Performance of low-rank Tensor Algorithms

Melven Röhrig-Zöllner[1], Jonas Thies[2] and Achim Basermann[1]
[1] Institute for Software Technology, German Aerospace Center (DLR)
[2] Delft Institute of Applied Mathematics, TU Delft

DLR

Problem 1

Low-rank approximation in tensor-train format (TT-SVD)

Given:

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

Calculate:

- tensor-train X_{TT} with:

$$\text{ranks}(X_{\text{TT}}) \leq r_{\text{max}} \quad \text{and} \quad \|X - X_{\text{TT}}\|_F \lesssim \epsilon_{\text{tol}}$$

Remarks:

- Focus on the tensor-train format; very similar approaches for some other formats
- Consider high-dimensional case ($d \gg 3$) and sufficiently small TT-ranks $r_1, \dots r_{d-1}$

Problem 2 (work-in-progress)

Linear solver in tensor-train format

Given:

- low-rank linear operator $\mathcal{A}_{\mathrm{TT}} \in \mathbf{R}^{(n_1 \times n_1) \times (n_2 \times n_2) \times \cdots \times (n_d \times n_d)}$
- low-rank right-hand side $B_{\mathrm{TT}} \in \mathbf{R}^{n_1 \times n_2 \times \cdots \times n_d}$
- desired tolerance ϵ_{tol}

Calculate:

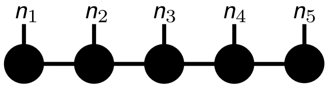
- iterative solution X_{TT} with

$$\|\mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}}\|_* \lesssim \epsilon_{\mathrm{tol}}$$

Remarks:

- Residual/error norm $\|\cdot\|_*$ depends on the solution method.

Tensor-Train Format [Oseledets]



(tensor-network notation)

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

$$T_1, \cdots, T_d, \text{ with } T_k \in \mathbf{R}^{r_{k-1}, n_k, r_k}, r_0 = r_d = 1$$

  with ranks $r_1, \ldots, r_{d-1}$ and dimensions $n_1, \ldots n_d$.
- Approximates high-dim. tensor $X \in \mathbf{R}^{n_1 \times n_2 \times \cdots \times n_d}$ with

$$X_{\mathrm{TT}} := T_1 \times T_2 \times \cdots \times T_d$$

  where $\cdot \times \cdot$ is the contraction: $T_i \times T_{i+1} := \sum_k (T_i)_{:,:,k} (T_{i+1})_{k,:,:} \in \mathbf{R}^{r_{i-1} \times n_i \times n_{i+1} \times r_{i+1}}$
- Generalizes a truncated SVD to higher dimensions.

DLR

"Refined" Roofline performance model

Consider 2 bottlenecks:

1. Max. performance: $P_{\text{max,op}}$ [GFlop/s] (for e.g., op = double-precision FMA)
2. Saturated memory bandwidth : $b_{s,pattern}$ [GByte/s] (for e.g., pattern = load / axpy)

$\Rightarrow$ Machine intensity: $I_m := \frac{P_{\text{max,op}}}{b_{s,pattern}}$

Analyze the algorithm:

1. Computations: n_{flops} [flop]
2. Data transfers: $V_{\text{load+store+update}}$ [byte]

$\Rightarrow$ Compute intensity: $I_c := \frac{n_{\text{flops}}}{V_{\text{load+store+update}}}$

Expected ideal runtime:

$$t = \max\left(\frac{n_{\text{flops}}}{P_{\text{max,op}}}, \frac{V_{\text{load+store+update}}}{b_{s,pattern}}\right) \text{ [s]}$$

Problem 1: TT-SVD

Standard algorithm

Input: Tensor X

for $i = 1, \ldots, d-1$ **do**

Reshape X to $\left(\prod_{k=i+1,d} n_k\right) \times (n_i r_{i-1})$

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 $(T_1, \ldots, T_d)$

DLR

Problem 1: TT-SVD

Standard algorithm

Input: Tensor X

for $i = 1, \dots, d-1$ **do**

Reshape X to $\left(\prod_{k=i+1,d} n_k\right) \times (n_i r_{i-1})$

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 $(T_1, \dots, T_d)$

Observations

- Based on SVDs, GEMMs, and reshaping.
- (Reshaping should copy to padded mem.-layout to avoid 2^k strides.)
- Cheap operations are grayed out.
- Large matrices are tall and skinny.
- Size of X ideally decreases in each step.

Problem 1: TT-SVD

Improved algorithm

Input: Tensor X

Skip first $j-1$ iterations

Reshape X to $\prod_{k=j+1,d} n_k \times (n_1 \cdots n_j)$

for $i = j, \ldots, d-1$ **do**

Tall-skinny QR decomposition: $QR = X$

Small SVD: $\bar{U}SV^T = R$

Choose rank r_i

$T_i \leftarrow V_{1:r_i}^T$, reshape to $r_{i-1} \times n_i \times r_i$

$X \leftarrow XV_{1:r_i}$, reshape to $\bar{n}_{i+1} \times (n_{i+1} r_i)$

end for

Recover $T_1, \ldots, T_j$ from T_j

Output: Tensor-train $(T_1, \ldots, T_{d-1}, X)$

Problem 1: TT-SVD

Improved algorithm

Input: Tensor X

Skip first $j-1$ iterations

Reshape X to $\prod_{k=j+1,d} n_k \times (n_1 \cdots n_j)$

for $i = j, \dots, d-1$ **do**

Tall-skinny QR decomposition: $QR = X$

Small SVD: $\bar{U}SV^T = R$

Choose rank r_i

$T_i \leftarrow V^T_{1:r_i}$, reshape to $r_{i-1} \times n_i \times r_i$

$X \leftarrow XV_{1:r_i}$, reshape to $\bar{n}_{i+1} \times (n_{i+1} r_i)$

end for

Recover $T_1, \dots, T_j$ from T_j

Output: Tensor-train $(T_1, \dots, T_{d-1}, X)$

Remarks

- Replaced costly SVD by tall-skinny QR
- Never use $Q \rightarrow$ Q-less TSQR
- Fused reshape and tall-skinny GEMM

$\rightarrow$ Reads the input data twice (1st iteration): (once for $QR = X$, once for $X \leftarrow XV_{1:r_1}$)

Optimized TT-SVD: performance analysis (1)

Building blocks

Q-less TSQR for $(X \in \mathbf{R}^{n \times m})$:

- $V_{\text{load}} = nm$
- $n_{\text{flops}} \approx 2nm^2$

TSMM+reshape for $(X \leftarrow XM,\ M \in \mathbf{R}^{m \times k})$:

- $V_{\text{load+store}} = n(m+k)$
- $n_{\text{flops}} = 2nmk$

Optimized TT-SVD: performance analysis (1)

Building blocks

Q-less TSQR for $(X \in \mathbf{R}^{n \times m})$:

- $V_{\text{load}} = nm$
- $n_{\text{flops}} \approx 2nm^2$

TSMM+reshape for $(X \leftarrow XM,\ M \in \mathbf{R}^{m \times k})$:

- $V_{\text{load+store}} = n(m+k)$
- $n_{\text{flops}} = 2nmk$

Complete algorithm

Assume size of X reduces by $f < 1$ in each iteration (so $k/m \leq f$).

$\Rightarrow$ Upper bound from the geometric series:

- $V_{\text{load+store}} \leq \frac{2N}{1-f} + \frac{fN}{1-f}$
- $n_{\text{flops}} \lesssim 2Nr_{\max}\left(\frac{1}{f} + \frac{2}{1-f}\right) + O(r_{\max}^3)$

with $N := \prod_{i=1}^{d} n_i$.

DLR

Optimized TT-SVD: performance analysis (2)

Interpretation

Influence compute intensity I_c through combining (or splitting) dimensions in the calculation:

- $f = 1/16$ (low rank): $V_{\text{load+store}} \lesssim 2.2N$ and $n_{\text{flops}} \lesssim 36Nr_{\text{max}}$
- $f = 1/2$ (medium rank): $V_{\text{load+store}} \lesssim 5N$ and $n_{\text{flops}} \lesssim 12Nr_{\text{max}}$

Optimized TT-SVD: performance analysis (2)

Interpretation

Influence compute intensity I_c through combining (or splitting) dimensions in the calculation:

- $f = 1/16$ (low rank): $V_{\text{load+store}} \lesssim 2.2N$ and $n_{\text{flops}} \lesssim 36 N r_{\text{max}}$
- $f = 1/2$ (medium rank): $V_{\text{load+store}} \lesssim 5N$ and $n_{\text{flops}} \lesssim 12 N r_{\text{max}}$

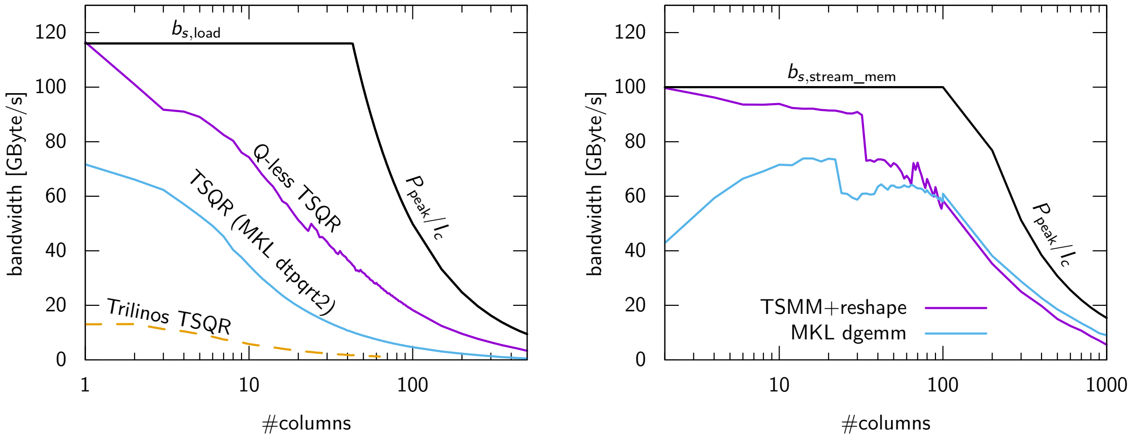
Comparison with measurements (using likwid-perfctr)

Decompose a double-precision 2^{30} tensor (8GB):

	r_{max}	operations (est.) [GFlop]	data transfers (est.) [GByte]
$f = 1/2$	1	14 (13)	43 (43)
$f = 1/16$	1	41 (39)	21 (19)
$f = 1/2$	31	417 (399)	43 (43)

(n_i and r_i are integers, so only some values for f are possible. Measured on an Intel Skylake Gold 6132.)

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



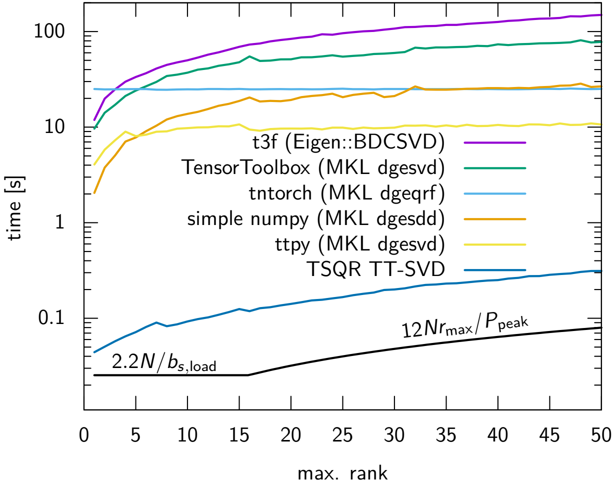
(($\sim 25 \cdot 10^6$) $\times$ m matrix in double-precision ($0.2m$ GB); 16-core Intel CascadeLake Gold 6242.)

TT-SVD: performance results

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

→ Existing software: >50x slower

- tntorch first constructs a full-rank TT, then truncates it.
- remark: my random number generator is slower than the TT-SVD for $r_{max} \lesssim 20$.



Problem 2: Linear solvers in TT format

Numerical methods

- TT-MALS (alternating least-squares):
  Optimize sub-tensors (T_i, T_{i+1}) for $i = 1, \ldots, d-1, \ldots, 1$ ("Sweeps")
  $\rightarrow$ sub-problem again in tensor-train format
- TT-GMRES (or other Krylov methods):
  Iterative algorithms based on arithmetic operations in TT format.
  $\rightarrow$ need *TT-truncation* to reduce ranks

All based on similar building blocks.

TT-truncation algorithm

- Given tensor-train X_{TT}, approximate by $\tilde{X}_{\text{TT}}$ with lower rank.
- Sweep left-to-right using QR decompositions,
  then sweep right-to-left using SVD decompositions (or vice versa).

Problem 2: Linear solvers in TT format (2)

Required decompositions for the TT-truncation

Given tall-skinny $X \in \mathbf{R}^{n \times m}$ (possibly rank-deficient!):

- QR-Sweep: actually need $X = QB$ with $Q^T Q = I$
  Possible implementations:
  - Pivoted QR: $X = Q(RP^T)$
  - SVQB: $M \leftarrow X^T X,\ B^T B = M \Rightarrow X = (XB^+)B$ (too inaccurate in my tests)
  - Q-less TSQR: $X = QRP^T$, recover $Q = XPR^{-1}$
- SVD-Sweep: actually need $X \approx QB$ with $Q^T Q = I$ and tolerance $\epsilon > \epsilon_{FP}$
  Possible implementations:
  - Truncated SVD: $X \approx U(SV^T)$
  - Gram-SVD: $M \leftarrow X^T X,\ M = VS^2V^T \Rightarrow X = (XVS^{-1})(SV^T)$ (too inaccurate in my tests)
  - Q-less TSQR "trick": $X = QR,\ R \approx USV^T$, recover $QU = XVS^{-1}$

DLR

Conclusion

- Goals:
  - low-rank approximation of large dense high-dimensional tensors
  - iterative algorithms in low-rank (tensor-train) format
- Roofline model for the TT-SVD algorithm:
  - low rank: $\sim$ access data twice
  - medium rank: $O(r_{max} \cdot N)$
- Almost optimal TT-SVD implementation: $\sim 50\times$ faster than others
- Difficult mapping of tensor algorithms to efficient building blocks (algorithms based on lots of (small) SVDs)
- Work-in-progress: operations for linear solvers in tensor-train format

Literature

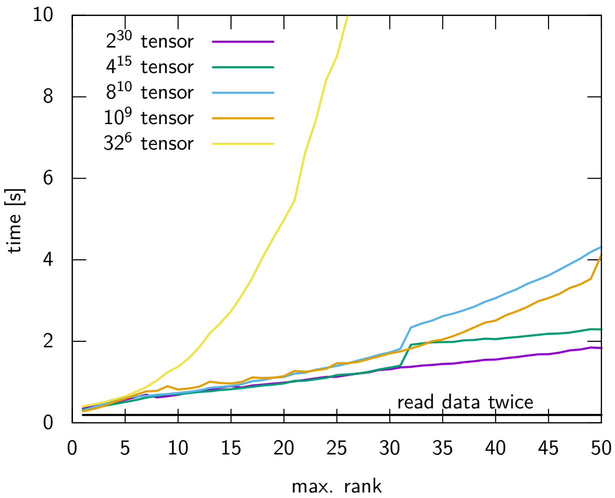
- Röhrig-Zöllner; Thies & Basermann: "Performance of the Low-Rank TT-SVD for Large Dense Tensors on Modern MultiCore CPUs", SISC, 2022
- Oseledets: "Tensor-Train Decomposition", SISC, 2011
- Demmel et.al.: "Communication-optimal Parallel and Sequential QR and LU Factorizations", SISC 2012
- Williams et.al.: "Roofline: An Insightful Visual Performance Model for Multicore Architectures", Comm. of the ACM, 2009

TT-SVD runtime: different tensor dimensions

- Decompose large random tensor, $r_{\max} = 1, \ldots, 50$ (double precision)
- Data size: $\sim$ 8GB
- Combine first dimensions only if beneficial
- 14-core Intel Skylake Gold 6132

→ Calculation more costly with fewer small dimensions!

- Jumps in runtime: switch from e.g. $8^8 \times 8^2$ to $8^7 \times 8^3$ in the first tsqr step



TT-SVD runtime: distributed memory (MPI)

- Decompose random 2^d tensor, $d = 29, \ldots, 36$, $r_{\max} = 1, \ldots, 50$ (double precision)
- Data size: 4GB, ..., 550GB
- Distributed parallel (user-defined MPI reduction for TSQR)
- Up to 4 nodes with 4x14-core Intel Skylake Gold 6132

→ Scales well onto multiple nodes