q,$$
$$X_2' = BX_2$$

Standard: truncated SVD

Optimized with Q-less TSQR:

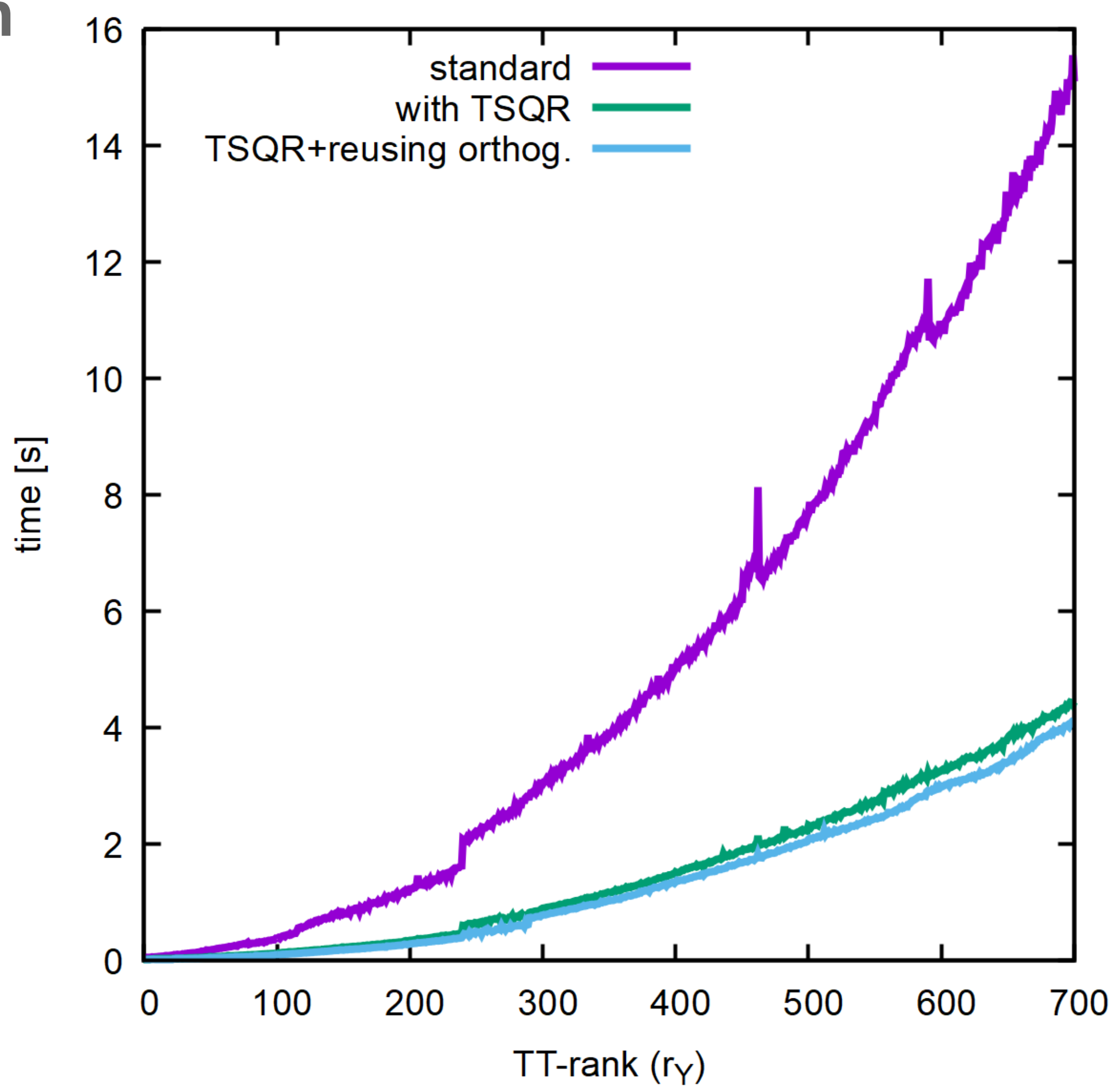
$$X_1 = QR,$$
$$R \approx USV^T,$$
$$X_1' = X_1 V S^{-1},$$
$$X_2' = SV^T X_2$$

Again less orthogonal X_1' but product $X_1 X_2$ ok!

QRs+SVDs in TT/MPS addition+truncation

High-level operation: $Z_{TT} \approx X_{TT} + Y_{TT}$
Setup: dim. 50^{10}, $r_X = 50$, $r_Y = 1, \dots, 700$

Background:
- Combines QR- / SVD-steps
- Additional optimization for previously orthog. X_{TT} or Y_{TT} → reuse orthogonal columns

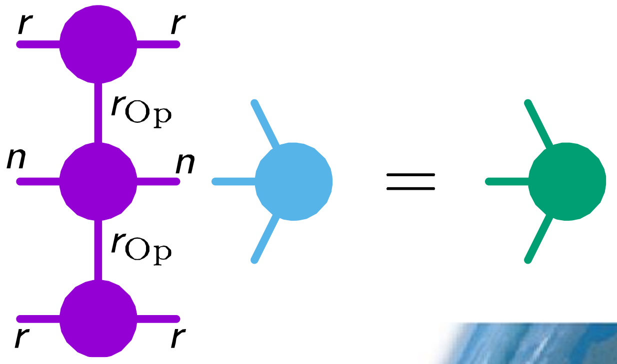


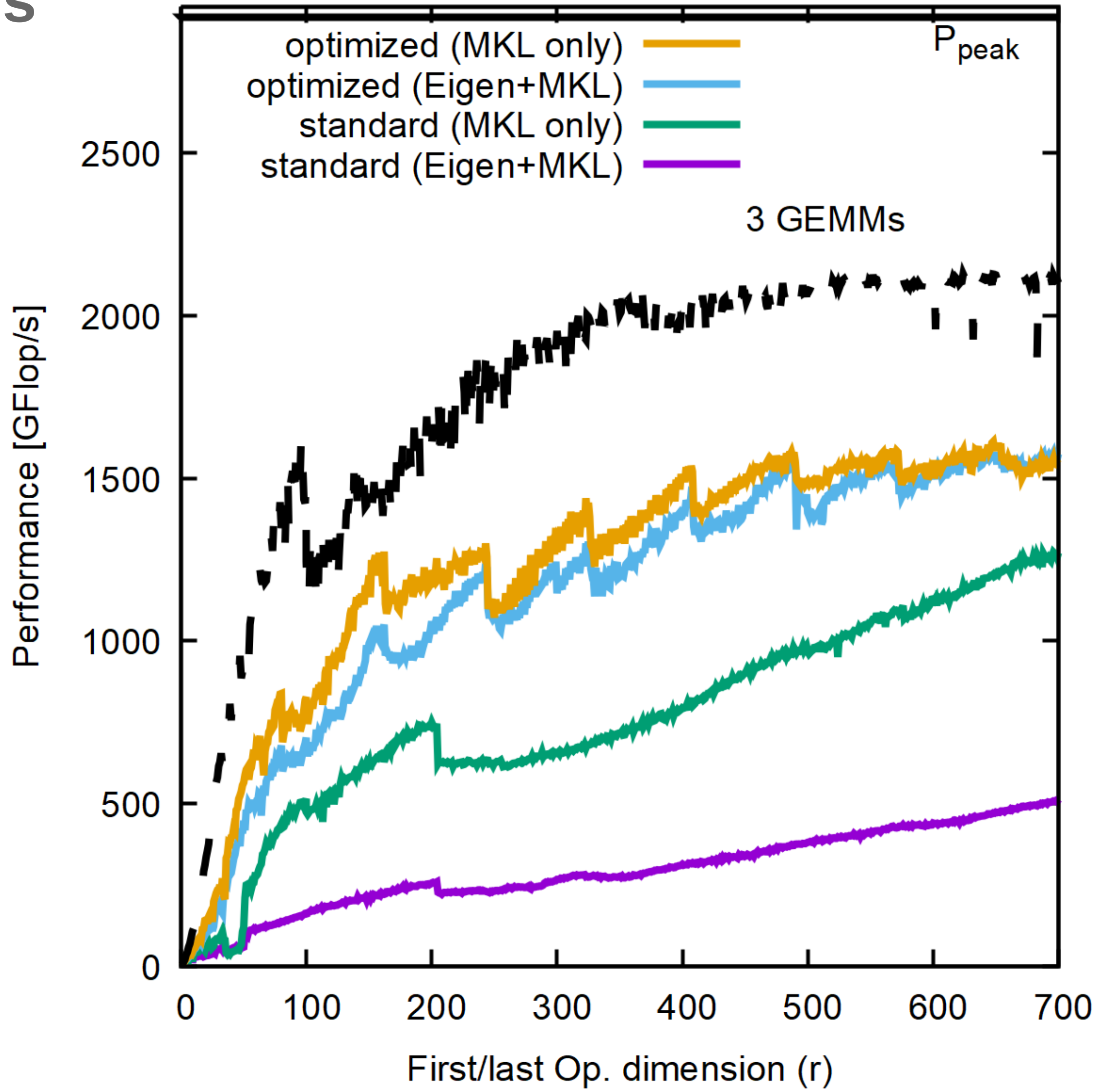
TT-AXPY+TRUNC for 50^{10} TTs on a 64-core AMD EPYC 7773X

Linear solver building blocks: contractions

Multiply TT operator (MPO) with dense array
- Easily leads to complicated array accesses
- Freedom in memory-layout and padding!
  (operator prepared once and applied often)

Optimizations:
- Reorder and combine dimensions
  (big 1st dim./ e.g. $\sum_{i,j} A_{:,i,j} B_{:,:,i,j}$ instead of $\sum_{i,j} A_{:,i,j} B_{i,:,:,j}$)
- Pad 1st dim. (introduces zeros the dense array!)
  (to avoid cache thrashing)





DLR

Conclusion

TT-SVD compression of large dense data:

- Common implementations are about >50x too slow

→ Allows extracting interesting quantities for data analysis

Linear solvers in TT / MPS format:

- Obtained ~5x speedup over the standard implementation
- Presented some ideas on numerical aspects
  - Generic rank-1 preconditioner
  - Non-symmetric projection

→ Allow solving really high-dimensional linear systems!

Generic optimizations* for building blocks of tensor-network algorithms.

*mostly for very non-square matrix operations

Ideas for future work?