Performance of linear solvers in tensor-train format on current multi-core architectures

Melven Röhrig-Zöllner
Institute for Software Technology, German Aerospace Center (DLR)

February 27, 2024

DLR

Goal

Show performance of mapping tensor algorithms onto linear algebra building blocks

Based on 2 examples...

Related talk: Paolo Bientinesi: The Linear Algebra Mapping Problem and how programming languages solve it

Problem definitions

Approx. large data with low-rank tensor

Given:

- dense tensor $X \in \mathbf{R}^{n_1 \times n_2 \times \cdots \times n_d}$
- desired tolerance ϵ_{tol} or max. rank r_{max}

Calculate:

- Low-rank approximation X_{TT} with

$$\|X - X_{\text{TT}}\|_F \leq \epsilon_{\text{tol}}$$

or

$$X_{\text{TT}} \approx X, \quad \text{with} \operatorname{rank}(X_{\text{TT}}) \leq r_{\text{max}}$$

Solve linear system in low-rank tensor format

Given:

- low-rank linear operator $\mathcal{A}_{\text{TT}} : \mathbf{R}^{n^d} \to \mathbf{R}^{n^d}$
- low-rank right-hand side $B_{\text{TT}} \in \mathbf{R}^{n^d}$
- desired tolerance ϵ_{tol}

Calculate:

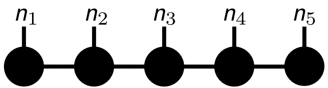
- iterative solution X_{TT} with

$$\|\mathcal{A}_{\text{TT}} X_{\text{TT}} - B_{\text{TT}}\|_* \leq \epsilon_{\text{tol}}$$

for some suitable norm $\|\cdot\|_*$

Tensor-train format

- Known as MPS (matrix-product states) in physics.
- Defined by series of 3d tensors



(tensor-network notation)

$$X_1, \cdots, X_d, \text{ with } X_k \in \mathbf{R}^{r_{k-1}, n_k, r_k}, r_0 = r_d = 1$$

with ranks (bond-dimensions) $r_1, \ldots, r_{d-1}$ and dimensions $n_1, \ldots n_d$.

- Approximates a high-dim. tensor $X \in \mathbf{R}^{n_1 \times n_2 \times \cdots \times n_d}$ with

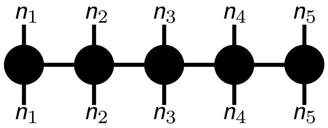
$$X_{\mathrm{TT}} := X_1 \times X_2 \times \cdots \times X_d$$

where $\cdot \times \cdot$ is the contraction: $X_i \times X_{i+1} := \sum_k (X_i)_{:,:,k} (X_{i+1})_{k,:,:} \in \mathbf{R}^{r_{i-1} \times n_i \times n_{i+1} \times r_{i+1}}$

- with a "TT-rank" of $r := \max(r_1, \ldots, r_{d-1})$

Tensor-train operator

- Known as MPO (matrix-product operator) in physics.



(tensor-network notation)

- Defined by series of 4d tensors

$$A_1, \cdots, A_d, \text{ with } A_k \in \mathbf{R}^{r_{\mathrm{Op},k-1}, n_k, n_k, r_{\mathrm{Op},k}}, r_{\mathrm{Op},0} = r_{\mathrm{Op},d} = 1$$

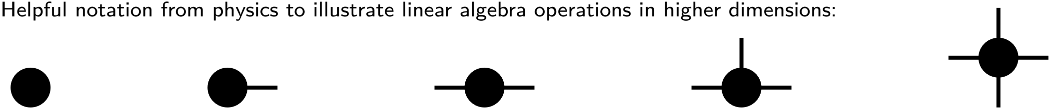
  with ranks $r_{\mathrm{Op},1}, \ldots, r_{\mathrm{Op},d-1}$ and dimensions $n_1 \times n_1, \ldots, n_d \times n_d$.
- Provides the high-dim. linear operator $\mathcal{A}_{\mathrm{TT}} \in \mathbf{R}^{(n_1 \times n_1) \times (n_2 \times n_2) \times \cdots \times (n_d \times n_d)}$ with

$$\mathcal{A}_{\mathrm{TT}} := A_1 \times A_2 \times \cdots \times A_d$$

In the following, we simply use $n_1 = \cdots = n_d = n$.

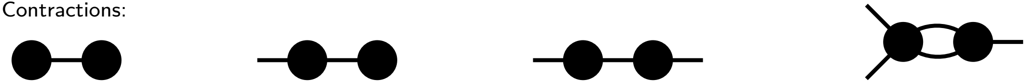
Tensor network notation

Helpful notation from physics to illustrate linear algebra operations in higher dimensions:



scalar | vector | matrix | 3d tensor | 4d tensor

Contractions:



dot-product | matrix-vector product | matrix-matrix product | contraction of 4d and 3d tensors

Orthogonalities and decompositions:

$v^T v = 1$ | $Q^T Q = I$ | QR | USV^T

Tensor unfoldings and orthogonalities

Unfolding a 3d tensor $T \in \mathbf{R}^{r_l, n, r_r}$ ("matrification"):

- "left-unfolding" combines first two dimensions:

$$T_{\text{left}} := \text{reshape}(T, r_l n, r_r) \in \mathbf{R}^{r_l n \times r_r}$$

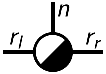
- "right-unfolding" combines last two dimensions:

$$T_{\text{right}} := \text{reshape}(T, r_l, n r_r) \in \mathbf{R}^{r_l \times n r_r}$$

Orthogonality of a 3d tensor:

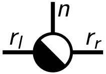
- T is "left-orthogonal" if its left-unfolding has orthonormal columns:

  $(T_{\text{left}}^T T_{\text{left}} = I \in \mathbf{R}^{r_r \times r_r})$



- T is "right-orthogonal" if its right-unfolding has orthonormal rows:

  $(T_{\text{right}} T_{\text{right}}^T = I \in \mathbf{R}^{r_l \times r_l})$



Relation between tensor-trains and 2d SVDs

Remark: tensor-train invariant wrt. multiplying with a matrix and its inverse ($M \in \mathbf{R}^{r_k \times r_k}$):

$$X'_{\mathrm{TT}} := X_1 \times \cdots \times (X_k \times M) \times (M^{-1} \times X_{k+1}) \times \cdots \times X_d = X_{\mathrm{TT}}$$

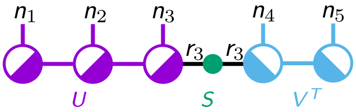
So we can left-orthogonalize X_1 then X_2, ..., up to X_{k-1}:

$$X'_1 := X_1 \times R_1^{-1}, \qquad X'_2 := R_1 \times X_2, \qquad \text{with } X_{1,\text{left}} = Q_1 R_1$$
$$X''_2 := X'_2 \times R_2^{-1}, \qquad X'_3 := R_2 \times X_3, \qquad \text{with } X'_{2,\text{left}} = Q_2 R_2$$

And similarly right-orthogonalize X_d to sub-tensor X_k... with an SVD in the last step:

$$X'''_k := X''_k \times U_{k+1}, \qquad X''_{k+1} := V^T_{k+1} \times X'_{k+1}, \qquad \text{with } X_{k+1,\text{right}} = U_{k+1} S V^T_{k+1}$$

Resulting in (for $k = 3$):



Performance of required dense linear algebra operations (on my machine...)

Matrix-matrix product (GEMM)

$$\begin{array}{cccc} C & \leftarrow & A & B \\ (n \times k) & & (n \times m) & (m \times k) \end{array}$$

Costs: $2nmk$ flop, $(nk + nm + mk)$ data transfers
compute-bound for $\min(n, m, k) \gg 100$
memory-bound for $\min(n, m, k) \lesssim 100$

(Pivoted) QR decomposition

$$AP = QR,$$

with $Q^T Q = I$, R upper triangular, $n \geq m$.

Costs: $2nm^2 - 2/3m^3$ flop, $2nm + 1/2m^2$ data transfers
memory-bound for $m \lesssim 100 \rightarrow$ tall-skinny QR (TSQR)

Singular value decomposition (SVD)

$$A = USV^T,$$

with $U^T U = I$, $V^T V = I$, $S = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$.

Costs: $> 7nm^2$ flop, $> 2nm + m^2$ data transfers

In practice: $t_{\mathrm{SVD}} \gg t_{\mathrm{QR}} > t_{\mathrm{GEMM}}$ for similar dimensions

Problem 1 – approximate large data with low-rank: TT-SVD

Idea

- Based on successive SVDs for each dimension.
- Truncated right-singular vectors become next sub-tensor.

Remarks

- Large matrices are tall and skinny (e.g., $n^{d-1} \times n$).
- Size of X (ideally) decreases in each step.
- Cheap operations are grayed out.

Algorithm [Oseledets, 2011]

Input: Tensor X
  for $i = 1, \dots, d-1$ **do**
    Reshape X to $\left(\prod_{k=i+1,d} n_k\right) \times (n r_{i-1})$
    Calculate SVD: $USV^T = X$
    Choose truncation rank r_i
    $T_i \leftarrow V^T_{1:r_i}$, reshape to $r_{i-1} \times n_i \times r_i$
    $X \leftarrow U_{1:r_i} S_{1:r_i}$
  end for
  $T_d \leftarrow X$, reshape to $(r_{d-1} \times n_d \times 1)$
Output: Tensor-train $(T_1, \dots, T_d)$

Problem 2 – solve linear systems: TT-AMEn

Idea

- Alternating least-squares (ALS):
  "optimize" one sub-tensor at a time
  sweep left-right until convergence
- Orthogonalize all other sub-tensors
  $\Rightarrow$ projection onto smaller problem
- Enrich subspace by a few directions of the residual

Remarks

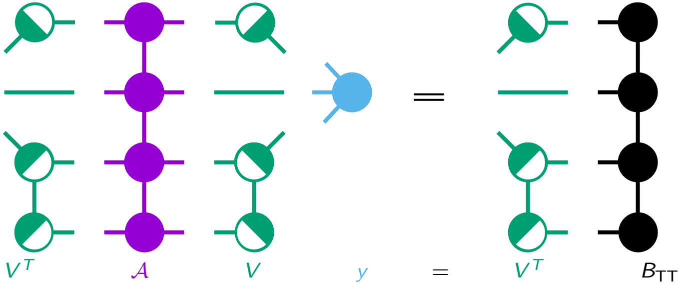
- iterative solver (GMRES, CG) for small problems
- Subspace enrichment needed to adapt ranks
  (for unknown solution rank)
- Complex algorithm with lots of different operations

Algorithm [Dolgov, 2014]

Input: Operator $\mathcal{A}_{\mathrm{TT}}$, RHS B_{TT}, initial guess X_{TT}
  Right-orthogonalize $X_d, \ldots, X_2$
  while not converged **do**
    for $i = 1, \ldots, d-1$ **do**
      $V_{\mathrm{left}} := (X_1, \ldots, X_{i-1})$, $V_{\mathrm{right}} := (X_{i+1}, \ldots, X_d)$
      $\mathcal{V} := V_{\mathrm{left}} \otimes I \otimes V_{\mathrm{right}}$
      Approx. solve $(\mathcal{V}^T \mathcal{A}_{\mathrm{TT}} \mathcal{V}) y = \mathcal{V}^T B_{\mathrm{TT}}$
      Left-orthogonalize $X_i \leftarrow y$
      Update (X_i, X_{i+1}) to enrich subspace
      (adds directions to X_i and zeros to X_{i+1})
    end for
    for $i = d, \ldots, 2$ **do**
      Same as above but right-to-left
    end for
  end while
Output: Approx. solution X_{TT}

Problem 2 – solve linear systems: projection onto small problem

Idea: minimize energy $J(u) := \frac{1}{2}\langle u, \mathcal{A}u\rangle - \langle u, b\rangle$ for (X_i)



Properties:

- $\mathcal{V}^T\mathcal{V} = I$
- $\mathcal{V}y = X_{\mathrm{TT}}$

For spd operator $\mathcal{A}$:

- minimizes $\|X_{\mathrm{TT}} - X^*_{\mathrm{TT}}\|_{\mathcal{A}}$
- $\mathrm{cond}(\mathcal{V}^T\mathcal{A}\mathcal{V}) \leq \mathrm{cond}(\mathcal{A})$

Alternative for non-symmetric $\mathcal{A}$:

$$\mathcal{W}^T\mathcal{A}\mathcal{V}y = \mathcal{W}^T\mathcal{A},$$

e.g., with $\mathcal{W}^T\mathcal{W} = I$ and $\mathcal{W}\mathcal{C} \approx \mathcal{A}\mathcal{V}$.

Problem 2 – solve linear systems: TT-AMEn subspace enrichment

Idea: directions from steepest descent step for minimizing $J(u)$

Basis enrichment (for left-to-right sweep):

1. With $\mathcal{V} := V_{\text{left}} \otimes I \otimes V_{\text{right}}$, calculate

$$Z_{\text{TT}} := V_{\text{left}}^T \left(B_{\text{TT}} - \mathcal{A}_{\text{TT}} X_{\text{TT}}\right)$$

2. Right-orthogonlize Z_{TT}
3. Add leading r_{add} directions of Z_1 to X_i:

$$(X_i)_{\text{left}} \leftarrow \begin{pmatrix} (X_i)_{\text{left}} & (Z_1)_{:,1:r_{\text{add}}} \end{pmatrix}, \qquad (X_{i+1})_{\text{right}} \leftarrow \begin{pmatrix} (X_{i+1})_{\text{right}} \\ 0 \end{pmatrix}$$

$\Rightarrow$ Increases rank by r_{add} in each sweep

Remark: this "full" variant needs another costly SVD, cheaper updates for approximating of Z_{TT} possible

Problem 2 – solve linear systems: TT-rank1 preconditioner

Idea:

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

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

$$\mathcal{L}_{\mathrm{TT}} \mathcal{A}_{\mathrm{TT}} \mathcal{R}_{\mathrm{TT}} \approx I$$

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

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

$\Rightarrow$ Reduces the condition number without increasing the rank of the operator

Underlying building blocks: TT-SVD

Required operation

For $X \in \mathbf{R}^{n \times m}$, $n \gg m$, we need:

$$\|X - BQ^T\|_F \leq \tau$$

$$X_1 \leftarrow \text{reshape}(B, \dots)$$

$$X_2 \leftarrow Q^T$$

Standard: truncated SVD

$$USV^T \approx X,$$

$$X_1 \leftarrow US,$$

$$X_2 \leftarrow V^T,$$

$$X_1' \leftarrow \text{reshape}(X_1, \dots)$$

Costs: $> 7nm^2$ flop, $> 2n(m+r)$ data transfers

Optimized: Q-less TSQR & TSMM+reshape

$$QR = X,$$

$$USV^T \approx R,$$

$$X_1 \leftarrow \text{reshape}(XV, \dots),$$

$$X_2 \leftarrow V^T$$

Costs: $2nm(m+r)$ flop, $2n(m+r)$ data transfers

TT-SVD performance results [Röhrig-Zöllner, 2022]

Further “tricks”

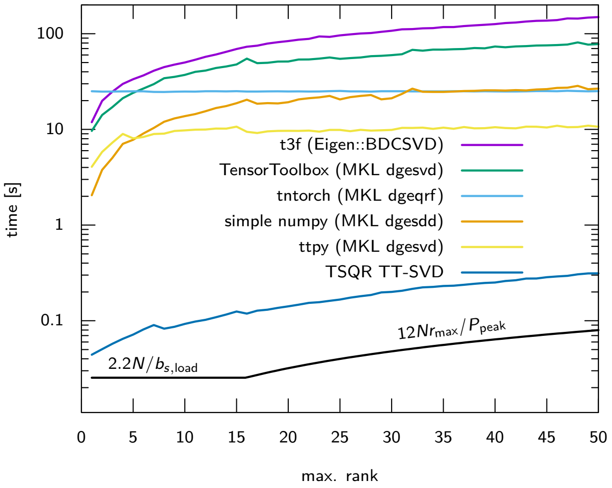
- Combine dimensions to increase compute intensity
- Add padding to avoid bad strides (multiples of 2^k → cache thrashing)

Setup & results

- Decompose random 2^{27} tensor
- Data size: 1GB
- 14-core Intel Skylake Gold 6132

→ Existing software: >50x slower

- Much closer to roofline performance ($N := n_1 n_2 \cdots n_d$)
- tntorch first constructs a full-rank TT, then truncates it.



Underlying building blocks: orthogonalization in linear solvers

Required operation

For $X := X_1 X_2$, we need:
(with $X_1 \in \mathbf{R}^{n\times m}$, $X_2 \in \mathbf{R}^{m\times k}$, $m \ll n \approx k$)

$$QB = X_1 \quad \text{(rank-revealing)}$$
$$X_1' = Q$$
$$X_2' = BX_2$$

Standard: pivoted QR

$$QR = X_1 P,$$
$$X_1' = Q,$$
$$X_2' = RP^T X_2$$

Costs: $5nm^2$ flop, $6nm$ data transfers

Optimized: Q-less (TS)QR:

$$QR = X_1 P$$
$$X_1' = X_1 P R^{-1} \quad \text{(backward subst.)}$$
$$X_2' = RP^T X_2$$

Costs: $4nm^2$ flop, $5nm$ data transfers
But X_1' inaccurate for $\mathrm{cond}(R) \gg 1 \Rightarrow$ track errors

Underlying building blocks: exploiting pre-existing orthogonalities

Setting: TT-axpby

(e.g., needed for residual $B_{\mathrm{TT}} - \mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}}$)

$$Z_{\mathrm{TT}} = \alpha X_{\mathrm{TT}} + \beta Y_{\mathrm{TT}} = Z_1 \times Z_2 \times \cdots \times Z_d$$

with

$$(Z_1)_{1,:,:} = \begin{pmatrix} (X_1)_{1,:,:} & (Y_1)_{1,:,:} \end{pmatrix},$$

$$(Z_i)_{:,j,:} = \begin{pmatrix} (X_i)_{:,j,:} & 0 \\ 0 & (Y_i)_{:,j,:} \end{pmatrix}, \forall j,\ i = 2, \ldots, d-1,$$

$$(Z_d)_{:,:,1} = \begin{pmatrix} \alpha (X_d)_{:,:,1} \\ \beta (Y_d)_{:,:,1} \end{pmatrix}.$$

Then orthogonalize Z_{TT}.

Idea: X_i usually already left-/right-orthogonal

$\Rightarrow$ blocks in Z_i already orthogonal.
Assuming left-orthogonal X_{TT}, calculate

$$Q_i R_i = (I - \bar{X}_i \bar{X}_i^T) \bar{Y}_i$$

in each step $i = 2, \ldots, d-1$ with

$$\bar{X}_j := \left(\begin{pmatrix} I \\ 0 \end{pmatrix} \times X_j \right)_{\text{left}}, \quad \bar{Y}_j = \left(\begin{pmatrix} M_{j-1} \\ R_{j-1} \end{pmatrix} \times Y_j \right)_{\text{left}}.$$

Underlying building blocks: truncation in linear solvers

Required operation

For $X = X_1 X_2$ with $X_2^T X_2 = I$, we need:
(with $X_1 \in \mathbf{R}^{n\times m}$, $X_2 \in \mathbf{R}^{m\times k}$, $m \ll n \approx k$)

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

Standard: truncated SVD

$$USV^T \approx X_1$$
$$X_1' \leftarrow U$$
$$X_2' \leftarrow SV^T X_2$$

Costs: $> 7nm^2 + 2nmr$ flop, $> 2n(m+r)$ data transfers

Optimized: Q-less (TS)QR + SVD

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

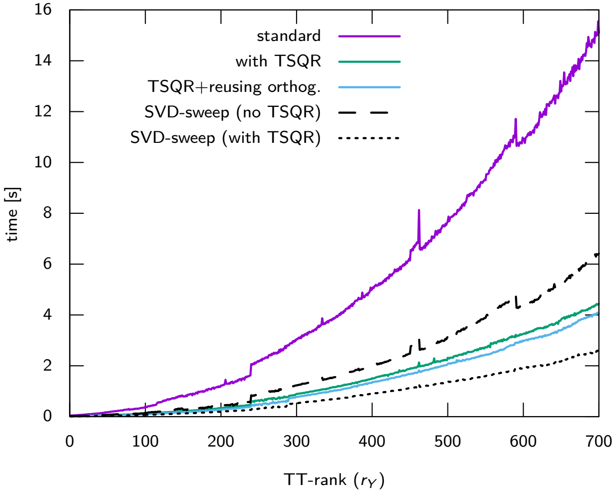
Costs: $2nm^2 + 4nmr$ flop, $nm + 2n(m+r)$ transfers
As before: X_1' less accurate in "unimportant" directions

Truncated TT-axpby performance

Setup & results

- Add 2 tensor-trains (X_{TT}, X_{TT}) of dim. 50^{10},
  TT-rank $r_X = 50$, varying r_Y
- both X_{TT}, X_{TT} previously left-orthogonal
- 64-core AMD EPYC 7773X ("Zen 3 V-Cache")
- Operations needed for $B_{TT} - \mathcal{A}_{TT} X_{TT}$

→ Roughly 4x speedup



Underlying building blocks: tensor contractions

Required operations

- Most costly part of inner solver (GMRES):
  Apply TT operator to dense array
- Easily sub-optimal array accesses (cache thrashing)
- Required contractions:
  (with e.g. $A_2 \in \mathbf{R}^{r_{\mathrm{Op}} \times n \times n \times r_{\mathrm{Op}}}$)

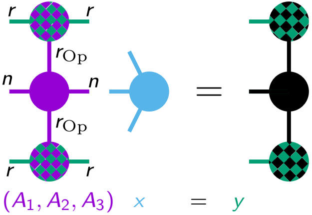
$$(z_1)_{:,:,:,:} \leftarrow \sum_i (A_3)_{:,:,i} x_{:,:,i}$$

$$(z_2)_{:,:,:,:} \leftarrow \sum_{i,j} (A_2)_{:,:,i,j} (z_1)_{j,:,:,i}$$

$$y_{:,:,:} \leftarrow \sum_{i,j} (A_1)_{:,i,j} (z_2)_{j,:,:,i}$$

Optimizations

- Reorder array dimensions
  $\rightarrow$ combine several small dimensions
- Padding (first dim.) to avoid bad strides



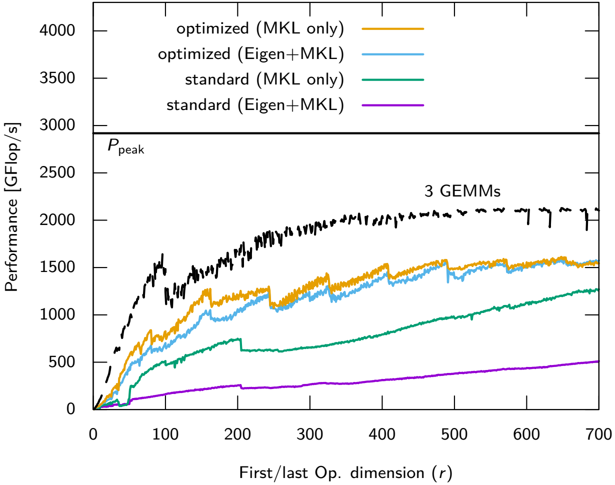
Tensor contractions performance

Setup & results

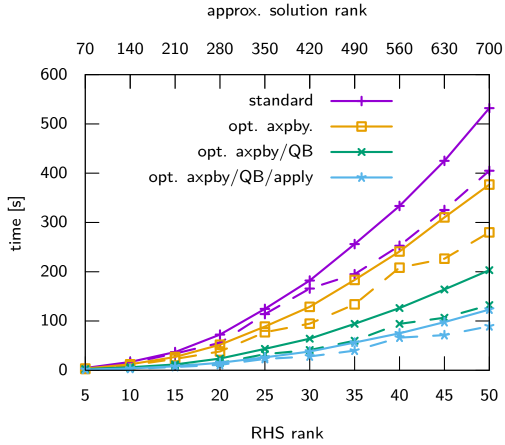
- Operator dimension $r \times 50 \times r$
- 64-core AMD EPYC 7773X ("Zen 3 V-Cache")
- Comparison to 3 GEMMS of similar dimensions

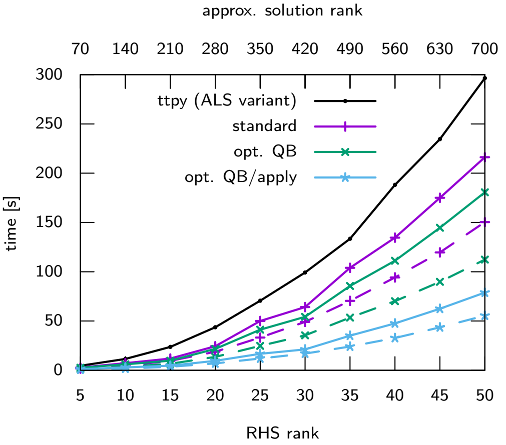
Remark

- Uses loop-over-GEMM with MKL GEMM
- More sophisticated implement. possible ([Springer 2018], no maintained library available?)



Complete TT-AMEn performance [Röhrig-Zöllner, 2023]





50^{10} conv.-diff. operator, random RHS, dashed lines with TT-rank1 preconditioner, AMD EPYC 7773X
left: "full" SVD variant, right: ALS/simplified variant

Background: self-implemented kernels

Unfortunately, optimizations need some "non-standard" operations...

Q-less (tall-skinny) QR

- Never stores Q
- Implementation based on [Demmel, 2012]
- First parallelize over blocks of rows
- Reduction parallelized over columns
  Background: e.g., $n/64 \times m$ not so tall-skinny
- Recursive blocking over columns

Memory-bound (fused) operations

- Just to optimize mem.-accesses
  (same distribution on cores for each call)
- Fused dense axpy+dot, axpy+norm, tall-skinny GEMM (TSMM) + reshape
- In-place triangular solve for very rectangular/tall-skinny matrices

Tensor operations mapping problem

Optimization steps

1. Reformulate tensor algorithm: actually required operations (often $\neq$ standard LAPACK operations)
2. Consider special properties/requirements: e.g., pre-existing orthogonalities, block-structure
3. Map required operations onto suitable building blocks
4. Optimize data layout: rearrange dimensions & padding
   (really crucial: e.g. if n mutliple of 4, unpadded n^d leads to bad strides)
5. Implement required "non-standard" kernels (like e.g., Q-less QR)

→ High speedups possible! (as illustrated)

Unfortunately, I don't see a generic/automated approach here (except for domain-specific algorithms)

Conclusion

Summary

- Optimized tensor-train/MPS decomposition (TT-SVD): $\sim 50\times$ speedup
- Optimized tensor-train/MPS linear solver: $\sim 5\times$ speedup
- Key ingredient: mapping of tensor algorithm onto (very) "rectangular" matrix operations

Possible next steps

- Other tensor-train/MPS algorithms (similar "rectangular" operations for $n_i \gg 2$)
- Extension to tree tensor-networks

Literature

- Röhrig-Zöllner; Thies & Basermann: "Performance of the Low-Rank TT-SVD for Large Dense Tensors on Modern MultiCore CPUs", SISC, 2022
- Röhrig-Zöllner; Becklas; Thies & Basermann: "Performance of linear solvers in tensor-train format on current multicore architectures", submitted to SISC, 2023
- Oseledets: "Tensor-Train Decomposition", SISC, 2011
- Dolgov & Savostyanov: "Alternating Minimal Energy Methods for Linear Systems in Higher Dimensions", SISC, 2014
- Demmel et.al.: "Communication-optimal Parallel and Sequential QR and LU Factorizations", SISC, 2012
- Williams et.al.: "Roofline: An Insightful Visual Performance Model for Multicore Architectures", Comm. of the ACM, 2009
- Anderson et.al.: "LAPACK Users' Guide", SIAM, 1999
- Springer & Bientinesi: "Design of a High-Performance GEMM-like Tensor-Tensor Multiplication", ACM TOMS, 2018

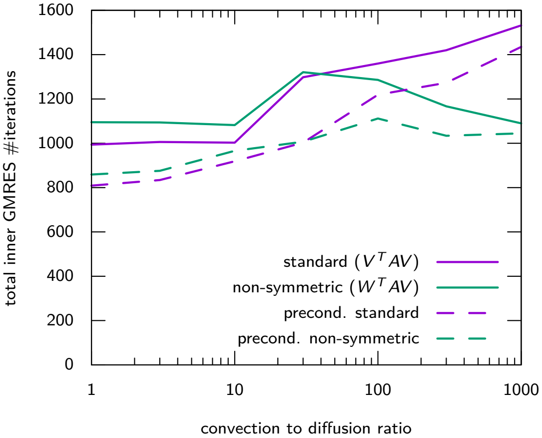
TT-AMEn: alternative projection for non-symmetric systems

Setup:

- TT-AMEn with inner GMRES
- varying asymmetry

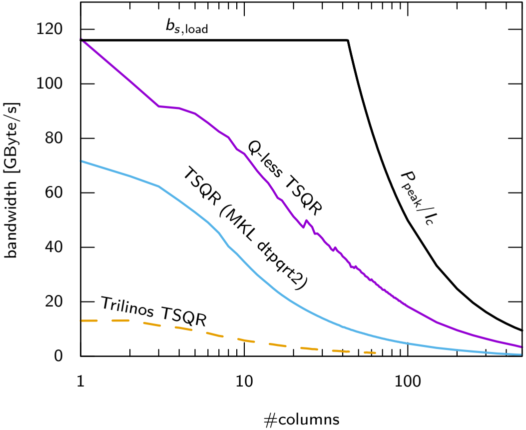
Observations: (work-in-progress!)

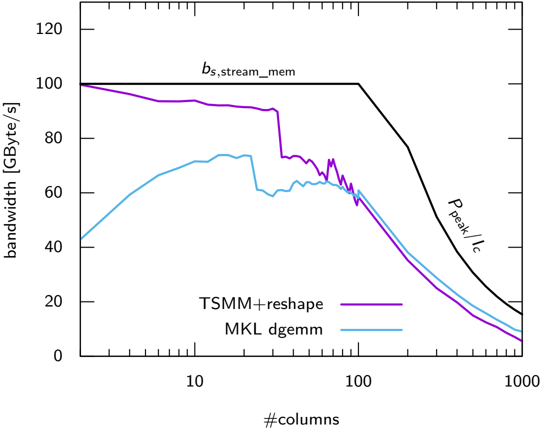
- alternative projection beneficial for strongly non-symmetric problems



Inner iterations for a 20^{10} conv.-diff. problem with RHS ones.

TT-SVD: Building blocks (TSQR and TSMM+reshape)





$((\sim 25 \cdot 10^6) \times m$ matrix in double-precision ($0.2m$ GB); 16-core Intel CascadeLake Gold 6242.)

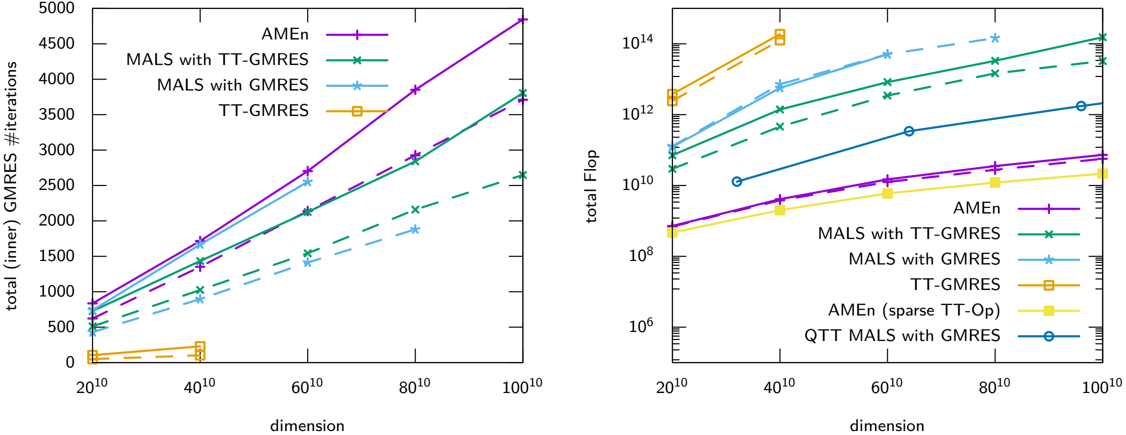
Comparison of methods: overview

method	idea	properties/problems
TT-GMRES	GMRES adaptive truncation tolerance	global, large intermediate ranks
TT-ALS (alternating least squares)	projection onto X_k, solve for $k = 1, \ldots, d$	predefined rank, stuck in local minima
TT-MALS (modified ALS)	projection onto $(X_k \times X_{k+1})$, solve for $k = 1, \ldots, d-1$	rank-adaptive, larger local problem
TT-AMEn (alternating minimal energy)	ALS + enrich basis	rank-adaptive
Riemannian optimization methods	fixed rank $\to$ smooth submanifold, search direction in tangent space	global, needs special preconditioner

Riemannian optimization not further discussed here (but promising for some applications!)

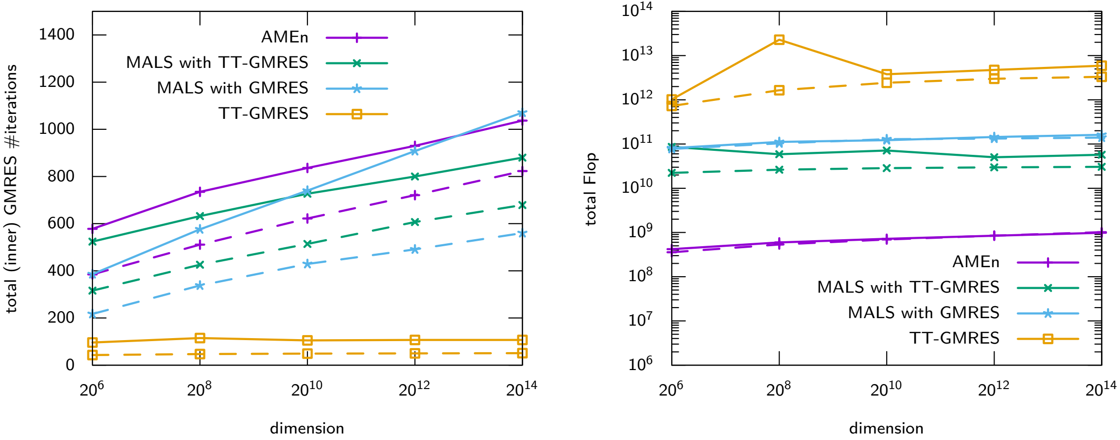
DLR

Comparison of methods: results for varying dimension n



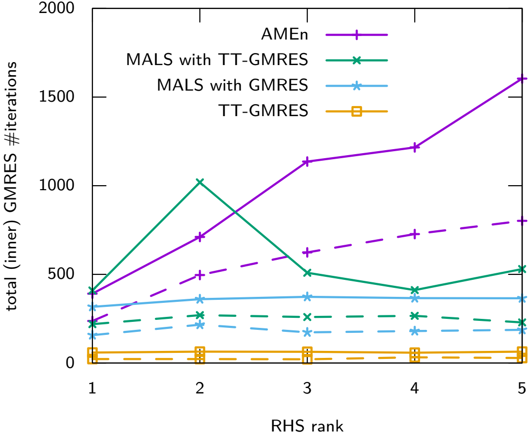
(Conv.-diff. problem with RHS ones and conv.-diff. ratio $n/2$. Dotted line with TT-rank1-preconditioner.)

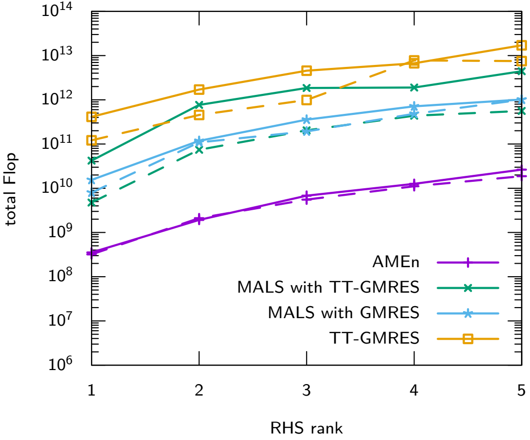
Comparison of methods: results for varying #dimensions d



(Conv.-diff. problem with RHS ones and conv.-diff. ratio 10. Dotted line with TT-rank1-preconditioner.)

Comparison of methods: results for varying rank r_B (and r_X)





(Conv.-diff. problem with random RHS and conv.-diff. ratio 10. Dotted line with TT-rank1-preconditioner.)