

Performance of linear algebra building blocks for low-rank tensor algorithms on current multi-core CPUs and GPUs

Melven Röhrig-Zöllner*, Jonas Thies, and Achim Basermann

German Aerospace Center (DLR) – Institute of Software Technology – High-Performance Computing – Cologne



- Goal: analyze building blocks of TT/MPS or TTN (tree tensor network) algorithms:
 1. (Series of) contractions
 2. Decompositions (QR/SVD)
- Disclaimer: not considered here (but definitely interesting!): randomized methods
- Target system: single CPU/GPU node (e.g., 2×192 core AMD Zen5 or NVidia H100)
- “Low-level” compared to some other talks in this session. . .

Introduction: characteristics of typical linear algebra operations



	dimensions	# FP operations	Flop/Byte
Matrix multiplication (GEMM)	$(n \times m) \cdot (m \times m)$	$2nm^2$	$(\frac{4}{m} + \frac{8}{n})^{-1}$
Pivoted QR decomposition (QR) (with forming Q explicitly)	$n \times m, \quad n \geq m$	$\sim 4nm^2 - \frac{4}{3}m^3$	$[\frac{m}{12}, \frac{m}{8}]$
Singular value decomposition (SVD) (with computing singular vectors)	$n \times m, \quad n \geq m$	$> 8nm^2 + \frac{4}{3}m^3$	—

Sources:

- Developer Reference for Intel®oneAPI Math Kernel Library for Fortran
- LAPACK Users' Guide

Introduction: performance of typical linear algebra operations



CPU:

	square	tall-skinny
GEMM	$0.74 P_{\text{peak}}$	$0.49 P_{\text{peak}}$
Pivoted QR	$0.03 P_{\text{peak}}$	$0.001 P_{\text{peak}}$
SVD	$0.03 P_{\text{peak}}$	$0.004 P_{\text{peak}}$

GPU:

	square	tall-skinny
GEMM	$0.78 P_{\text{peak}}$	$0.57 b_{\text{mem}}$
Unpivoted QR	$0.03 P_{\text{peak}}$	$0.006 b_{\text{mem}}$
SVD	$\gtrsim 5 \cdot 10^{-5} P_{\text{peak}}$	$0.006 b_{\text{mem}}$

Setup:

- CPU: 2×64 core AMD Epyc 7773X, Intel MKL 2024.0
- GPU: NVidia H100 with cuSOLVER (CUDA 13.1, SVD with gesvdp)
- square: 2000×2000 , tall-skinny: $2.5 \cdot 10^6 \times 20$ (fits into CPU L3 cache)

Introduction: algorithms built on top of GEMM, QR and SVD...



- GEMM works fine → **lets analyze series of tensor contractions (part one)**
- QR / SVD reach no usable performance.
- Idea 1: improve QR / SVD software—difficult within constraints of LAPACK (prescribed memory layout, accuracy, and robustness)
- Idea 2: Use more GEMMs instead—workarounds: (mostly for tall-skinny cases)
 - normal equations (construct $QR = X$ using $M := X^T X$)
 - sketching / randomized linear algebra
- **Idea 3: analyze algorithm/operations “around” QR / SVD (part two)**

Contractions: local operator in TT algorithm (1)

Motivation:

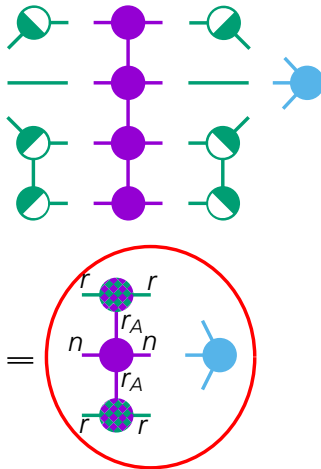
Inner operation in ALS / single-site DMRG:

$$y_{i'j'k'} \leftarrow \sum_{i,j,k,l,m} A_{i'li} B_{j'lmj} C_{k'mk} x_{ijk}$$

- project operator onto one tensor
→ local problem
- usually solved iteratively

Ordering of contractions influences #Flops:

→ use opt_einsum (Smith et al.2018) as starting point

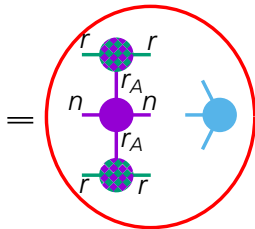
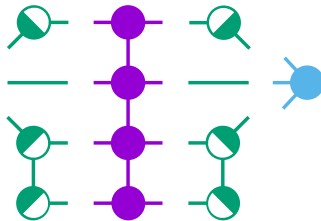


Contractions: local operator in TT algorithm (2)



Optimizations:

- Memory layout: group contracted dimensions, padding
- Special GEMM kernels (result stored reshaped/transposed)
- Multi-threaded loop-over-GEMM



Contractions: local operator in TT algorithm (2)



Optimizations:

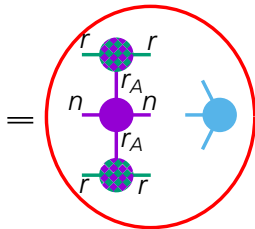
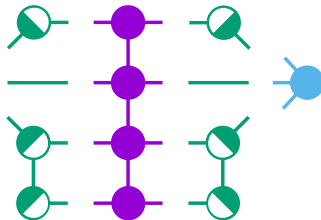
- Memory layout: group contracted dimensions, padding
- Special GEMM kernels (result stored reshaped/transposed)
- Multi-threaded loop-over-GEMM

Performance results (CPU):

	$n = 50, r = 500$	$n = 2, r = 2000$
naive opt-einsum	$0.015 P_{\text{peak}}$	$0.19 P_{\text{peak}}$
optimized	$0.33 P_{\text{peak}}$	$0.38 P_{\text{peak}}$

Setup:

- 192 core AMD Epyc 9965, Intel MKL 2025.3/2024.0
- Operator rank: $r_A = 2$



Operation:

$$y_{i'j'k'} \leftarrow \sum_{i,j,k,n} A_{ni} B_{nj} C_{nlk} A_{ni'} B_{nj'} C_{nlk'} x_{ijk}$$

- most costly step in a tree tensor network machine-learning algorithm
- gradient of loss wrt. one tensor
- dimension n is large (data dimension)

Operation:

$$y_{i'j'k'} \leftarrow \sum_{i,j,k,n} A_{ni} B_{nj} C_{nlk} A_{ni'} B_{nj'} C_{nlk'} x_{ijk}$$

- most costly step in a tree tensor network machine-learning algorithm
- gradient of loss wrt. one tensor
- dimension n is large (data dimension)

Optimizations:

- Parallelization over n .
- Iterating in batches over n
s.t. parts of A, B, C fit into the cache.
- Memory layout (as before).

Operation:

$$y_{i'j'k'} \leftarrow \sum_{i,j,k,n} A_{ni} B_{nj} C_{nlk} A_{ni'} B_{nj'} C_{nlk'} x_{ijk}$$

- most costly step in a tree tensor network machine-learning algorithm
- gradient of loss wrt. one tensor
- dimension n is large (data dimension)

Optimizations:

- Parallelization over n .
- Iterating in batches over n
s.t. parts of A, B, C fit into the cache.
- Memory layout (as before).

Performance results (CPU):

	$n=1.9 \cdot 10^7$
naive opt-einsum	$7 \cdot 10^{-4} P_{\text{peak}}$
optimized	$0.60 P_{\text{peak}}$

Setup:

- 192 core AMD Epyc 9965, Intel MKL 2024.0/2025.3
- $l=10, i,j,k,i',j',k'=50$
- mem.-bound without optimization (opt-einsum reaches $0.34 b_{\text{mem}}$)

Decomposition step (TT-SVD):

Given: $X \in \mathbf{R}^{n \times m}$, $n \gg m$, tolerance τ

Find: $B, Q : \|BQ^T - X\|_F \leq \tau$

Standard way (truncated SVD):

$$X = USV^T$$

$$B \leftarrow U_{:,1:r} S_{1:r,1:r}, \quad Q \leftarrow V_{:,1:r}$$

Optimization (Q-less TSQR + small SVD):

$$X = \bar{Q}R, \quad R = \bar{U}SV^T$$

$$B \leftarrow X V_{:,1:r}, \quad Q \leftarrow V_{:,1:r}$$

Decomposition step (TT-SVD):

Given: $X \in \mathbf{R}^{n \times m}$, $n \gg m$, tolerance τ

Find: $B, Q : \|BQ^T - X\|_F \leq \tau$

Standard way (truncated SVD):

$$X = USV^T$$

$$B \leftarrow U_{:,1:r} S_{1:r,1:r}, \quad Q \leftarrow V_{:,1:r}$$

Optimization (Q-less TSQR + small SVD):

$$X = \bar{Q}R, \quad R = \bar{U}SV^T$$

$$B \leftarrow X V_{:,1:r}, \quad Q \leftarrow V_{:,1:r}$$

Recompression step:

Given: $X \in \mathbf{R}^{n \times m}$, $Q \in \mathbf{R}^{k \times m}$ Find:

$\hat{Q}, Y : \|\hat{Q}Y - XQ^T\|_F \leq \tau$

Standard way (truncated SVD):

$$X = USV^T$$

$$\hat{Q} \leftarrow U_{:,1:r}, \quad Y \leftarrow S_{1:r,1:r} V^T Q^T$$

Optimization (Q-less TSQR + small SVD):

$$X = \bar{Q}R, \quad R = \bar{U}SV^T$$

$$\hat{Q} \leftarrow X V_{:,1:r} S_{1:r,1:r}^{-1}, \quad Y \leftarrow S_{1:r,1:r} V^T Q^T$$

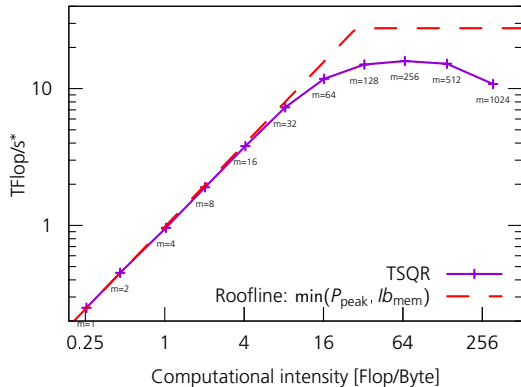
Decompositions: Q-less (TS)QR algorithm



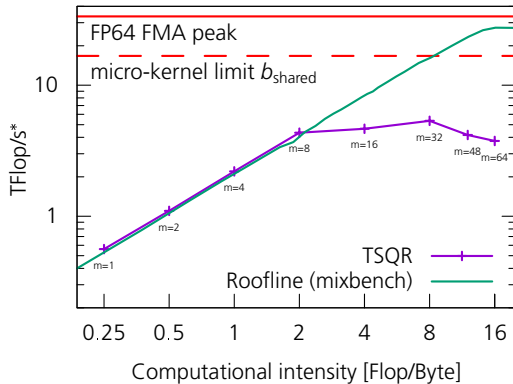
- Based on (Demmel et al.2012)
- Avoid storing Q : fewer data transfers, simpler to implement
- Lots of optimizations:
 - built upon trapezoidal QR in cache / GPU shared memory (GPU limited by shared mem. size!)
 - applies 2 Householder reflections to 4 vectors (micro-kernel)
 - CPU: parallel recursive blocking over columns within one block of the matrix
 - GPU: interleaved data transfers and computations on multiple levels
 - ...
- Complicated implementations (≥ 1500 lines templated C++ code for both CPU and GPU)

Decompositions: Q-less (TS)QR performance

CPU: (2×192 core AMD Epyc 9965)



GPU: (Nvidia H100, preliminary results)

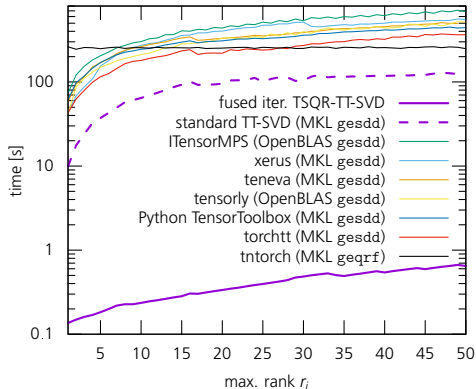


*based on the ideal complexity $2nm^2$, dimensions $n \times m$ with $nm = 6.6 \cdot 10^9$, mixbench from (Konstantinidis et al.2017)

Results: complete algorithms (64 core AMD 7773X)



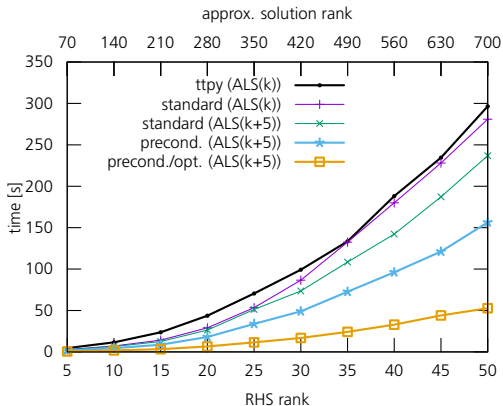
TT-SVD: (Oseledets2011)



Decompose 2^{30} tensor with prescribed rank.
(Röhrig-Zöllner et al.2022).

M. Röhrig-Zöllner, DLR Institute of Software Technology, 2026-03-05

TT-AMEn (ALS): (Dolgov et al.2014)



Solve $50^{10} \times 50^{10}$ linear system (conv. diff. problem).
(Röhrig-Zöllner et al.2025).

Contributions:

- Optimizations for series of contractions: speedup $2\times$ - $20\times$ (local TT operator) and $\sim 10^4$ (TTN local gradient)
- High-performance Q-less TSQR implementation: $0.57 P_{\text{peak}}$ (CPU) and $0.16 P_{FMA}$ (GPU) ($\sim 10\times$ faster than cuSOLVER QR)
- Speedup of complete algorithms: $3\times$ (TT-ALS) to $500\times$ (TT-SVD)

Ongoing and future work:

- Compare Q-less QR algorithms for tall-skinny matrices on current CPUs and GPUs.
- Vision: Q-less QR and faster tall-skinny SVD in LAPACK

Acknowledgements:

Gerhard Wellein & the RRZE team for their support and access to the Helma cluster

-  Demmel, J. et al. (Jan. 2012). "Communication-optimal Parallel and Sequential QR and LU Factorizations". In: *SIAM Journal on Scientific Computing* 34.1, A206–A239. ISSN: 1095-7197. DOI: 10.1137/080731992.
-  Dolgov, Sergey V. and Dmitry V. Savostyanov (Jan. 2014). "Alternating Minimal Energy Methods for Linear Systems in Higher Dimensions". In: *SIAM Journal on Scientific Computing* 36.5, A2248–A2271. DOI: 10.1137/140953289.
-  Konstantinidis, Elias and Yiannis Cotronis (Sept. 2017). "A quantitative roofline model for GPU kernel performance estimation using micro-benchmarks and hardware metric profiling". In: *Journal of Parallel and Distributed Computing* 107, pp. 37–56. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2017.04.002.
-  Oseledets, I. V. (Jan. 2011). "Tensor-Train Decomposition". In: *SIAM Journal on Scientific Computing* 33.5, pp. 2295–2317. DOI: 10.1137/090752286.
-  Röhrig-Zöllner, Melven et al. (Feb. 2025). "Performance of linear solvers in tensor-train format on current multicore architectures". In: *The International Journal of High Performance Computing Applications*. ISSN: 1741-2846. DOI: 10.1177/10943420251317994.
-  Röhrig-Zöllner, Melven, Jonas Thies, and Achim Basermann (July 2022). "Performance of the Low-Rank TT-SVD for Large Dense Tensors on Modern MultiCore CPUs". In: *SIAM Journal on Scientific Computing* 44.4, pp. C287–C309. DOI: 10.1137/21m1395545.
-  Smith, G. A. Daniel and Johnnie Gray (June 2018). "opt_einsum - A Python package for optimizing contraction order for einsum-like expressions". In: *Journal of Open Source Software* 3.26, p. 753. ISSN: 2475-9066. DOI: 10.21105/joss.00753.