Fast & Layout-Oblivious Tensor-Matrix Multiplication with BLAS

Cem Bassoy
cem.bassoy@gmail.com

July 4, 2024

TUHH
Hamburg
University of
Technology

Tensor-Matrix Multiplication (TTM)

Function Arguments

- dense input tensor $\underline{\mathbf{A}} \in \mathbb{R}^{n_1 \times \cdots \times n_q \times \cdots \times n_p}$ with order $p > 1$
- dense input matrix $\mathbf{B} \in \mathbb{R}^{m \times n_q}$
- dense output tensor $\underline{\mathbf{C}} \in \mathbb{R}^{n_1 \times \cdots \times m \times \cdots \times n_p}$
- contraction mode q with $1 \leq q \leq p$

Mode-q Tensor-Matrix Multiplication (Mode-q Product)

$$\underline{\mathbf{C}} = \underline{\mathbf{A}} \times_q \mathbf{B} \quad \Leftrightarrow \quad \underline{\mathbf{C}}([\mathbf{i}_1, j, \mathbf{i}_2]) = \sum_{i_q=1}^{n_q} \underline{\mathbf{A}}([\mathbf{i}_1, i_q, \mathbf{i}_2]) \cdot \mathbf{B}(j, i_q)$$

with $\mathbf{i}_1 = (i_1, \ldots, i_{q-1})$ and $\mathbf{i}_2 = (i_{q+1}, \ldots, i_p)$.

Linear Tensor Layout

Linear Tensor Layout (informal)

Given an order-p tensor $\underline{\mathbf{A}}$ with dimension tuple $\mathbf{n} = (n_1, \ldots, n_p)$, a linear tensor layout of $\underline{\mathbf{A}}$ is given by a **permutation tuple** $\boldsymbol{\pi}$ of length p which specifies the priority of its tensor modes.

Example

$$\underline{\mathbf{A}} = \left(\begin{array}{cc|cc} a_{1,1,1} & a_{1,2,1} & a_{1,1,2} & a_{1,2,2} \\ a_{2,1,1} & a_{2,2,1} & a_{2,1,2} & a_{2,2,2} \end{array} \right) \in \mathbb{R}^{2\times 2\times 2}$$

first-order layout (column-major): $\boldsymbol{\pi}_F := (1, 2, 3)$

$$\mathbf{a}_{\boldsymbol{\pi}_F} = (\, a_{1,1,1},\, a_{2,1,1},\, a_{1,2,1},\, a_{2,2,1},\, \ldots,\, a_{2,2,2} \,)_{\boldsymbol{\pi}_F}$$

last-order layout (row-major): $\boldsymbol{\pi}_L := (3, 2, 1)$

$$\mathbf{a}_{\boldsymbol{\pi}_L} = (\, a_{1,1,1},\, a_{1,1,2},\, a_{1,2,1},\, a_{1,2,2},\, \ldots,\, a_{2,2,2} \,)_{\boldsymbol{\pi}_L}$$

A dense order-p tensor can be stored according to any of the $p!$ different tensor layouts.

BLAS-based Approach

Loops-over-GEMM (LoG)

Y. Shi et al. (2016). "Tensor Contractions with Extended BLAS Kernels on CPU and GPU". In: *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, pp. 193–202

Jiajia Li et al. (2015). "An input-adaptive and in-place approach to dense tensor-times-matrix multiply". In: *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*. IEEE, pp. 1–12

Cem Bassoy (2019). "Design of a High-Performance Tensor-Vector Multiplication with BLAS". In: *International Conference on Computational Science*. Springer, pp. 32–45

Cem Savaş Başsoy (2024). "Fast and Layout-Oblivious Tensor-Matrix Multiplication with BLAS". In: *International Conference on Computational Science*. Springer, pp. 256–271

Benefits of LOG Approach

- \+ in-place operation: does not need to flatten the matrix (cf. TTGT)
- \+ uses of BLAS: no need to fine tune for a specific processor (cf. GETT)
- \+ easier to handle multiple layouts
- \+ can be as efficient as TBLIS and other high-performance implementations

Layout-Oblivious TTM with BLAS

8 Cases with gemv & gemm

order p	layout $\boldsymbol{\pi}$	c.mode q	blas	T	M	N	K	A	LDA	B	LDB	LDC
1	-	1	gemv	-	m	n1	-	B	n1	A	-	-
2	cm	1	gemm	tB	n2	m	n1	A	n1	B	n1	m
2	cm	2	gemm	-	m	n1	n2	B	n2	A	n1	n1
2	rm	1	gemm	-	m	n2	n1	B	n1	A	n2	n2
2	rm	2	gemm	tB	n1	m	n2	A	n2	A	n2	m
>2	any	π_1	gemm	tB	nn	m	nq	A	nq	B	nq	m
>2	any	π_p	gemm	-	m	nn	nq	B	nq	A	nn	nn
>2	any	$\pi_2,..,\pi_{p-1}$	gemm*	-	m	n_{π_1}	nq	B	nq	A	wq	wq

Features

- eight cases cover all linear tensor layouts $\boldsymbol{\pi}$ and contraction modes q
- table contains the maximum amount of (useful) cases$^{(\star)}$
- no copy operation and only $\mathbf{B}$ is transposed
- only case 8 calls `gemm` multiple times or once with `batch_gemm`

Skeleton for LoG-based TTM

Inner Product of Tensor Fibers

function `ttm`$(\underline{\mathbf{A}}, \mathbf{B}, \underline{\mathbf{C}}, \mathbf{i}, \mathbf{n}, m, q \neq 1, p > 1)$

for $i_p \leftarrow 1$ **to** n_p **do**
- $\ldots$
- **for** $i_{q+1} \leftarrow 1$ **to** n_{q+1} **do**
  - **for** $i_{q-1} \leftarrow 1$ **to** n_{q-1} **do**
    - $\ldots$
    - **for** $i_1 \leftarrow 1$ **to** n_1 **do**
      - **for** $j \leftarrow 1$ **to** m **do**
        - **for** $i_q \leftarrow 1$ **to** n_q **do**
          - $\underline{\mathbf{C}}(i_1, .., i_{q-1}, j, i_{q+1}, .., i_p)$ += $\underline{\mathbf{A}}(i_1, .., i_q, .., i_p)\ \mathbf{B}(j, i_q)$

Nested Recursion with Loops-over-Dot

Inner Product of Tensor Fibers with

---

function `ttm`$(\underline{\mathbf{A}}, \mathbf{B}, \underline{\mathbf{C}}, \mathbf{i}, \mathbf{n}, m, q \neq 1, p > 1)$

- **if** $r = q$ **then**
  - `ttm`$(\underline{\mathbf{A}}, \mathbf{B}, \underline{\mathbf{C}}, \mathbf{n}, \mathbf{i}, m, q, r-1)$
- **else if** $r \geq 1$ **then**
  - **for** $i_r \leftarrow 1$ **to** n_r **do**
    - `ttm`$(\underline{\mathbf{A}}, \mathbf{B}, \underline{\mathbf{C}}, \mathbf{n}, \mathbf{i}, m, q, r-1)$
- **else**
  - **for** $j \leftarrow 1$ **to** m **do**
    - $\underline{\mathbf{C}}(\mathbf{i}_1, j, \mathbf{i}_2) =$ `dot`$(n_q, \underline{\mathbf{A}}(\mathbf{i}_1, :_q, \mathbf{i}_2), w_q, \mathbf{B}(j, :), 1)$

- Strided access of $\underline{\mathbf{A}}$ and $\underline{\mathbf{C}}$ with step $w_q = \prod_{r=1}^{q-1} n_r$.
- No temporal data locality larger n.

Nested Recursion with Loops-over-GEMM

TTM with Slice-Matrix Multiplication (<seq-loop,par-gemm,slice>)

function `ttm`$(\underline{\mathbf{A}}, \mathbf{B}, \underline{\mathbf{C}}, \mathbf{n}, \mathbf{i}, m, q, r)$
- **if** $r = q$ **then**
  - `ttm`$(\underline{\mathbf{A}}, \mathbf{B}, \underline{\mathbf{C}}, \mathbf{n}, \mathbf{i}, m, q, r-1)$
- **else if** $r > 1$ **then**
  - **for** $i_r \leftarrow 1$ **to** n_r **do**
    - `ttm`$(\underline{\mathbf{A}}, \mathbf{B}, \underline{\mathbf{C}}, \mathbf{n}, \mathbf{i}, m, q, r-1)$
- **else**
  - `gemm`$(\texttt{rm}, \texttt{-}, \texttt{-}, m, n_{\pi_1}, n_q, 1, \mathbf{B}, n_q, \underline{\mathbf{A}}(:, \mathbf{i}'_1, :, \mathbf{i}_2), w_q, 1, \underline{\mathbf{C}}(:, \mathbf{i}'_1, :, \mathbf{i}_2), w_q)$

- $\mathbf{i}'_1 = (i_2, \ldots, i_{q-1})$ and $\mathbf{i}'_1 = (i_{q+1}, \ldots, i_p)$
- tensor order p must be $p > 2$ and contraction mode $q \neq \pi_1$
- `else` branch performs a slice-matrix multiplication
- preserves spatial data locality for any contraction mode and tensor layout

Tensor Slicing and Parallelization

Tensor slicing for e.g. $\boldsymbol{\pi} = (1, 2, \ldots, p)$

- slice 2 dimensions of $\underline{\mathbf{A}}$ and $\underline{\mathbf{C}}$, e.g. slice $\mathbf{A}' = \underline{\mathbf{A}}(:, i_2, \ldots, i_{q-1}, :, i_{q+1}, \ldots, i_p)$ or
- slice q dimensions of $\underline{\mathbf{A}}$ and $\underline{\mathbf{C}}$, e.g. subtensor $\mathbf{A}' = \underline{\mathbf{A}}(:, \ldots, :, i_{q+1}, \ldots, i_p)$
- call `GEMM` executing $\mathbf{C}' = \mathbf{B} \cdot_{\text{row}} \mathbf{A}'$

Tensor Slicing and Parallelization

Tensor slicing for e.g. $\boldsymbol{\pi} = (1, 2, \ldots, p)$

- slice 2 dimensions of $\underline{\mathbf{A}}$ and $\underline{\mathbf{C}}$, e.g. slice $\mathbf{A}' = \underline{\mathbf{A}}(:, i_2, \ldots, i_{q-1}, :, i_{q+1}, \ldots, i_p)$ or
- slice q dimensions of $\underline{\mathbf{A}}$ and $\underline{\mathbf{C}}$, e.g. subtensor $\mathbf{A}' = \underline{\mathbf{A}}(:, \ldots, :, i_{q+1}, \ldots, i_p)$
- call `GEMM` executing $\mathbf{C}' = \mathbf{B} \cdot_{\text{row}} \mathbf{A}'$

Multithreaded Parallel Execution

- outer modes/loops (OpenMP Parallelization)
  - slice: $(2, \ldots, q-1)$ $(q+1, \ldots, p)$
  - subtensor: $(q+1, \ldots, p)$
- Inner modes/loops (Multithreaded `GEMM`)
  - slice: 1 and q
  - subtensor: $1, 2, \ldots, q$

Tensor Slicing and Parallelization

Tensor slicing for e.g. $\boldsymbol{\pi} = (1, 2, \ldots, p)$

- slice 2 dimensions of $\underline{\mathbf{A}}$ and $\underline{\mathbf{C}}$, e.g. slice $\mathbf{A}' = \underline{\mathbf{A}}(:, i_2, \ldots, i_{q-1}, :, i_{q+1}, \ldots, i_p)$ or
- slice q dimensions of $\underline{\mathbf{A}}$ and $\underline{\mathbf{C}}$, e.g. subtensor $\mathbf{A}' = \underline{\mathbf{A}}(:, \ldots, :, i_{q+1}, \ldots, i_p)$
- call `GEMM` executing $\mathbf{C}' = \mathbf{B} \cdot_{\text{row}} \mathbf{A}'$

Multithreaded Parallel Execution

- outer modes/loops (OpenMP Parallelization)
  - slice: $(2, \ldots, q-1)$ $(q+1, \ldots, p)$
  - subtensor: $(q+1, \ldots, p)$
- Inner modes/loops (Multithreaded `GEMM`)
  - slice: 1 and q
  - subtensor: $1, 2, \ldots, q$

Examples

- Variants with slices: `<par-loops,slice>`, `<par-gemm,slice>`
- Variants with subtensors: `<par-loops,subtensor>`, `<par-gemm,subtensor>`

Experimental Setup

Computing system

- Intel Xeon Gold 5318Y with 2×24 cores running 2.1 GHz (base freq.)
  - measured max. 3.80 TFLOPS (79.25 GFLOPS/core).
- AMD EPYC 9354 with 2×32 cores running 3.25 GHz (base freq.)
  - measured max. 3.87 TFLOPS (60.5 GFLOPS/core).
- `GCC v11.2` with `-Ofast` and `OpenMP 4.5` and `MKL 2022` and `AOCL 4.2`.

Tensor elements

- are contiguously stored according to the first-order storage format
- are floating-point numbers in double precision

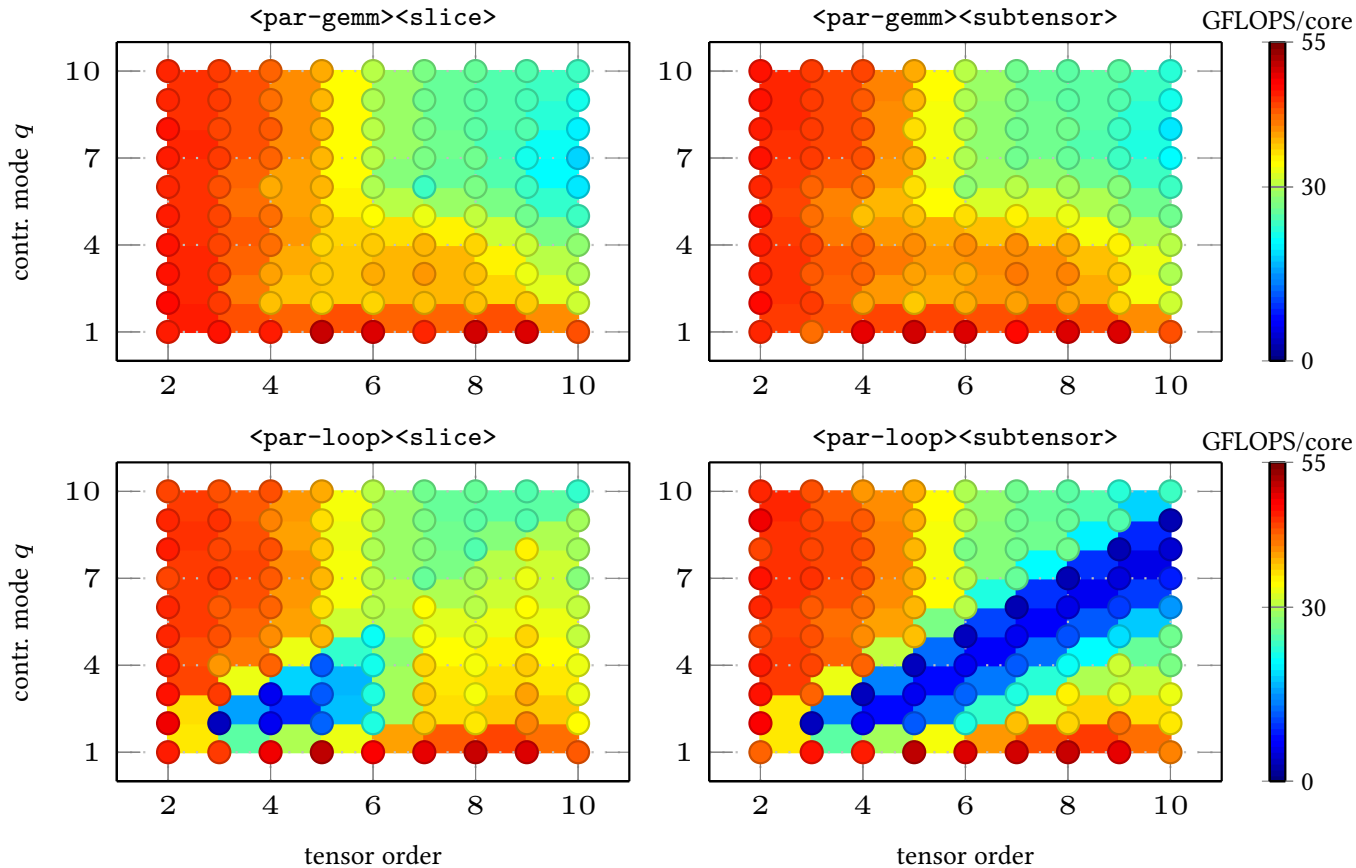
Test Tensors

- 1440 artificial non-symmetrically-shaped tensors, e.g. $1024 \times 2 \times 2048 \times 2 \times 2$.
- 336 artificial symmetrically-shaped tensors, e.g. $256 \times 256 \times 256$.
- 8 real-world order-4 tensors from SDRbench, e.g. $256 \times 384 \times 384 \times 7$.

`https://sdrbench.github.io/`

Multi-Threaded Parallel Execution (1)

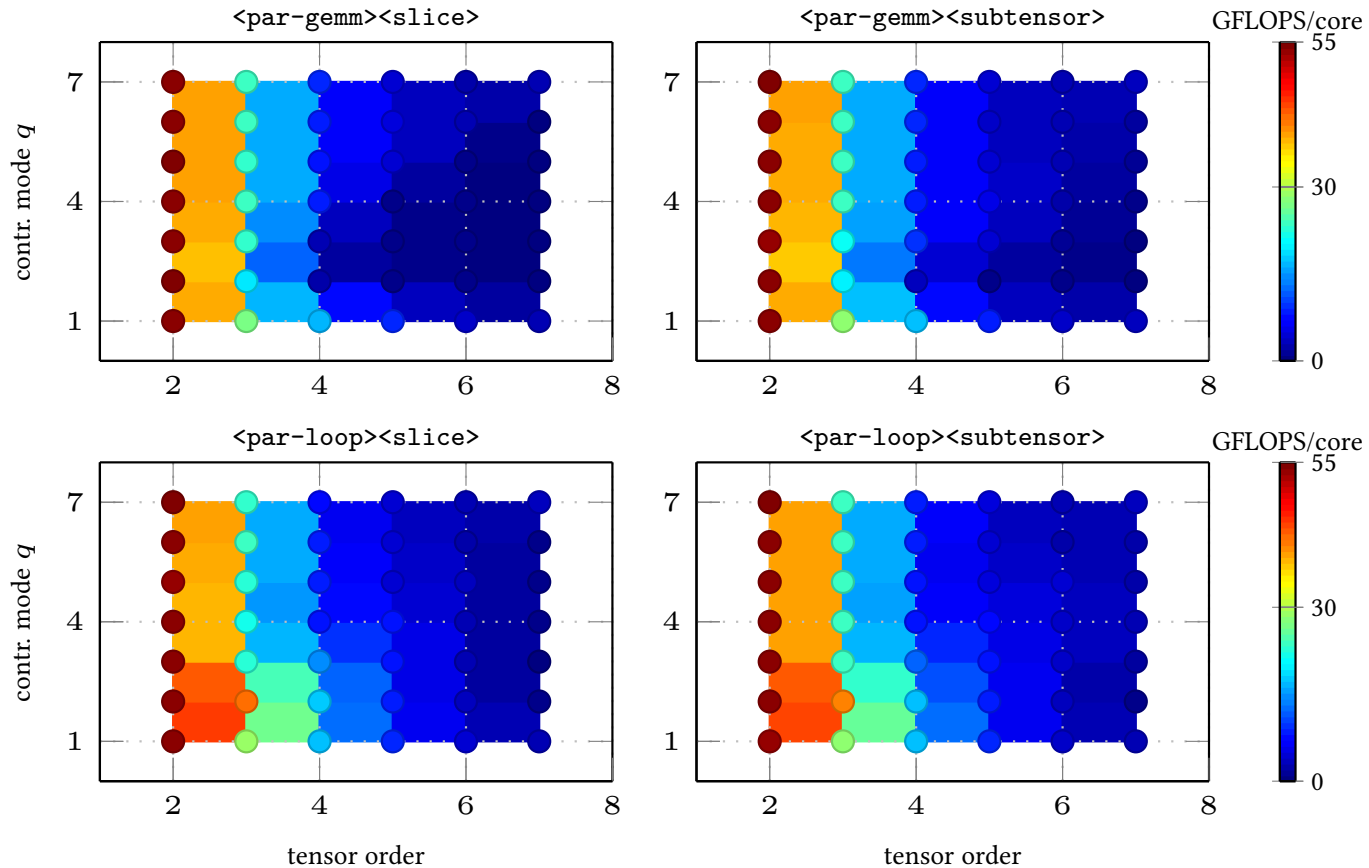
Asymmetrically Shaped Tensors



- `<par-gemm,*>` achieves 36 GFLOPS/core and outperforms `<par-gemm,*>` by 2.31x
- `<par-loops,*>` shows a slow performance due to large tensor slices

Multi-Threaded Parallel Execution(2)

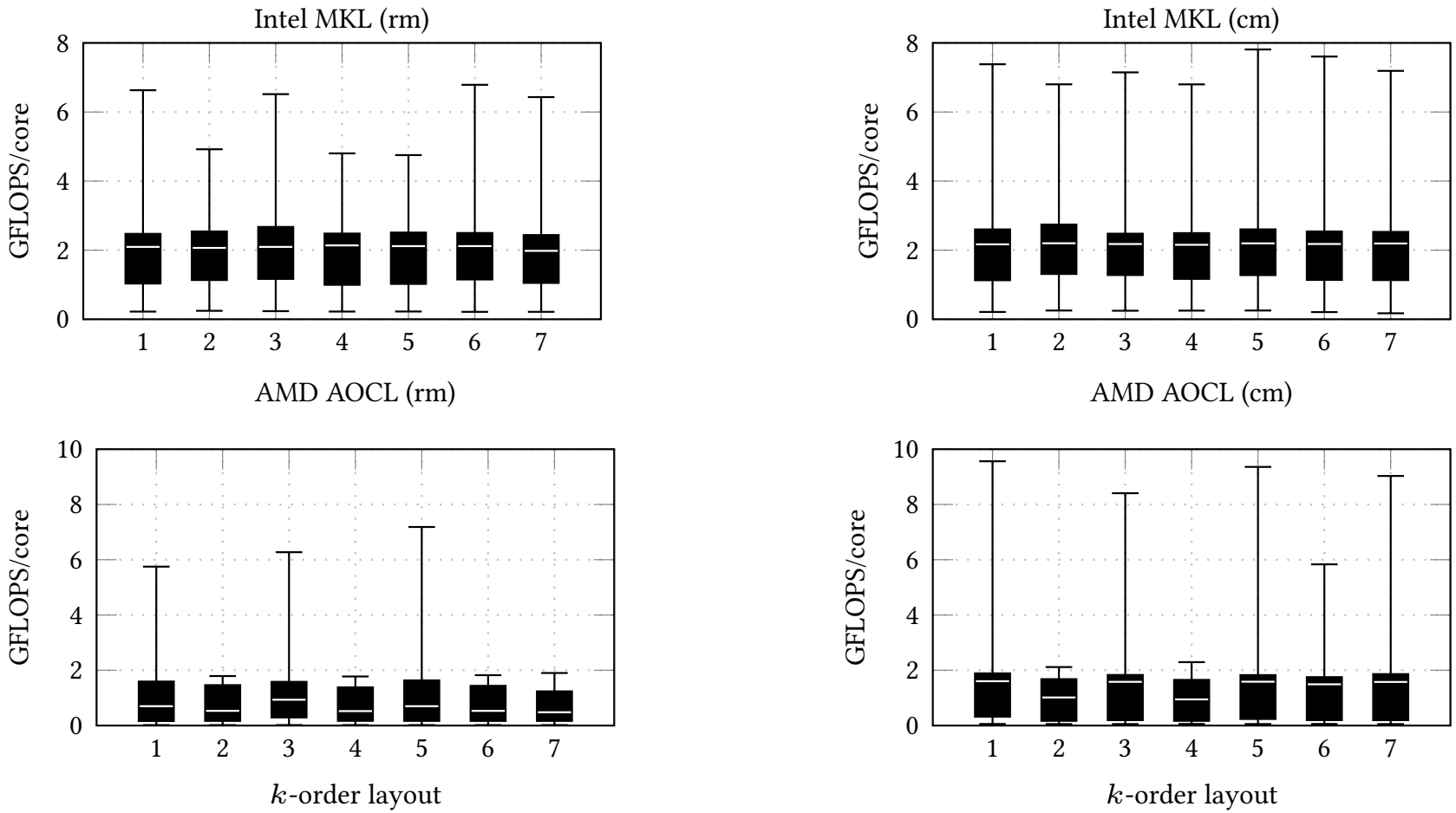
Symmetrically Shaped Tensors



- `<par-loop,*>` outperforms `<par-gemm,*>` by 24% and 33%
- `<par-loop,*>` performs best for $1 < q < p$.

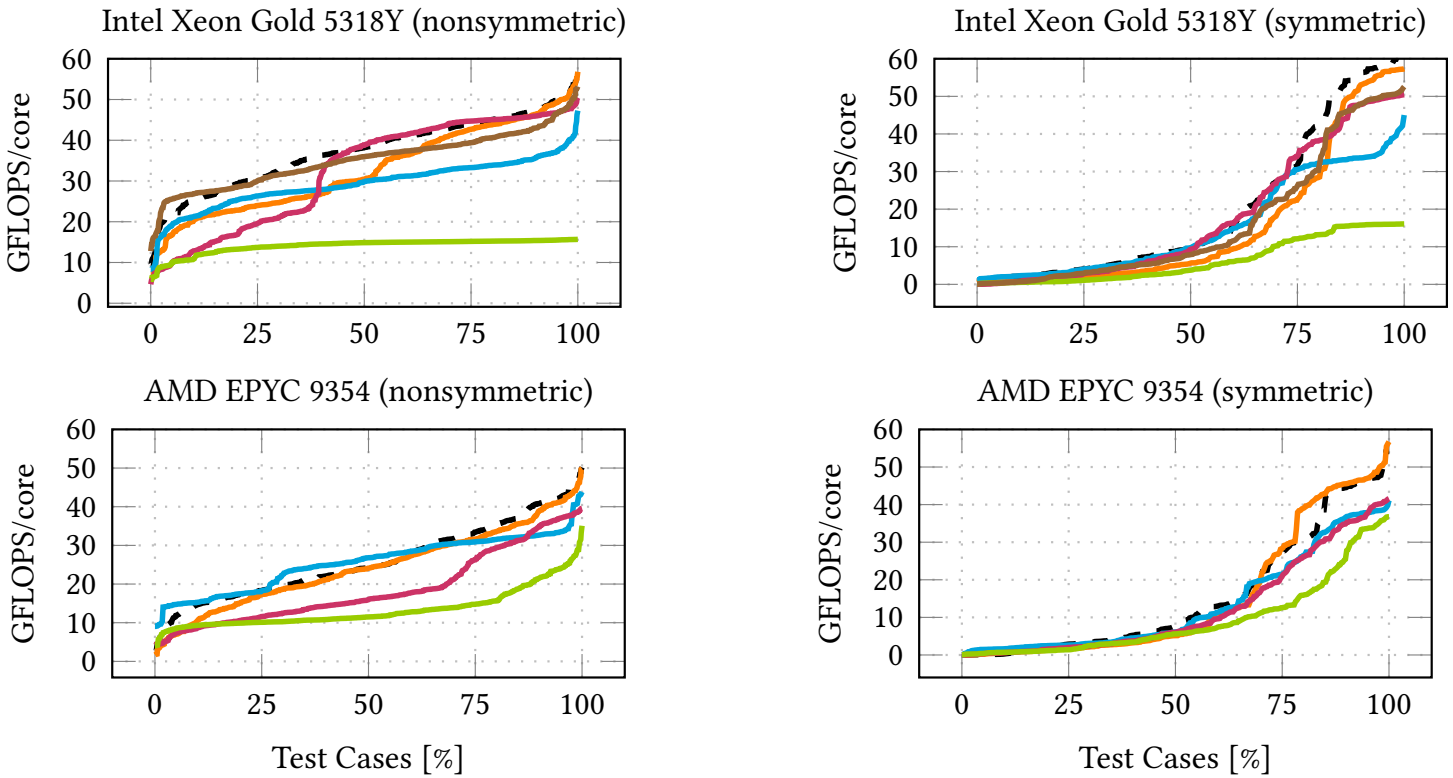
Layout Agnostic Performance

Symmetric Order-8 Tensor



- Median throughput across all layouts is ca. 2 GFLOPS/core (Intel) and ca. 1.5 GFLOPS/core (AMD)
- Low performance due to small slice dimensions (at most 48) and the contraction dimension (8)

Performance Comparison with other Approaches



`TLIB[ours]` (- - -), `TCL` (——), `TBLIS 1.2.` (——), `LibTorch 2.5` (——), `Eigen` (——), `TuckerMPI` (——)

Asymmetrically Shaped Tensors:

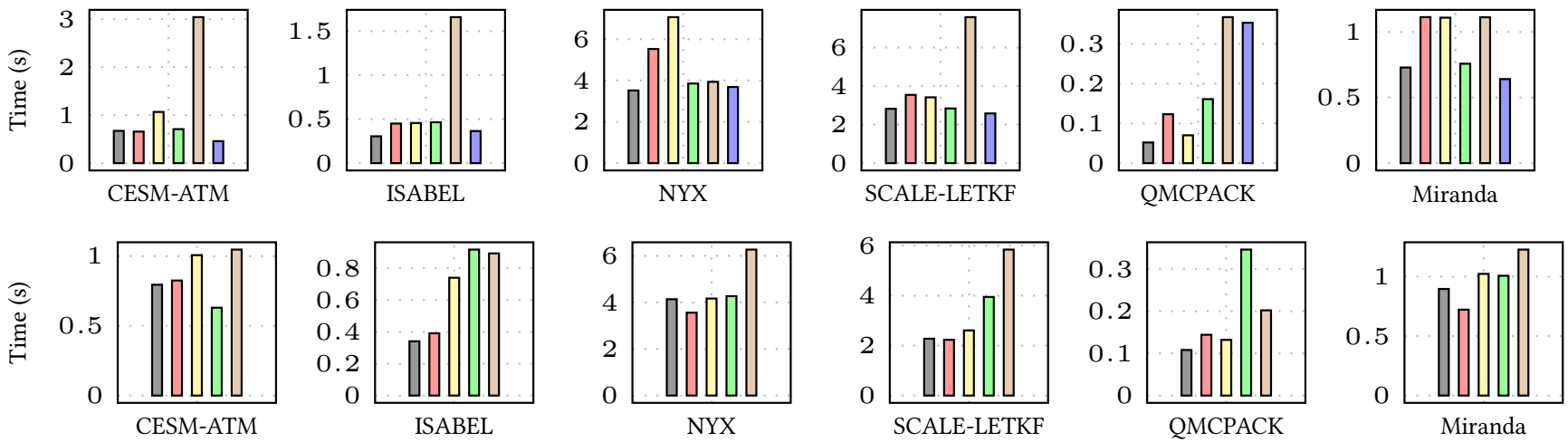
- Intel: `tlib` 38.27, `torch`: 38.17, `tcl`: 30.46, `tblis`: 29.14
- AMD: `tblis` 26.81, `tlib`: 24.28, `tcl`: 24.11, `torch`: 16.04

Symmetrically Shaped Tensors (right):

- Intel: `tblis` 9.73, `torch`: 9.31, `tlib`: 8.99, `tucker`: 7.91
- AMD: `tlib` 7.75, `tblis`: 6.14, `torch`: 6.04, `eigen`: 5.58

Performance Comparison with other Approaches

Tensors from the SDR Benchmarks



`TLIB[ours]` ( ), `TCL` ( ), `TBLIS` ( ), `LibTorch` ( ), `Eigen` ( ) and `TuckerMPI` ( ).

- Intel (top): on par with `tblis` or faster in some instances
- AMD (bottom): on par with `tblis` or slower in some instances

Paper & Implementation

`https://github.com/bassoy/ttm`

README LGPL-3.0 license

High-Performance Tensor-Matrix Multiplication Library - TLIB(TTM)

C++ 17 | license GPL | ttm wiki | ttm discussions | Linux passing

Summary

TLIB(TTM) is C++ high-performance tensor-matrix multiplication **header-only library**. It provides free C++ functions for parallel computing the **mode-`q` tensor-times-matrix product** of the general form

$$\underline{\mathbf{C}} = \underline{\mathbf{A}} \times_q \mathbf{B} \quad :\Leftrightarrow \quad \underline{\mathbf{C}}(i_1,...,i_{q-1},j,i_{q+1},...,i_p) = \sum_{i_q=1}^{n_q} \underline{\mathbf{A}}(i_1,...,i_q,...,i_p) \cdot \mathbf{B}(j,i_q)\,.$$

where q is the contraction mode, $\underline{\mathbf{A}}$ and $\underline{\mathbf{C}}$ are tensors of order p with shapes $\mathbf{n}_a = (n_1,...n_{q-1},n_q,n_{q+1},...,n_p)$ and $\mathbf{n}_c = (n_1,...,n_{q-1},m,n_{q+1},...,n_p)$, respectively. The order 2 tensor $\mathbf{B}$ is a matrix with shape $\mathbf{n}_b = (m,n_q)$.

Usage & Installation

Please have a look at the wiki page for more informations about library **usage**, function **interfaces** and the **parameters** settings.

Key Features

Flexibility

- Contraction mode q, tensor order p, tensor extents n and tensor layout π can be chosen at runtime
- Supports any linear tensor layout inlcuding the first-order and last-order storage layouts
- Offers two high-level and one C-like low-level interfaces for calling the tensor-times-matrix multiplication
- Implemented independent of a tensor data structure (can be used with `std::vector` and `std::array`)
- Currently supports float and double

Cem Savaş Başsoy (2025). “Design of a high-performance tensor–matrix multiplication with BLAS”. In: *Journal of Computational Science* 87

Future Work

- Fine-granular slicing and parallelization
- Apply performance modeling based on the `gemm` runtime
- Implement **TTT** cases using **TTM** as a base

Fin