UNIVERSITY OF OREGON

Human vs. AI – Who Is Better at Sparse Computation?

Jee Whan Choi

University of Oregon

March 3rd, 2026 @ NHR PerfLab Seminar Series

Motivation

Sparse computation is difficult – structure of the data becomes (a big) part of the problem

Motivation

Sparse computation is difficult – structure of the data becomes (a big) part of the problem

- Non-contiguous data makes accessing data and parallel computation less efficient
- Overhead of indexing and metadata
- (Parallel) Algorithmic and performance modeling complexity
- Under-utilization of specialized hardware resources (e.g., tensor cores, prefetchers)
- And more

Motivation

What can we do?

- Specialized data structures/formats (e.g., CSR, ELL, DIA, etc.), if data is structured
- What if we (human) can't see the pattern? **Can AI come to our aid**?
- We want to answer this question for **sparse tensor computation**

Can we make this better?

- What if you asked ChatGPT?
  - “Can you design a sparse tensor format for MTTKRP that's better than ALTO?”

Motivation

[1] J. Choi, X. Liu, S. Smith and T. Simon, "Blocking Optimization Techniques for Sparse Tensor Computation," *IPDPS'18*

[2] A. E. Helal *et al*., "ALTO: Adaptive Linearized Storage of Sparse Tensors," ICS '21

[3] A. Nguyen *et al*., "Efficient, out-of-memory sparse MTTKRP on massively parallel architectures," ICS '22

[4] Y. Soh *et al*., "Dynamic Tensor Linearization and Time Slicing for Efficient Factorization of Infinite Data Streams," *IPDPS'23*

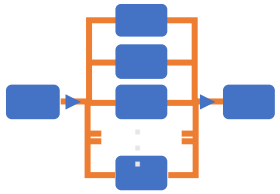
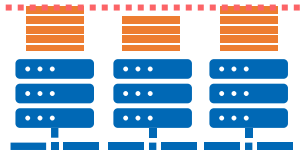
[5] Y. Soh et al., "Accelerated Constrained Sparse Tensor Factorization on Massively Parallel Architectures," *ICPP'24*

[6] J. Laukemann *et al*., "Accelerating Sparse Tensor Decomposition Using Adaptive Linearized Representation," TPDS, 2025

[7] S. Smith *et al*., "*FROSTT: The formidable repository of open sparse tensors and tools,"* online, 2017 [http://frostt.io]

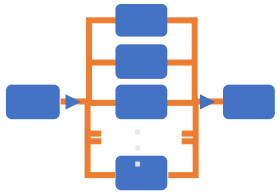
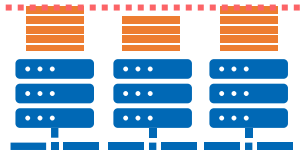
Motivation

- A fundamental problem in sparse tensor computation is how to *store*, *group*, and *organize* the nonzero elements to

1. reduce tensor storage 
2. improve data locality 
3. increase parallelism 
4. decrease workload imbalance & synchronization overhead 

Motivation

- A fundamental problem in sparse tensor computation is how to *store*, *group*, and *organize* the nonzero elements to

1. reduce tensor storage 
2. improve data locality 
3. increase parallelism 
4. decrease workload imbalance & synchronization overhead 

Conflicting goals

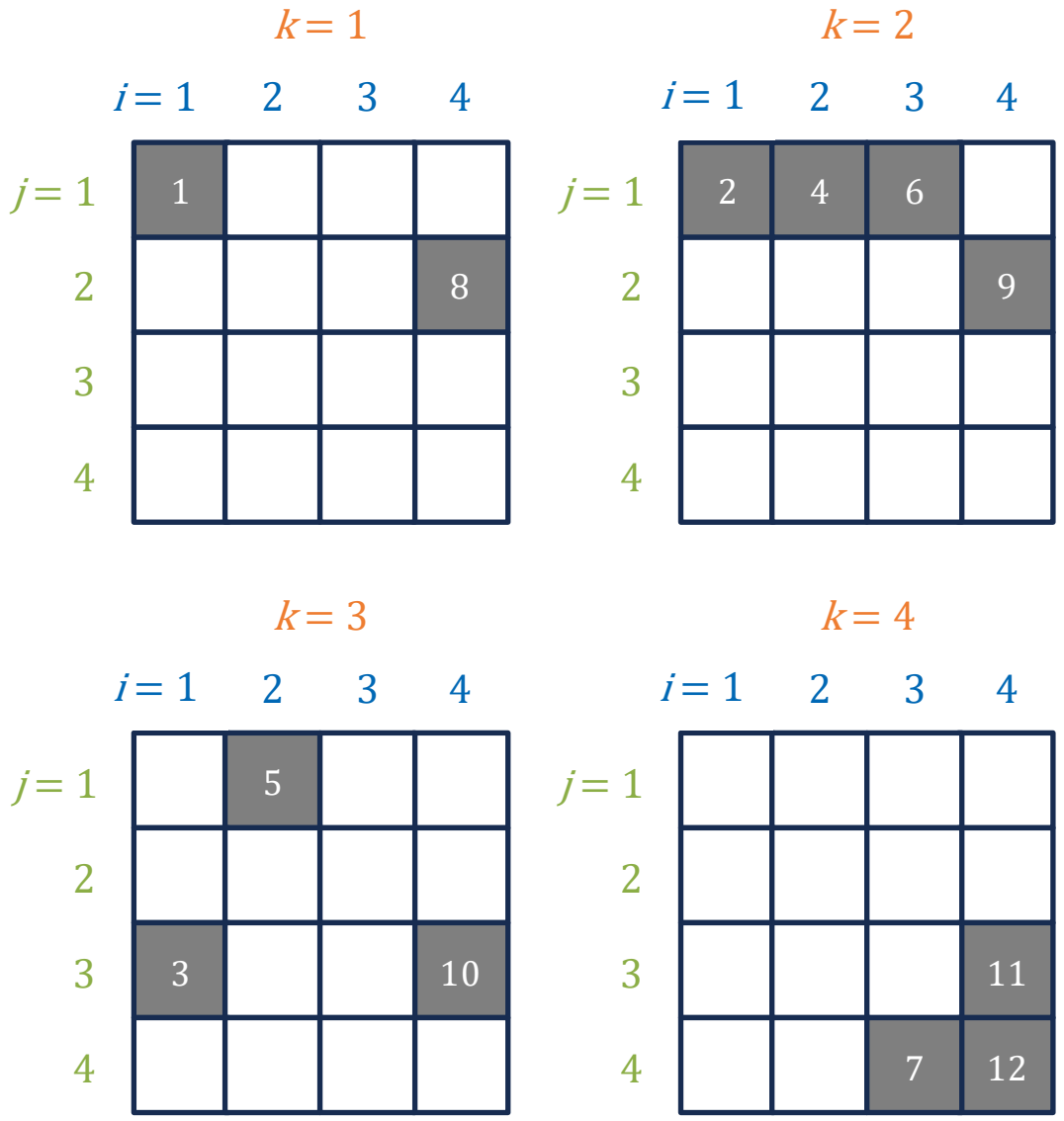
State-of-the-art

- Leverage (decades of) experience & knowledge from spares matrix computation to design compressed sparse tensor formats
- Can be classified based on their encoding of the indexing metadata into:
  1. List-based formats
  2. Tree-based formats
  3. Block-based formats

State-of-the-art

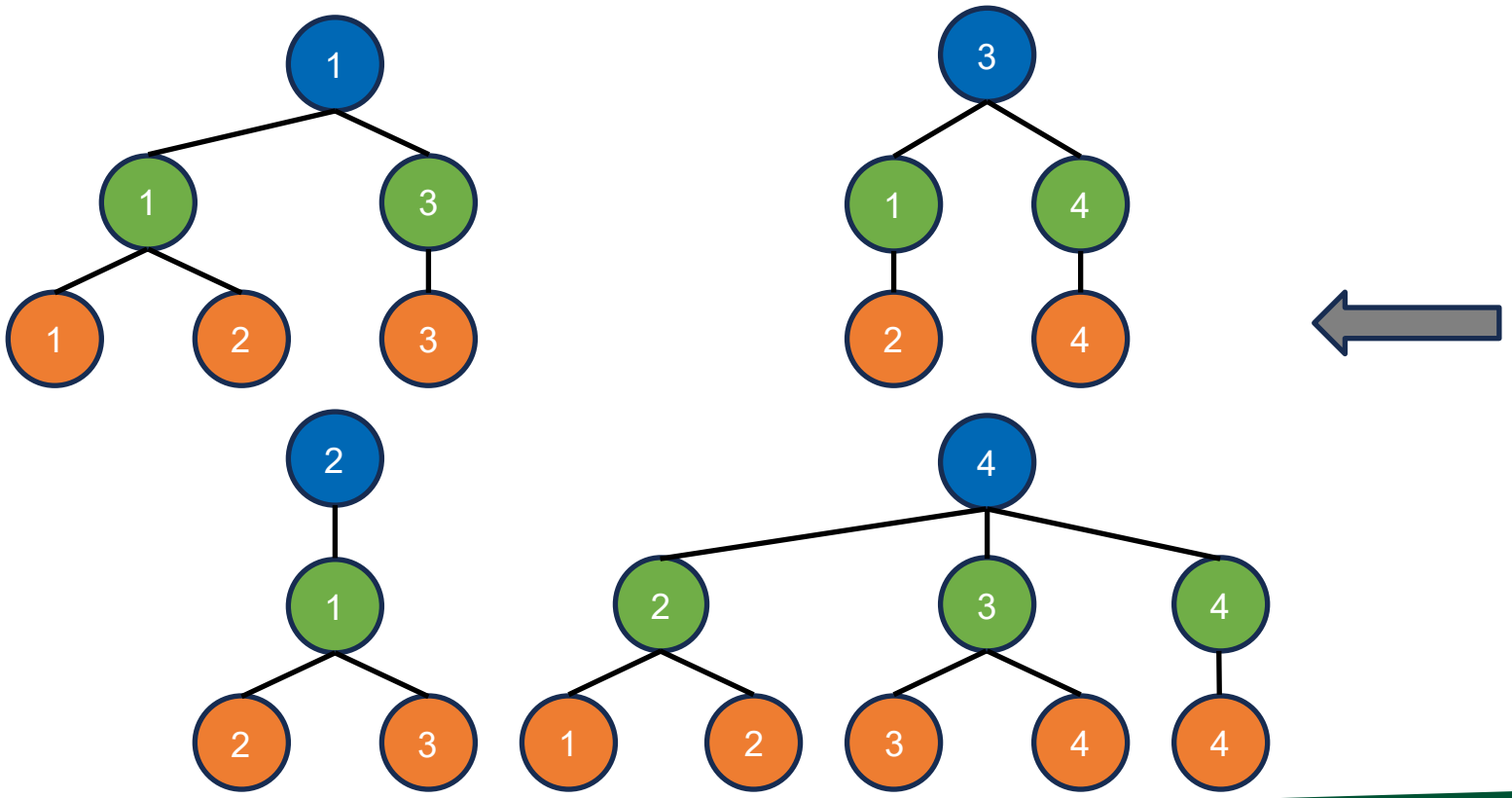
- List-based format: COO (coordinate)

i	j	k	v
1	1	1	1
1	1	2	2
1	3	3	3
2	1	2	4
2	1	3	5
3	1	2	6
3	4	4	7
4	2	1	8
4	2	2	9
4	3	3	10
4	3	4	11
4	4	4	12



State-of-the-art

- Tree-based format: CSF (compressed sparse fiber)



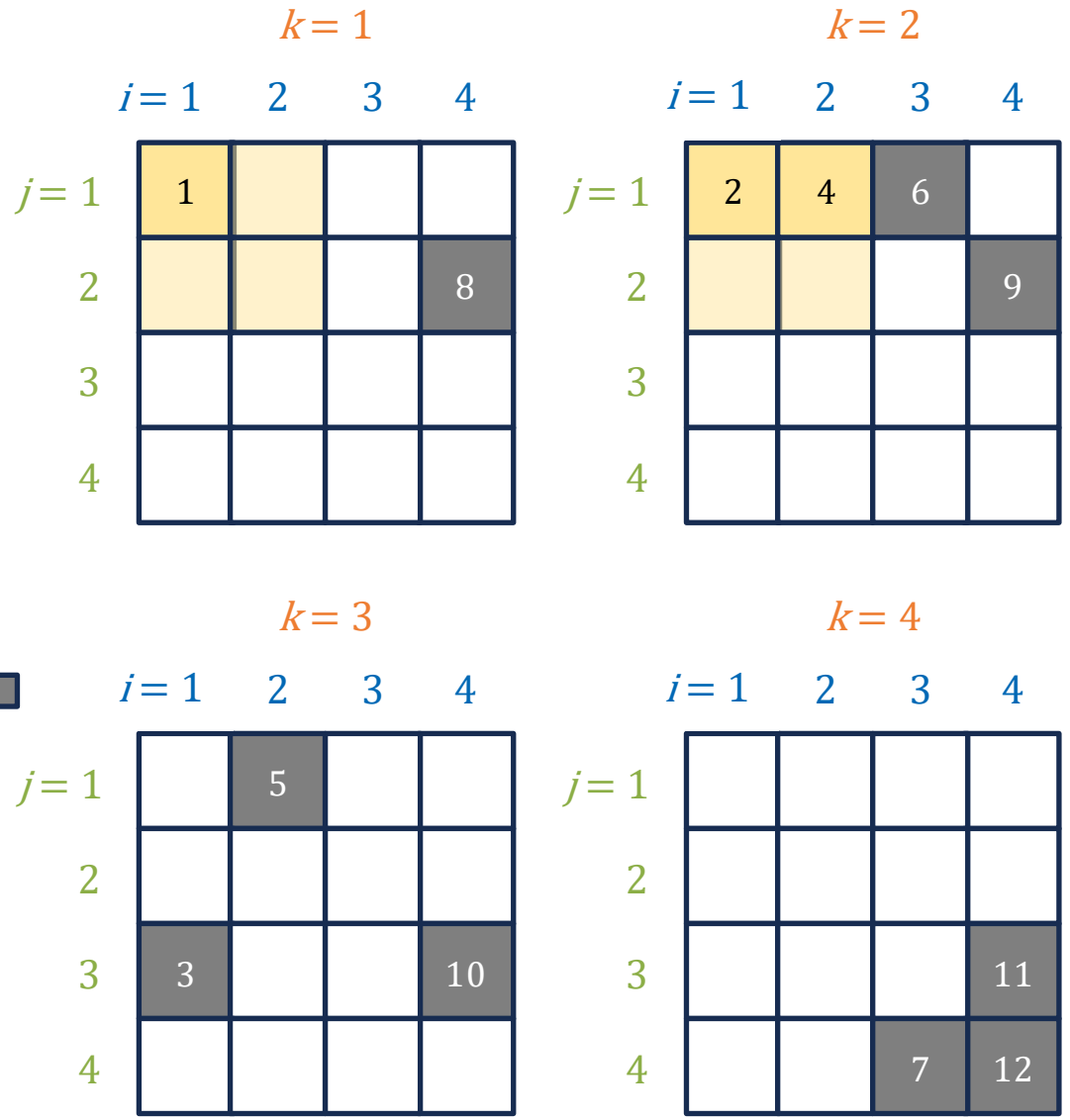
i	j	k	v
1	1	1	1
1	1	2	2
1	3	3	3
2	1	2	4
2	1	3	5
3	1	2	6
3	4	4	7
4	2	1	8
4	2	2	9
4	3	3	10
4	3	4	11
4	4	4	12

State-of-the-art

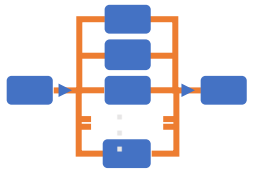
- Block-based format: HiCOO (hierarchical COO)

b_{ptr}	b_i	b_j	b_k	e_i	e_j	e_k	v
1	0	0	0	1	1	1	1
1	0	0	1	1	1	0	2
1	1	0	1	0	1	0	4
2	1	0	1	0	1	1	5
3	0	1	1	1	1	1	3
4	1	1	0	2	0	1	8
4	1	0	1	1	1	0	6
4	1	1	1	2	0	0	9
5	1	1	1	2	1	1	10
5	1	1	1	1	2	2	7
5	1	1	1	2	1	2	11
5	1	1	1	2	2	2	12

2x2x2 block



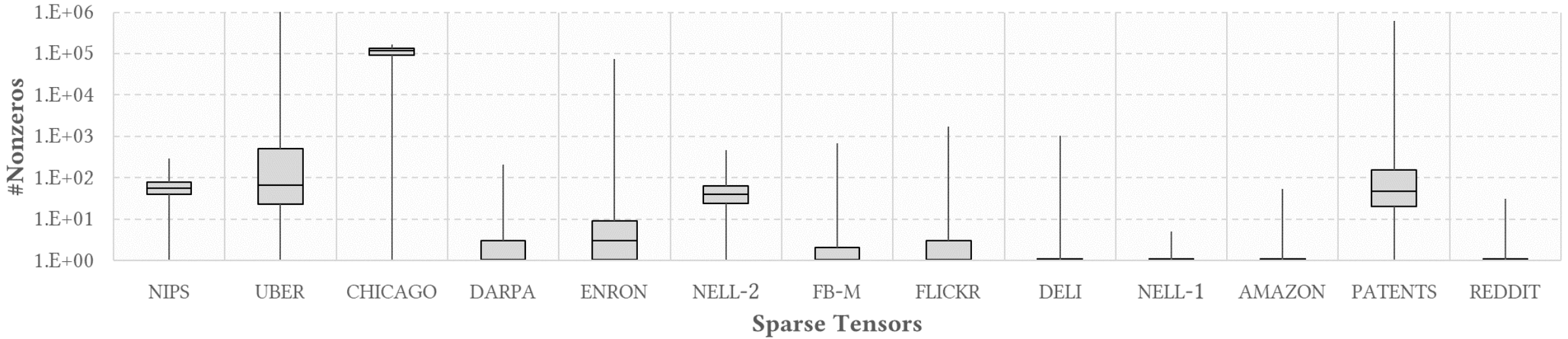
State-of-the-art


Formats	Granularity	Mode Orientation	Data Locality	Parallelism	Load balance
List-based (COO)	Nonzero element	Mode-agnostic	Poor	Suffer from conflicts	Maximized
Tree-based (CSF)	Compressed tree	Mode-specific	Improved for a specific mode	Improved for a specific mode	Work imbalance (especially in short modes)
Block-based (HiCOO)	Compressed block	Mode-agnostic	Improved	Suffer from conflicts	Work imbalance across blocks

Popularity of tree- and block-based formats

Spatial data distribution



A box plot of the data (nonzero elements) distribution across multi-dimensional blocks. The multi-dimensional subspace size is 128^N, where N is the number of dimensions (modes) [2]

Popularity of tree- and block-based formats

- Tree- and block-based formats are extensions of sparse matrix formats
- However, they are ill-suited for sparse tensors

Popularity of tree- and block-based formats

- Tree- and block-based formats are extensions of sparse matrix formats
- However, they are ill-suited for sparse tensors

1. Tensor computation often operate over every dimension
   - This contrasts with sparse matrix-vector multiply (*one dimension*) and sparse matrix-matrix multiply (*two dimensions*)

Popularity of tree- and block-based formats

- Tree- and block-based formats are extensions of sparse matrix formats
- However, they are ill-suited for sparse tensors

1. Tensor computation often operate over every dimension
   - This contrasts with sparse matrix-vector multiply (*one dimension*) and sparse matrix-matrix multiply (*two dimensions*)
2. Tensor sparsity >> matrix sparsity
   - Due to their dimensionality, tensors are extremely sparse – it's difficult to find dense blocks
   - Tensor sparsity ranges from 1.5×10^{-2} to 4.3×10^{-15} [7]

Popularity of tree- and block-based formats

- Tree- and block-based formats are extensions of sparse matrix formats
- However, they are ill-suited for sparse tensors

1. Tensor computation often operate over every dimension
   - This contrasts with sparse matrix-vector multiply (*one dimension*) and sparse matrix-matrix multiply (*two dimensions*)
2. Tensor sparsity >> matrix sparsity
   - Due to their dimensionality, tensors are extremely sparse – it's difficult to find dense blocks
   - Tensor sparsity ranges from 1.5×10^{-2} to 4.3×10^{-15} [7]

Our hypothesis – *simple mode-agnostic, list-based formats are the best for sparse tensor computation*

Linearized Formats

- ALTO (Adaptive Linearized Tensor Order) for CPUs [2]
- BLCO (Block Linearized Coordinate) for GPUs [3]
- Application of linearized formats to
  - *streaming* tensor decomposition [4]
  - *on-the-fly* Khatri-Rao product for CP-APR [6]
  - non-negative sparse tensor factorization for GPUs via PLANC [5]

ALTO – Adaptive Linearized Tensor Order

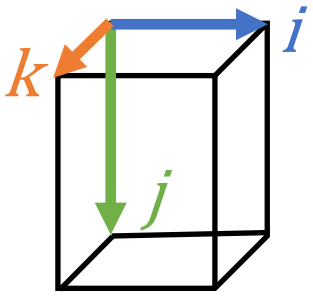
Tammy Kolda

T. G. Kolda, B. W. Bader, ”**Tensor Decompositions and Applications,”** *SIAM Review*, 2009.

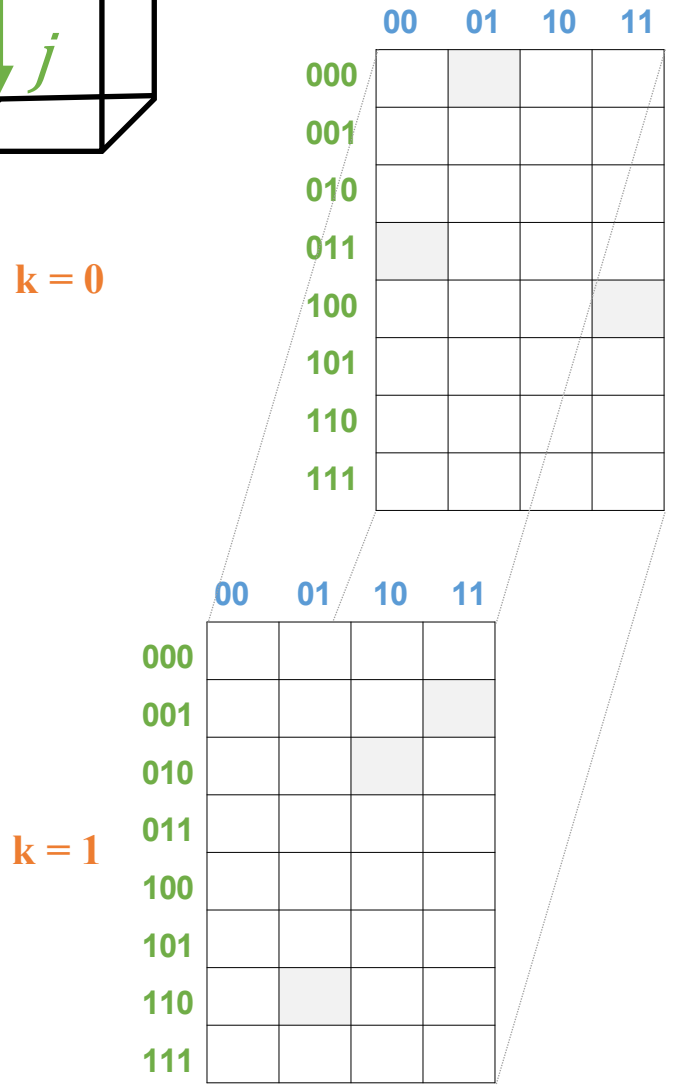
Harrison, Adam P., and Dileepan Joseph, "High Performance Rearrangement and Multiplication Routines for Sparse Tensor Arithmetic." *SIAM Journal on Scientific Computing,* 2018

ALTO

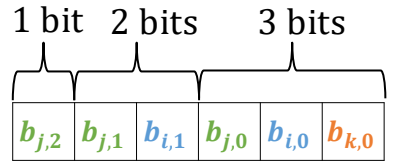
- ALTO interleaves the index bits by grouping them by their positions
- Within each group, the bits are arranged from the longest mode (most-significant) to the shortest (least-significant)



4x8x2 tensor



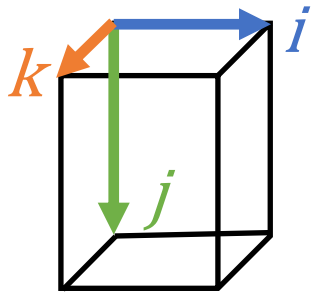
ALTO bit mask



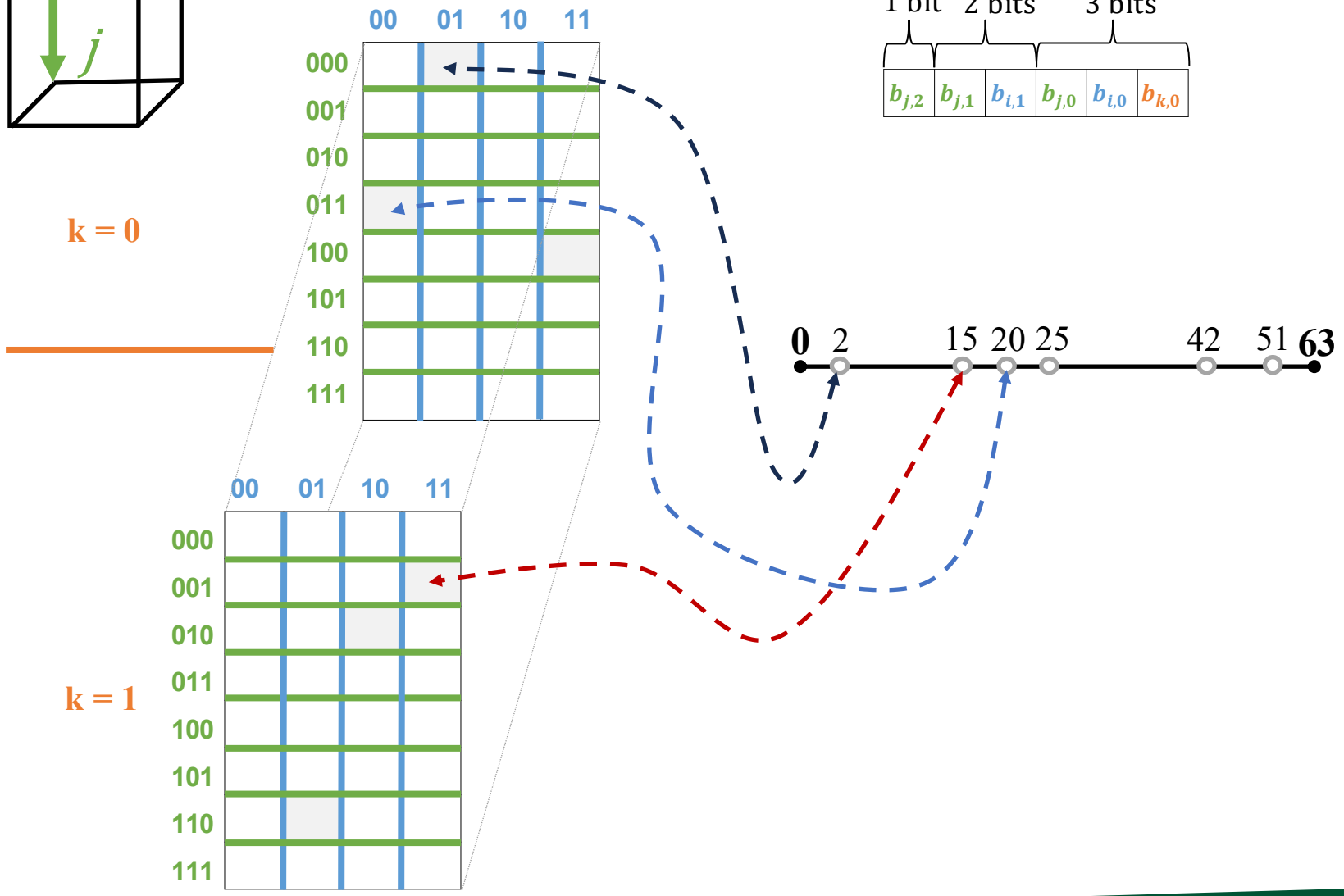
ALTO

ALTO

Value	Position
$x_{1,0,0}$	2 (000010)
$x_{3,1,1}$	15 (001111)
$x_{0,3,0}$	20 (010100)
$x_{2,2,1}$	25 (011001)
$x_{3,4,0}$	42 (101010)
$x_{1,6,1}$	51 (110011)

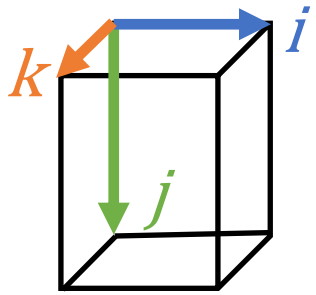


4x8x2 tensor

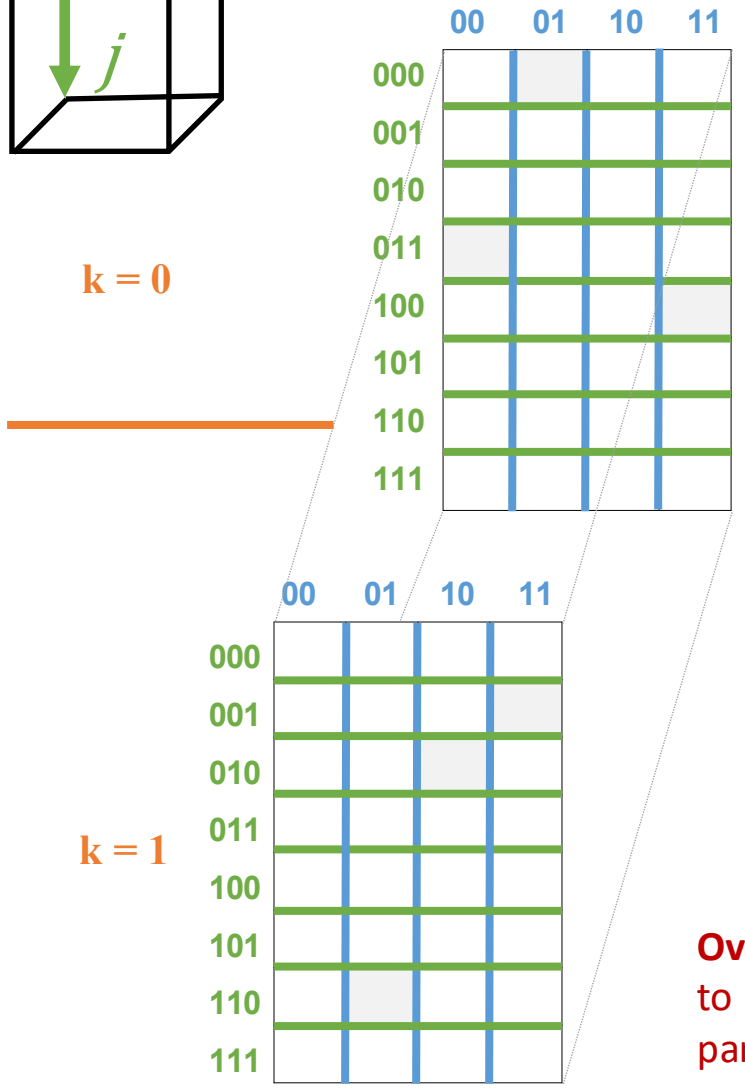


ALTO bit mask

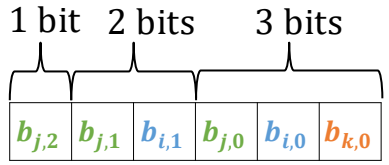
Partitioning



4x8x2 tensor

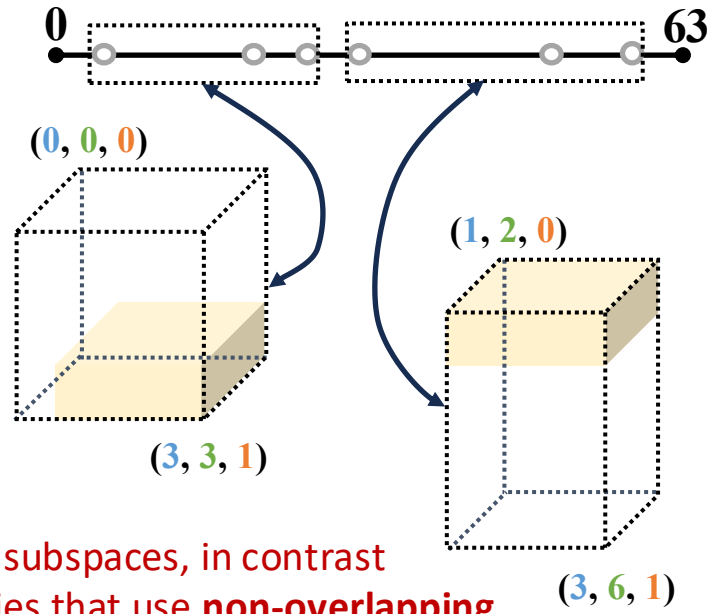


ALTO bit mask



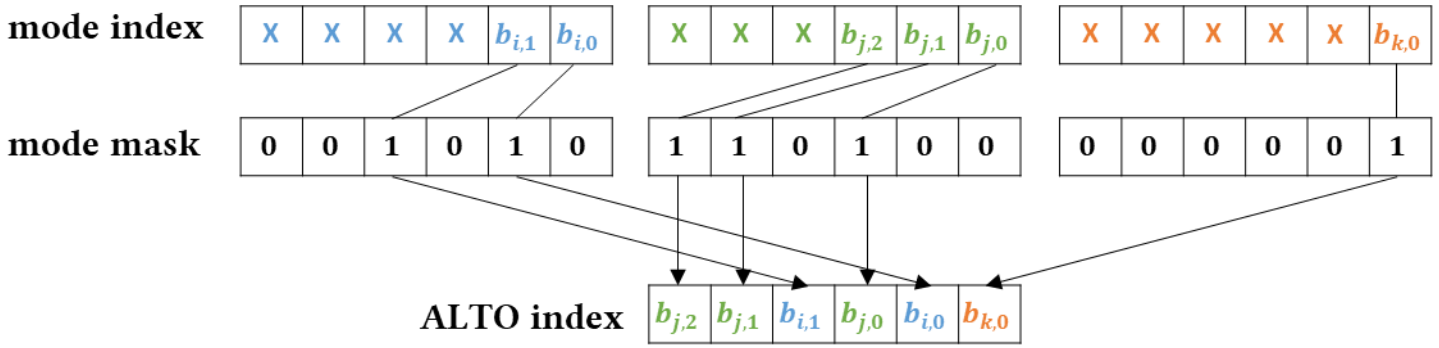
ALTO

Value	Position
$x_{1,0,0}$	2 (000010)
$x_{3,1,1}$	15 (001111)
$x_{0,3,0}$	20 (010100)
$x_{2,2,1}$	25 (011001)
$x_{3,4,0}$	42 (101010)
$x_{1,6,1}$	51 (110011)

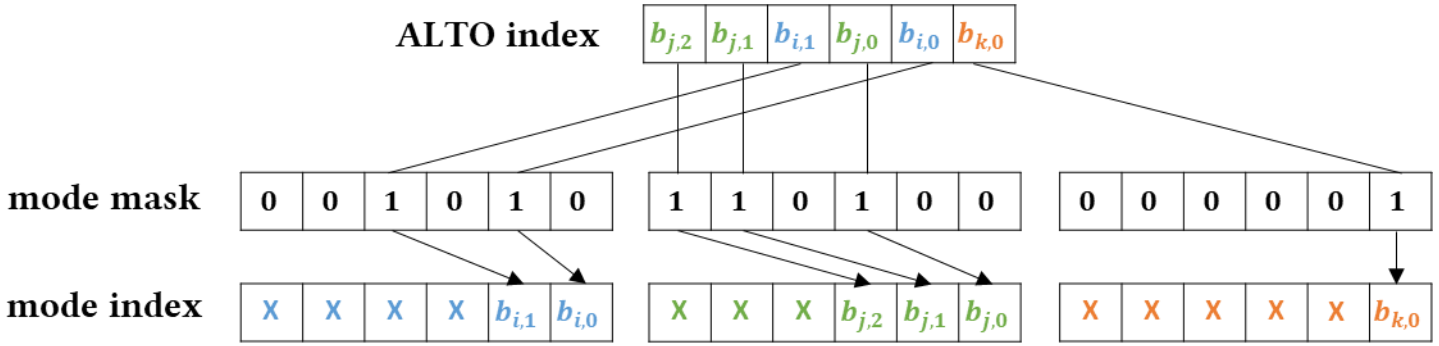


Overlapping subspaces, in contrast to prior studies that use **non-overlapping** partitions, that often lead to **imbalance**.

Fast Index Encoding & Decoding



Encoding (bit-level gather)



Decoding (bit-level scatter)

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
for l = 1,…,L in parallel do
        for all x ∈ X_l do
                fetch i_x,j_x,k_x,v_x
                fetch v_1 = A^(2)(j_x,:)
                fetch v_2 = A^(3)(k_x,:)
                scratch += v_x · (v_1 * v_2)
                A^(1)(i_x,:) += scratch
        endfor
endfor
```

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
for l = 1,…,L in parallel do
        for all x ∈ X_l do
                fetch i_x,j_x,k_x,v_x
                fetch v_1 = A(2)(j_x,:)
                fetch v_2 = A(3)(k_x,:)
                scratch += v_x · (v_1 * v_2)
                A(1)(i_x,:) += scratch
        endfor
endfor
```

Each partition l could be a HiCOO block or a CSF sub-tree

Each nonzero element in the partition

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
for l = 1,…,L in parallel do
        for all x ∈ X_l do
                fetch i_x,j_x,k_x,v_x
                fetch v_1 = A(2)(j_x,:)
                fetch v_2 = A(3)(k_x,:)
                scratch += v_x · (v_1 * v_2)
                A(1)(i_x,:) += scratch
        endfor
endfor
```

Each partition l could be a HiCOO block or a CSF sub-tree

Each nonzero element in the partition

Fetch the indices and value associated with nonzero x

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
for l = 1,…,L in parallel do
        for all x ∈ X_l do
                fetch i_x,j_x,k_x,v_x
                fetch v_1 = A^(2)(j_x,:)
                fetch v_2 = A^(3)(k_x,:)
                scratch += v_x · (v_1 * v_2)
                A^(1)(i_x,:) += scratch
        endfor
endfor
```

Each partition l could be a HiCOO block or a CSF sub-tree

Each nonzero element in the partition

Fetch the indices and value associated with nonzero x

Fetch row j_x from mode-2 factor matrix

Fetch row k_x from mode-3 factor matrix

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
for l = 1,…,L in parallel do
        for all x ∈ X_l do
                fetch i_x,j_x,k_x,v_x
                fetch v_1 = A(2)(j_x,:)
                fetch v_2 = A(3)(k_x,:)
                scratch += v_x · (v_1 * v_2)
                A(1)(i_x,:) += scratch
        endfor
endfor
```

Each partition l could be a HiCOO block or a CSF sub-tree

Each nonzero element in the partition

Fetch the indices and value associated with nonzero x

Fetch row j_x from mode-2 factor matrix

Fetch row k_x from mode-3 factor matrix

MTTKRP

Update row i_x from mode-1 factor matrix, either **atomically** (e.g., COO, or HiCOO), or **freely** (e.g., CSF)

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
for l = 1,…,L in parallel do  <----------  ALTO line segment
        for all x ∈ X_l do
                fetch i_x,j_x,k_x,v_x
                fetch v_1 = A^(2)(j_x,:)
                fetch v_2 = A^(3)(k_x,:)
                scratch += v_x · (v_1 * v_2)
                A^(1)(i_x,:) += scratch
        endfor
endfor
```

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
for l = 1,…,L in parallel do
        for all x ∈ X_l do
                fetch i_x,j_x,k_x,v_x
                fetch v_1 = A^(2)(j_x,:)
                fetch v_2 = A^(3)(k_x,:)
                scratch += v_x · (v_1 * v_2)
                A^(1)(i_x,:) += scratch
        endfor
endfor
```

ALTO line segment

fetch a_x (ALTO index), v_x

i_x, j_x, k_x = decode(a_x)

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
for l = 1,…,L in parallel do
        for all x ∈ X_l do
                fetch i_x,j_x,k_x,v_x
                fetch v_1 = A^(2)(j_x,:)
                fetch v_2 = A^(3)(k_x,:)
                scratch += v_x · (v_1 * v_2)
                A^(1)(i_x,:) += scratch
        endfor
endfor
```

ALTO line segment

```
fetch a_x (ALTO index),v_x
i_x,j_x,k_x = decode(a_x)
```

This can be done efficiently using parallel bit extract/deposit (`pext`/`pdep`) instructions on x86 CPUs

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
for l = 1,…,L in parallel do
        for all x ∈ X_1 do
                fetch i_x,j_x,k_x,v_x
                fetch v_1 = A(2)(j_x,:)
                fetch v_2 = A(3)(k_x,:)
                scratch += v_x · (v_1 * v_2)
                A(1)(i_x,:) += scratch
        endfor
endfor
```

ALTO line segment

```
fetch a_x (ALTO index),v_x
i_x,j_x,k_x = decode(a_x)
```

This can be done efficiently using parallel bit extract/deposit (`pext`/`pdep`) instructions on x86 CPUs

Adaptive update mechanism

- If rows (i_x) have limited reuse → update using atomic ops.
- If rows (i_x) have large reuse → use scratchpad to combine local updates and parallel reduction to merge globally
- `reuse = nnz / mode_length`

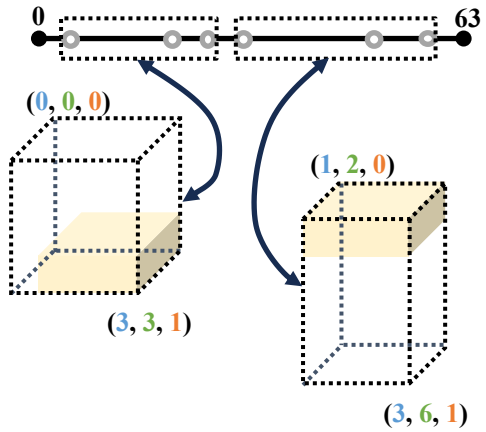
Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
for l = 1,…,L in parallel do
    for all x ∈ X_l do
        fetch i_x,j_x,k_x,v_x
        fetch v_1 = A(2)(j_x,:)
        fetch v_2 = A(3)(k_x,:)
        scratch += v_x · (v_1 * v_2)
        A(1)(i_x,:) += scratch
    endfor
endfor
```

ALTO line segment

```
fetch a_x (ALTO index),v_x
i_x,j_x,k_x = decode(a_x)
```

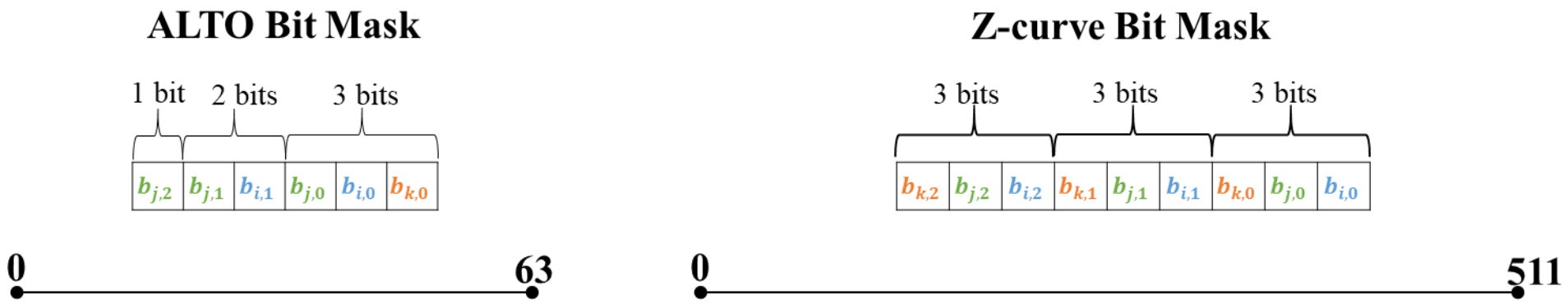
This can be done efficiently using parallel bit extract/deposit (`pext`/`pdep`) instructions on x86 CPUs



Adaptive update mechanism

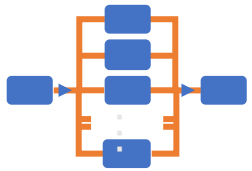
- If rows (i_x) have limited reuse → update using atomic ops.
- If rows (i_x) have large reuse → use scratchpad to combine local updates and parallel reduction to merge globally
- `reuse = nnz / mode_length`
- Global reduction uses the boundary information to "pull" rows (i_x) from the appropriate scratchpad

ALTO vs. Z-curve



- In contrast to Z-ordering, ALTO uses a non-fractal encoding to
  1. adapt to irregularly shaped tensors,
  2. further reduce storage, and
  3. reduce encoding/decoding time

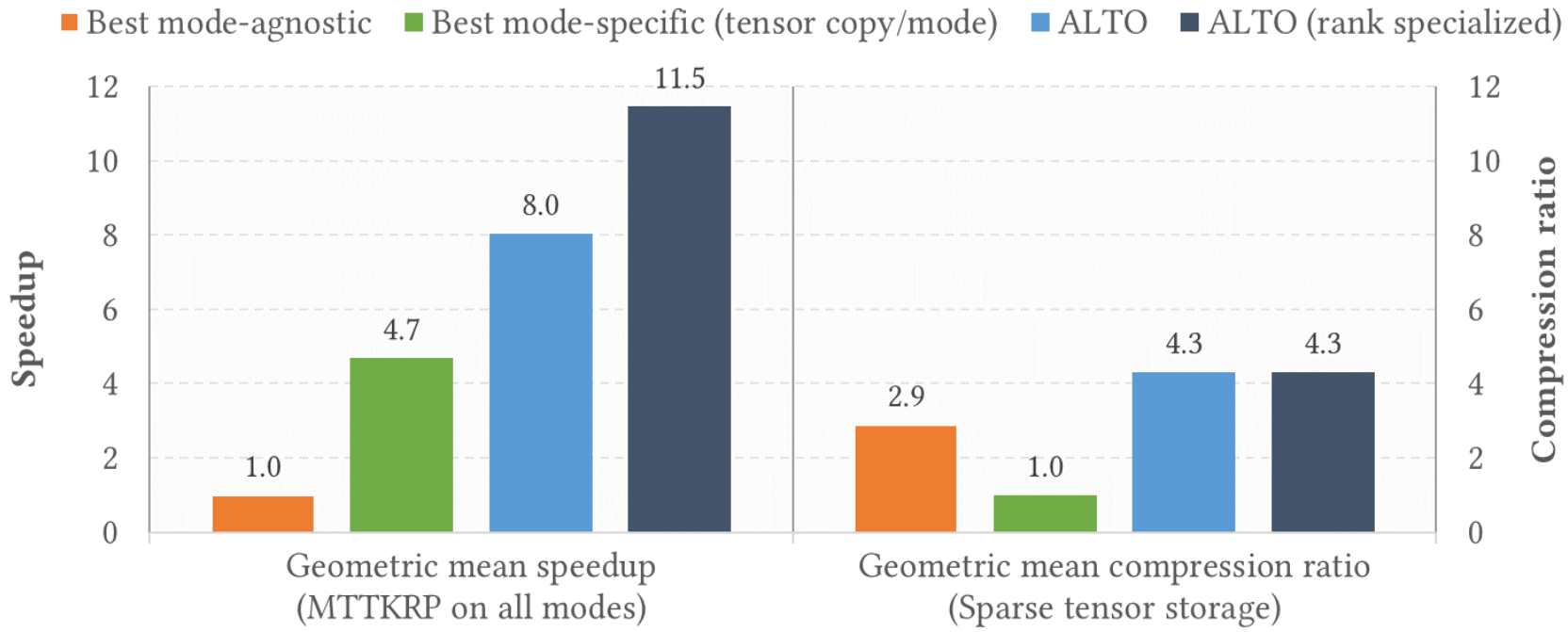
ALTO vs. State-of-the-art


Formats	Granularity	Mode Orientation	Data Locality	Parallelism	Load balance
List-based (COO)	Nonzero element	Mode-agnostic	Poor	Suffer from conflicts	Maximized
Tree-based (CSF)	Compressed tree	Mode-specific	Improved for a specific mode	Improved for a specific mode	Work imbalance (especially in short modes)
Block-based (HiCOO)	Compressed block	Mode-agnostic	Improved	Suffer from conflicts	Work imbalance across blocks
ALTO	Nonzero element	Mode-agnostic	Improved (via nearest-neighbor traversal)	Improved (via adaptive update)	Maximized

Performance Summary

- Intel Cascade Lake-X
  - 28 x 2 cores @ 1.8 GHz
- Oracle selects the best mode-agnostic and mode-specific format for each of the 15 tensors
- Mode-agnostic formats: COO and HiCOO
- Mode-specific formats: CSF and CSF with tiling with N copies



Performance Portability

- Can ALTO perform well on GPUs?
- If not, what are the challenges?

MTTKRP on GPUs

```
for l = 1,…,L in parallel do
        for all x ∈ X_l do
                fetch i_x,j_x,k_x,v_x
                fetch v_1 = A(2)(j_x,:)
                fetch v_2 = A(3)(k_x,:)
                scratch += v_x · (v_1 * v_2)
                A(1)(i_x,:) += scratch
        endfor
endfor
```

Each partition l could be a thread block

Each thread operates on a 1~k non-zero element

fetch a_x (ALTO index), v_x

$i_x, j_x, k_x = decode(a_x)$

GPUs do **NOT** support parallel bit extract/deposit instructions

Adaptive update mechanism

Large number of threads → synchronization is more expensive

Encoding/Decoding on GPUs

- Blocked Linearized Coordinates (BLCO)
  - Linearize and order using ALTO
  - Re-linearize indices by grouping bits from same index together
  - Bit-wise shift and masking is cheap on GPUs

ALTO

l		v
0	$(000000)_2$	1.0
4	$(000100)_2$	2.0
5	$(000101)_2$	4.0
10	$(001010)_2$	8.0
12	$(001100)_2$	6.0
15	$(001111)_2$	9.0
33	$(100001)_2$	5.0
48	$(110000)_2$	3.0
57	$(111001)_2$	10.0
61	$(111101)_2$	11.0
62	$(111110)_2$	7.0
63	$(111111)_2$	12.0

BLCO

l		v
0	$(00000)_2$	1.0
16	$(10000)_2$	2.0
17	$(10001)_2$	4.0
6	$(00110)_2$	8.0
18	$(10010)_2$	6.0
23	$(10111)_2$	9.0
1	$(00001)_2$	5.0
8	$(01000)_2$	3.0
11	$(01011)_2$	10.0
27	$(11011)_2$	11.0
30	$(11110)_2$	7.0
31	$(11111)_2$	12.0

Encoding/Decoding on GPUs

- Blocked Linearized Coordinates (BLCO)
  - Linearize and order using ALTO
  - Re-linearize indices by grouping bits from same index together
  - Bit-wise shift and masking is cheap on GPUs
- Adaptive Blocking
  - Use uppermost bits from every mode that exceeds target integer size (e.g., 64 bits) to form the initial blocks, then further divided to meet GPU memory constraints
  - Leverages native integer instructions
  - Reduces overall storage
  - Does not require expensive tuning
  - Leverages GPU schedulers to workload balance

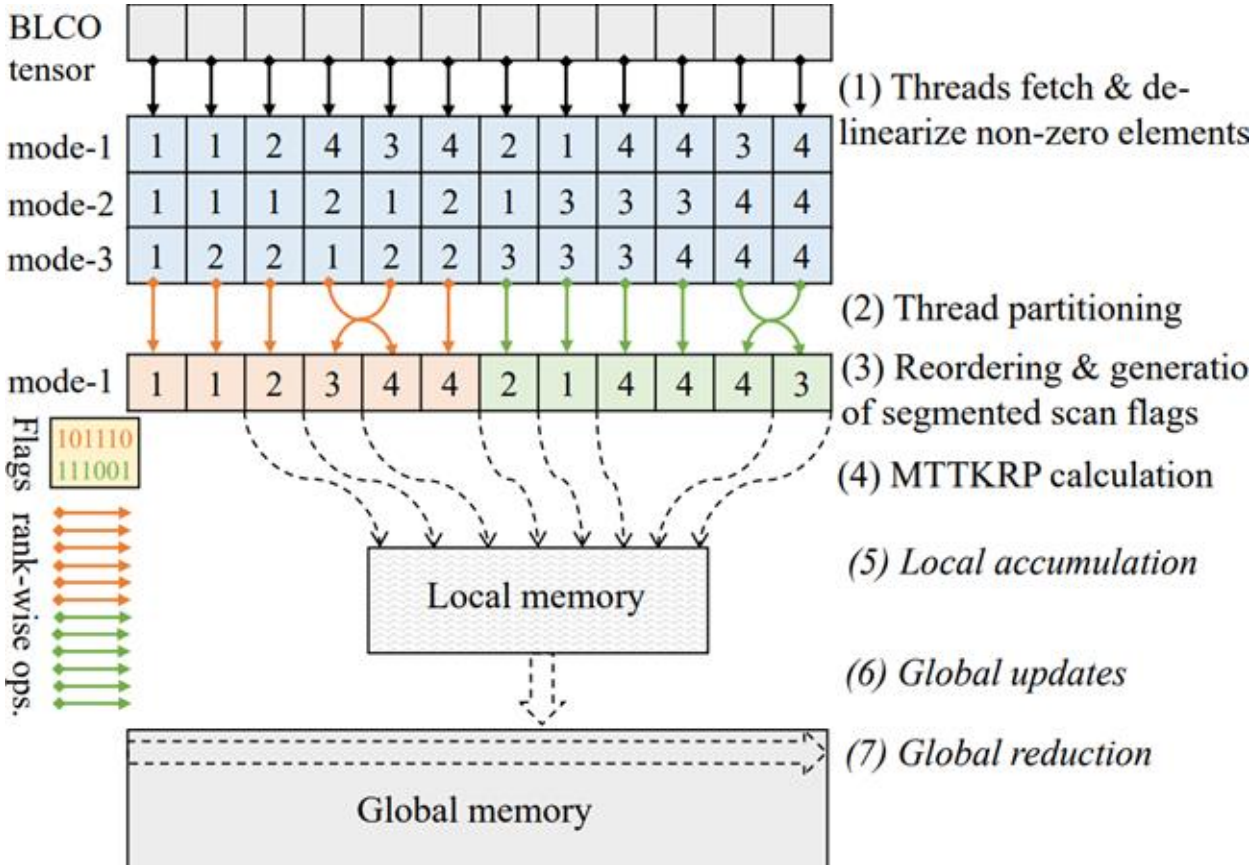
ALTO

l		v
0	$(000000)_2$	1.0
4	$(000100)_2$	2.0
5	$(000101)_2$	4.0
10	$(001010)_2$	8.0
12	$(001100)_2$	6.0
15	$(001111)_2$	9.0
33	$(100001)_2$	5.0
48	$(110000)_2$	3.0
57	$(111001)_2$	10.0
61	$(111101)_2$	11.0
62	$(111110)_2$	7.0
63	$(111111)_2$	12.0

BLCO

b	l		v
0	0	$(00000)_2$	1.0
	16	$(10000)_2$	2.0
	17	$(10001)_2$	4.0
	6	$(00110)_2$	8.0
	18	$(10010)_2$	6.0
	23	$(10111)_2$	9.0
1	1	$(00001)_2$	5.0
	8	$(01000)_2$	3.0
	11	$(01011)_2$	10.0
	27	$(11011)_2$	11.0
	30	$(11110)_2$	7.0
	31	$(11111)_2$	12.0

Synchronization



→ Global memory

→ Shared memory

→ Register memory

→ Register-level accumulation

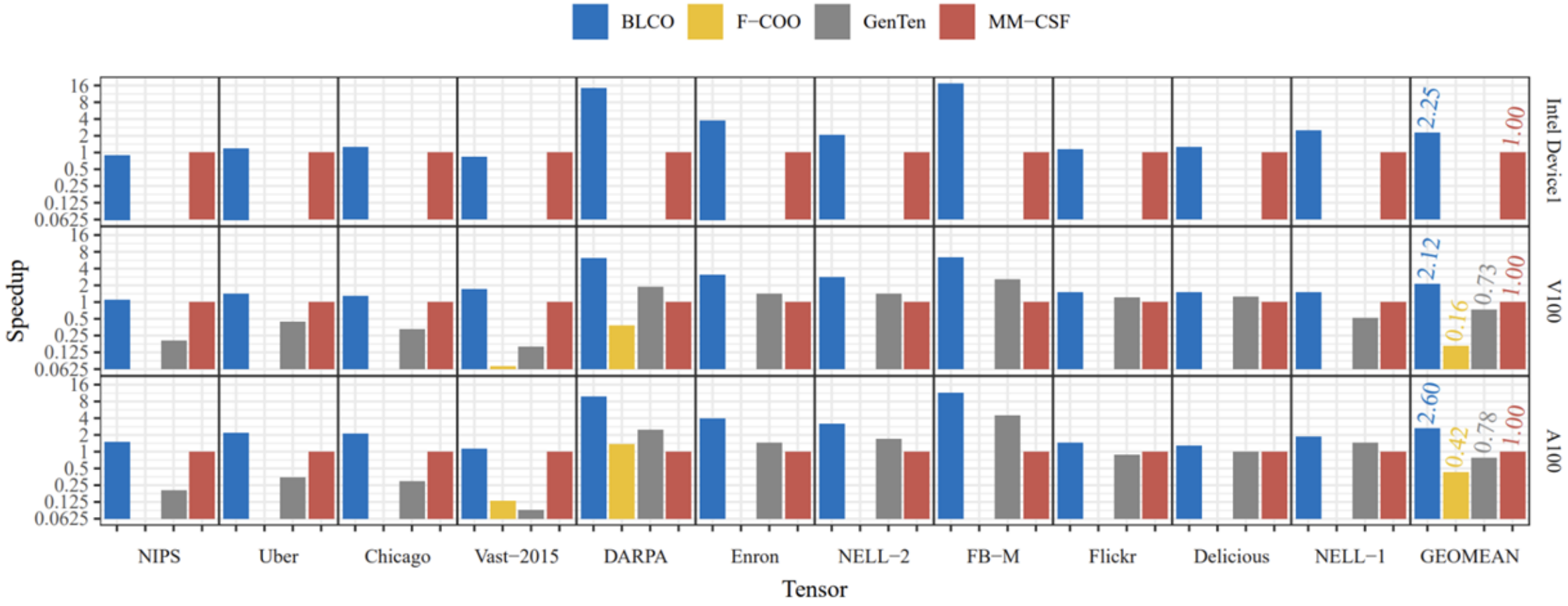
→ Shared memory accumulation

→ Global accumulation

→ Global accumulation

Performance

- MM-CSF is the baseline
- `Amazon`, `Patents`, and `Reddit` do not run on any prior frameworks
- F-COO only supports 3D tensors and `segfaults` on some tensors



Performance

- Consistent memory throughput for every mode

Data Set	Format	n	Vol[1]	TP[2]	Data Set	Format	n	Vol[1]	TP[2]
Uber	BLCO	1	2.78	3.60	Enron	BLCO	1	44.82	4.11
		2	2.75	3.61			2	46.23	4.62
		3	2.75	3.53			3	47.88	4.92
		4	2.73	2.77			4	47.22	4.70
	MM-CSF	1	1.68	1.68		MM-CSF	1	41.39	0.31
		2	1.33	2.03			2	62.83	3.16
		3	1.33	1.93			3	37.15	2.29
		4	2.12	0.32			4	37.05	3.01
Vast-2015	BLCO	1	16.91	3.92	NELL-1	BLCO	1	107.5	2.44
		2	16.73	3.77			2	104.5	2.32
		3	13.92	2.90			3	110.7	2.39
	MM-CSF	1	9.19	1.19		MM-CSF	1	123.1	2.21
		2	8.36	1.57			2	118.5	2.19
		3	8.36	1.45			3	122.1	0.86

[1] Memory volume in GB, measured by *l1tex__t_bytes.sum* in Nsight Compute [3]
[2] Memory throughput in TB/s, calculated by (Vol / total execution time)

Productivity (subjective)

- ALTO/BLCO
  - A single, very clean code that is easy to read
- HiCOO/CSF/MM-CSF
  - More complex algorithm(s)

Q&A

Can we make this better?

- Different sparsity patter would benefit from a different traversal order
  - Different space filling curves? Hilbert, Peano, etc.?
- What about AI? Can they produce a traversal order customized for the input tensor's sparsity pattern?
  - What type of AI model?
  - How do we generate/gather training data?
- What if you asked ChatGPT?

Can we make this better?

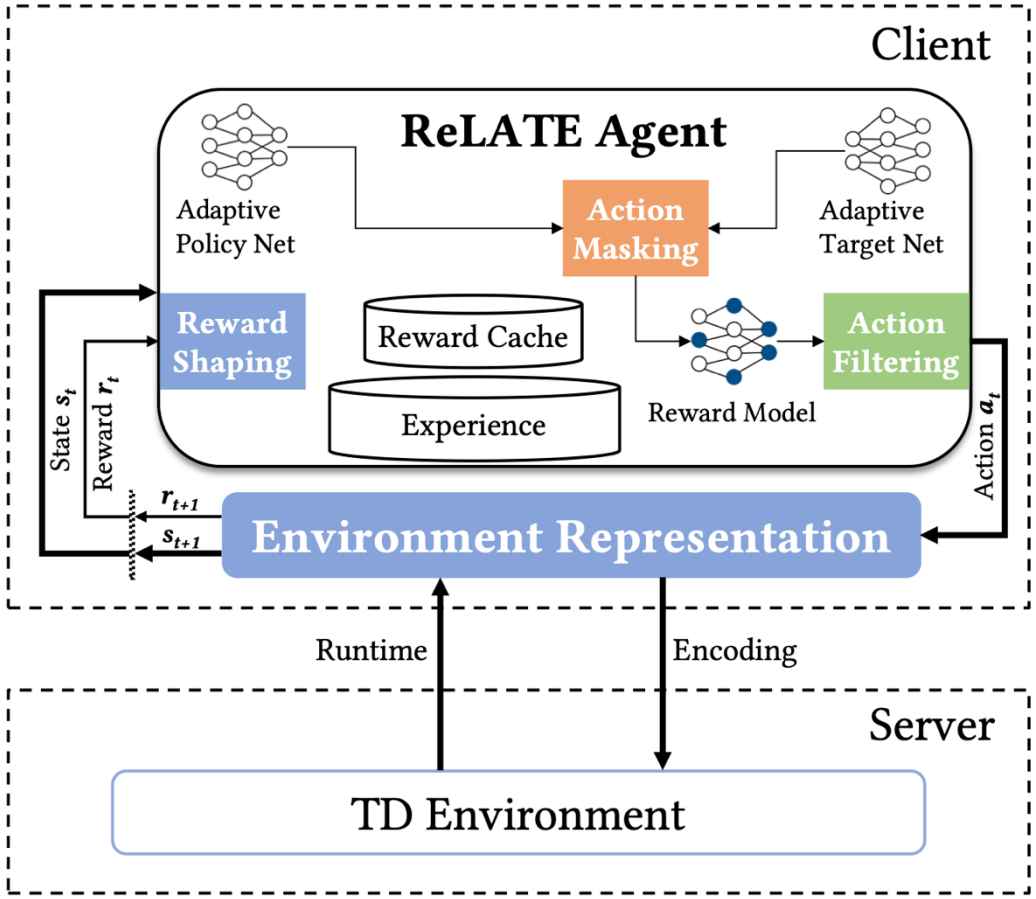
- What if you asked ChatGPT?
  - “Can you design a sparse tensor format for MTTKRP that's better than ALTO?”
  - Result: A mode-specific tree-structured format like CSF with tiling (i.e., tiled CSF)
    - Admits that they are similar under specific conditions (when tiling size is well-chosen and mode ordering is optimal) and performance difference maybe within 10 – 20%.
    - Requires N copies like CSF
    - However, claims to be better than ALTO
  - It did a good job of justifying its decision choices (e.g., that majority of the data comes from the factor matrices, and grouping by common index to maximize reuse, etc.)
  - It seems to be good at analyzing ”static” metrics (i.e., flops, # of memory requests, etc.)
  - However, it was unable to account for variation in input pattern and architectural properties
  - So, maybe not quite there yet

Motivation

- Number of ways to traverse an N-dimensional space = $(I^N)!$
- Generating training data
  - Randomly generate traversal -> measure performance -> repeat
  - Impractically expensive
- Other formats?
  - CSF – N(N-1)!
  - HiCOO – hyper-sparsity limits the impact of blocking size (unless you permute the rows)
- Linearization maps naturally to traversal
  - Different bit interleaving maps to different linearized traversal

ReLATE – Reinforcement Learned Adaptive Tensor Encoding

- Deep Reinforcement Learning (DRL)-based framework for deriving a linearized encoding for sparse tensors
- Key features
  - **Client-server model**
  - Double DQN
  - Reward cache
  - Prioritized Replay Buffer
  - **Hybrid model-free and model-based learning**
  - Reward shaping mechanism

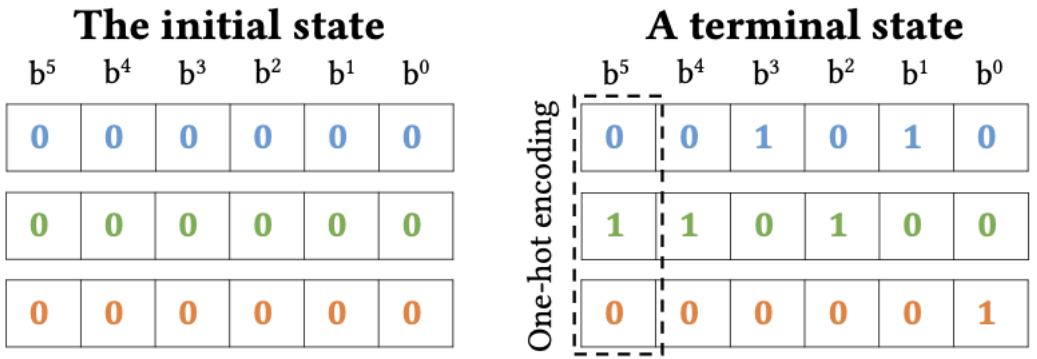


Problem Formulation

- Bit interleaving
  - Total number of ways for I^N space = $(N \log_2(I))!$
- Bit from the index tuple -> a bit location in the linearized index
  - Sequential decision problem -> MDP -> **reinforcement learning**
- Still very expensive
  - 256 x 256 x 256 x 256 tensor -> $(4 \times 8)! = 2.63 \times 10^{35}$
  - How do we **reduce the search space**?

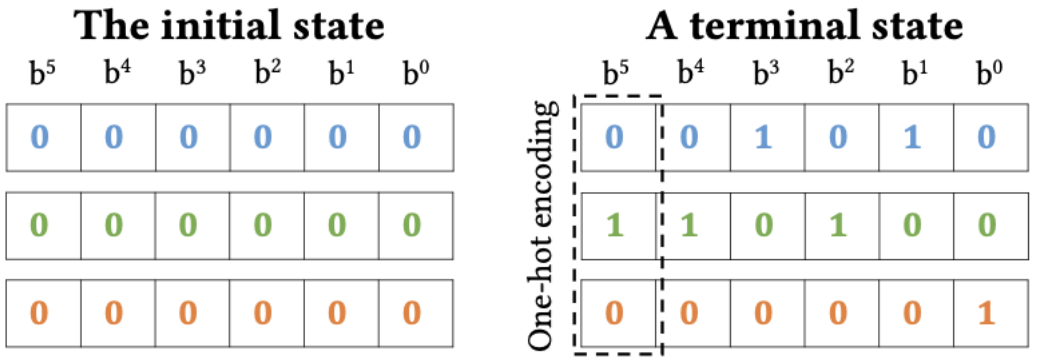
Encoding

- Represented as a matrix with N rows and (N $\log_2 I$) columns
  - For example, for a (4 x 8 x 2) tensor, that would be 3 rows and (2 + 3 + 1 = 6) columns
- At each state at step t, we decide which bit from the index tuple will be selected for the bit position t in the linear index
- In each column only one element can be 1 and everything else will be 0 (one-hot encoding),
  - One row can be used to represent the encoding more compactly, but implies ordinal relationship across modes -> poor learning performance



Encoding

- We limit the encoding so that bits from a given mode are assigned to the linearized index from most-significant to least-significant , which reduces the total search space from (N $\log_2$I) -> (N $\log_2$I) / (($\log_2$1)! ($\log_{22}$)! … ($\log_2$N)!)
- We can restrict how the encoding bits are selected to ensure a correct encoding is always chosen (e.g., # of bits in row n == $\log_2 I_n$)



Reward calculation

- Client-server model
  - Client – trains the model
  - Server – evaluates the learned encoding by executing MTTKRP and measuring its performance
  - This allows us to leverage idle data center capacity and evaluate rewards in parallel to improve learning efficiency
  - Minimize noise/jitter
- Using raw execution time -> exploding/vanishing error gradients
  - Using speedup (over ALTO) improved stability, same hyperparameter for different input tensors

Learning Algorithm

- Reward evaluation is expensive
  - Large number of non-zero elements, large and skewed dimensions, large number of modes
  - For example, Reddit takes 10+ seconds to execute 1 MTTKRP
- We use a hybrid model-free (double DQN) and model-based DRL

Learning Algorithm

- Reward evaluation is expensive
  - Large number of non-zero elements, large and skewed dimensions, large number of modes
  - For example, Reddit takes 10+ seconds to execute 1 MTTKRP
- We use a hybrid model-free (double DQN) and model-based DRL
  - Double DQN improves stability by
    - decoupling action selection (online network, θ) and action evaluation (target network, θ^-)
    - using a prioritized replay buffer to store the latest significant transitions observed by the agent, and sampling mini-batches from the buffer
    - θ and θ^- are adaptive CNN – number of hidden units increases with state-action space and CNN encodes the hierarchical spatial information of the target environment

Learning Algorithm

- Reward evaluation is expensive
  - Large number of non-zero elements, large and skewed dimensions, large number of modes
  - For example, Reddit takes 10+ seconds to execute 1 MTTKRP
- We use both a model-free (double DQN) and model-based DRL
  - During early stages of exploration, we have high ϵ to select more random actions that are evaluated on the target environment (i.e., we physically run MTTKRP)
  - During this exploration stage, we also **train a fully connected network** to form a simple reward model
  - Once this model becomes accurate enough (> 90%), when we find an action with a high reward (no worse than the highest observed reward +- error), it is evaluated and used to further update the model
  - This allows us to skip the expensive evaluation step – expensive during the early stage of training, but becomes more efficient during the later stage

Learning Algorithm

- Reward evaluation is expensive
  - Large number of non-zero elements, large and skewed dimensions, large number of modes
  - For example, Reddit takes 10+ seconds to execute 1 MTTKRP
- We use both a model-free (double DQN) and model-based DRL
- We also use a reward cache to avoid evaluating encodings we have already seen

Learning Algorithm

- Reward evaluation is expensive
  - Large number of non-zero elements, large and skewed dimensions, large number of modes
  - For example, Reddit takes 10+ seconds to execute 1 MTTKRP
- We use both a model-free (double DQN) and model-based DRL
- We also use a reward cache to avoid evaluating encodings we have already seen

Learning Algorithm

- Delayed reward
  - Reward is calculated at the very end (when we have a real encoding), which results in sparse/delayed reward -> instability and slower learning
  - We introduce a reward shaping mechanism that allocates credit from the reward to all actions leading up to the final encoding
    - Uniform credit distribution showed the best result

Evaluation

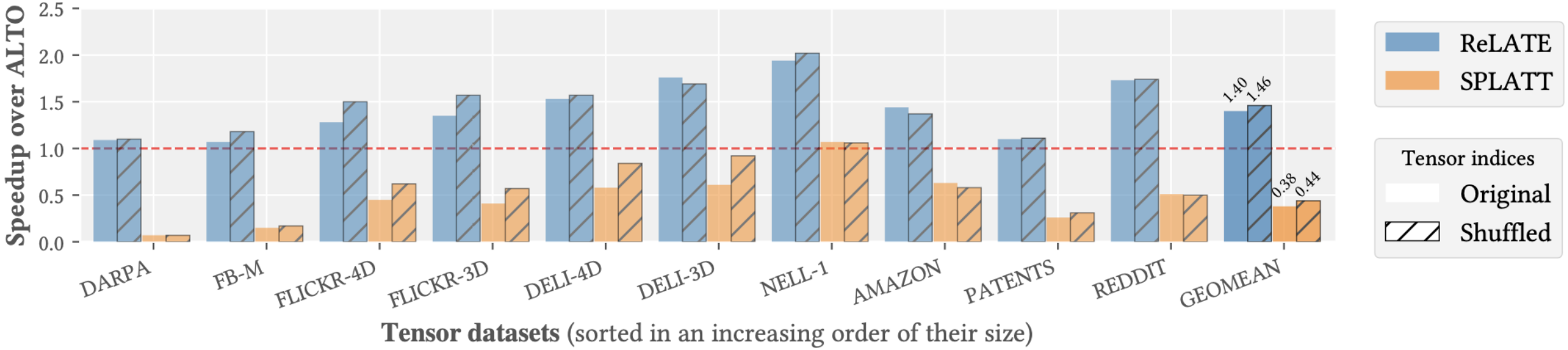


Figure 5: The speedup of our DRL-based sparse encoding (ReLATE) compared to ALTO and SPLATT on a 128-core EMR system.

Evaluation

- Speedup over ALTO (generally) improves with tensor size and sparsity
  - This suggests that DRL is effective compared to input-agnostic (human) expert-designed formats – higher improvements for more difficult problems
- Shuffling improves CSF as well
  - Improves workload balance

Learning Effectiveness

- Error = |estimated – measured|/ measured
- Averaged over 10 samples model had not seen during training
- Reward Model
  - Highly accurate
  - Learned quickly (< 100 ~ 500) episodes

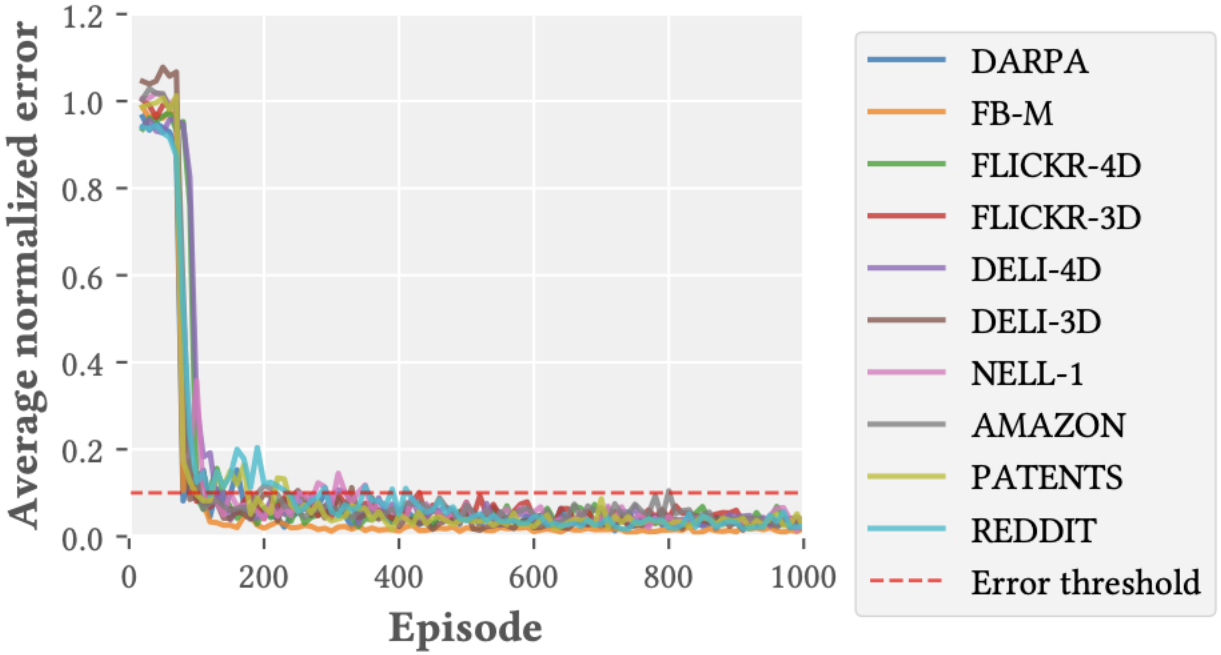


Figure 8: Average estimation error of ReLATE's reward model, relative to the actual reward, over the first thousand episodes.

Learning Effectiveness

- Error = |estimated – measured|/ measured
- Averaged over 10 samples model had not seen during training
- Reward Model
  - Highly accurate
  - Learned quickly (< 100 ~ 500) episodes
- Large % of rewards are modeled for many tensors
  - On average 34% are real

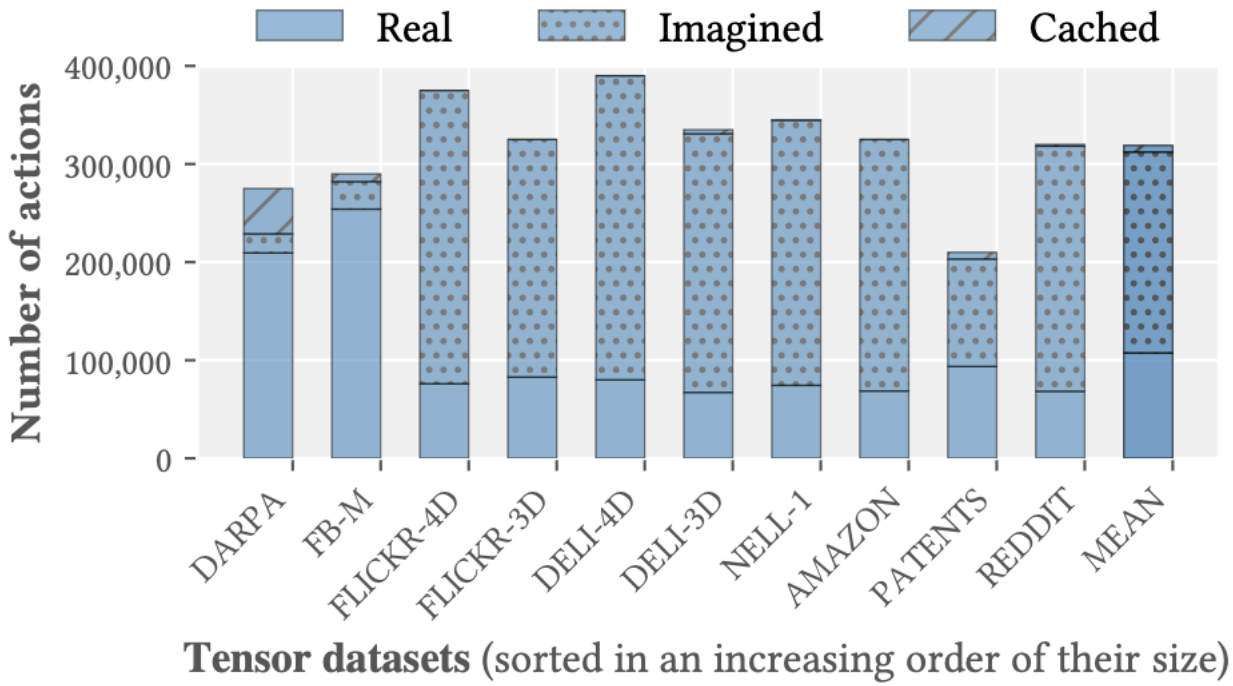


Figure 9: The number of real, cached, and imagined actions taken by the ReLATE agent across sparse tensors.

Practical?

- Yes and no
- Yes
  - In practical data analysis using tensor decomposition, you run a) on several randomly initialized factors, b) multiple factor ranks, c) different data processing methods (to handle noise, outliers, etc.) -> tens to hundreds of thousands of MTTKRP
  - Amortizable for real-world data analysis applications
- No
  - We need to train for each new input tensor (cumbersome, unless seamlessly integrated into the workflow)
  - Current speedup (1.4x geometric mean) may not seem “worth the trouble.”

Human vs. AI. Who Wins?

- In the context of sparse tensor formats…
  - We need both to maximize performance
  - AI is a useful tool, but cannot (yet) create something beyond what we've already thought of
- However, AI models are involving quickly, so who knows what'll happen in a few years?

Questions?