FIST-HOSVD:
Fused In-place Sequentially Truncated Higher Order Singular Value Decomposition

SIAM PP
February 26, 2022

Benjamin Cobb*, Hemanth Kolla›, Eric Phipps›, Ümit V. Çatalyürek*

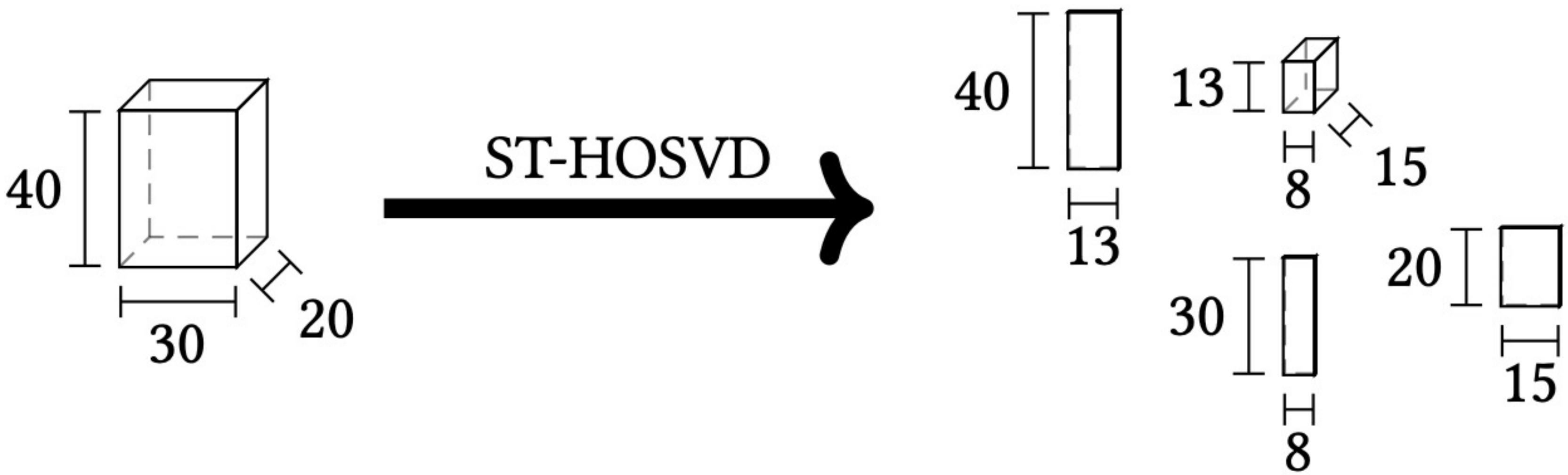
**Georgia Institute of Technology*
›*Sandia National Labs*

Sandia National Laboratories

EXASCALE COMPUTING PROJECT

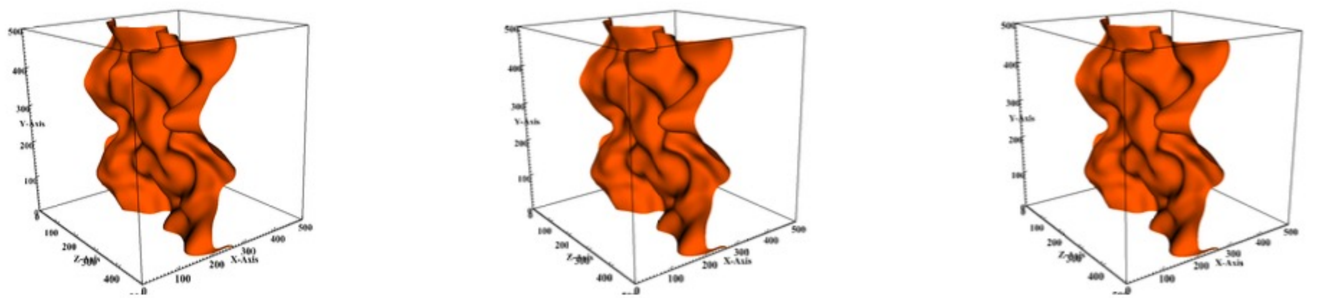
Georgia Tech

Tucker Decomposition



- Tensor is a multi-dimensional array
- Tucker decomposition compresses tensor
  - Smaller core tensor
  - Set of factor matrices corresponding to each dimension

Applications of Tucker Decomposition



(a) Original (b) Low compression (ϵ=1e-4) (c) High compression (ϵ=1e-2)

Figure courtesy of:

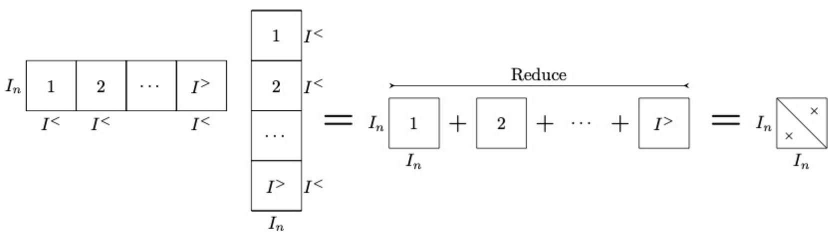
Grey Ballard, Alicia Klinvex, and Tamara G. Kolda. 2020. TuckerMPI: A Parallel C++/MPI Software Package for Large-Scale Data Compression via the Tucker Tensor Decomposition. *ACM Trans. Math. Softw.* 46, 2, Article 13 (June 2020), 31 pages.

- Data Compression
  - TuckerMPI compressed over 6TB of simulation data into 167MB with 1e-2 relative error
  - Can reconstruct entire or part of dataset
  - Error tolerance related to truncation
- Computer Vision
  - TensorFaces, Vasilescu et al.
- Signal Porcessing
  - Lathauwer et al, Muti et al, and more
- And many more…

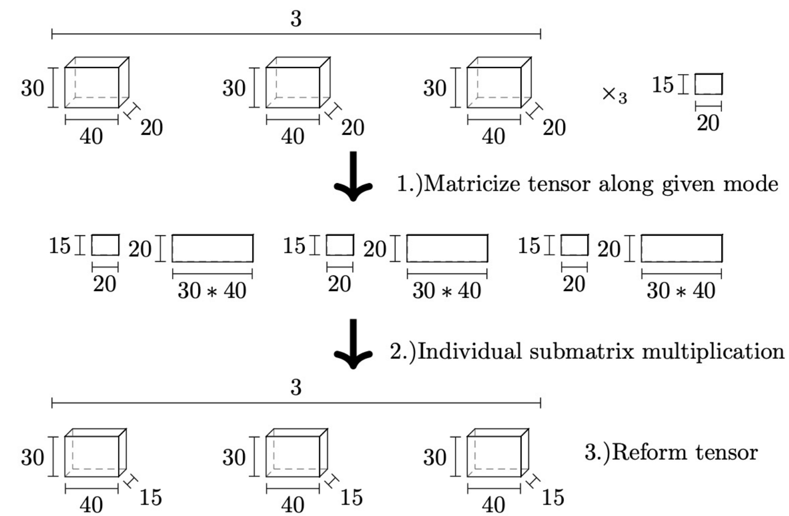
- Good at capturing latent multi-way relationships between variables
  - Lose information by just viewing as matrix

Computing Tucker via ST-HOSVD

Gram



TTM



- Two computational bottlenecks
  - Tensor Times Matrix (TTM)
  - Gram
- For tensor X with dimensions $I_1 \times \cdots \times I_N$, let:
  - $I_n^* = \prod_{r=1}^{N} I_r$
  - $I_n^> = \prod_{r=n+1}^{N} I_r$
  - $I_n^< = \prod_{r=1}^{n-1} I_r$
- mode-n TTM with $J \times I_n$ matrix: $O(J I_n^*)$
- Gram along mode-n: $O(I_n I_n^*)$
- Focusing on dense, single-node case

Algorithm 1: ST-HOSVD

Data: Tensor $\mathcal{X}$, accuracy bound ϵ

Result: Tensor core $\mathcal{G}$, factor matrices $\mathcal{F}$

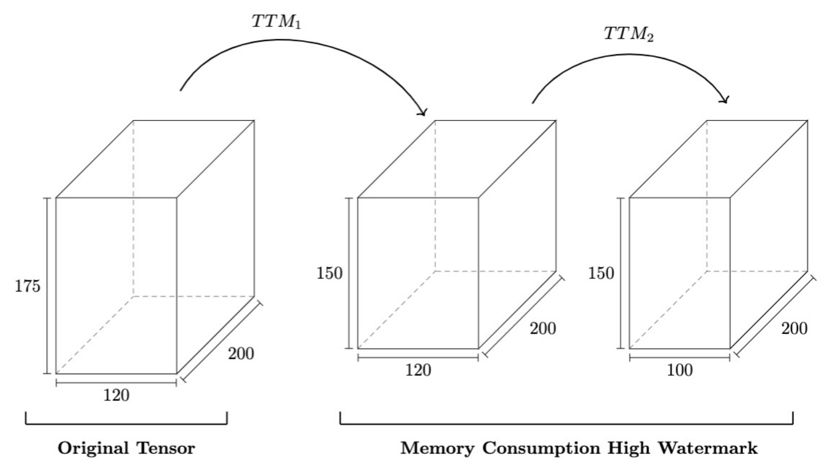
```
G ← X; /* Initialize G */
for n = 1 : N do
    S ← G_n G_n^T; /* Gram matrix */
    [λ, V] ← eig(S)
    /* R_n is smallest value that satisfies ε */
    U_n ← V(:, 1 : R_n)
    G ← G ×_n U_n^T; /* TTM */
F ← U_1 ... U_N
return G, F
```

ST-HOSVD Limitations

Algorithm 1: ST-HOSVD

Data: Tensor $\mathcal{X}$, accuracy bound ϵ
Result: Tensor core $\mathcal{G}$, factor matrices $\mathcal{F}$

$\mathcal{G} \leftarrow \mathcal{X}$; /* Initialize $\mathcal{G}$ */
for $n = 1 : N$ **do**
    $S \leftarrow \mathcal{G}_n \mathcal{G}_n^T$; /* Gram matrix */
    $[\lambda, V] \leftarrow eig(S)$
    /* R_n is smallest value that satisfies ϵ */
    $U_n \leftarrow V(:, 1 : R_n)$
    $\mathcal{G} \leftarrow \mathcal{G} \times_n U_n^T$; /* TTM */
$\mathcal{F} \leftarrow U_1 \ldots U_N$
return $\mathcal{G}, \mathcal{F}$



- In addition to original tensor, must allocate memory to store intermediate TTM results
  - In the worst case when there is little to no truncation, consumes 2x tensor size in memory
- Memory is a limited resource
  - Often bottlenecked by tensor exceeding available memory
- When tensor is larger than $\frac{1}{3}$ main memory (RAM), ST-HOSVD either:
  - Is unable to complete
  - Goes out-of-core → ST-HOSVD thrashes between RAM and Disk, potentially leading to catastrophic performance degradation
- We aim to alleviate this by computing the Tucker decomposition in-place
  - Overwrite the original tensor with core of Tucker decomposition
  - Memory Consumption: $O(\,I_{max}^2 + \prod_{r=1}^{N} I_r) \rightarrow O(\,I_{max}^2)$

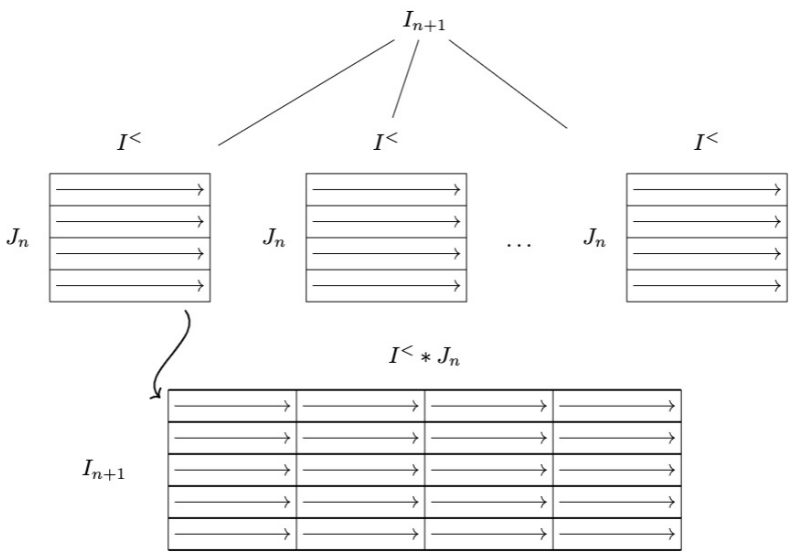
Overview

- Memory is a valuable, limited resource
- Significantly decreased memory consumption of computing dense Tucker
  - $O(I_{max}^2 + \prod_{r=1}^{N} I_r) \rightarrow O(I_{max}^2)$
  - If the tensor can be held in memory then we can most likely compute tucker
  - Maintain comparable or decrease runtime

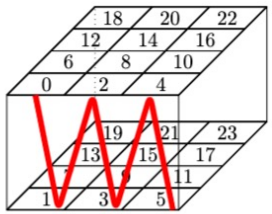
Optimizations

- Develop 3 novel optimizations to efficiently compute Tucker Decomposition in-place:
  - Kernel Fusion
    - Fuse TTM and Gram kernels together to improve memory locality
  - Tensor Tiling
    - Extend matrix tiling and cache blocking to fused kernel operation
  - In-place Transpose
    - Develop blocked in-place transpose algorithm based on cycle-following to prepare cache blocks in-place

Kernel Fusion



$$I_n^* = \prod_{r=1}^{N} I_r$$
$$I_n^> = \prod_{r=n+1}^{N} I_r$$
$$I_n^< = \prod_{r=1}^{n-1} I_r$$



TTM_2

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 6 & 7 \\ 8 & 9 \\ 10 & 11 \end{bmatrix} \begin{bmatrix} 12 & 13 \\ 14 & 15 \\ 16 & 17 \end{bmatrix} \begin{bmatrix} 18 & 19 \\ 20 & 21 \\ 22 & 23 \end{bmatrix}$$

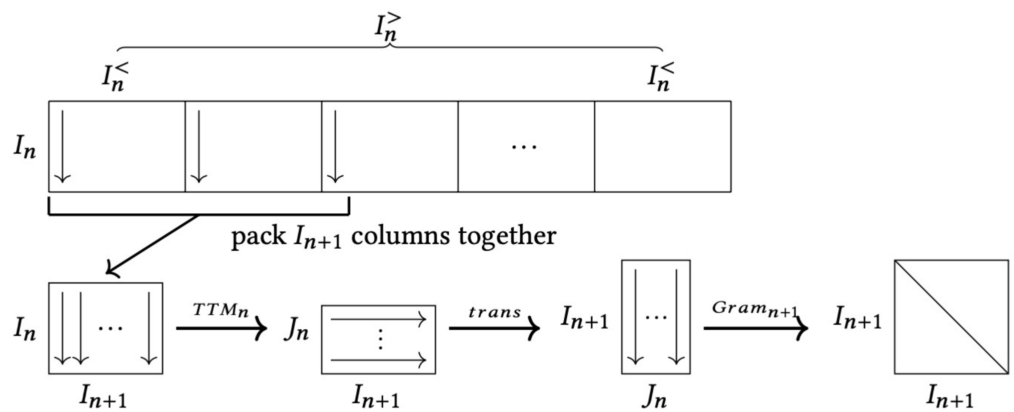
$$= \begin{bmatrix} 6 & 9 \\ 12 & 18 \end{bmatrix} \begin{bmatrix} 24 & 27 \\ 48 & 54 \end{bmatrix} \begin{bmatrix} 42 & 45 \\ 84 & 90 \end{bmatrix} \begin{bmatrix} 60 & 63 \\ 120 & 126 \end{bmatrix}$$

$Gram_3$

$$\begin{bmatrix} 6 & 9 & 12 & 18 \\ 24 & 27 & 48 & 54 \\ 42 & 45 & 84 & 90 \\ 60 & 63 & 120 & 126 \end{bmatrix} \begin{bmatrix} 6 & 24 & 42 & 60 \\ 9 & 27 & 45 & 63 \\ 12 & 48 & 84 & 120 \\ 18 & 54 & 90 & 126 \end{bmatrix} = \begin{bmatrix} 585 & 1935 & 3285 & 4635 \\ 1935 & 6525 & 11115 & 15705 \\ 3285 & 11115 & 18945 & 26775 \\ 4635 & 15705 & 26775 & 37845 \end{bmatrix}$$

- Compute mode-$(n+1)$ Gram whilst computing mode-n TTM
  - Fuse TTM and Gram kernels together
- Aim to keep everything in cache
- Fusing kernels together is known to improve memory locality
  - Especially effective in GPU case → future work

Tensor Tiling



$I_n^* = \prod_{r=1}^{N} I_r$

$I_n^> = \prod_{r=n+1}^{N} I_r$

$I_n^< = \prod_{r=1}^{n-1} I_r$

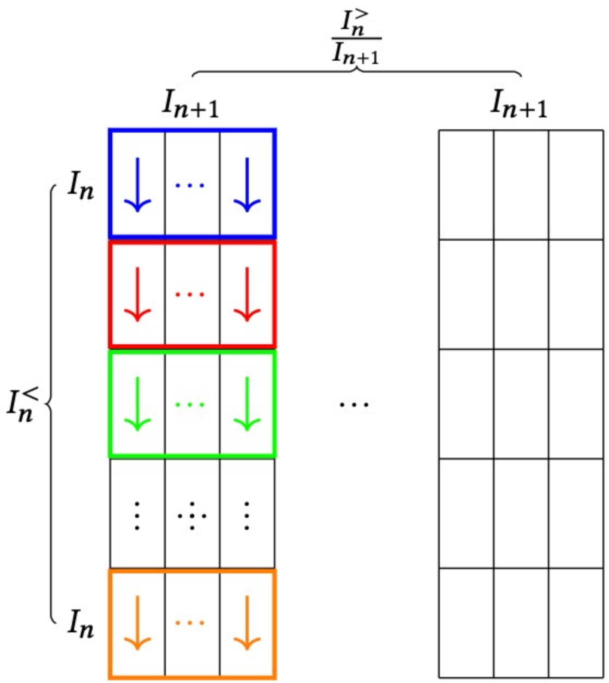
Algorithm 3: FaST-HOSVD

Data: Tensor $\mathcal{X}$, accuracy bound ϵ
Result: Tensor core $\mathcal{G}$, factor matrices $\mathcal{F}$

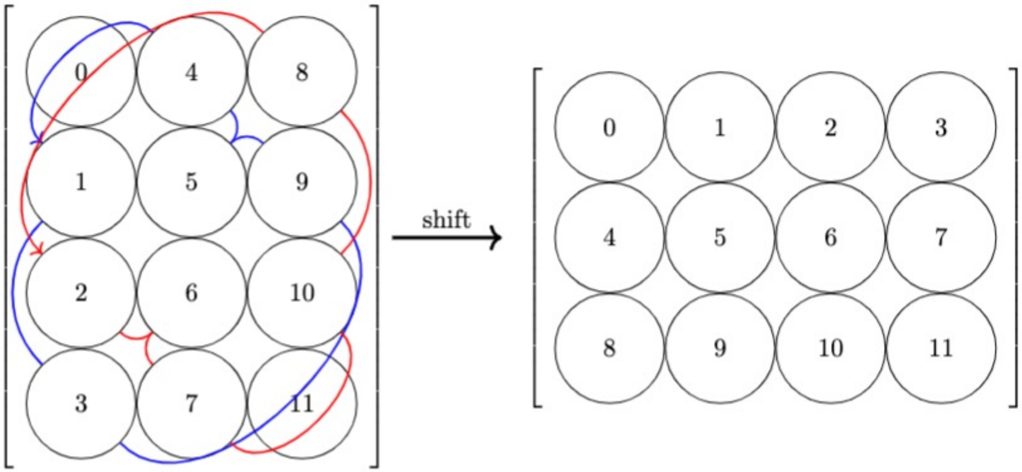
```
/* Initialize G                                              */
G ← X
/* First Gram matrix                                         */
S_1 ← G_1 G_1^T
[λ, V] ← eig(S_1)
U_1 ← V(:, 1 : R_1)
/* R_1 is smallest value that satisfies ε                    */
for n = 1 : N − 1 do
    [G_{n+1}, S_{n+1}] ← Fused_Packed_Kernel(G_n, U_n, n)
    [λ, V] ← eig(S_{n+1})
    U_{n+1} ← V(:, 1 : R_{n+1})
/* Last TTM                                                  */
G ← G ×_n U_N^T
F ← U_1 . . . U_N
return G, F
```

- Pack columns into $I_n \times I_{n+1}$ cache blocks
  - Write $J \times I_{n+1}$ TTM submatrix results in row-major order
  - Then logically transpose to $I_{n+1} \times J$ column-major submatrices
    - No data movement required for logical transpose
- Requires I_{n+1} discontiguous memory accesses on I_n contiguous entries per block
- Tensor layout evolves in memory over course of computation
  - Prepares tensor for subsequent iterations
  - Next dimension contiguous in memory

In-place Transpose



$A \in \mathbb{R}^{m \times n}$
$i \to$ linear index
$q = m * n - 1$
next cycle element linear index $= (i * n)\%q$



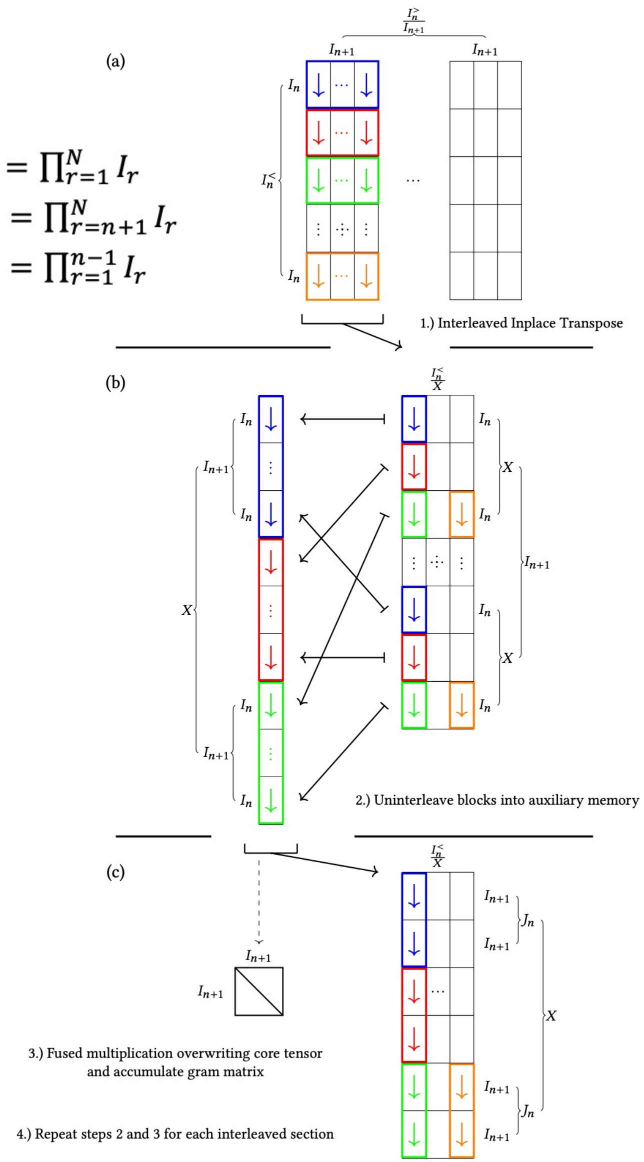
$I_n^* = \prod_{r=1}^{N} I_r$
$I_n^> = \prod_{r=n+1}^{N} I_r$
$I_n^< = \prod_{r=1}^{n-1} I_r$

- Traditional Cycle-Following based In-place Transpose algorithm
  - Requires less element access than other in-place transpose algorithms
  - In practice suffers from poor memory locality due to almost pseudo-random element access
- Developed blocked variant that improves memory locality, referred to as Interleaved In-Place Transpose (IIPT)
- Plan to compare performance against existing in-place transpose algorithms in future work

FIST-HOSVD

- Interleaved In-place Transpose to prepare cache blocks
- Copy cache blocks into auxiliary memory allocation
  - deinterleave cache blocks during copy
- Perform fused multiplication on each cache block
  - Result overwrites corresponding section of tensor
- Avoids allocating memory to hold intermediate TTM results
- If the tensor can be held in memory with at least $O(I_{max}^2)$ additional elements worth of memory, then we can compute Tucker

$$I_n^* = \prod_{r=1}^{N} I_r$$
$$I_n^> = \prod_{r=n+1}^{N} I_r$$
$$I_n^< = \prod_{r=1}^{n-1} I_r$$



Algorithm 6: FIST-HOSVD

Data: Tensor $\mathcal{X}$, auxiliary memory limit in β

Result: $\mathcal{X}$ overwritten with core tensor, Factor matrices $\mathcal{F}$

$S_0 \leftarrow \mathcal{X}_1 \mathcal{X}_1^T$; /* First Gram matrix */

for $n = 1 : N-1$ **do**

  $[\lambda, V] \leftarrow eig(S_n)$

  $U_n \leftarrow V(:, 1 : R_n)$; /* R_n, smallest value that satisfies ϵ */

  $S_{n+1} \leftarrow$ Fused_Inplace_kernel($\mathcal{G}$, U_n, β)

$[\lambda, V] \leftarrow eig(S_N)$

$U_N \leftarrow V(:, 1 : R_N)$

$\mathcal{X} \leftarrow \mathcal{X} \times_N U_N^T$; /* Last Inplace TTM */

$\mathcal{F} \leftarrow U_1 \ldots U_N$

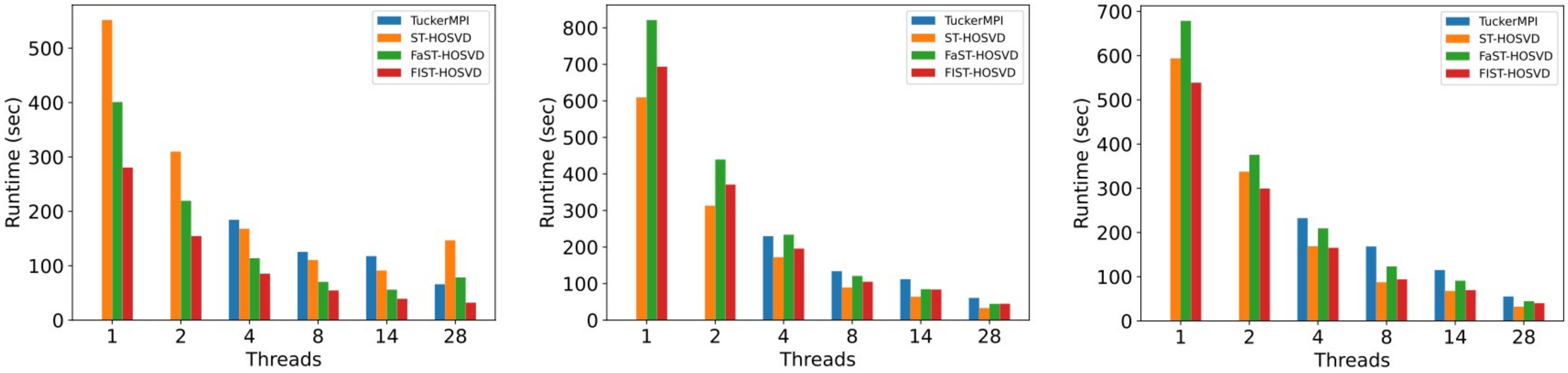
return $\mathcal{F}$

Experimental set-up

- Each node has 28 cores and 256GB memory
  - Allocating more than this allocation either causes the program to terminate or the node to crash

- Three different datasets:
  - Randomly generated
    - Used to represent high-rank tensor
    - Each timeslice is: $64 \times 64 \times 64 \times 64 \times 64$
  - Homogeneous Charge Compression Ignition (HCCI)
    - 4-th order tensor from a simulation of turbulent autoignition over a 2D spatial domain
    - Each timeslice is: $672 \times 672 \times 33$
  - Statistically Planar (SP)
    - 5-th order tensor from a simulation over a 3D spatial domain, 4-th mode is 11 solution variables at each grid point
    - Each timeslice is: $500 \times 500 \times 500 \times 11$

Runtime Results



(a) Random: $64 \times 64 \times 64 \times 64 \times 64 \times 4$

(b) HCCI: $672 \times 672 \times 33 \times 326$

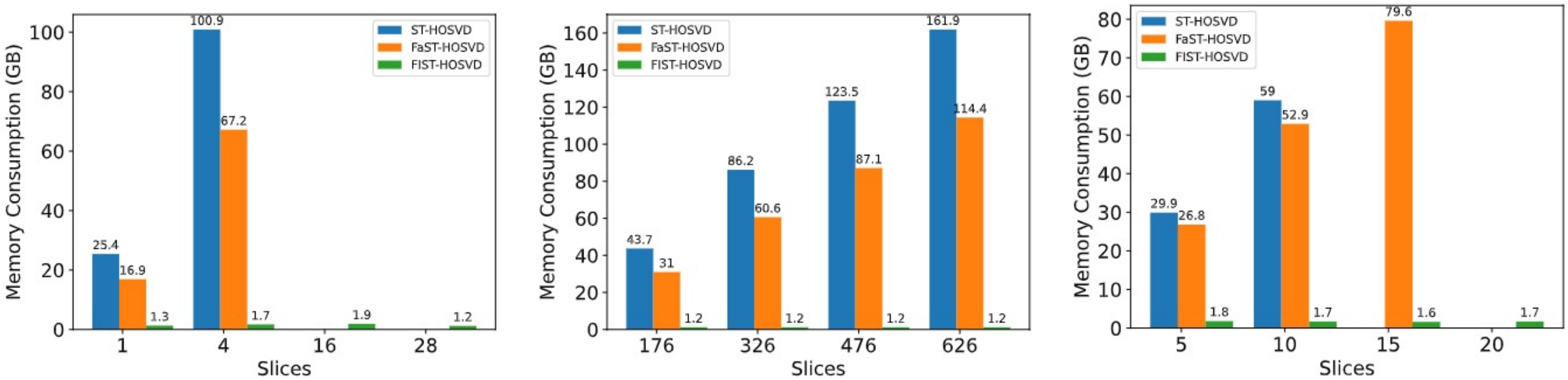
(c) SP: $500 \times 500 \times 500 \times 11 \times 10$

Sample bar charts of runs with an error-tolerance (ϵ) of 1e-07.

Bars not shown did not complete due to running out of memory.

- Fused implementations performs better along later dimensions due to cache blocking
- Cache blocking incurs data movement overhead
- Plan to add support for processing dimensions out of order in future work
- Maintain comparable runtime

Memory Consumption



(a) Random: 1 slice is $64 \times 64 \times 64 \times 64 \times 64$ **(b) HCCI: 1 slice is** $672 \times 672 \times 33$ **(c) SP: 1 slice is** $500 \times 500 \times 500 \times 11$

Memory consumption over different timeslice counts for an error-tolerance (ϵ) of 1e-07.
Bars not shown did not complete due to running out of memory.

- FIST-HOSVD consumes significantly less memory than the other algorithms
  - $O(I_{max}^2 + \prod_{r=1}^{N} I_r) \rightarrow O(I_{max}^2)$
- Compute memory consumption as: $memory\ highwater\ mark\ of\ program - size\ of\ orignal\ tensor$
- Allocated 1GB of auxiliary memory for FIST-HOSVD
  - Additional memory usage comes from Gram reduction

Summary

- Recap:
  - Memory Consumption: $O(\, I_{max}^2 + \prod_{r=1}^{N} I_r) \;\rightarrow O(\, I_{max}^2)$
    - Significantly decreased memory consumption of ST-HOSVD for dense Tucker
    - If tensor fits in memory, then FIST-HOSVD can most likely compute Tucker
    - Maintained comparable or decreased runtime

- Future Work:
  - Add in support for processing dimensions in any order
    - Kernel fusion provides biggest performance improvements along later dimensions
  - Compare IIPT algorithm to other in-place transpose algorithms
  - Complete GPU port
    - Everything implemented in Kokkos (portable framework)
    - Kernel fusion originally intended for GPU case
    - Device memory even more limited than host memory

Thanks!

Questions?

Backup Slides

Runtime Tables

Table 1: Random tensor runtime (in seconds).
1 slice: $64 \times 64 \times 64 \times 64 \times 64$

ϵ	Slices:	1	4	16	28
1e-09	TuckerMPI	13.4	66.5	—	—
	ST-HOSVD	4.4	142.1	—	—
	FaST-HOSVD	**4.4**	70.0	—	—
	FIST-HOSVD	5.5	**32.3**	**107.6**	**219.3**
1e-05	TuckerMPI	13.4	66.4	—	—
	ST-HOSVD	4.4	143.8	—	—
	FaST-HOSVD	**4.3**	74.8	—	—
	FIST-HOSVD	5.5	**32.1**	**107.3**	**219.8**
1e-03	TuckerMPI	13.5	65.6	—	—
	ST-HOSVD	4.4	143.3	—	—
	FaST-HOSVD	**4.4**	70.8	—	—
	FIST-HOSVD	5.6	**32.7**	**107.4**	**219.6**

Table 2: HCCI tensor runtime (in seconds).
1 slice: $672 \times 672 \times 33$

ϵ	Slices:	176	326	476	626
1e-09	TuckerMPI	35.4	71.4	104.9	—
	ST-HOSVD	18.0	**37.4**	**58.6**	—
	FaST-HOSVD	**23.9**	47.8	76.0	125.2
	FIST-HOSVD	25.0	51.2	77.6	**105.7**
1e-05	TuckerMPI	20.0	43.7	63.8	84.2
	ST-HOSVD	**9.6**	**23.3**	**35.6**	**47.7**
	FaST-HOSVD	12.0	31.3	44.8	66.7
	FIST-HOSVD	13.5	31.5	45.7	60.2
1e-03	TuckerMPI	11.4	27.1	38.3	49.1
	ST-HOSVD	**5.7**	**12.6**	**19.5**	**25.6**
	FaST-HOSVD	6.9	16.0	24.3	33.5
	FIST-HOSVD	7.0	16.5	24.7	31.6

Table 3: SP tensor runtime (in seconds).
1 slice: $500 \times 500 \times 500 \times 11$

ϵ	Slices:	5	10	15	20
1e-09	TuckerMPI	34.0	—	—	—
	ST-HOSVD	**24.9**	38.6	—	—
	FaST-HOSVD	35.1	54.2	—	—
	FIST-HOSVD	25.6	**49.3**	**72.7**	**92.6**
1e-05	TuckerMPI	12.9	25.4	38.1	—
	ST-HOSVD	**10.1**	**19.2**	**28.9**	—
	FaST-HOSVD	12.2	22.8	35.5	—
	FIST-HOSVD	12.5	24.3	36.4	**48.2**
1e-03	TuckerMPI	8.4	16.6	24.8	33.3
	ST-HOSVD	**7.0**	**13.9**	**21.36**	**27.6**
	FaST-HOSVD	9.5	18.9	29.0	37.9
	FIST-HOSVD	9.9	19.4	28.9	38.4

ϵ	Dataset	Slices	Resulting Core
1e-09	Random	4	$64 \times 64 \times 64 \times 64 \times 64 \times 4$
	HCCI	326	$631 \times 610 \times 31 \times 326$
	SP *	20	$187 \times 288 \times 278 \times 9 \times 20$
1e-05	Random	4	$64 \times 64 \times 64 \times 64 \times 64 \times 4$
	HCCI	326	$433 \times 410 \times 33 \times 234$
	SP *	20	$79 \times 116 \times 117 \times 7 \times 5$
1e-03	Random	4	$64 \times 64 \times 64 \times 64 \times 64 \times 4$
	HCCI	326	$232 \times 217 \times 29 \times 81$
	SP	20	$27 \times 48 \times 48 \times 2 \times 3$

Memory Consumption Tables

Table 5: Random tensor memory consumption (in GB). 1 slice is ~ 8 GB

ϵ	Slices:	1	4	16	28
1e-09	ST-HOSVD	24.2	96.2	—	—
	FaST-HOSVD	16.2	64.1	—	—
	FIST-HOSVD	**1.2**	**1.6**	**1.9**	**1.2**
1e-05	ST-HOSVD	24.2	96.2	—	—
	FaST-HOSVD	16.2	64.1	—	—
	FIST-HOSVD	**1.2**	**1.6**	**1.9**	**1.2**
1e-03	ST-HOSVD	24.2	96.2	—	—
	FaST-HOSVD	16.2	64.1	—	—
	FIST-HOSVD	**1.2**	**1.6**	**1.9**	**1.2**

Table 6: HCCI tensor memory consumption (in GB). 1 slice is ~ 0.12 GB

ϵ	Slices:	176	326	476	626
1e-09	ST-HOSVD	49.0	94.0	135.6	—
	FaST-HOSVD	34.2	65.2	94.1	123.0
	FIST-HOSVD	**1.1**	**1.1**	**1.1**	**1.1**
1e-05	ST-HOSVD	18.4	47.3	65.8	82.5
	FaST-HOSVD	15.2	37.9	54.1	70.1
	FIST-HOSVD	**1.1**	**1.2**	**1.1**	**1.1**
1e-03	ST-HOSVD	6.7	17.7	24.1	30.6
	FaST-HOSVD	6.8	17.0	23.4	29.8
	FIST-HOSVD	**1.1**	**1.2**	**1.3**	**1.3**

Table 7: SP tensor memory consumption (in GB). 1 slice is ~ 11 GB

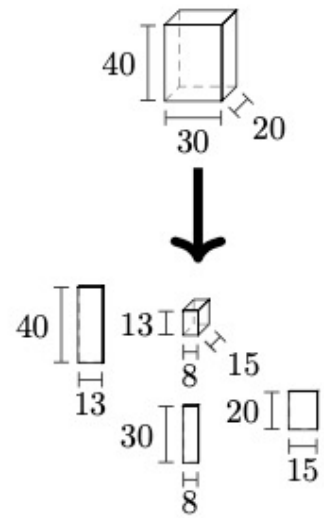
ϵ	Slices:	5	10	15	20
1e-09	ST-HOSVD	35.5	70.3	—	—
	FaST-HOSVD	30.8	60.2	—	—
	FIST-HOSVD	**1.5**	**1.8**	**1.5**	**1.7**
1e-05	ST-HOSVD	10.4	20.4	30.4	—
	FaST-HOSVD	10.3	20.2	30.2	—
	FIST-HOSVD	**1.2**	**1.2**	**1.4**	**1.3**
1e-03	ST-HOSVD	3.2	6.3	9.3	12.3
	FaST-HOSVD	3.3	6.3	9.4	12.4
	FIST-HOSVD	**1.1**	**1.1**	**1.1**	**1.1**

ϵ	Dataset	Slices	Resulting Core
1e-09	Random	4	$64 \times 64 \times 64 \times 64 \times 64 \times 4$
	HCCI	326	$631 \times 610 \times 31 \times 326$
	SP *	20	$187 \times 288 \times 278 \times 9 \times 20$
1e-05	Random	4	$64 \times 64 \times 64 \times 64 \times 64 \times 4$
	HCCI	326	$433 \times 410 \times 33 \times 234$
	SP *	20	$79 \times 116 \times 117 \times 7 \times 5$
1e-03	Random	4	$64 \times 64 \times 64 \times 64 \times 64 \times 4$
	HCCI	326	$232 \times 217 \times 29 \times 81$
	SP	20	$27 \times 48 \times 48 \times 2 \times 3$

How to compute Tucker

Algorithm 2 ST-HOSVD

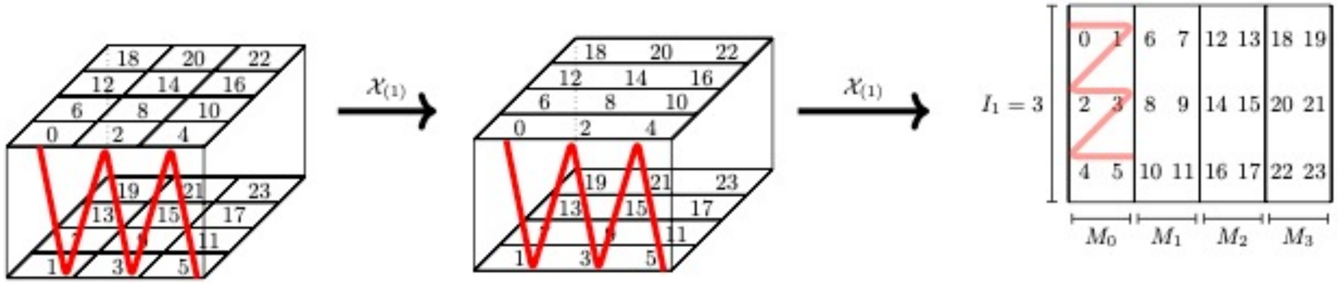
function ST-HOSVD(Tensor $\mathcal{X}$, accuracy bound ϵ)
  $\mathcal{G} \leftarrow \mathcal{X}$ $\quad \triangleright$ Initialize $\mathcal{G}$
  for $n = 0, 1 \ldots N-1$ **do**
    $S \leftarrow \mathcal{G}_n \mathcal{G}_n^T$ $\quad \triangleright$ Gram matrix
    $[\lambda, V] \leftarrow eig(S)$
    $U_n \leftarrow V(:, 1 : R_n)$ $\quad \triangleright$ R_n is smallest value that satisfies ϵ
    $\mathcal{G} \leftarrow \mathcal{G} \times_n U_n^T$ $\quad \triangleright$ TTM
  end for
  $\mathcal{F} \leftarrow U_0 \ldots U_{N-1}$
  return $\mathcal{G}, \mathcal{F}$
end function



- Several algorithms
  - HOOI, HOSVD, T-HOSVD, ST-HOSVD etc
- Sequentially Truncated Higher Order Singular Value Decomposition
  - ST-HOSVD
  - Truncates tensor at each iteration to save on FLOPs
  - Arguably fastest and most common method to compute Tucker

Helpful Notation

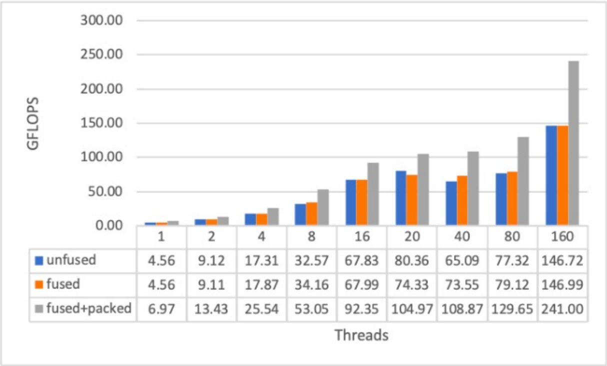
- $X\ is\ a\ tensor\ of\ order\ N\ with\ dimension\ sizes: I_1 \times\ \ldots\ I_N$
  - Assume starts stored in column major order
- $Mode - n\ fibers: set\ of\ vectors\ resulting\ from\ holding\ the\ n^{th}\ mode\ constant\ and\ iterating\ over\ all\ other\ dimensions$
- $Mode - n\ matricization: matrix\ whose\ columns\ are\ the\ mode - n\ tensor\ fibers\ of\ X, denoted\ X_{(n)}$
- $Useful\ values: I_n^{>} = \prod_{r=n+1}^{N} I_r\ ,\quad I_n^{<} = \prod_{r=1}^{n-1} I_r$



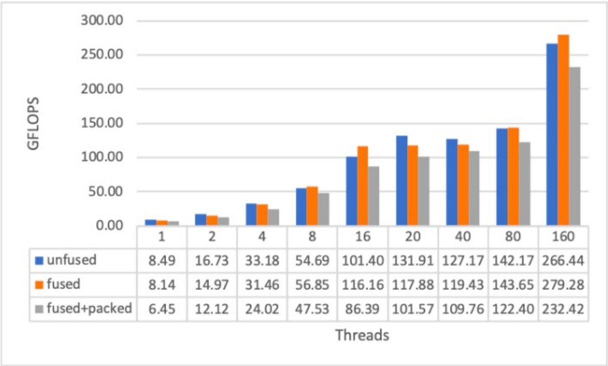
ST-HOSVD Bottlenecks

- Two kernels: TTM and Gram
- Tensor Times Matrix (TTM)
  - Can be viewed as batched matrix multiplication
  - Multiplies tensor along n^{th} dimension by $R_n \times I_n$ matrix
  - Input tensor dimensions are: $I_1 \times \ldots I_n \ldots \times I_N$
  - Output tensor dimensions are: $I_1 \times \ldots R_n \ldots \times I_N$
- Gram
  - Matricized tensor multiplied by its transpose
  - $I_n \times (I_1 * \ldots I_{n-1} * I_{n+1} \ldots * I_N) * I_n \times (I_1 * \ldots I_{n-1} * I_{n+1} \ldots * I_N)^T$
  - Result is: symmetric $I_n \times I_n$ matrix
- Depending on size of R_n relative to I_n, require asymptotically comparable amounts of work

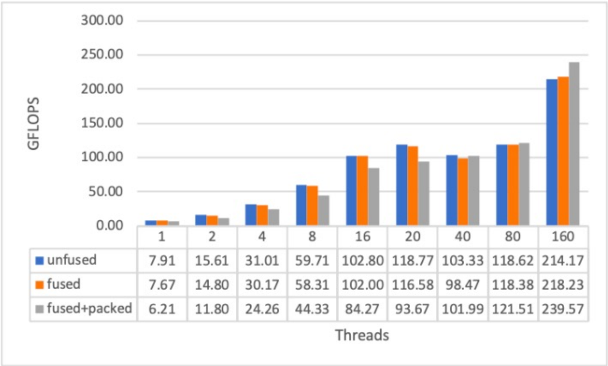
Benchmark results



(a) mode-0



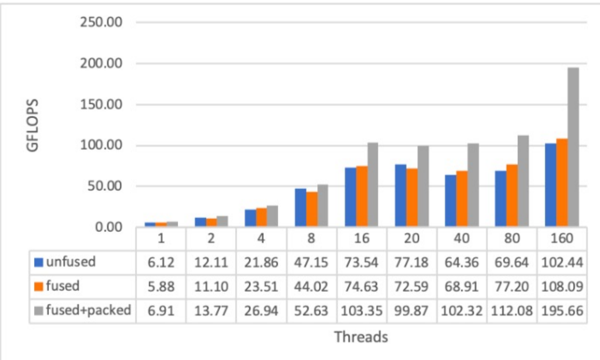
(b) mode-1



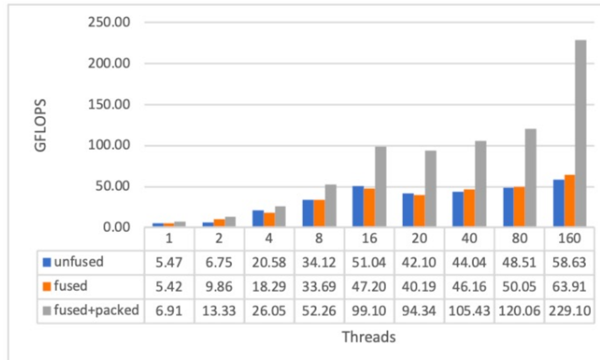
(c) mode-2

- $16\times16\times16\times16\times16\times16\times16$
  - Dense, random tensor
- Uses KokkosKernel’s SerialGemm
- Run on IBM Power 9
- Comparison of:
  - Unfused TTM + Gram
  - Fused
  - Fused+packed

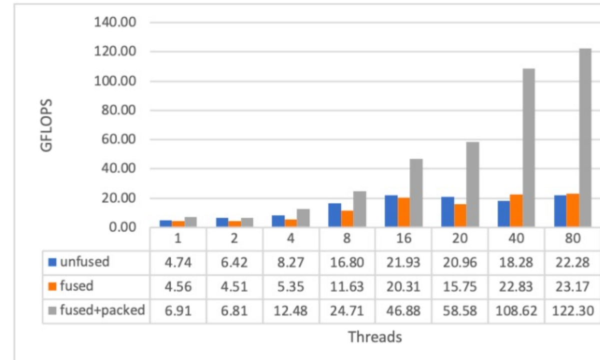
Another page of graphs



(d) mode-3



(e) mode-4

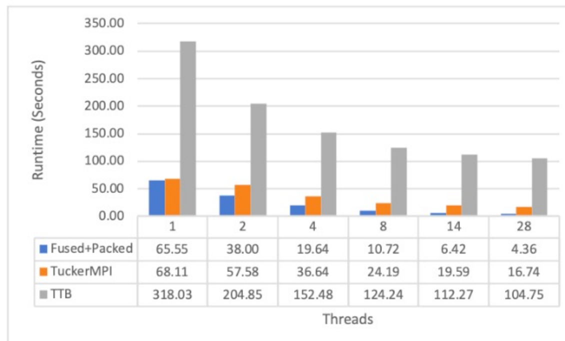


(f) mode-5

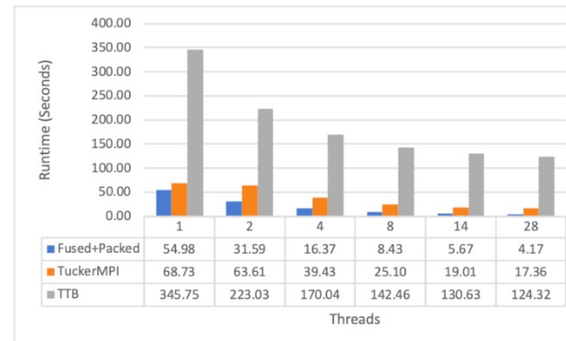
- Later modes submatrices become very long
  - Start falling out of cache
  - Begin to require skinny matrix multiplications that many GEMM kernels are not optimized for
- Packed blocks maintain performance for later dimensions
- Well worth the packing overhead

Benchmark results (cont.)

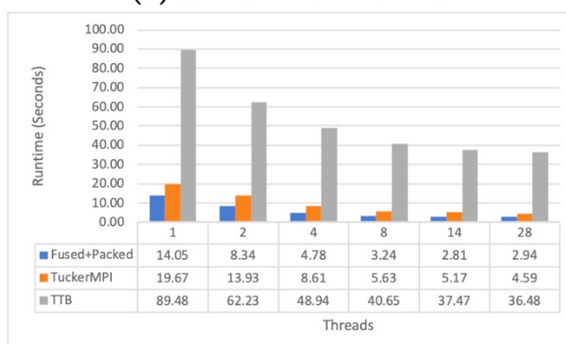
- 4 dense, random tensors
  - Error tolerance = 0
- All use MKL
- Run on Intel Xeon E5
  - 14 cores per socket
  - 2 sockets
- Comparison of:
  - Proposed Fused+Packed
  - TuckerMPI
  - Matlab Tensor Toolbox
- Fused+Packed scales better



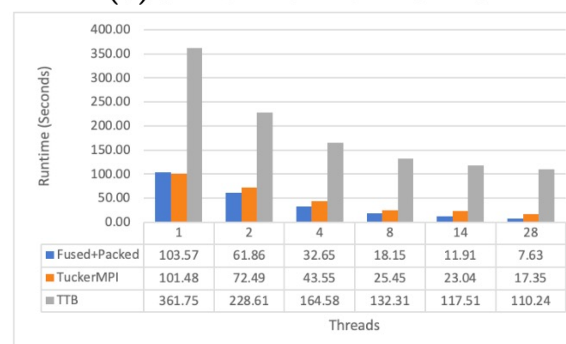
(a) $64 \times 64 \times 64 \times 64 \times 64$



(b) $32 \times 32 \times 32 \times 32 \times 32 \times 32$



(c) $16 \times 16 \times 16 \times 16 \times 16 \times 16 \times 16$



(d) $4 \times 128 \times 128 \times 128 \times 128$

References

[1] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, "A new truncation strategy for the higher-order singular value decomposition," *SIAM Journal on Scientific Computing*, vol. 34, pp. 1027–1052, 04 2012.

[2] L. R. Tucker, "Implications of factor analysis of three-way matrices for measurement of change," in *Problems in measuring change.*, C. W. Harris, Ed. Madison WI: University of Wisconsin Press, 1963, pp. 122–137.

[3] L. Omberg, G. Golub, and O. Alter, "A tensor higher-order singular value decomposition for integrative analysis of dna microarray data from different studies," *Proceedings of the National Academy of Sciences*, vol. 104, no. 47, pp. 18 371—-18 376, November 2007.

[4] T. Souza, A. L. Aquino, and D. Gomes, "An online method to detect urban computing outliers via higher-order singular value decomposition," *Sensors (Basel, Switzerland)*, vol. 19, 2019.

[5] G. Ballard, A. Klinvex, and T. G. Kolda, "Tuckermpi: A parallel C++/MPI software package for large-scale data compression via the tucker tensor decomposition," *CoRR*, vol. abs/1901.06043, 2019. [Online]. Available: http://arxiv.org/abs/1901.06043

[6] H. Kolla, X.-Y. Zhao, J. H. Chen, and N. Swaminathan, "Velocity and reactive scalar dissipation spectra in turbulent premixed flames," *Combustion Science and Technology*, vol. 188, no. 9, pp. 1424–1439, 2016. [Online]. Available: https://doi.org/10.1080/00102202.2016.1197211

[7] S. Lyra, B. Wilde, H. Kolla, J. M. Seitzman, T. C. Lieuwen, and J. H. Chen, "Structure of hydrogen-rich transverse jets in a vitiated turbulent flow," *Combustion and Flame*, vol. 162, no. 0, 11 2014.

[8] T. Shead, H. Kolla, A. Konduri, G. Papoola, W. L. Davis, D. Dunlavy, and K. Reed, "A framework for in-situ anomaly detection in hpc environments." 9 2019.

[9] K. Aditya, H. Kolla, W. P. Kegelmeyer, T. M. Shead, J. Ling, and W. L. Davis, "Anomaly detection in scientific data using joint statistical moments," *Journal of Computational Physics*, vol. 387, p. 522–538, Jun 2019. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2019.03.003

[10] "Pelec, version 00," 5 2017. [Online]. Available: https://www.osti.gov//servlets/purl/1374142

[11] E. T. Phipps and T. G. Kolda, "Software for sparse tensor decomposition on emerging computing architectures," *CoRR*, vol. abs/1809.09175, 2018. [Online]. Available: http://arxiv.org/abs/1809.09175

[12] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202 – 3216, 2014.

References (Cont.)

[13] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[14] J. Choi, X. Liu, and V. Chakaravarthy, "High-performance dense tucker decomposition on gpu clusters," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 543–553.

[15] J. Li, C. Battaglino, I. Perros, J. Sun, and R. Vuduc, "An input-adaptive and in-place approach to dense tensor-times-matrix multiply," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12.

[16] F. G. Van Zee and R. A. van de Geijn, "BLIS: A framework for rapidly instantiating BLAS functionality," *ACM Transactions on Mathematical Software*, vol. 41, no. 3, pp. 14:1–14:33, Jun. 2015. [Online]. Available: http://doi.acm.org/10.1145/2764454

[17] "An updated set of basic linear algebra subprograms (blas)," *ACM Trans. Math. Softw.*, vol. 28, no. 2, p. 135–151, Jun. 2002. [Online]. Available: https://doi.org/10.1145/567806.567807

[18] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[19] J. Filipovič, M. Madzin, J. Fousek, and L. Matyska, "Optimizing cuda code by kernel fusion: application on blas," *The Journal of Supercomputing*, vol. 71, no. 10, p. 3934–3957, Jul 2015. [Online]. Available: http://dx.doi.org/10.1007/s11227-015-1483-z

[20] S. Tabik, G. Ortega Lopez, and E. M. Garzon, "Performance evaluation of kernel fusion blas routines on the gpu: iterative solvers as case study," *The Journal of Supercomputing*, vol. 70, pp. 577–587, 11 2014.

[21] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, 1996.

[22] M. D. Lam, E. E. Rothberg, and M. E. Wolf, "The cache performance and optimizations of blocked algorithms," *SIGPLAN Not.*, vol. 26, no. 4, p. 63–74, Apr. 1991. [Online]. Available: https://doi.org/10.1145/106973.106981

[23] C. Rivera, J. Chen, N. Xiong, S. L. Song, and D. Tao, "Tsm2x: High-performance tall-and-skinny matrix-matrix multiplication on gpus," 2020.

[24] "Intel math kernel library users manual," Intel Corporation, 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top.html