Distributed Dense Tucker Decomposition and GPUs

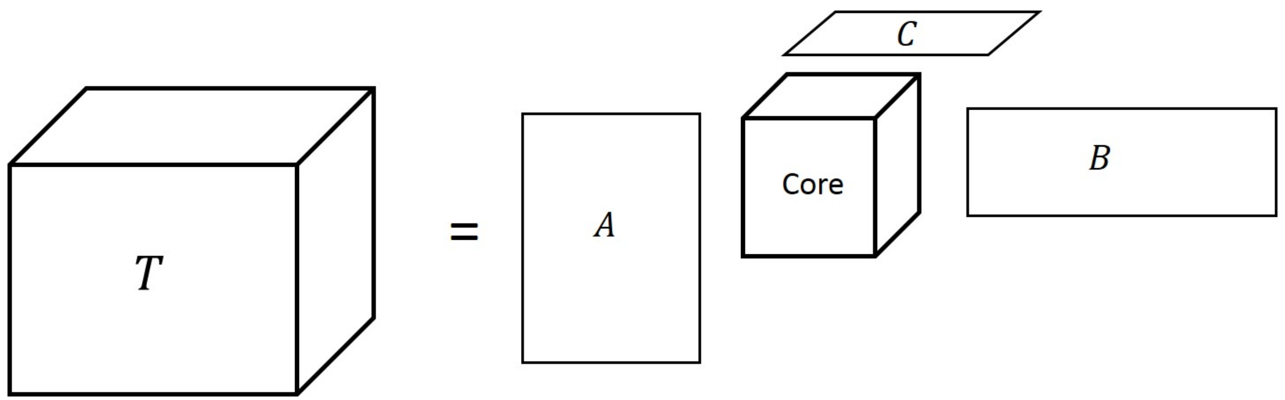
Jee Choi, Venkatesan Chakaravarthy, Prakash Murali, Xing Liu
IBM T. J. Watson Research

Presented at SIAM CSE
March 2nd, 2017.

Overview

- **Distributed Dense Tucker (HOOI) Framework**
  - Up to **7×** speedup over prior state-of-the-art
  - Optimal computational load and communication volume
    - Optimal balanced tree to minimize computation
    - Optimal static grid + dynamic gridding mechanism
- **Accelerating HOSVD via GPUs**
  - Up to **5.4×** speedup on 4 P100 GPUs vs. 2-socket, 20-core CPU system
  - Matricization re-use for (potentially) further **1.4×** speedup

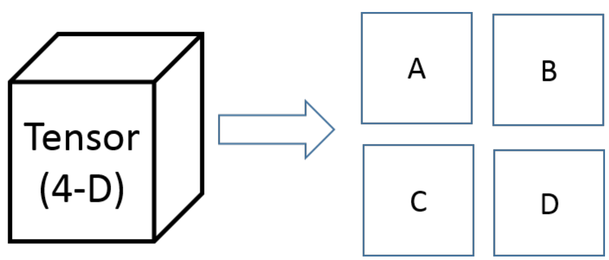
Tucker Decomposition



HOSVD-HOOI Algorithms

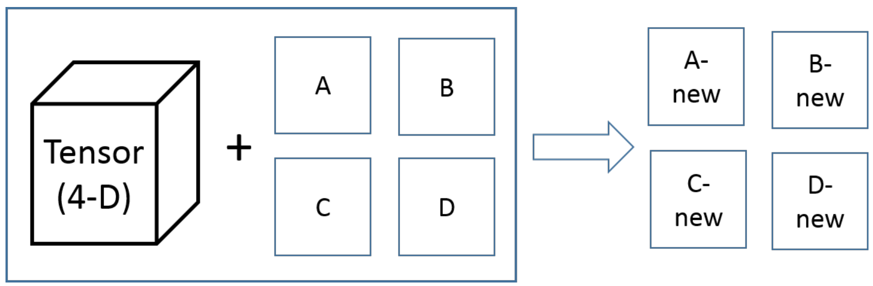
HOSVD

- Produces an initial solution

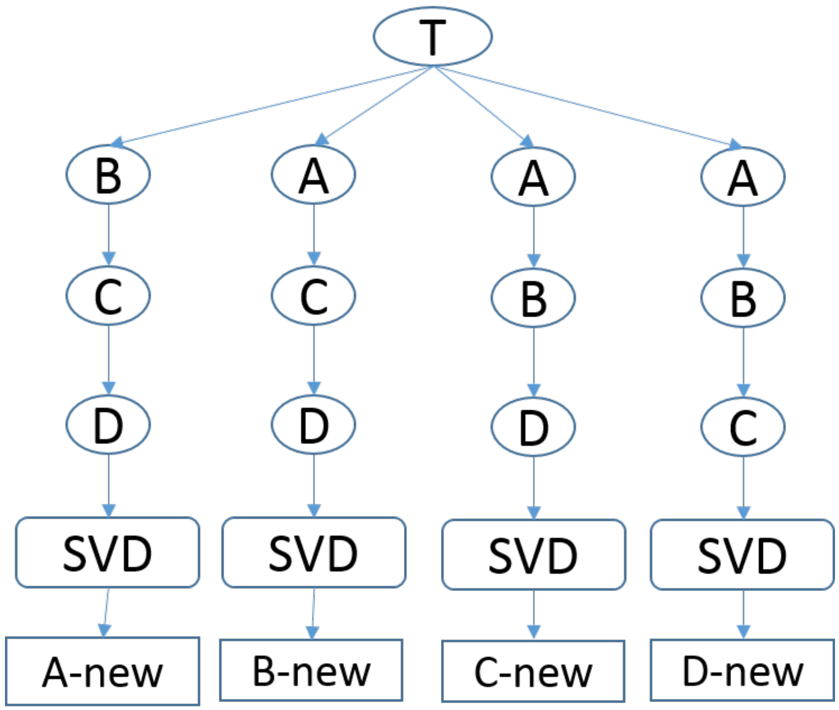


HOOI Iterator

- Refinement : improve accuracy
- Applied multiple times to get increasing accuracy



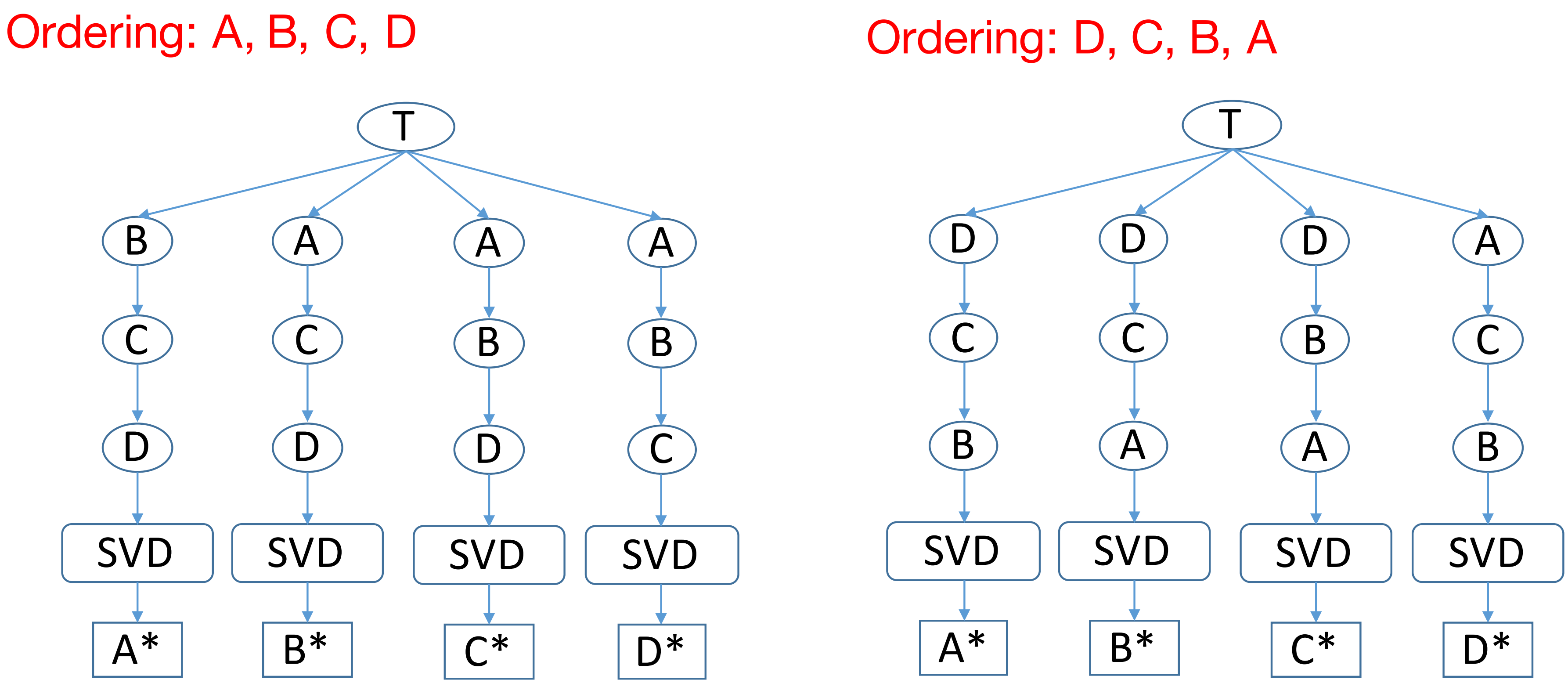
HOOI Algorithm



Goals

- Minimize computational load
- Minimize computational volume

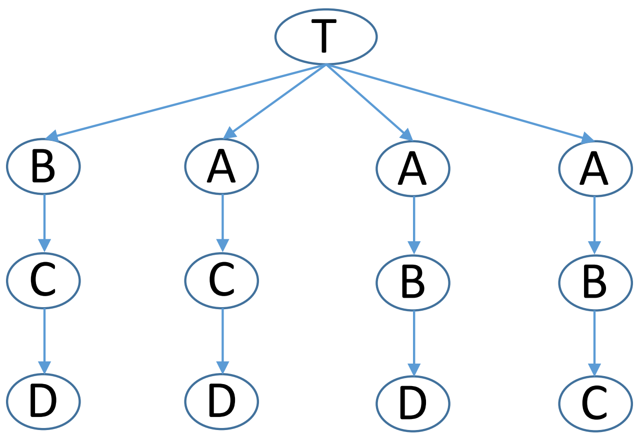
[ABK'16]: Importance of Mode Ordering



Performance is determined by the **mode ordering**

[Kaya-Ucar '16]: Balanced Trees

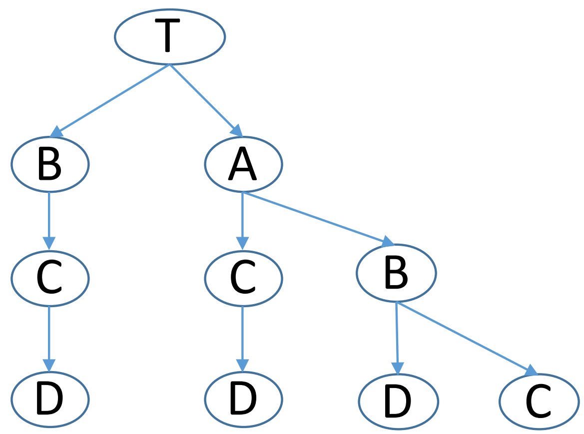
Chain trees



#TTM = 4 x 3 = 12

(N – 1) x N

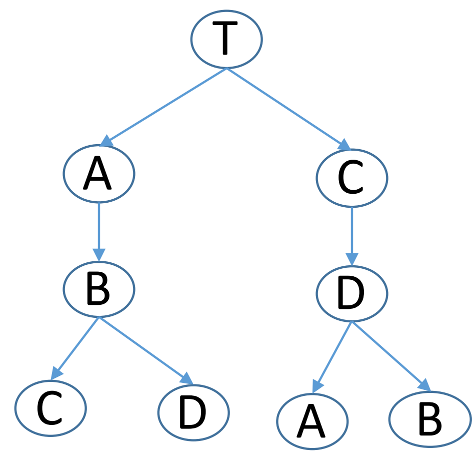
Chain reuse trees



#TTM = 9

(N-1) (1 + N/2)

Balanced Trees



#TTM = 8

N log N

Theorem:

Any tree must use at least **N log N** multiplications

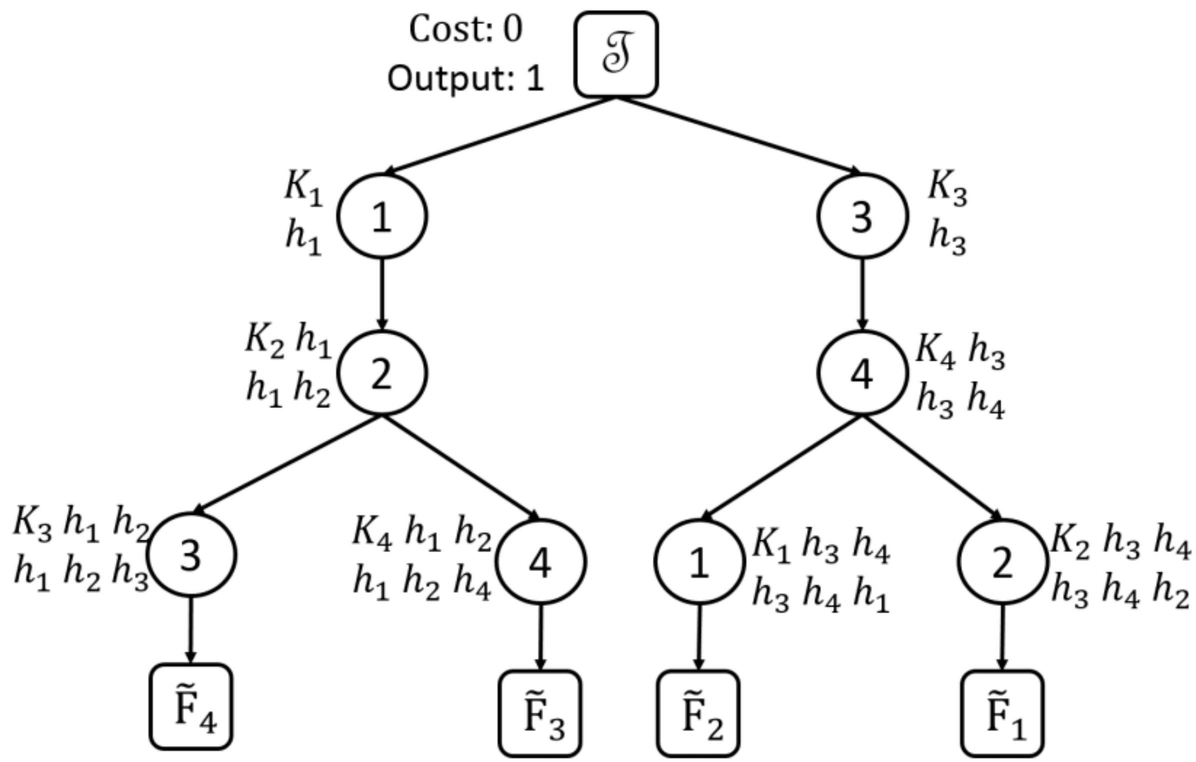
[ABK'16]: Heuristics for Mode Ordering

- Input tensor: **L1 × L2 × L3 × L4**
- Output core: **K1 × K2 × K3 × K4**

- Important parameters: K1, K2, K3, K4
- Compression: **h1** = K1/L1, **h2** = K2/L2, **h3** = K3/L3, **h4** = K4/L4

- Ordering:
  - K-ordering – order modes in **increasing K** value
  - h-ordering – order modes in **increasing h** value

- Matrix-multiplication cost (node 1) :
  A: K1 × L1
  T: L1 x L2 * L3 * L4
  K1 * L1 * L2 * L3 * L4 = **K1* |T|**
- Output dimension:
  K1 * L2 * L3 * L4 = **h1 * |T|**



Optimal Trees

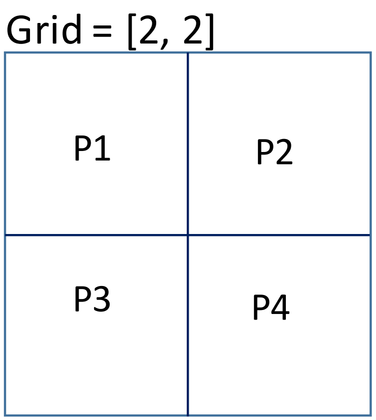
- Enumerate all possible trees and choose the best?

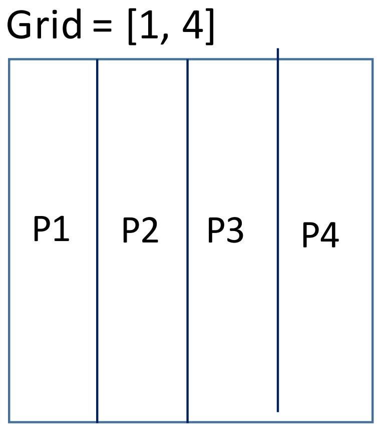
Number of "distinct" trees is at least [factorial(N-1)]^N

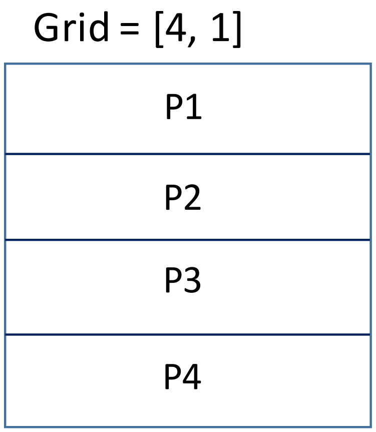
N	#trees	4^N
3	8	64
4	1296	256
5	8 Million	1024
6	3 Trillion	4096

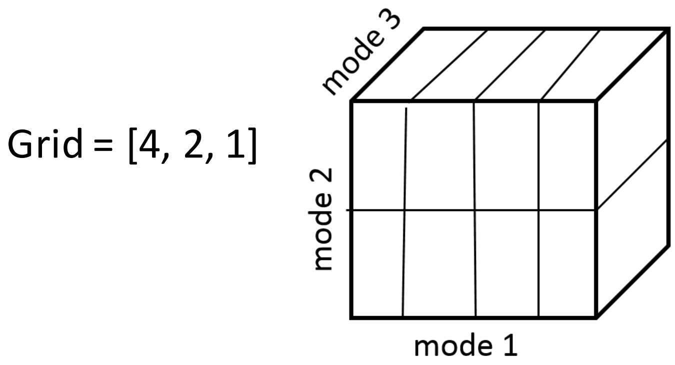
Theorem:
Optimal tree can be found (via dynamic programming) in time **O(4^N)**

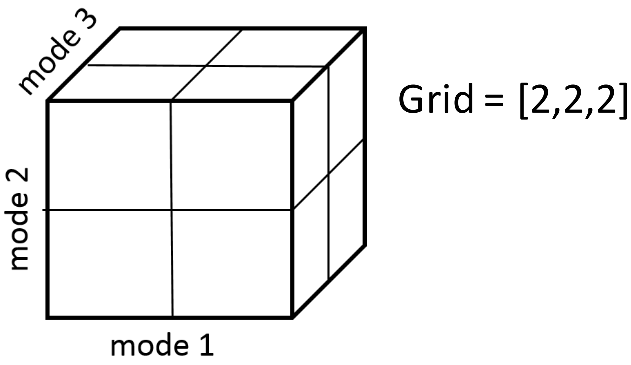
Communication Volume : Grid Selection







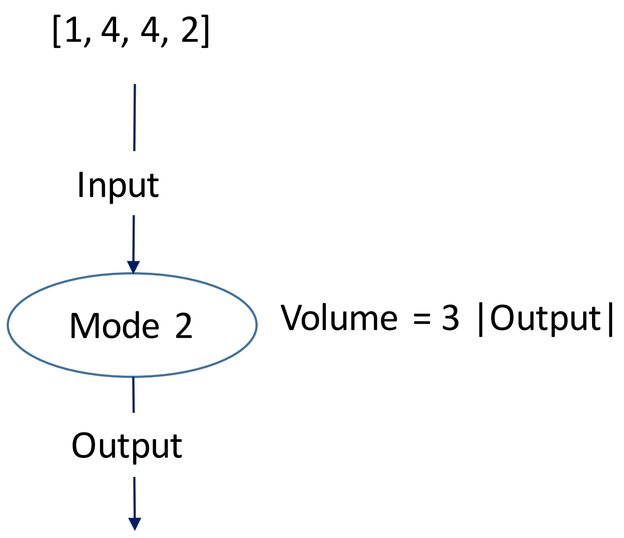


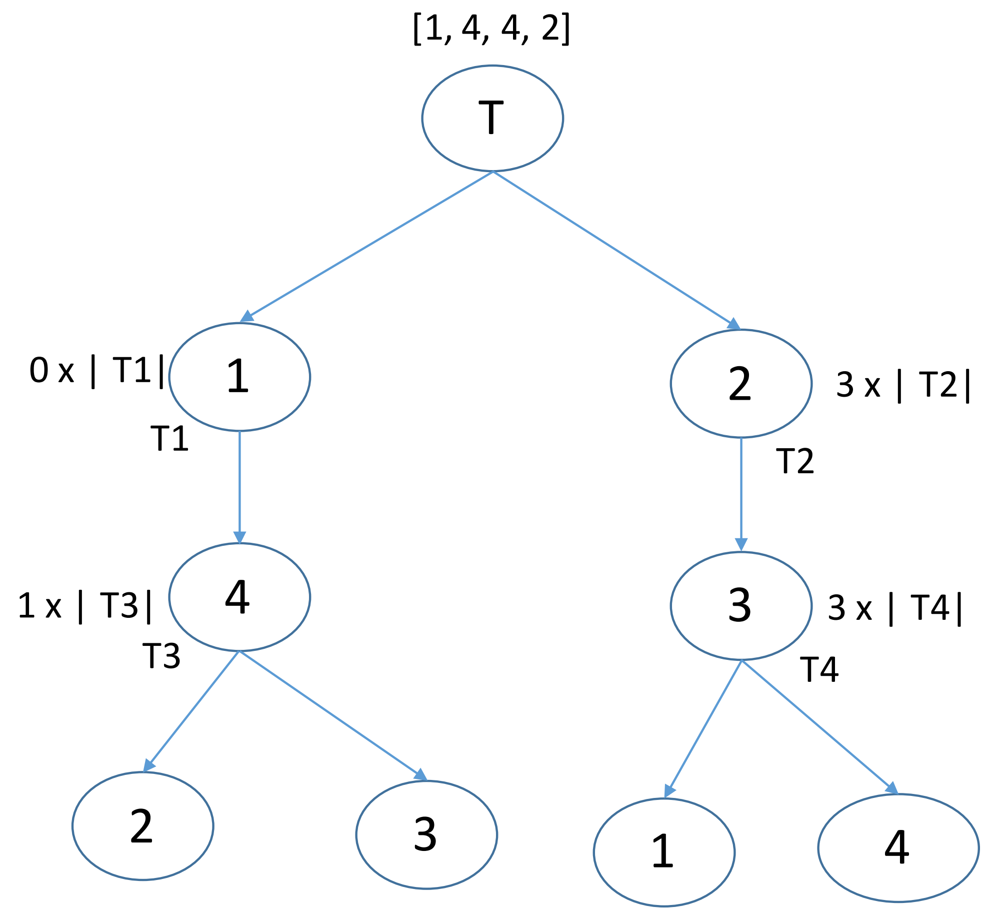


Communication Volume

[ABK '16]

- Volume depends on grid selection.
- For a TTM on mode s: volume = (p_s – 1) x |output|





Theorem:
An algorithm for finding optimal grid – the one with minimum communication volume

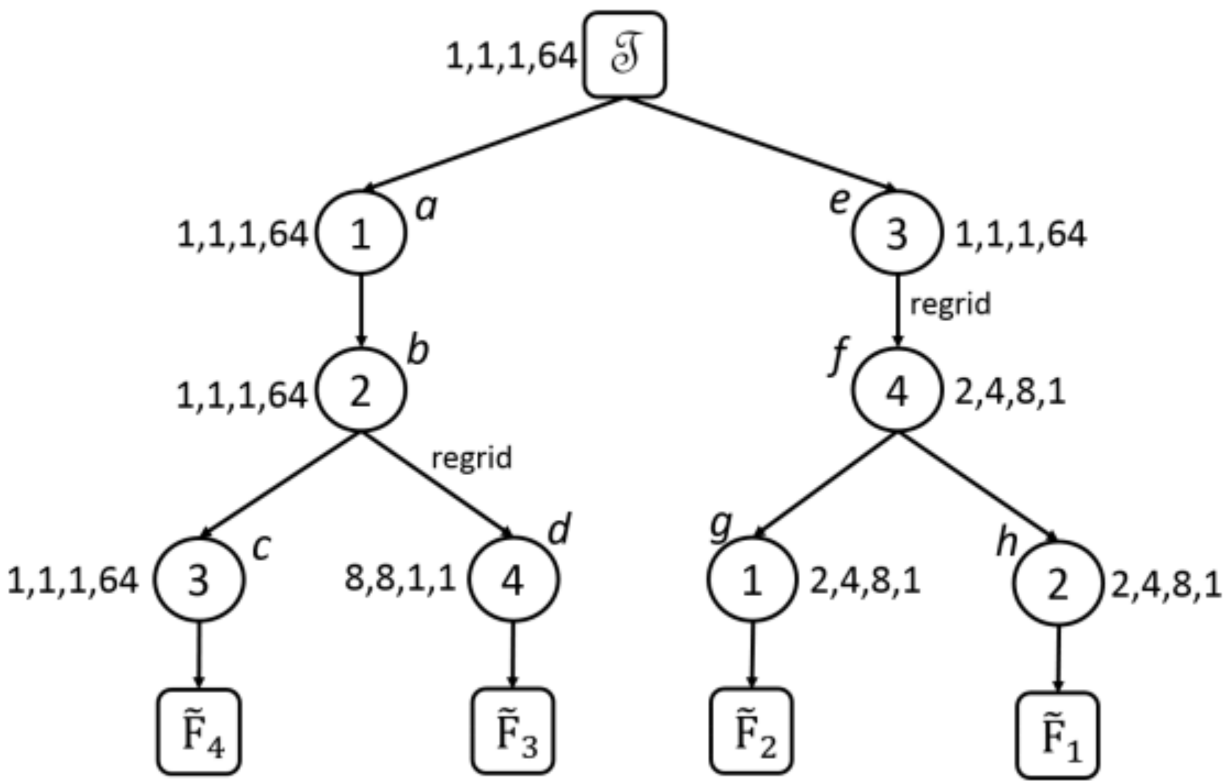
Finding the Optimal Static Grid

$$P = p_1^{e1} * p_2^{e2} \ldots p_s^{es}$$

$$\psi(P, N) = \prod_{i=1}^{s} \binom{e_i + N - 1}{N - 1}$$

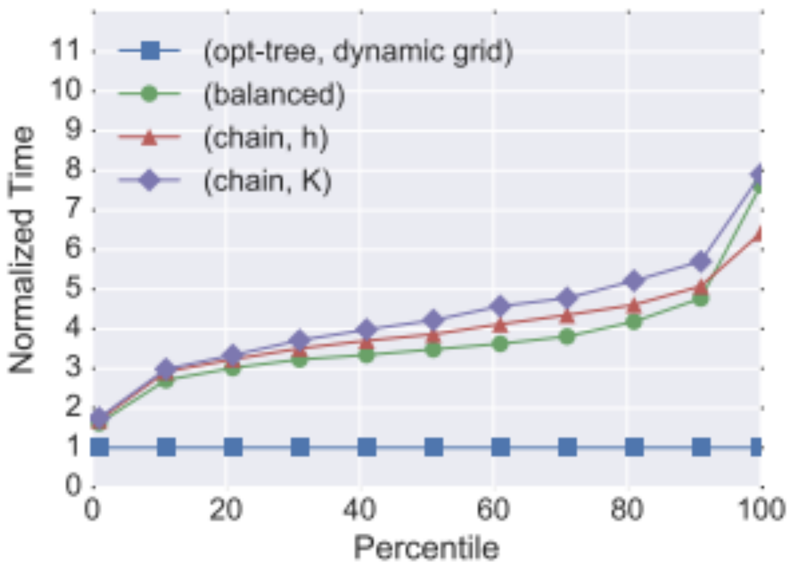
	$N = 5$	6	7	8	9	10
$P = 2^5$	126	252	562	792	1287	2002
2^{10}	1001	3003	8008	19448	43758	92378
2^{20}	10626	53130	230K	880K	3.1M	10M

Theorem
An algorithm for finding optimal dynamic grids

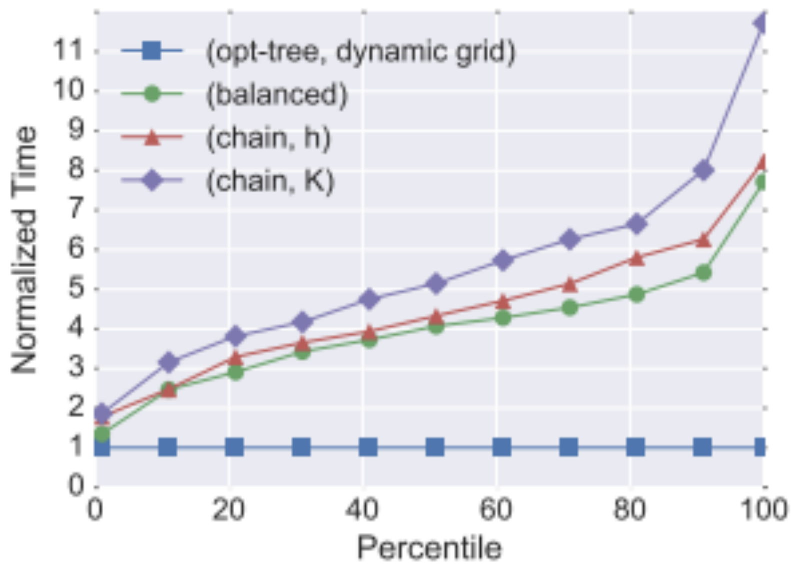


Experimental Results

Synthetic benchmark: About 1700 tensors of different dimension sizes



5D



6D

Prior Work

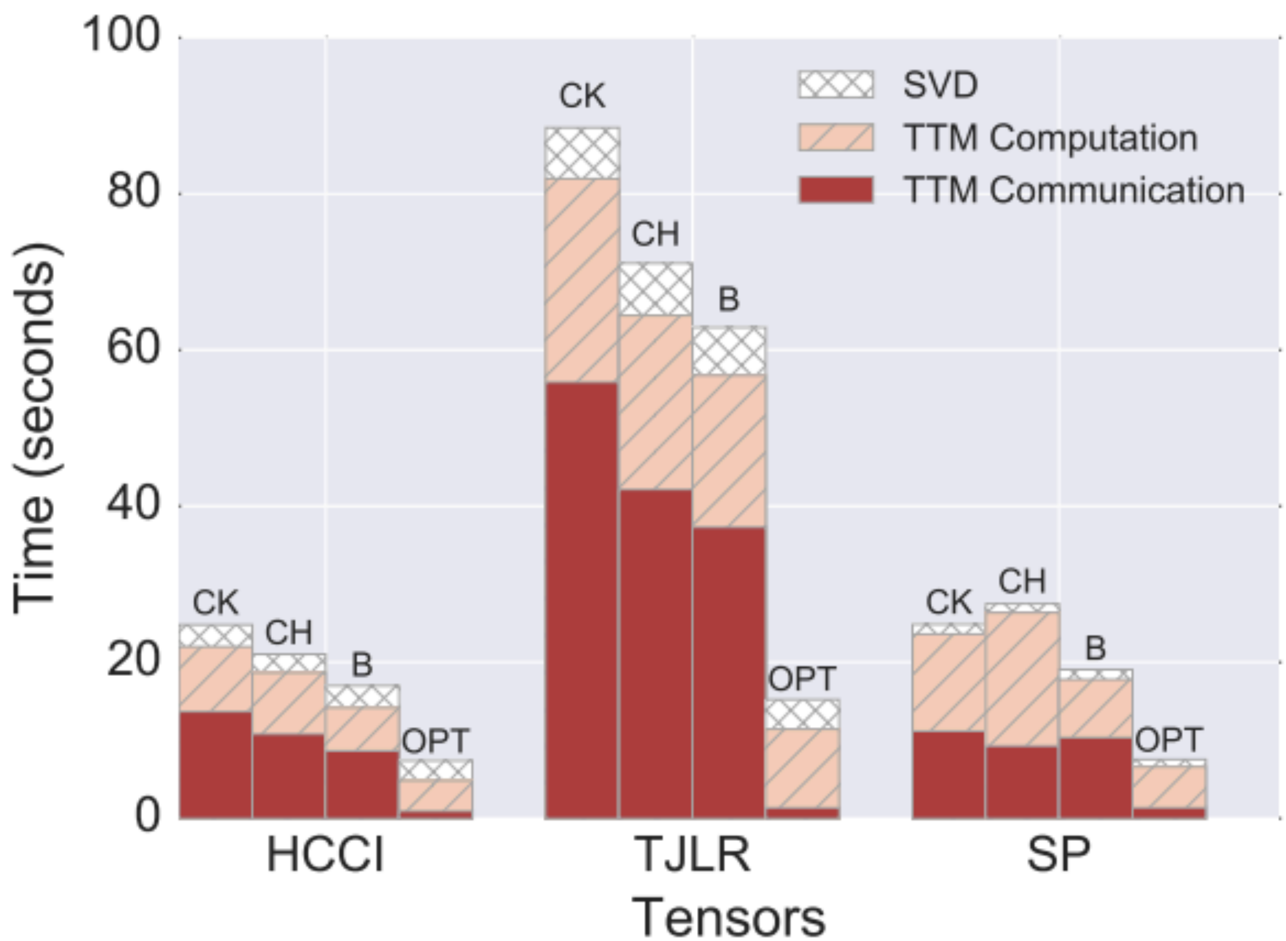
- CK – chain tree (K ordering) + static grid
- CH – Chain tree (H ordering) + static grid
- B – Balanced tree + static grid

Our work

- Opt – optimal tree + optimal dynamic grids

Experimental Results

Real tensors – combustion simulation



	Tensor Size Core Size
HCCI	672 x 672 x 627 x 16 279 x 279 x 153 x 14
TJLR	460 x 700 x 360 x 16 x 4 306 x 232 x 239 x 16 x 4
SP	500 x 500 x 500 x 11 x 10 81 x 129 x 127 x 7 x 6

GPU Acceleration

- **Accelerators**
  - High-bandwidth, high-flop devices
  - Requires regularized memory access and large amount of data parallelism
  - Is it suitable for tensor decomposition?
- **Case Study of Dense Tucker Decomposition**
  - If the right algorithm is used, it can yield good speedup for dense tensors
  - Likely to be true for sparse tensors

- **High-order singular value decomposition (HOSVD)**

procedure HOSVD($\mathcal{X}$, R_1, R_2, ..., R_N)
  for $n = 1, \ldots, N$ **do**
    $\mathbf{A}^{(n)} \leftarrow R_n$ leading left singular vectors of $\mathbf{X}_{(n)}$
  end for
  $\mathcal{G} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)\mathsf{T}} \times_2 \mathbf{A}^{(2)\mathsf{T}} \cdots \times_N \mathbf{A}^{(N)\mathsf{T}}$
  return $\mathcal{G}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$
end procedure

- **First instinct – use optimized library (e.g., cusolverDn)**
  - SVD requires that the entire matrix be on the GPU memory
  - Matricized tensor has one large dimension
  - SVD Performance is low (< 150 GFLOP/s)
  - Matricization on the GPU is non-trivial

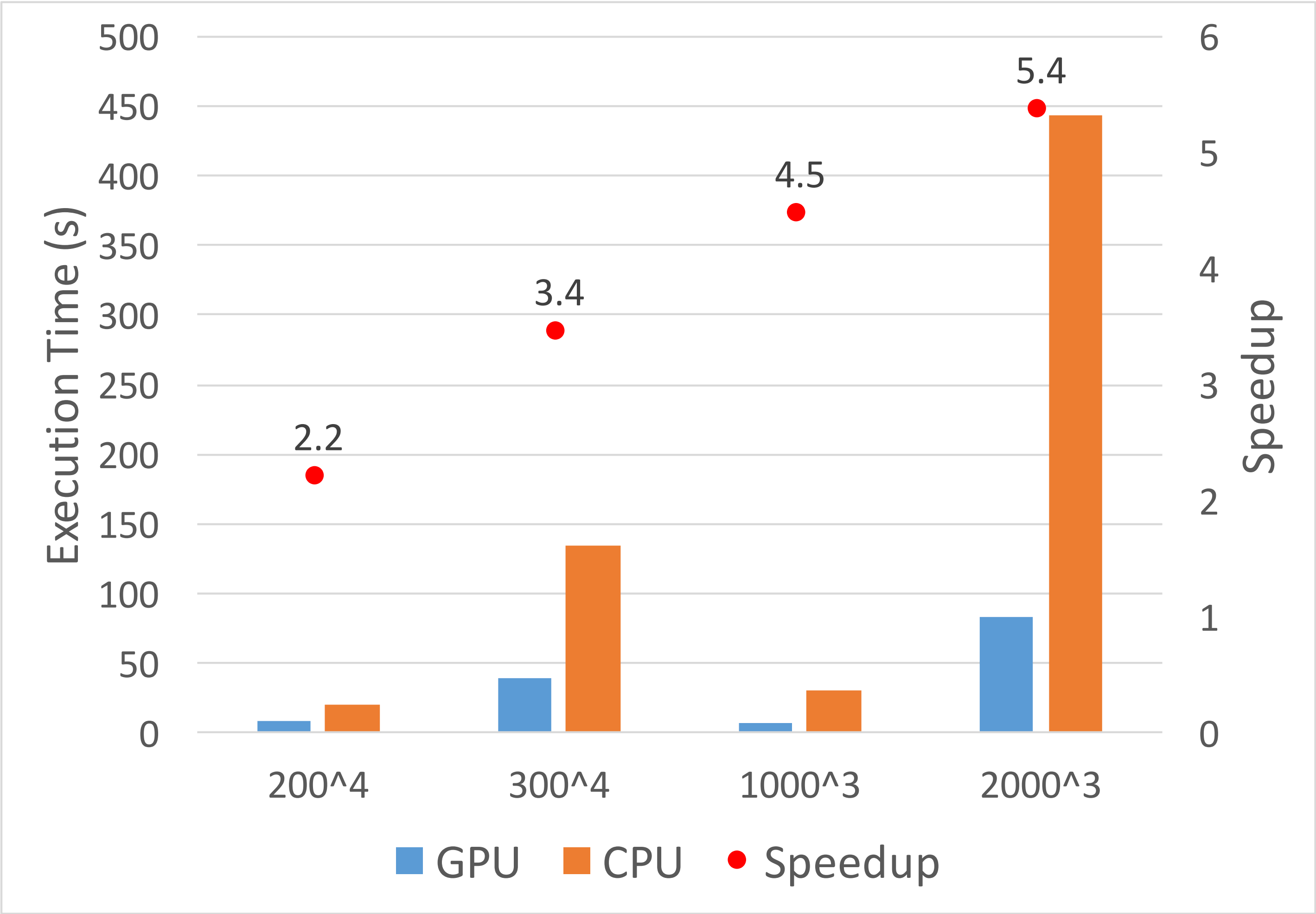
Tucker Decomposition

- **Simple solution**
  - Optimize the $X_{(n)}X_{(n)}^T$ (DGEMM)
  - Eigendecomposition
  - DGEMM dominates the execution time, as Eigendecomposition is done on a much smaller matrix

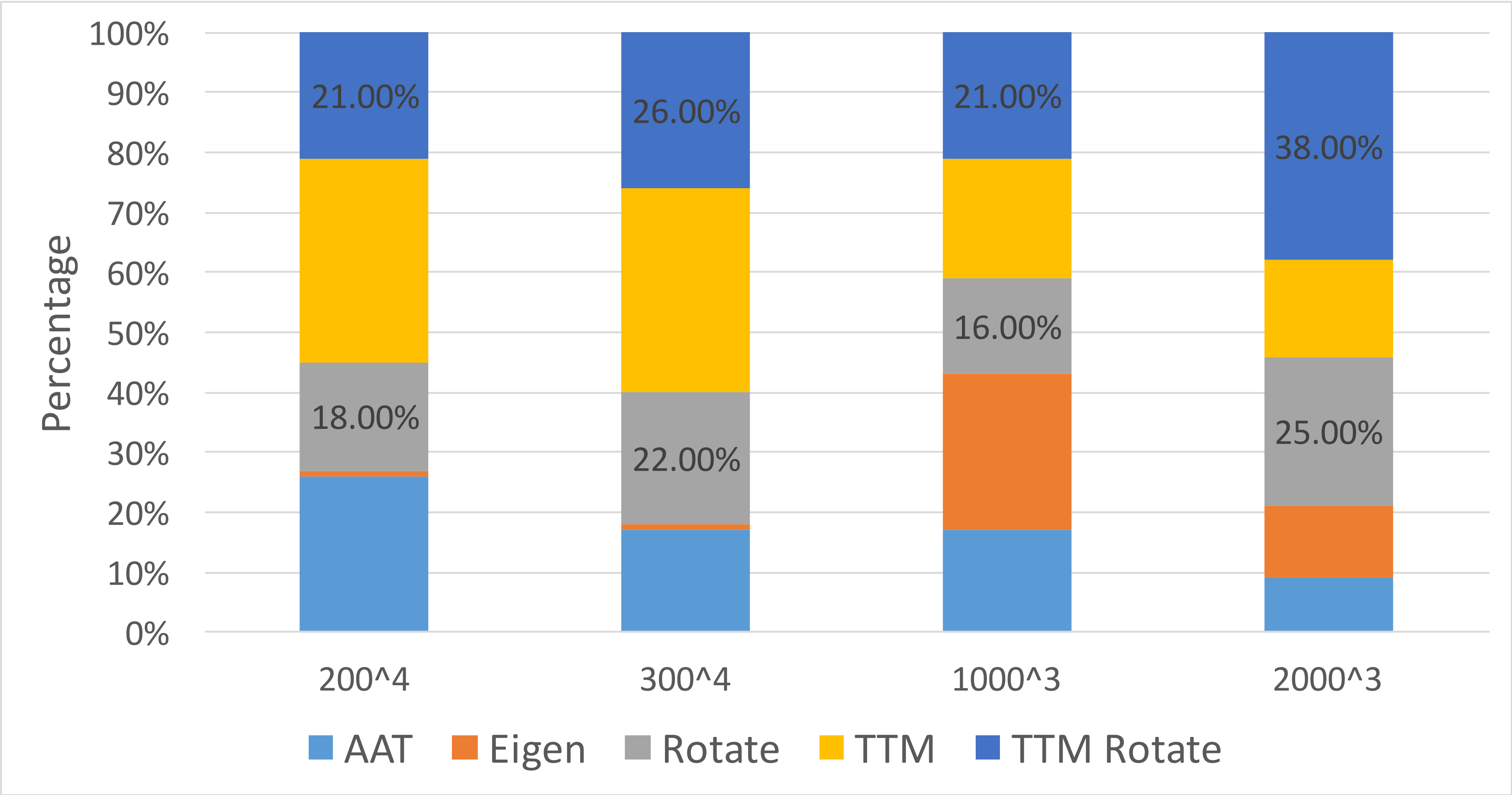
Algorithm 1 Sequentially-Truncated HOSVD (ST-HOSVD)

procedure ST-HOSVD($\mathcal{X}$, ϵ)
    $\mathcal{Y} \leftarrow \mathcal{X}$
    for $n = 1, \ldots, N$ **do**
        $\mathbf{S} \leftarrow \mathbf{Y}_{(n)} \mathbf{Y}_{(n)}^{\mathsf{T}}$
        $R_n \leftarrow \min R$ such that $\sum_{r>R} \lambda_r(\mathbf{S}) \le \epsilon^2 \|\mathcal{X}\|^2 / N$
        $\mathbf{U}^{(n)} \leftarrow$ leading R_n eigenvectors of $\mathbf{S}$
        $\mathcal{Y} \leftarrow \mathcal{Y} \times_n \mathbf{U}^{(n)\mathsf{T}}$
    end for
    $\mathcal{G} \leftarrow \mathcal{Y}$
    return $(\mathcal{G}, \{ \mathbf{U}^{(n)} \})$
end procedure

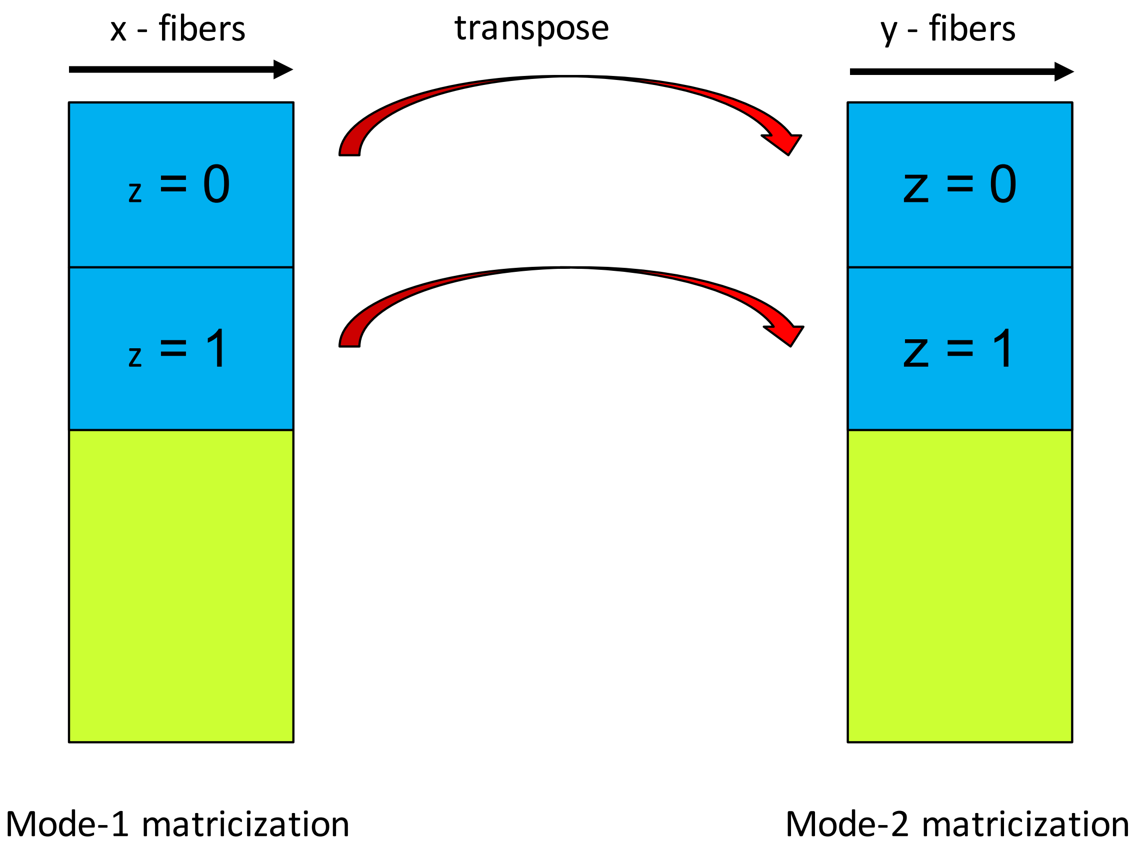
Performance Result



Time Breakdown



Matricization Re-use



Mode ordering

Mode 1 – A x BCD
Mode 2 – B x ACD
Mode 3 – C x ABD
Mode 4 – D x ABC

Mode rotation

Mode 1 – A x BCD
Mode 2 – B x CDA
Mode 3 – C x DAB
Mode 4 – D x ABC

Tucker Decomposition

- Original algorithm
  - Mode-1 matricization -> DGEMM -> Eigen -> Mode-2 matricization -> DGEMM -> Eigen -> …
- Unfold-reuse algorithm
  - Mode-1 matricization -> DGEMM -> Eigen -> In-GPU transpose -> DGEMM -> Eigen
  - Eliminate ½ matricization and transfer cost
- Expected performance improvement
  - ~1.2 - 1.4× additional speedup
  - Work in progress

Conclusion

- Distributed performance
  - 7× speedup over prior methods
  - Optimal computation and communication
- GPUs
  - 5.4× (4 GPUs vs. 20 CPU cores)
  - Potential for up to 1.4× (7.5× total) using unfold re-use.