Tensor-Times-Vector: A Use-Case for "Loop-over-BLIS"

Cem (Gemm) Bassoy
cem.bassoy@gmail.com

September 30, 2022

TUHH
Hamburg
University of
Technology

A tensor $\underline{\mathbf{A}}$ of order p is an element of the tensor space A with $A = V_1 \otimes \cdots \otimes V_p$ being the outer product of p vector spaces V_i[a].

[a] Wolfgang Hackbusch (2014). "Numerical tensor calculus". In: *Acta numerica* 23, pp. 651–742.

A tensor can be thought of as a multi-index numerical array of which the order is the number of its modes / dimensions[a].

[a] Andrzej Cichocki et al. (2015). "Tensor decompositions for signal processing applications: From two-way to multiway component analysis". In: *IEEE Signal Processing Magazine* 32.2, pp. 145–163.

0^{th}-order tensor is a **scalar** denoted by $a \in \mathbb{F}$

1^{st}-order tensor is a **vector** denoted by $\mathbf{a} \in \mathbb{F}^n$

2^{nd}-order tensor is a **matrix** denoted by $\mathbf{A} \in \mathbb{F}^{n_1 \times n_2}$

p^{th}-order tensor is a **tensor** denoted by $\underline{\mathbf{A}} \in \mathbb{F}^{n_1 \times n_2 \times \cdots \times n_p}$ of order $p > 2$

Tensor-Vector Multiplication (TTV)

- Given a p^{th}-order tensor $\underline{\mathbf{A}} \in \mathbb{F}^{n_1 \times \cdots \times n_p}$,
- $(p\text{-}1)^{th}$-order tensor $\underline{\mathbf{C}} \in \mathbb{F}^{n_1 \times \cdots \times n_{q-1} \times n_{q+1} \times \cdots \times n_p}$ and
- a vector $\mathbf{b} \in \mathbb{F}^{n_q}$.

A q-mode **tensor-vector multiplication** (TTV) is denoted by $\underline{\mathbf{C}} = \underline{\mathbf{A}} \times_q \mathbf{b}$ where q is the contraction mode with $1 \leq q \leq p$ and

$$\underline{\mathbf{C}}(i_1, \ldots, i_{q-1}, i_{q+1}, \ldots, i_p) = \sum_{i_q=1}^{n_q} \underline{\mathbf{A}}(i_1, \ldots, i_q, \ldots, i_p) \cdot \mathbf{b}(i_q)$$

is an element of $\underline{\mathbf{C}}$ with $1 \leq i_r \leq n_r$ for $1 \leq r \leq p$.

Higher-Order Power-Method for Rank-1 Approximation

$$\min_{\underline{\hat{\mathbf{C}}}} ||\underline{\mathbf{C}} - \underline{\hat{\mathbf{C}}}|| \quad \text{with} \quad \underline{\hat{\mathbf{C}}} = \sigma \cdot \mathbf{v}_1 \circ \mathbf{v}_2 \circ \cdots \circ \mathbf{v}_p$$

function $[\, \sigma_p^{(k)}, \mathbf{v}_1^{(k)}, \ldots, \mathbf{v}_p^{(k)} \,]$ = `hopm(`$\underline{\mathbf{A}}, \epsilon, \mathbf{v}_1^{(0)}, \ldots, \mathbf{v}_p^{(0)}$`)`

- $k \leftarrow 1$
- **repeat**
  - **for** $j \leftarrow 1$ **to** p **do**
    - $\mathbf{v}_j^{(k+1)} \leftarrow \underline{\mathbf{A}} \times_1 \mathbf{v}_1^{(k)} \cdots \times_{j-1} \mathbf{v}_{j-1}^{(k)} \times_{j+1} \mathbf{v}_{j+1}^{(k)} \cdots \times_p \mathbf{v}_p^{(k)}$
    - $\sigma_j^{(k+1)} \leftarrow \|\mathbf{v}_j^{(k+1)}\|$
    - $\mathbf{v}_j^{(k+1)} \leftarrow \mathbf{v}_j^{(k+1)} / \sigma_j^{(k+1)}$
  - $k \leftarrow k + 1$
- **until** *converged* **or** *maximum iteration count reached*

Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle (2000). "On the Best Rank-1 and Rank-(R1 ,R2 ,. . .,RN) Approximation of Higher-Order Tensors". In: *SIAM Journal on Matrix Analysis and Applications* 21.4, pp. 1324–1342

Eleftherios Kofidis and Phillip a. Regalia (2002). "On the Best Rank-1 Approximation of Higher-Order Supersymmetric Tensors". In: *SIAM Journal on Matrix Analysis and Applications* 23.3, pp. 863–884

Brett W. Bader and Tamara G. Kolda (2006). "Algorithm 862: MATLAB tensor classes for fast algorithm prototyping". In: *ACM Trans. Math. Softw.* 32 (4), pp. 635–653

Alternating Least-Squares for CP-Decomposition

$$\min_{\hat{\underline{\mathbf{C}}}} ||\underline{\mathbf{C}} - \hat{\underline{\mathbf{C}}}|| \quad \text{with} \quad \hat{\underline{\mathbf{C}}} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \cdots \circ \mathbf{c}_r$$

function $[\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_p] = \texttt{cp-als}(R, \underline{\mathbf{C}}, \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_p)$

- **repeat**
  - **for** $n = 1, \ldots, p$ **do**
    - **for** $r = 1, \ldots, R$ **do**
      - $\mathbf{M}(:, r) \leftarrow \underline{\mathbf{C}} \times_1 \mathbf{a}_1^{(r)} \cdots \times_{n-1} \mathbf{a}_{n-1}^{(r)} \times_{n+1} \mathbf{a}_{n+1}^{(r)} \cdots \times_p \mathbf{a}_p^{(r)}$
    - $\mathbf{H} \leftarrow \mathbf{A}_1^T \mathbf{A}_1 * \cdots * \mathbf{A}_{n-1}^T \mathbf{A}_{n-1} * \mathbf{A}_{n+1}^T \mathbf{A}_{n+1} * \cdots * \mathbf{A}_p^T \mathbf{A}_p$
    - $\mathbf{A}_n \leftarrow \mathbf{M}\mathbf{H}^\dagger$
  - **until** *converged* ***or*** *maximum iteration count reached*

estimator of the best-rank-1 approximation of $\underline{\mathbf{A}}$.

Tamara G Kolda and Brett W Bader (2009). "Tensor decompositions and applications". In: *SIAM review* 51.3, pp. 455–500

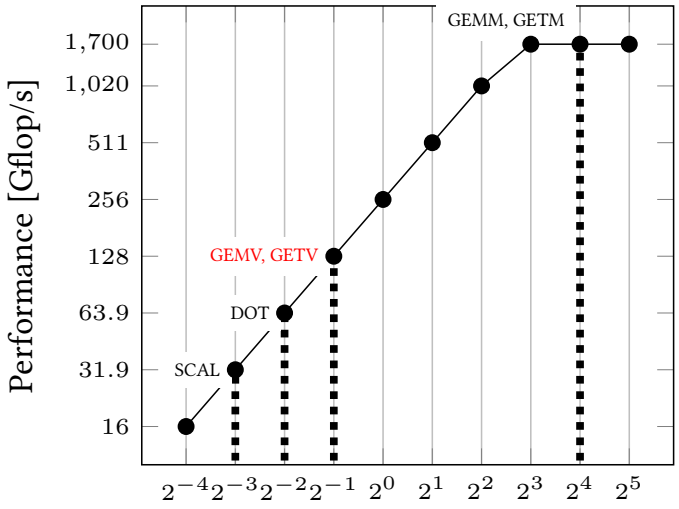
Basic Multilinear Algebra Subroutines (BMAS)

Level	Operation	BMAS	Boost.uBLAS Expression
1	$\underline{\mathbf{A}} \leftarrow \alpha \cdot \underline{\mathbf{A}}$	`TSCAL`	`x *= A`
1	$\alpha \leftarrow \langle \underline{\mathbf{A}} \cdot \underline{\mathbf{B}} \rangle$	`TDOT`	`inner_prod(A, B)`
1	$\alpha \leftarrow \sqrt{\langle \underline{\mathbf{A}}, \underline{\mathbf{A}} \rangle}$	`TNRM2`	`norm_2(A)`
1	$\underline{\mathbf{B}} \leftarrow \alpha \cdot \underline{\mathbf{A}} + \underline{\mathbf{B}}$	`TAXPY`	`B += a * A`
1	$\cdots$	$\cdots$	$\cdots$
2	$\mathbf{A} \leftarrow \alpha \cdot \underline{\mathbf{A}} \odot \underline{\mathbf{B}}$	`TGER`	`C += a * outer_prod(A, B)`
2	$\underline{\mathbf{C}} \leftarrow \alpha \cdot \underline{\mathbf{A}} \times_q \mathbf{x} + \beta \cdot \underline{\mathbf{C}}$	`GETV`	`C = a * prod(A, x, q) + b * C`
2	$\cdots$	$\cdots$	$\cdots$
3	$\underline{\mathbf{C}} \leftarrow \alpha \cdot \underline{\mathbf{A}} \times_q \mathbf{B} + \beta \cdot \underline{\mathbf{C}}$	`GETM`	`C = a * prod(A, B, q) + b * C`
3	$\cdots$	$\cdots$	$\cdots$
3/4	$\underline{\mathbf{C}} \leftarrow \alpha \cdot \underline{\mathbf{A}} \times_{q,r} \underline{\mathbf{B}} + \beta \cdot \underline{\mathbf{C}}$	`GETT`	`C = a * prod(A,B,q,r) + b * C`

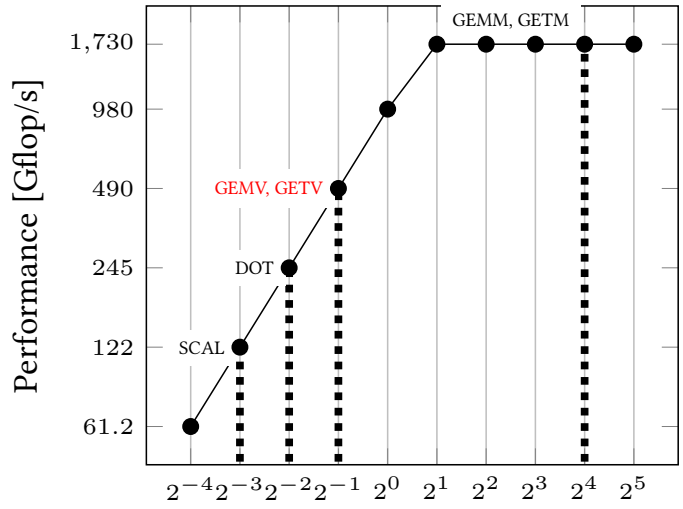
https://www.boost.org/doc/[..]/overview.html#functionality

Roofline Models

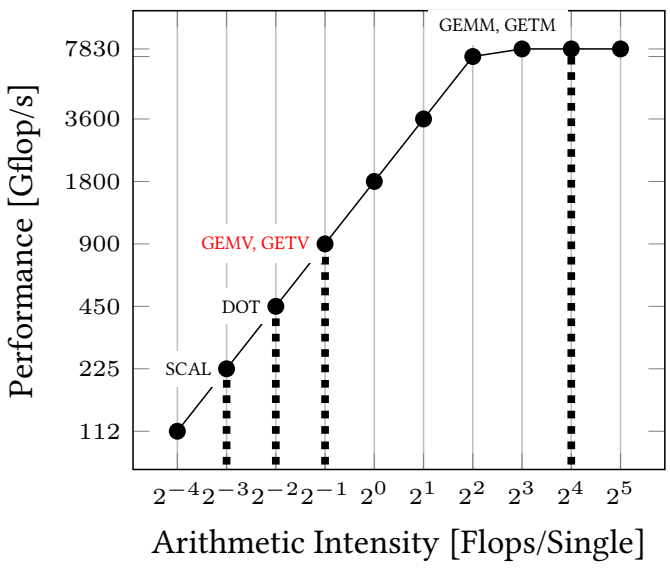
Intel-Xeon-8180M



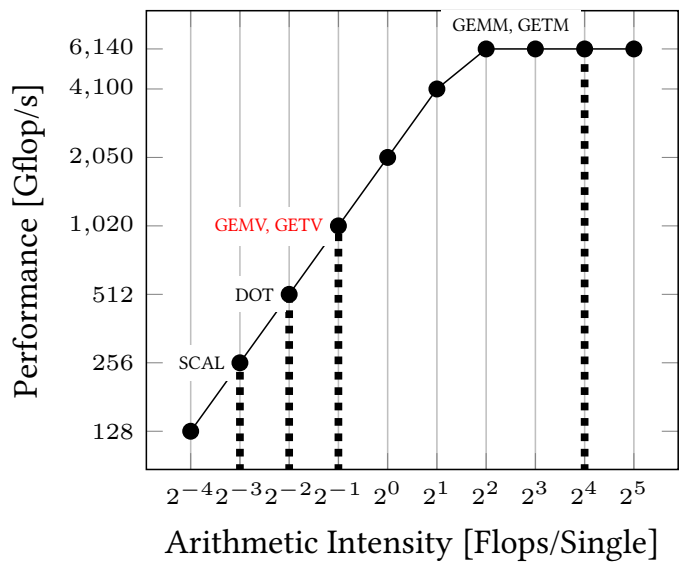
Intel-Xeon-Phi-7920



Nvidia-Tesla-V100



AMD-Radeon-Instinct-MI60



GEMV and GETV

$\mathbf{c} \leftarrow \alpha \cdot \mathbf{A}^{(T)} \cdot \mathbf{b} + \beta \cdot \mathbf{c}$	`GEMV(F,tA,m,n,alpha,A,lda,b,incb,beta,c,incc)`
$\underline{\mathbf{C}} \leftarrow \alpha \cdot \underline{\mathbf{A}}^{(\tau)} \times_q \mathbf{b} + \beta \cdot \underline{\mathbf{C}}$	`GETV(F,tA,n,p,q,alpha,A,lda,b,incb,beta,C,ldc)`

`F` : layout tuple $(\pi_1, \ldots, \pi_p)$ with $p!$ variations (or just k-mode)

`tA` : transposition tuple $(\tau_1, \ldots, \tau_p)$ with $p!$ variations

`n` : dimension tuple $(n_1, \ldots, n_p)$

`p` : number of dimensions/modes (order) with $p \in \mathbb{N}$

`q` : contraction mode with $1 \leq q \leq p$

`lda`: stride tuple with multiple leading dimensions

`ldc`: stride tuple with multiple leading dimensions

Baseline Algorithm

// $\underline{\mathbf{C}}(i_1, \ldots, i_{q-1}, i_{q+1}, \ldots, i_p) = \sum_{i_q=1}^{n_q} \underline{\mathbf{A}}(i_1, \ldots, i_q, \ldots, i_p) \cdot \mathbf{b}(i_q)$

---

function $[\ \underline{\mathbf{C}}\ ]$ = `ttv`$(\underline{\mathbf{A}}, \mathbf{b}, \underline{\mathbf{C}}, \mathbf{i}, \mathbf{n}, q \neq 1, p > 1)$

```
for i_p ← 1 to n_p do
    ...
    for i_{q+1} ← 1 to n_{q+1} do
        for i_{q-1} ← 1 to n_{q-1} do
            ...
            for i_1 ← 1 to n_1 do
                for i_q ← 1 to n_q do
                    C(i_1, ..., i_{q-1}, i_{q+1}, ..., i_p) += A(i_1, ..., i_q, ..., i_p) · b(i_q)
```

For any contraction mode $q \neq 1$, spatial data locality cannot be preserved if $\pi_1 \neq q$.

Implementation of High-Performance Tensor Contractions

Transpose-(Transpose)-BLAS-Transpose - TTBT (TTGT)

+ Use of efficient BLAS / BLIS
- (Un)Folding can take considerable amount of space and time[(1)]

Extended-GOTO or TBLIS (GETT)

+ High-performance kernels
- Implementation efforts (portability?)

Loop-Over-BLAS - LOB (LOG)

+ Uses BLAS without (un)folding
- Not applicable for all GETT constellations [(2)]

---

(1) 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

(2) Edoardo Di Napoli et al. (2014). "Towards an efficient use of the BLAS library for multilinear tensor contractions". In: *Applied Mathematics and Computation* 235, pp. 454 –468

TTV Algorithm using DOT & GEMV

Case	Order (p)	Layout ($\boldsymbol{\pi}$)	Mode (q)	Routine (BLAS)	Format	M	N	LDA
1	1	-	1	DOT	-	n_1	-	-
2	2	$(1,2)$	1	GEMV	ROW	n_2	n_1	n_1
3	2	$(1,2)$	2	GEMV	COL	n_1	n_2	n_1
4	2	$(2,1)$	1	GEMV	COL	n_2	n_1	n_2
5	2	$(2,1)$	2	GEMV	ROW	n_1	n_2	n_2
6	> 2	any	π_1	GEMV	ROW	$\bar{n}_q$	n_q	n_q
7	> 2	any	π_p	GEMV	COL	$\bar{n}_q$	n_q	$\bar{n}_q$
8	> 2	any	$\pi_2, \ldots, \pi_{p-1}$	GEMV*	COL	$\hat{n}_q$	n_q	$\hat{n}_q$

$\hat{n}_q = \prod_{r=1}^{q-1} \mathbf{n}(\pi_r)$ and $\bar{n}_q = 1/\mathbf{n}(\pi_q) \prod_{r=1}^{p} \mathbf{n}(r)$,

TTV cases

case 1: execute one parallel DOT

case 2-4: execute one parallel GEMV with a matrix

case 6-7: execute one parallel GEMV with a reinterpreted tensor as matrix

case 8: execute sequential GEMV* with subtensors in parallel

Case 8: Loops-over GEMV with Subtensors

On a single core

Interpret a q^{th}-order subtensor of $\underline{\mathbf{A}}$ as a matrix $\mathbf{A}'$ of size $(n_1 \cdots n_{q-1}, n_q)$

$$\mathbf{A}' = \underline{\mathbf{A}}(:, \ldots, :, i_{q+1}, \ldots, i_p)$$

Interpret $(q-1)^{th}$-order subtensor of $\underline{\mathbf{C}}$ as a vector $\mathbf{c}'$ of size $(n_1 \cdots n_{q-1}, 1)$

$$\mathbf{c}' = \underline{\mathbf{C}}(:, \ldots, :, i_{q+1}, \ldots, i_p)$$

Execute a GEMV

$$\mathbf{c}' = \mathbf{A}' \cdot \mathbf{b}$$

On multiple cores

Execute multiple GEMV $\mathbf{c}' = \mathbf{A}' \cdot \mathbf{b}$ for all $i_{q+1}, \ldots, i_p$ in parallel

Experimental Setup

Computing system

- Core i9-7900X Intel Xeon processor with **10 cores** running 3.3 GHz
- 10 threads are set using `<openmp,mkl>_set_num_threads`
- `GCC v7.4` with `-Ofast`, `OpenMP 4.5` and `OpenBLAS 0.2.20` or `MKL 2019`.

Tensor elements

- are contiguously stored according to the 1^{st}**-order storage format**
- are floating-point numbers in **single precision**

Sustained performance of `GEMV` (expectation)

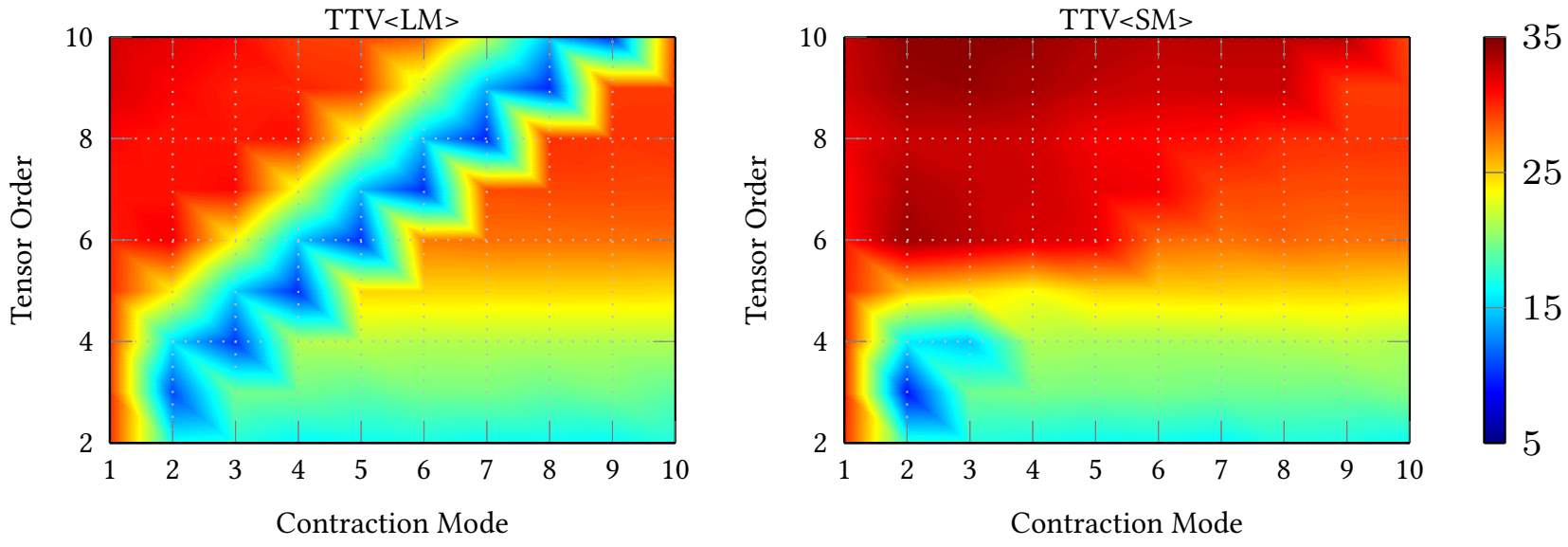
- MKL: 33 Gflops/s (Row-major), 30 Gflops/s (Column-major),
- OpenBLAS: 30 Gflops/s (Row-major), 29 Gflops/s (Column-major),

Test tensor sets

- $2880 = 9 \times 32 \times 10$ **a**symmetrically-shaped tensors ($\geq$ 32 MiB and $\leq$ 4 GiB)
- $336 = 6 \times 8 \times 7$ symmetrically-shaped tensors ($\geq$ 32 MiB and $\leq$ 4 GiB)

GETV Performance Maps

GETV executed with GEMV using different matrix sizes:
LargeMatrix [LM] vs. SmallMatrix [SM]

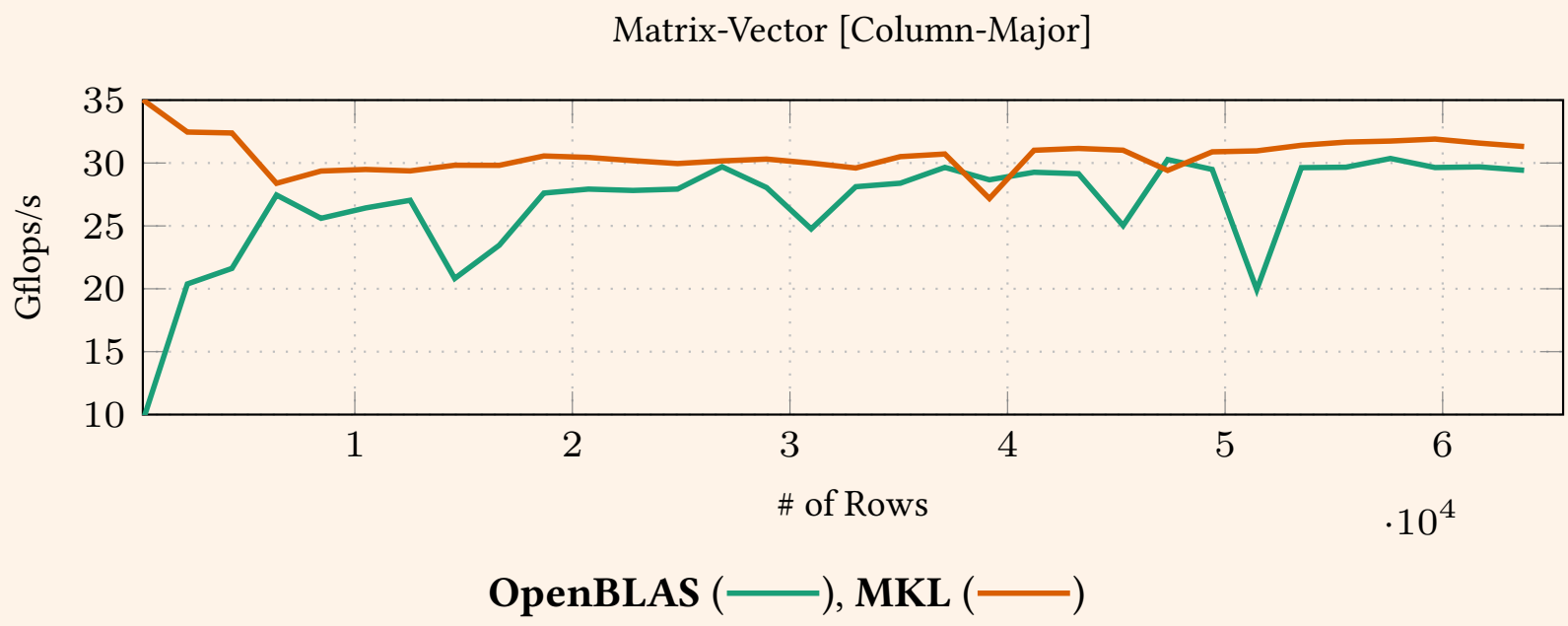


Using OpenBLAS with Setup (1)

`TTV<LM>`: Max. 32 Gflops/s with $p - q$ fused loops.

`TTV<SM>`: Max. 34 Gflops/s with $p - 2$ fused loops (matrix dimensions: (π_1, q))

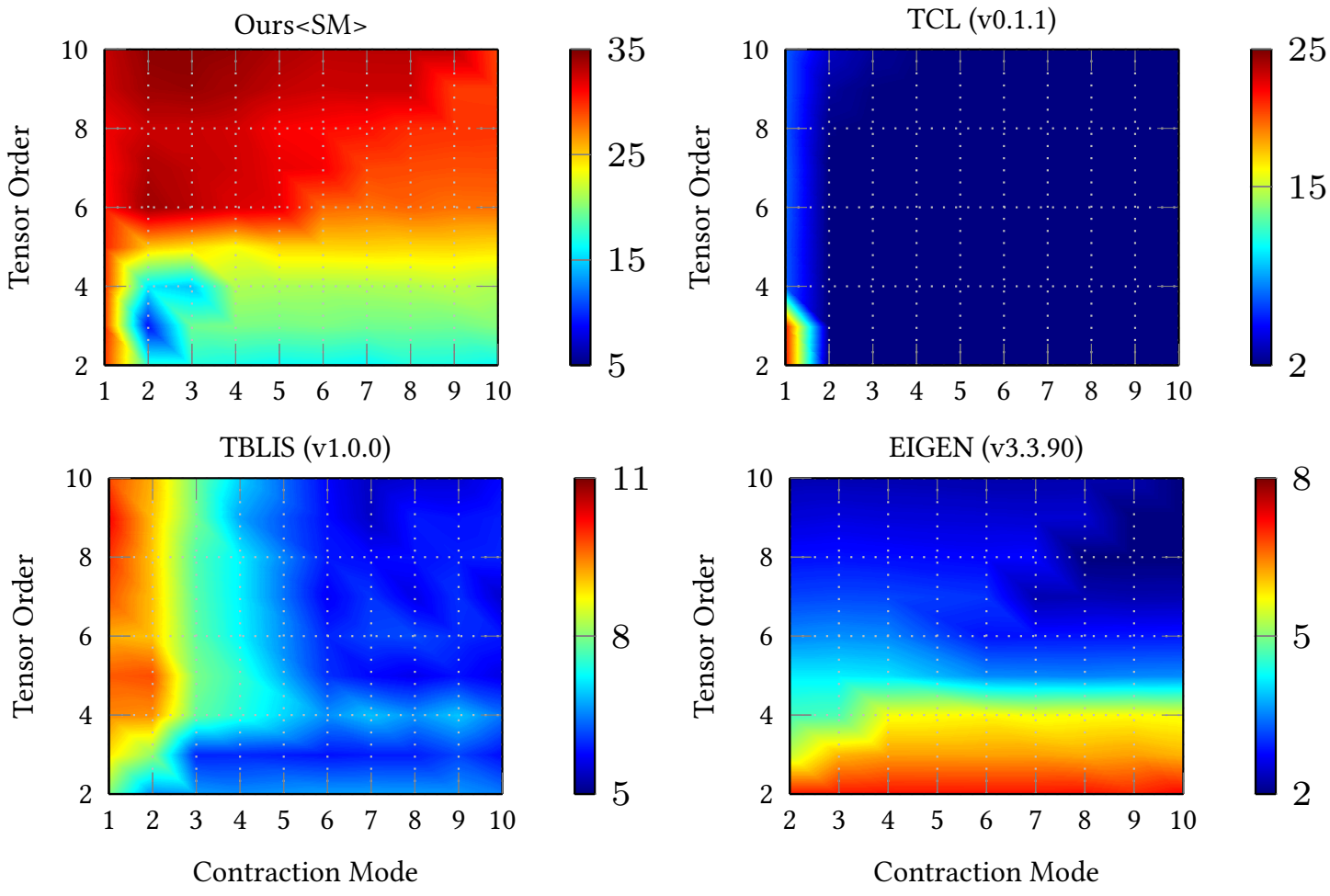
Runtime Measurments of OpenBLAS' `GEMV`



Mode-2 **GETV** with $p = 2$ and $\pi = (1, 2)$ is a **GEMV** in **column-major** and slows down with decreasing number of rows.

Cases 3, 7 and 8: performance of `GEMV` for the column-major format decreases with decreasing number of rows $\hat{n}_q$ with $\hat{n}_q = 1/n_p \prod_{r=1}^{p} n_r$.

Comparison with other approaches



Throughput is in Gflops/s single-precision. Tensors are asymmetrically-shaped. Tensor elements are stored according to the first-order storage format.

Summary

All implementations supporting all combinations of **contraction mode**, **tensor order**, **dimensions** and any **non-hierarchical storage format** which may be set at runtime.

LOG-based TTV and TTM implementations are **easy to implement** with a performance comparable to `TTBT`- and `TBLIS`-based implementations.

Design and **detailed analysis** see Cem Bassoy (2019). “Design of a High-Performance Tensor-Vector Multiplication with BLAS”. In: *International Conference on Computational Science*. Springer, pp. 32–45 with implementation corresponding `https://github.com/bassoy/ttv`.

Fin