UNIVERSITY OF OREGON

Linearized Tensor Format for Performance-Portable Sparse Tensor Computation

Andy Nguyen[1], **Ahmed E. Helal**[2], Fabio Checconi[2], Jan Laukemann[2], Jesmin Jahan Tithi[2], Yongseok Soh[1], Teresa Ranadive[3], Fabrizio Petrini[2], Tammy Kolda[4], Jee Whan Choi[1]

[1] University of Oregon [2] Intel Labs [3] Laboratory for Physical Sciences [4] Sandia National Laboratory

Motivation

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

1. reduce tensor storage 

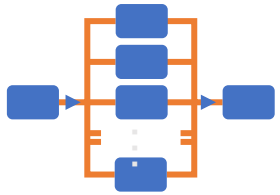
Motivation

- 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 

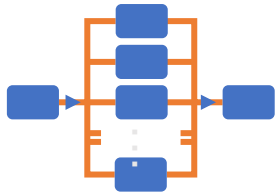
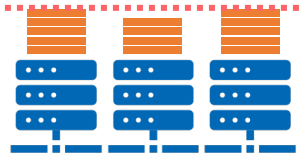
Motivation

- 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 

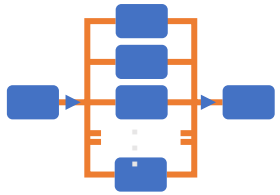
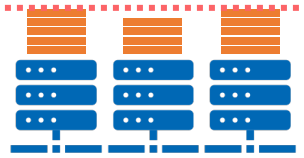
Motivation

- 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

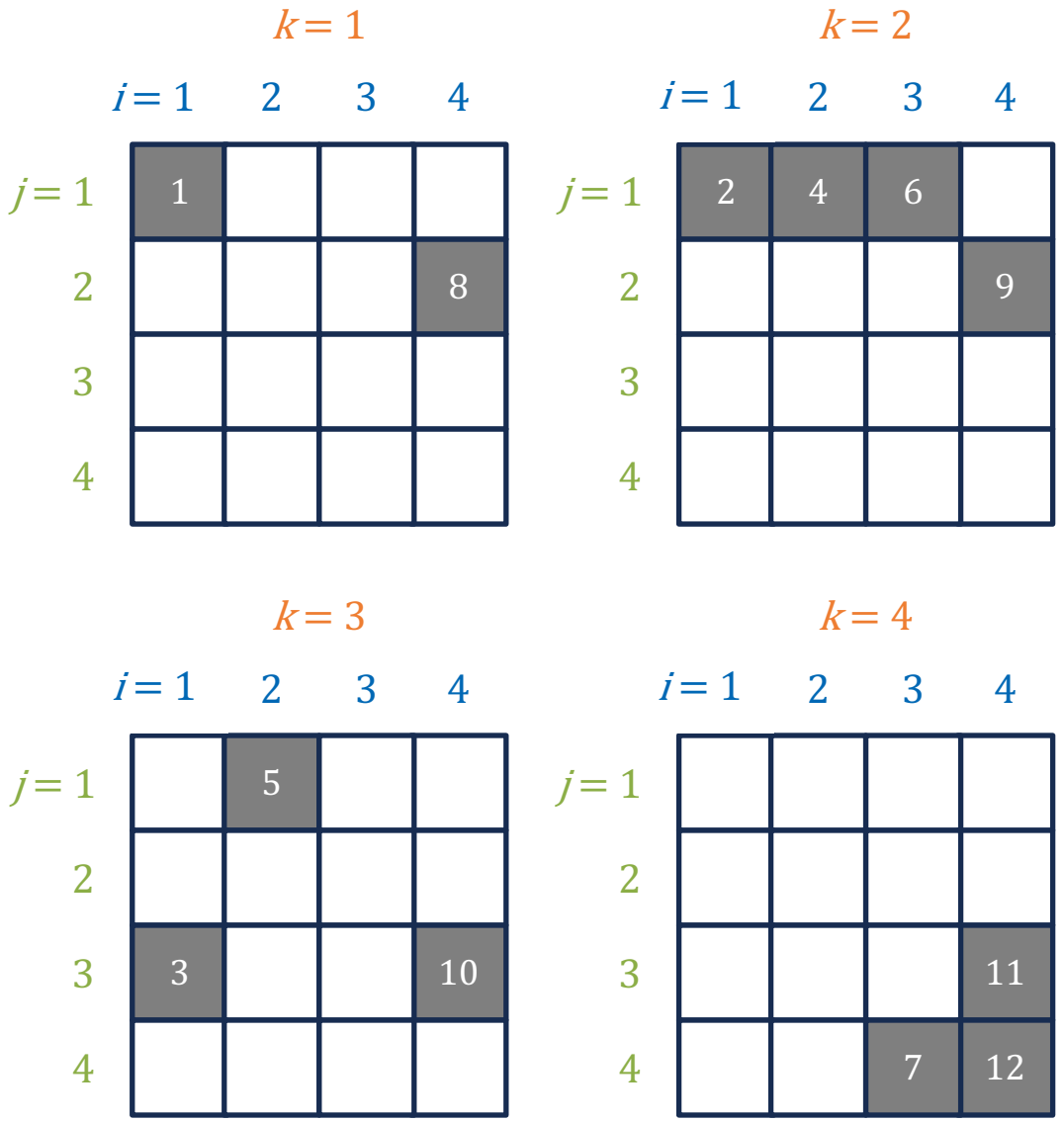
- Can be classified based on their encoding of the indexing metadata into:

1. List-based formats
2. Tree-based formats
3. Block-based formats

State-of-the-art

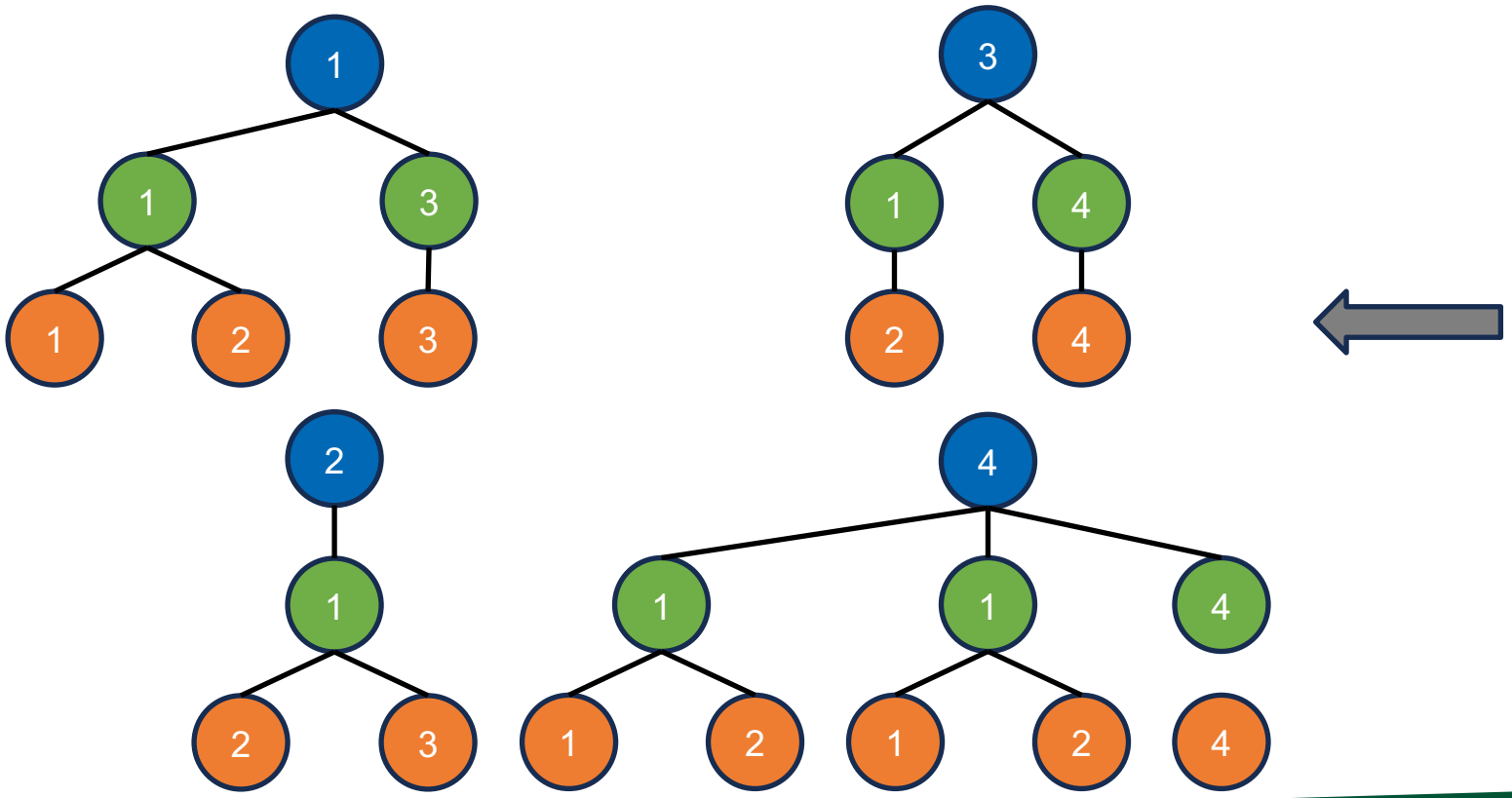
- List-based format: COO (coordinate)

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)



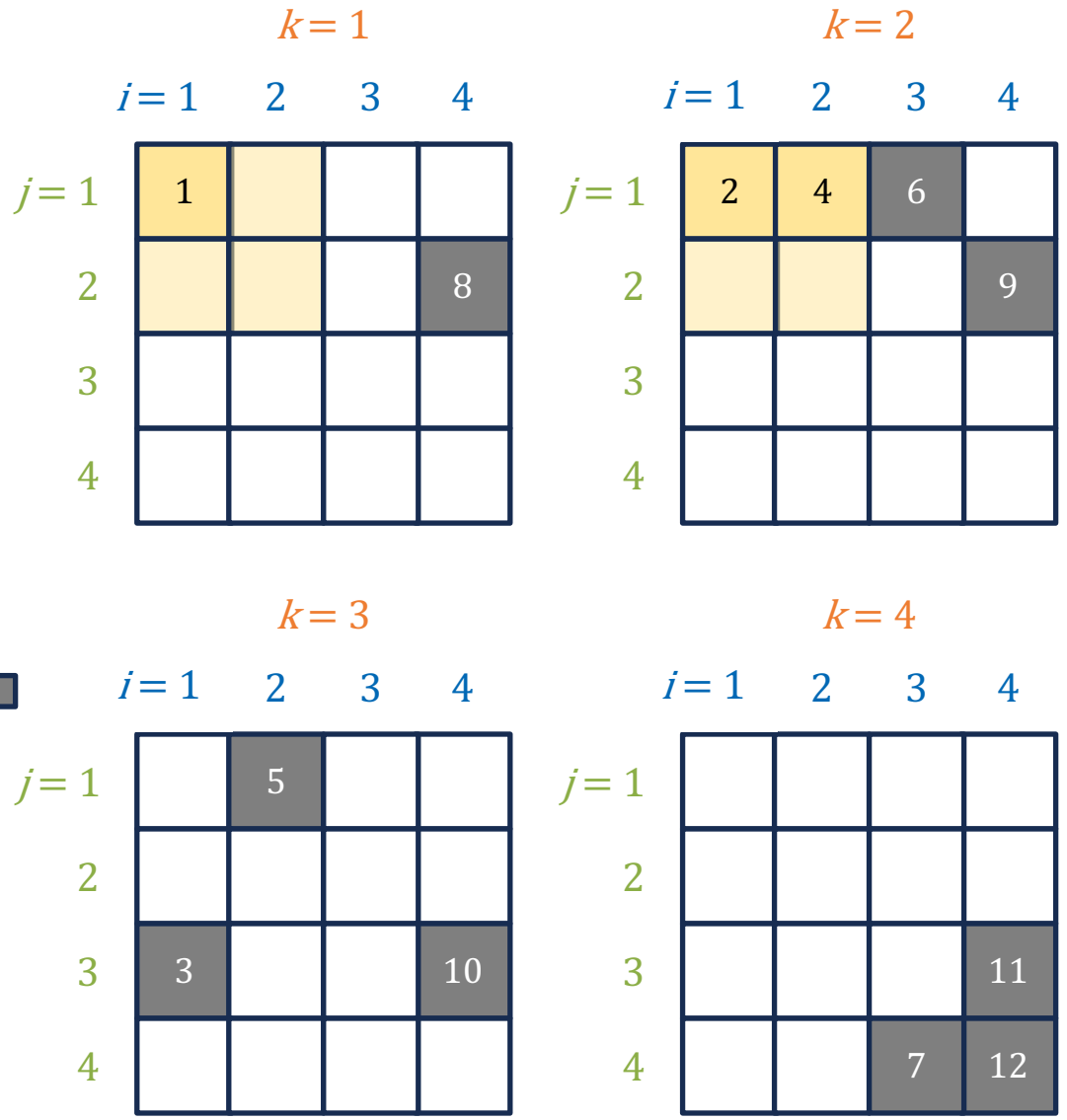
i	j	k	v
1	1	1	1
1	1	2	2
1	3	3	3
2	1	2	4
2	1	3	5
3	1	2	6
3	4	4	7
4	2	1	8
4	2	2	9
4	3	3	10
4	3	4	11
4	4	4	12

State-of-the-art

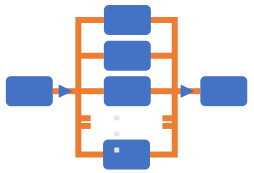
- Block-based format: HiCOO (hierarchical COO)

2x2x2 block

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



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

- 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} [1]

[1] Smith et al. 2017. FROSTT: The Formidable Repository of Open Sparse Tensors and Tools. http://frostt.io/

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} [1]

- *Our hypothesis – simple mode-agnostic, list-based formats are the best for sparse tensors*
  - *How do we improve data locality and reduce conflicts?*
  - *How do we make it simple and performance-portable?*

[1] Smith et al. 2017. FROSTT: The Formidable Repository of Open Sparse Tensors and Tools. http://frostt.io/

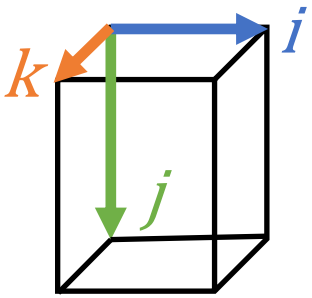
Linearized Formats

- ALTO (Adaptive Linearized Tensor Order) for CPUs
- BLCO (Block Linearized Coordinate) for GPUs
- Application of linearized formats to
  - *streaming* tensor decomposition [1]
  - *on-the-fly* Khatri-Rao product for CP-APR (WIP)
  - non-negative sparse tensor factorization for GPUs via PLANC (WIP)

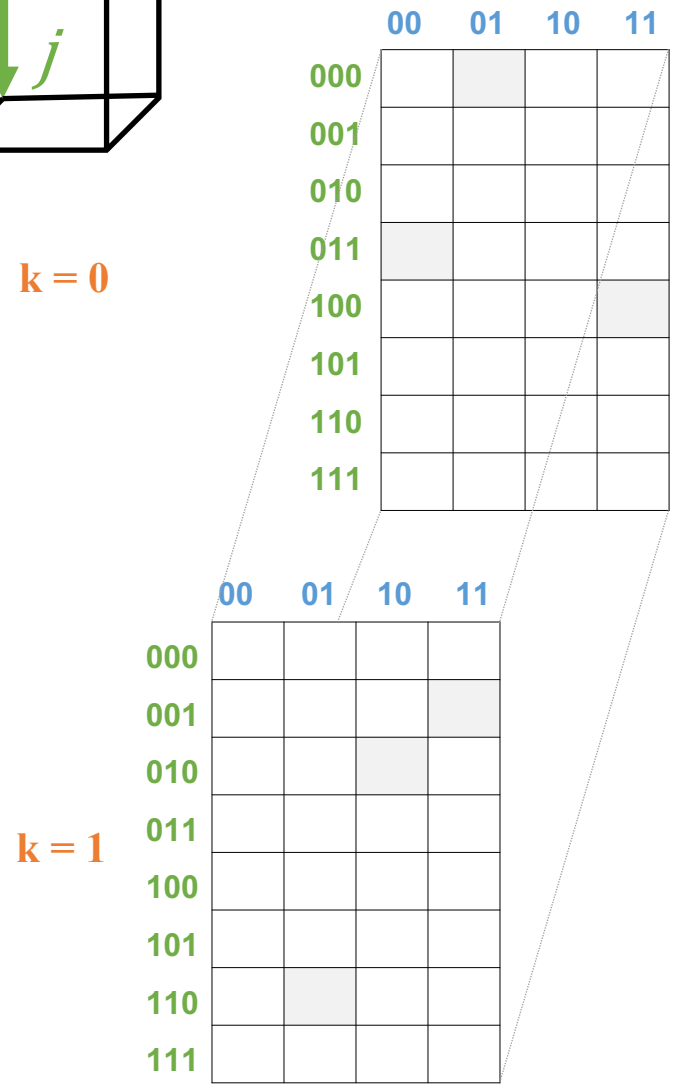
[1] Soh et al. Dynamic Tensor Linearization and Time Slicing for Efficient Factorization of Infinite Data Streams. IPDPS'23

ALTO

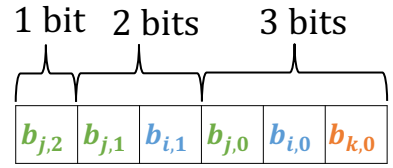
- ALTO interleaves the index bits by grouping them by their positions
- Within each group, the bits are arranged from the longest mode (most-significant) to the shortest (least-significant)



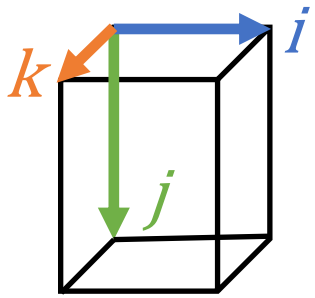
4x8x2 tensor



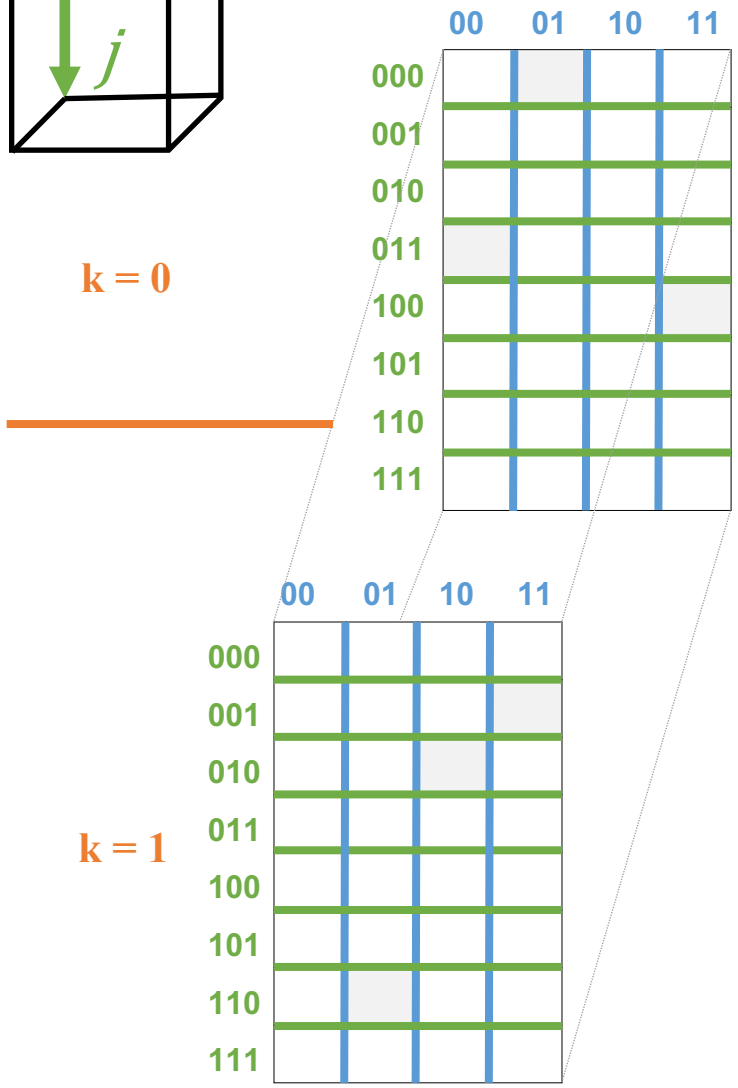
ALTO bit mask



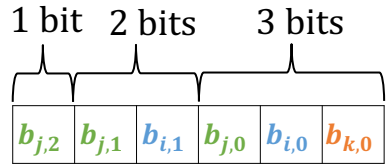
ALTO

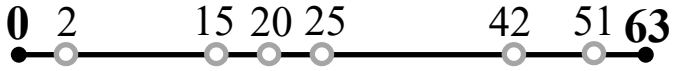


4x8x2 tensor



ALTO bit Mask





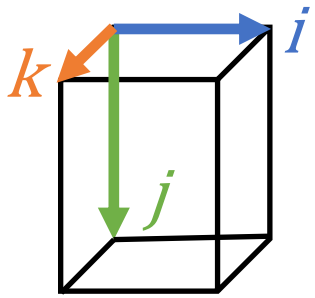
ALTO

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)

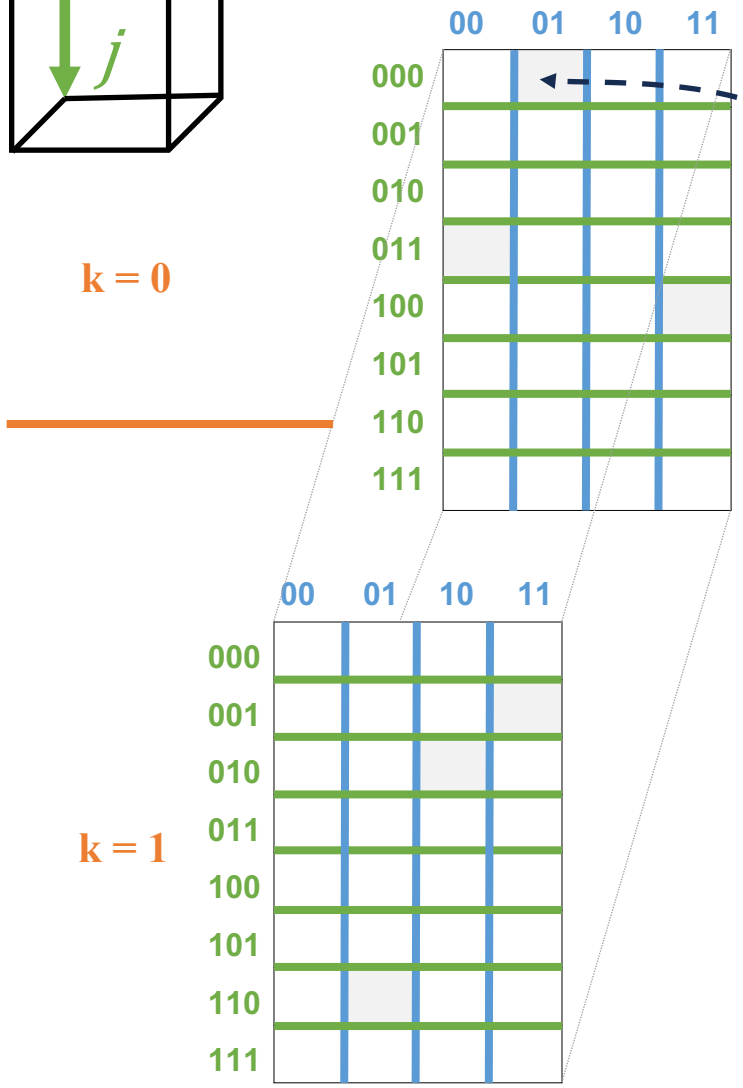
ALTO

ALTO

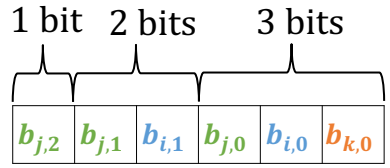
Value	Position
$x_{1,0,0}$	2 (000010)
$x_{3,1,1}$	15 (001111)
$x_{0,3,0}$	20 (010100)
$x_{2,2,1}$	25 (011001)
$x_{3,4,0}$	42 (101010)
$x_{1,6,1}$	51 (110011)

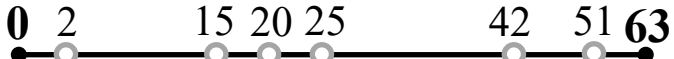


4x8x2 tensor

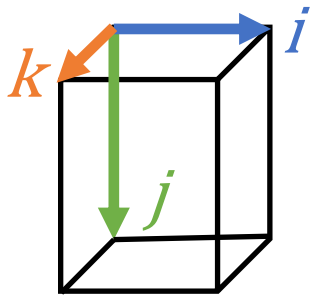


ALTO bit Mask

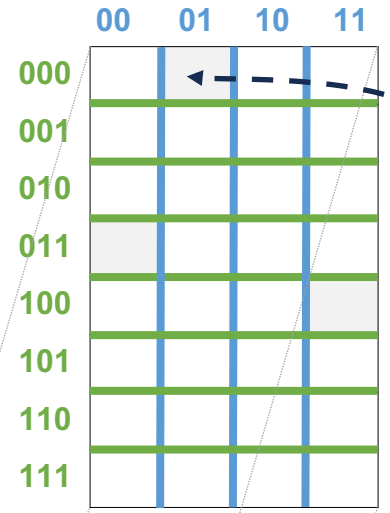




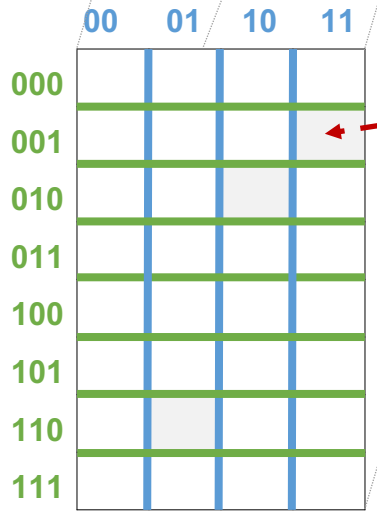
ALTO



4x8x2 tensor

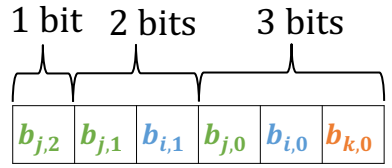


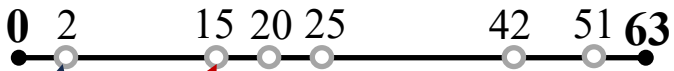
k = 0



k = 1

ALTO bit mask

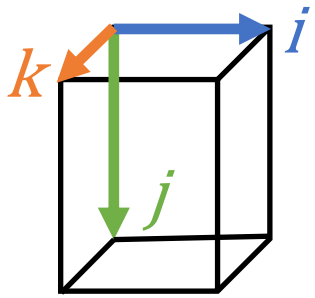




ALTO

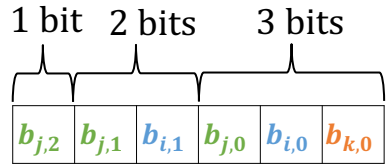
Value	Position
$x_{1,0,0}$	2 (000010)
$x_{3,1,1}$	15 (001111)
$x_{0,3,0}$	20 (010100)
$x_{2,2,1}$	25 (011001)
$x_{3,4,0}$	42 (101010)
$x_{1,6,1}$	51 (110011)

ALTO



4x8x2 tensor

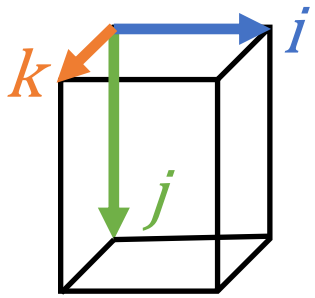
ALTO bit mask



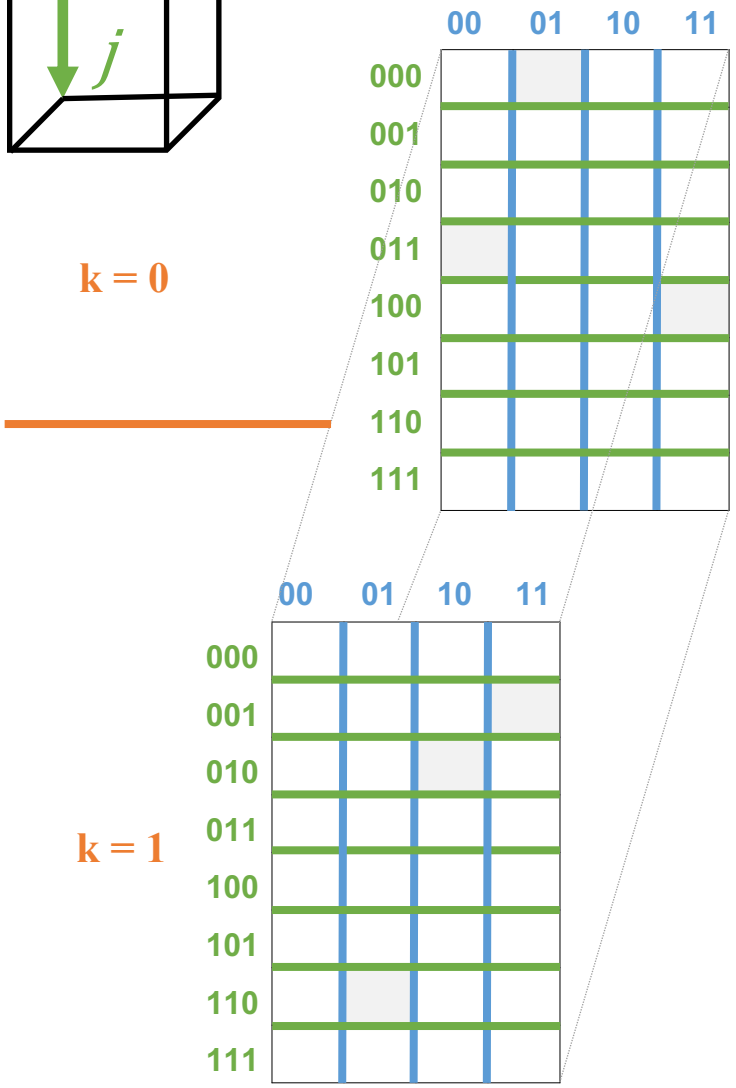
ALTO

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)

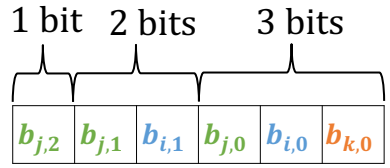
Partitioning



4x8x2 tensor

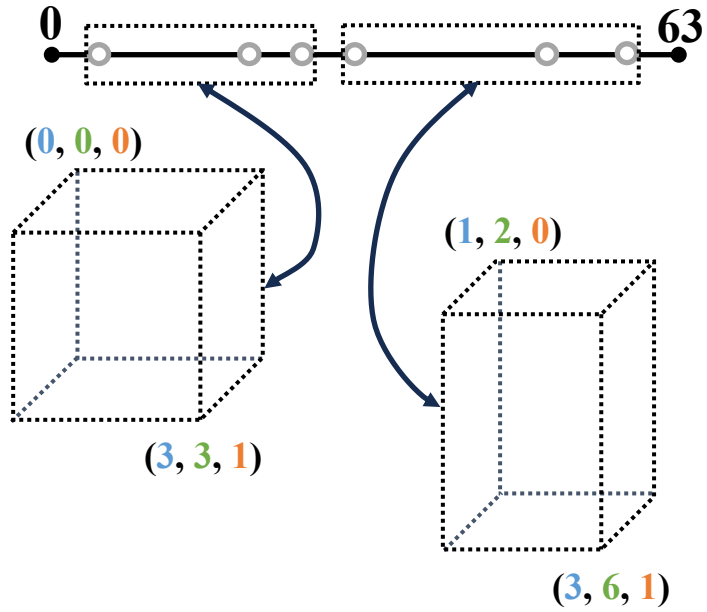


ALTO bit mask

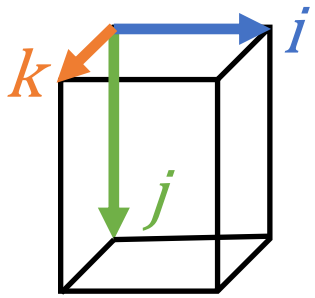


ALTO

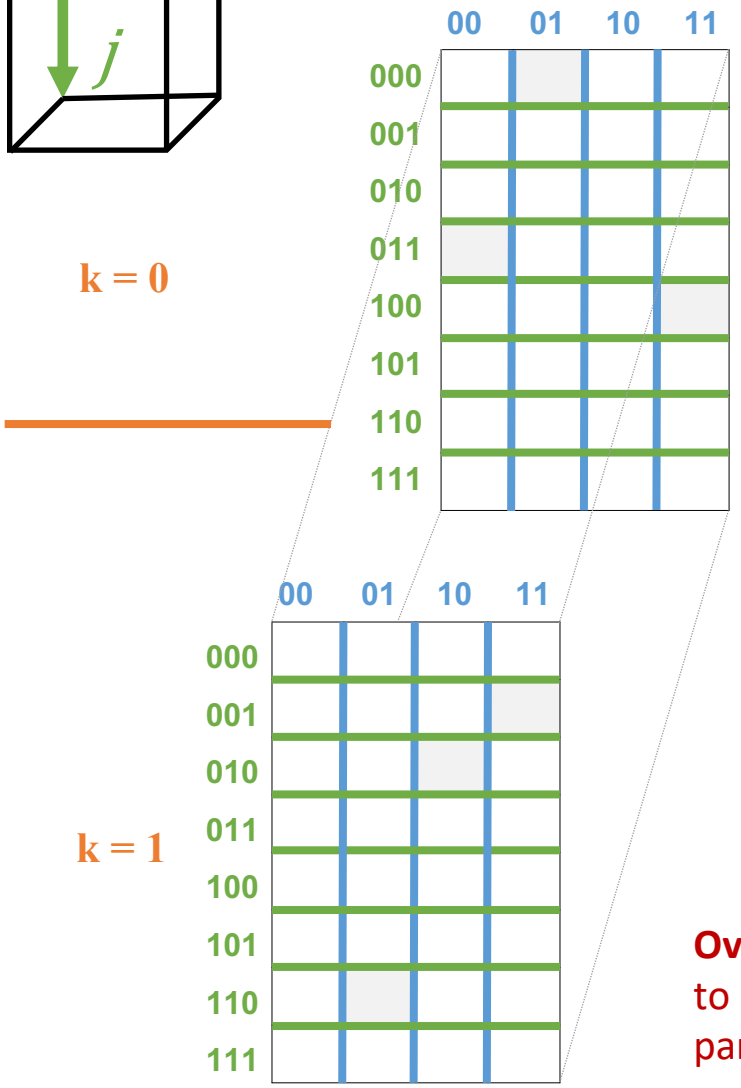
Value	Position
$x_{1,0,0}$	2 (000010)
$x_{3,1,1}$	15 (001111)
$x_{0,3,0}$	20 (010100)
$x_{2,2,1}$	25 (011001)
$x_{3,4,0}$	42 (101010)
$x_{1,6,1}$	51 (110011)



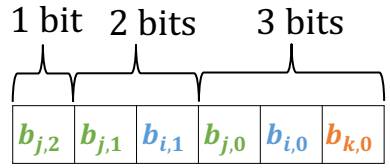
Partitioning



4x8x2 tensor

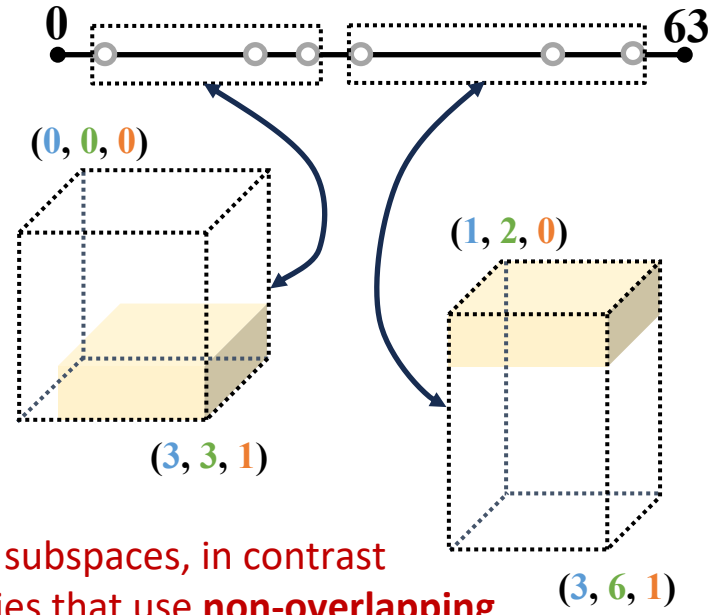


ALTO bit mask



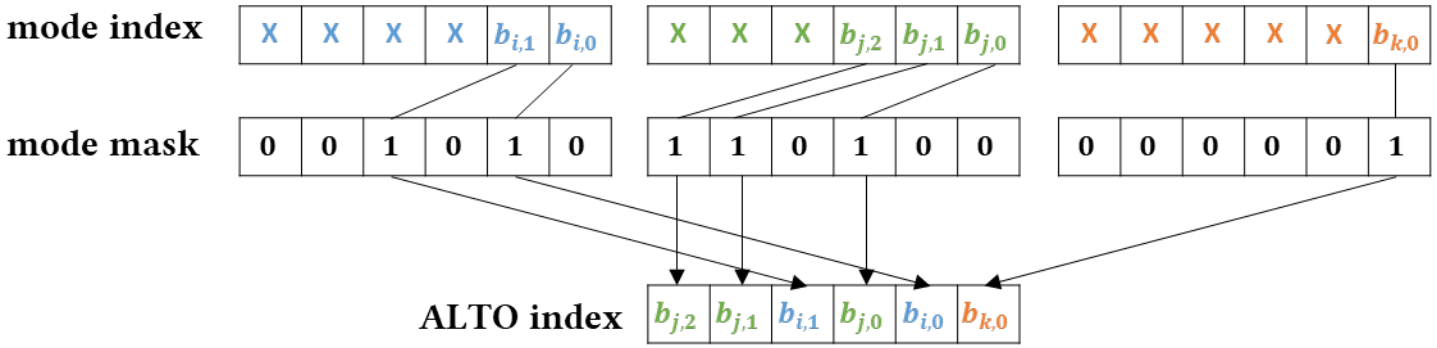
ALTO

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)

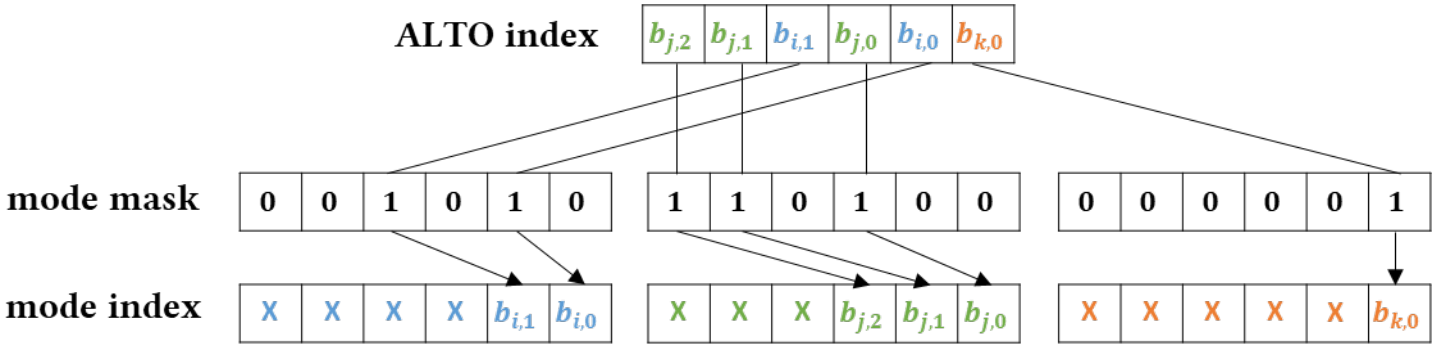


Overlapping subspaces, in contrast to prior studies that use **non-overlapping** partitions, that often lead to **imbalance**.

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          <----------  ALTO line segment
        for all x ∈ 𝒳_l do
                fetch i_x,j_x,k_x,v_x      ------>  fetch a_x (ALTO index),v_x
                fetch v_1 = A(2)(j_x,:)            i_x,j_x,k_x = decode(a_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)
```

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_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`

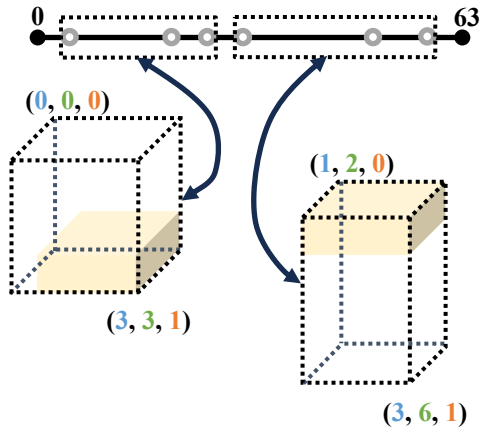
Matricized Tensor Times Khatri-Rao Product (MTTKRP)

```
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)
```

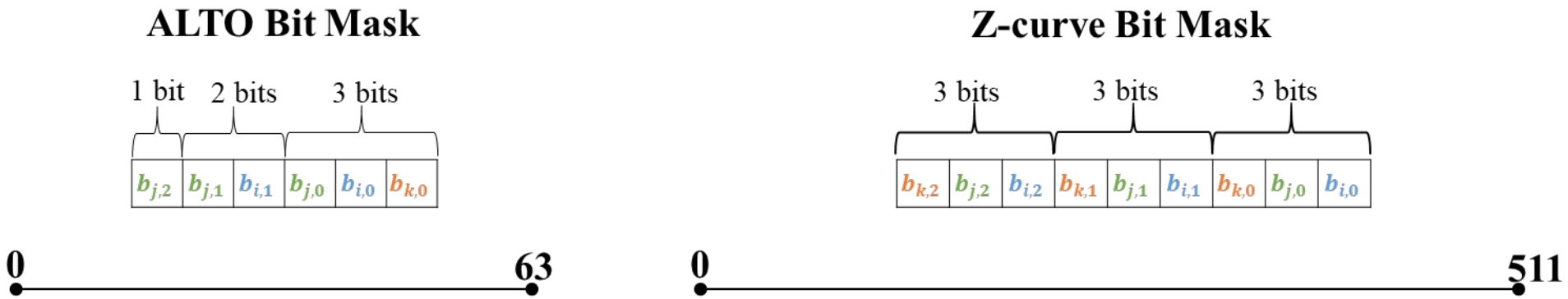
This can be done efficiently using parallel bit extract/deposit (`pext`/`pdep`) instructions on x86 CPUs



Adaptive update mechanism

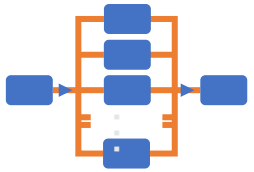
- If rows (i_x) have limited reuse → update using atomic ops.
- If rows (i_x) have large reuse → use scratchpad to combine local updates and parallel reduction to merge globally
- `reuse = nnz / mode_length`
- Global reduction uses the boundary/overlap information to "pull" rows (i_x) from the appropriate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

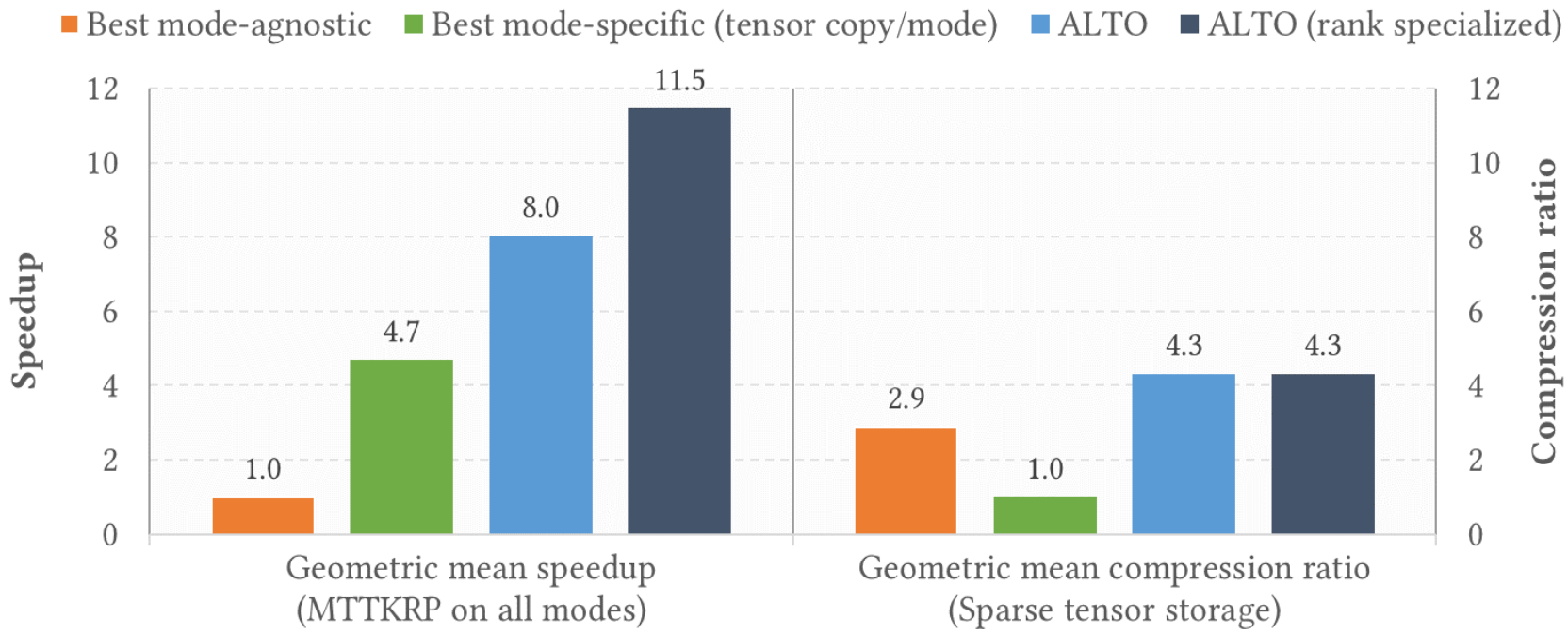
ALTO vs. State-of-the-art


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

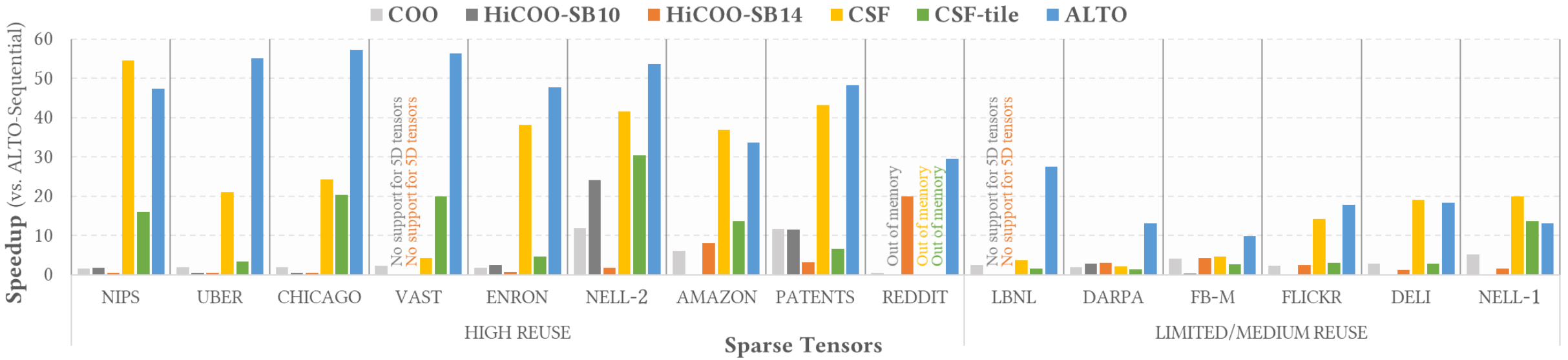
Performance Summary

- 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



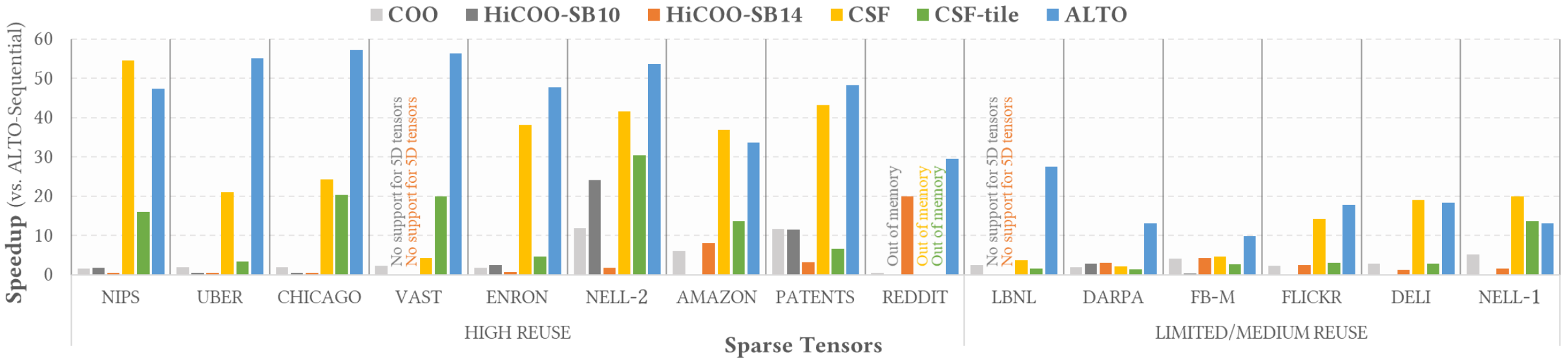
Speedup

- Speedup against serial MTTKRP ALTO implementation
- HiCOO does not support 5D tensors
- HiCOO and CSF runs out of memory for REDDIT



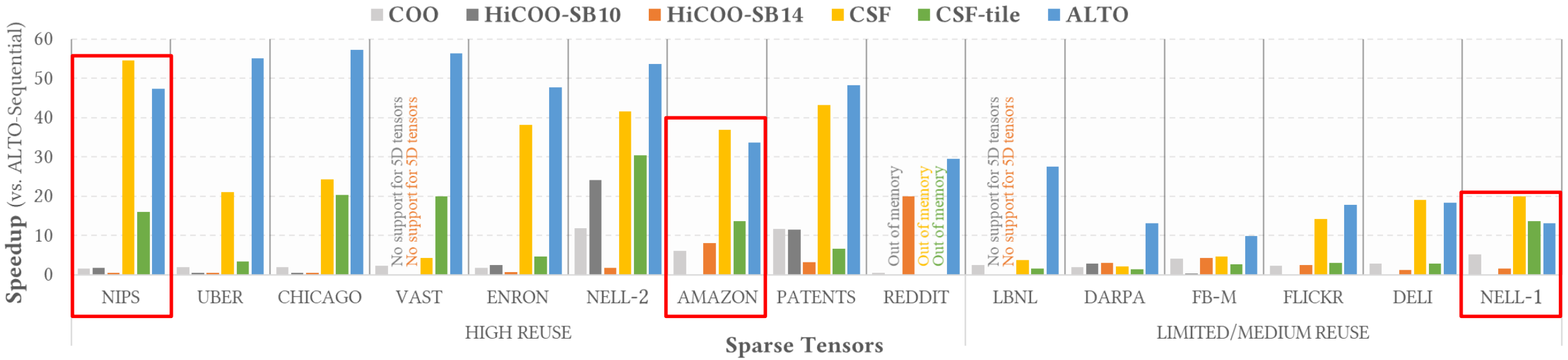
Speedup

- ALTO achieves ~80% of realizable speedup across tensors with different characteristics
- For high reuse, ALTO achieves 47x geo-mean speedup
- For limited reuse, ALTO achieves 16x geo-mean speedup



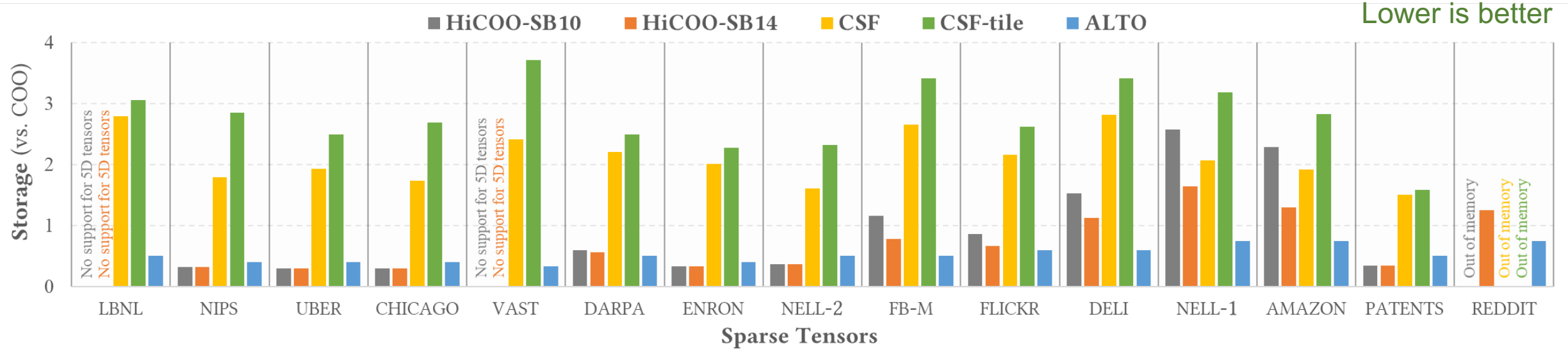
Speedup

- CSF shows slightly better performance for NIPS, AMAZON, and NELL-1 (geo-mean speedup of 1.2x over ALTO) by keeping N copies of the tensor, optimized for each mode
- Performance of COO, HiCOO, and CSF are sensitive to irregular tensor shape and data distribution



Storage

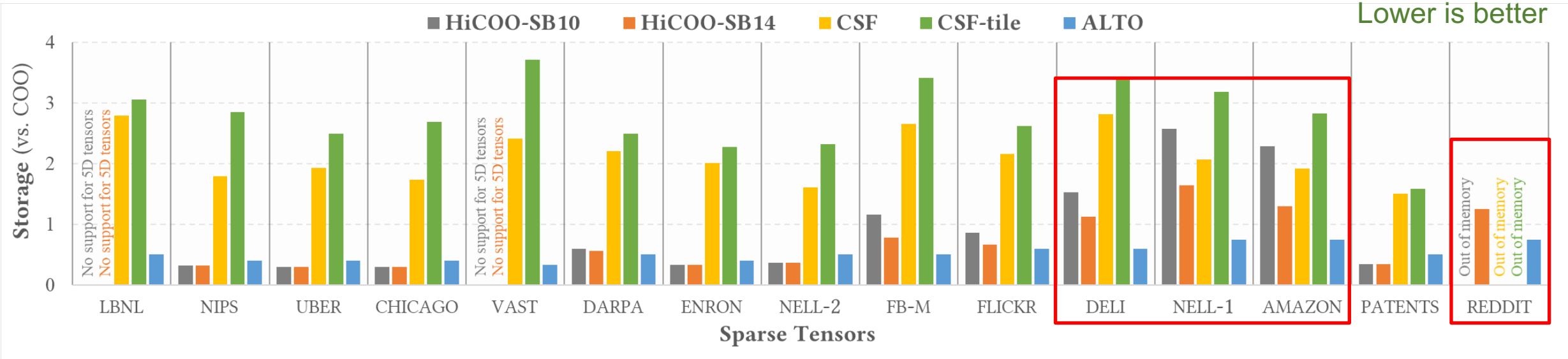
- ALTO always requires less storage than COO due to linearization



Helal et al. ALTO: Adaptive Linearized Storage of Sparse Tensors. ICS'21

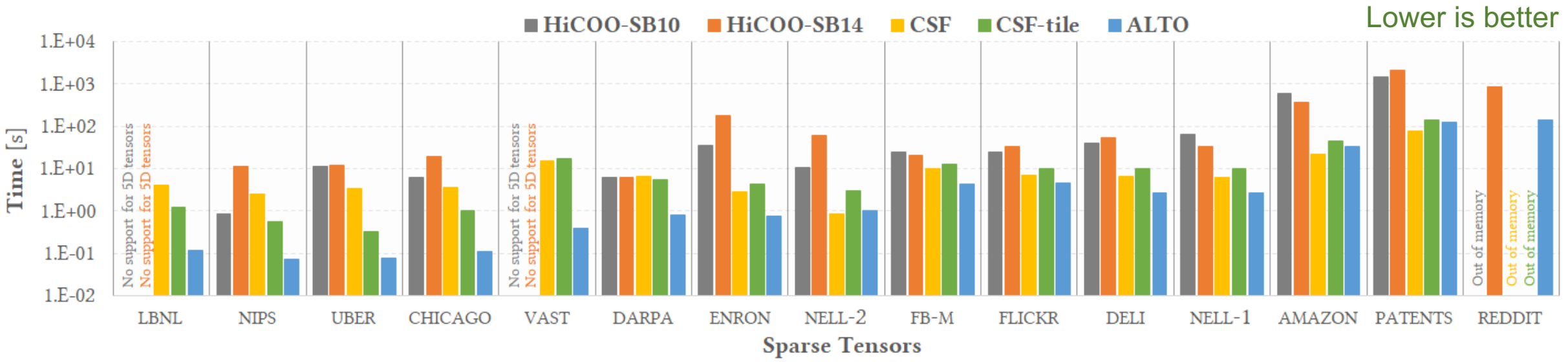
Storage

- ALTO always requires less storage than COO due to linearization
- HiCOO storage depends on the block/superblock sizes and spatial distribution of nonzero elements
  - For hyper-sparse tensors, HiCOO consumes more storage than COO



ALTO Generation Overhead

- ALTO substantially decreases sorting time by reducing the # of comparison operations
- Block-based formats require expensive clustering (and sometimes reordering) and scheduling of non-zero elements
- CSF is generated from **pre-sorted** tensors, but are still slower on average to construct than ALTO



Helal et al. ALTO: Adaptive Linearized Storage of Sparse Tensors. ICS'21

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
```

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

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

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

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

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

Adaptive update mechanism

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

Large number of threads → synchronization is more expensive

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

GPUs have limited on-chip memory

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

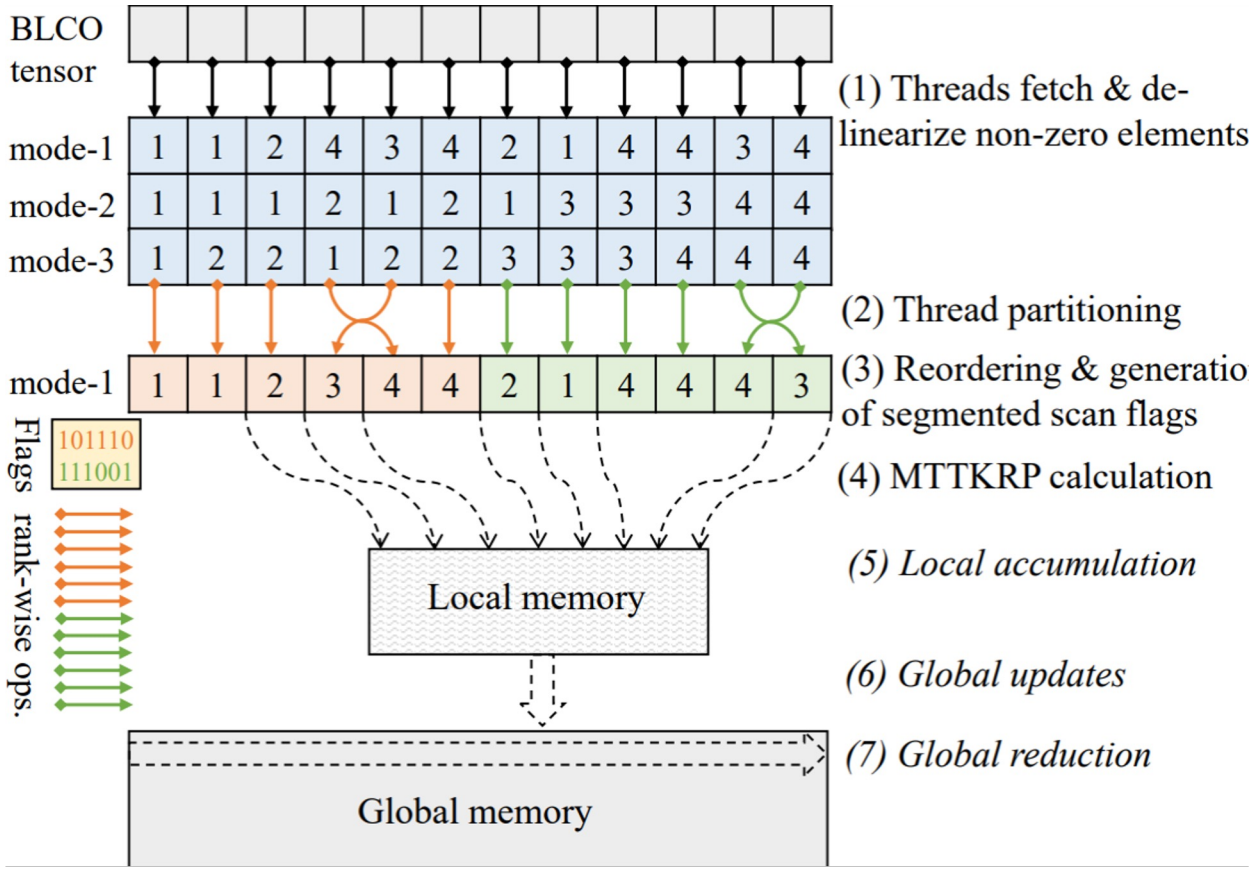
ALTO

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

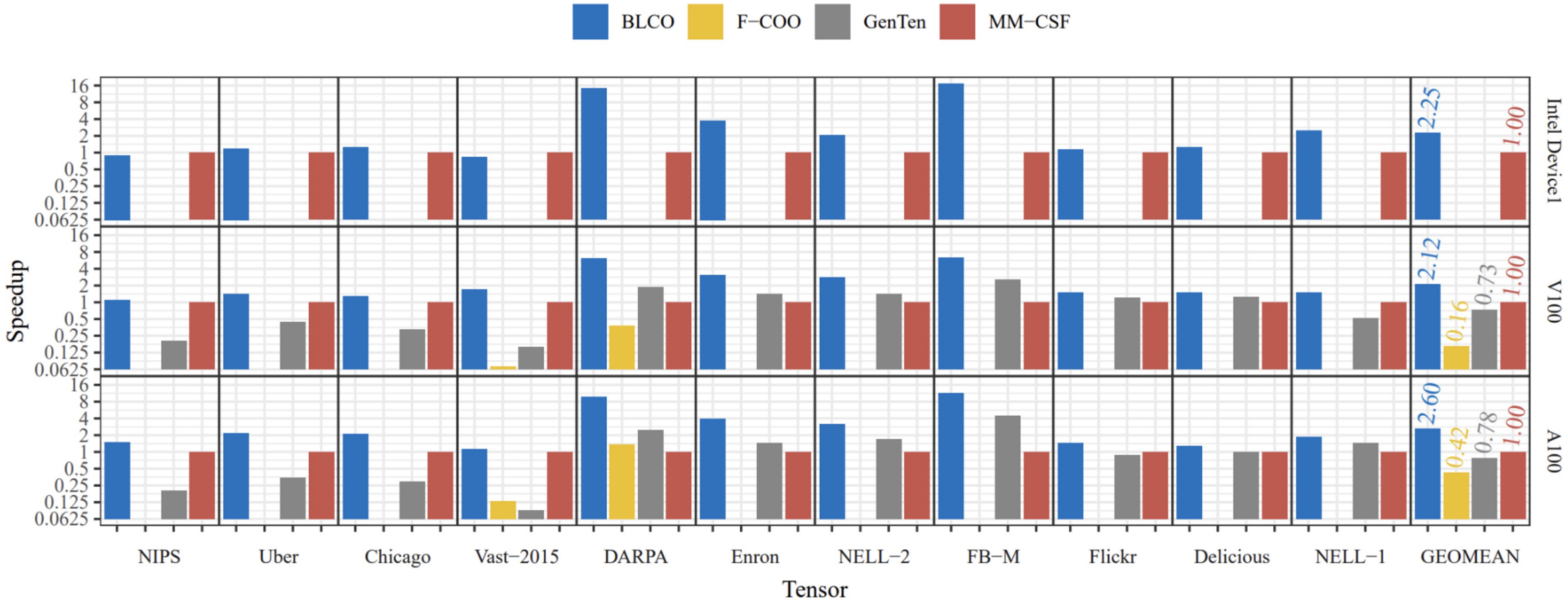
b	l		v
0	0	$(00000)_2$	1.0
	16	$(10000)_2$	2.0
	17	$(10001)_2$	4.0
	6	$(00110)_2$	8.0
	18	$(10010)_2$	6.0
	23	$(10111)_2$	9.0
1	1	$(00001)_2$	5.0
	8	$(01000)_2$	3.0
	11	$(01011)_2$	10.0
	27	$(11011)_2$	11.0
	30	$(11110)_2$	7.0
	31	$(11111)_2$	12.0

Synchronization



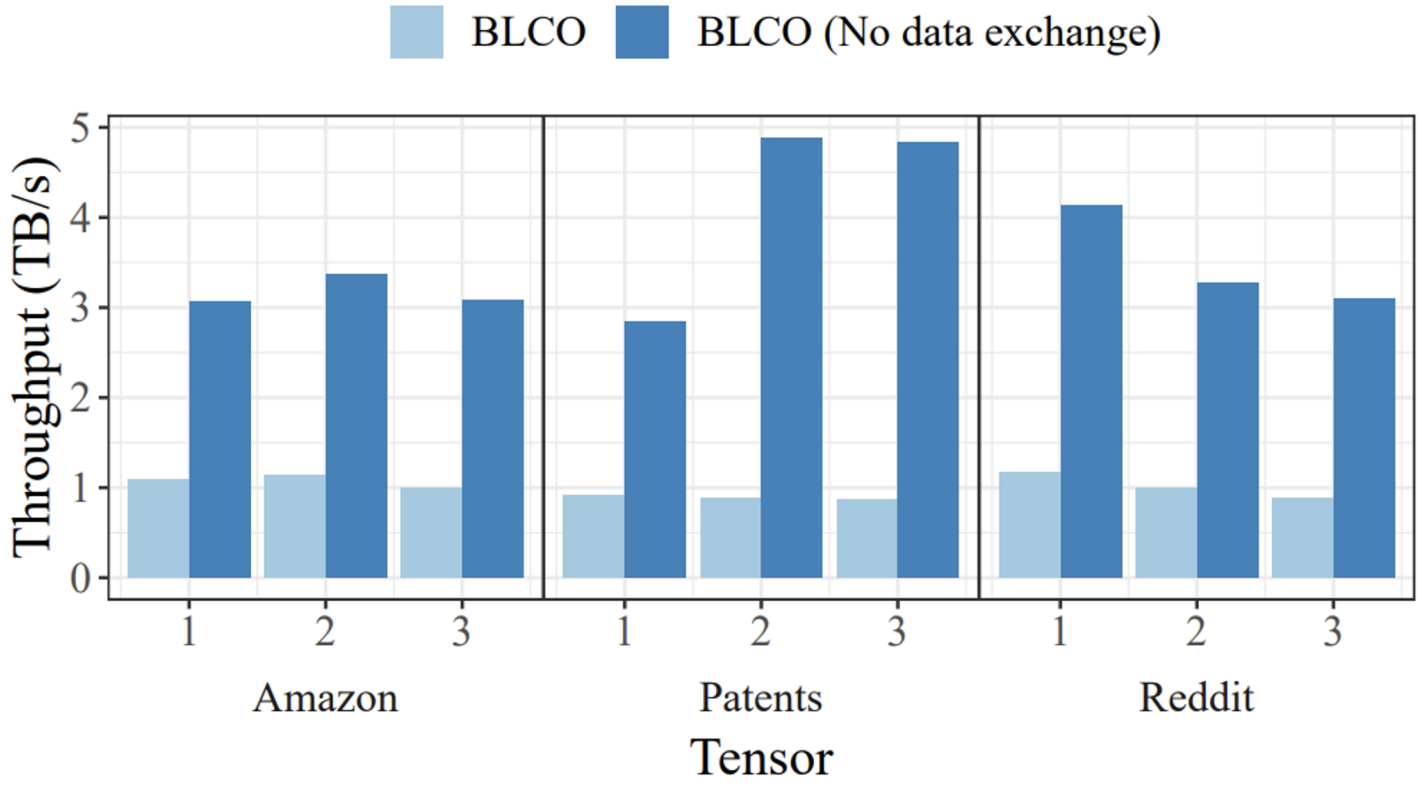
Performance

- 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



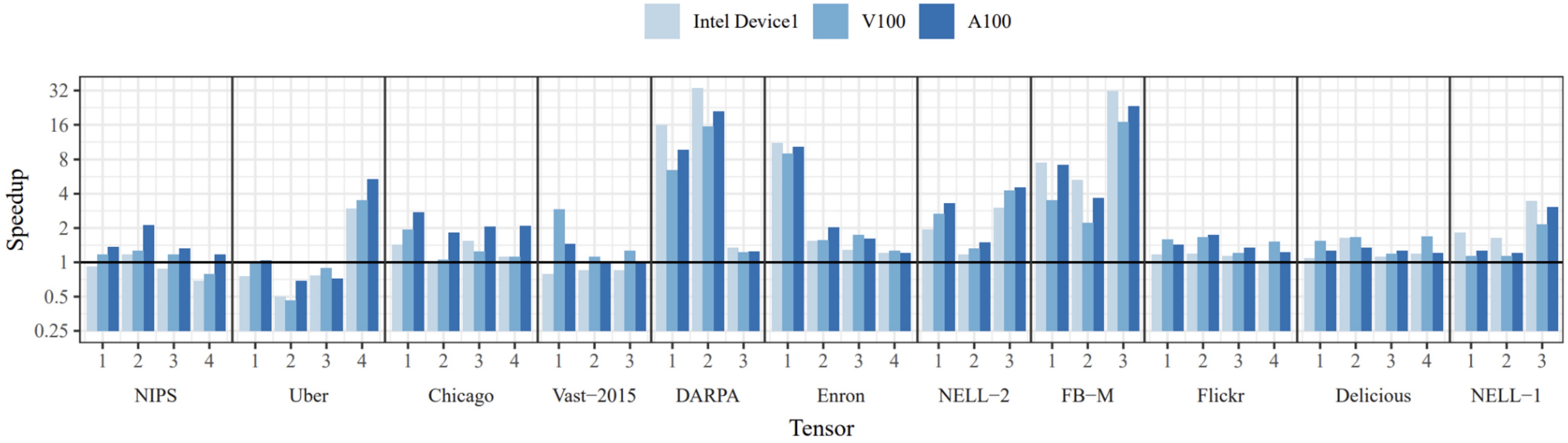
Performance

- Throughput limited by host-to-device data transfer
- In-memory performance on par with other tensors (measured via Nsight System Profiler)

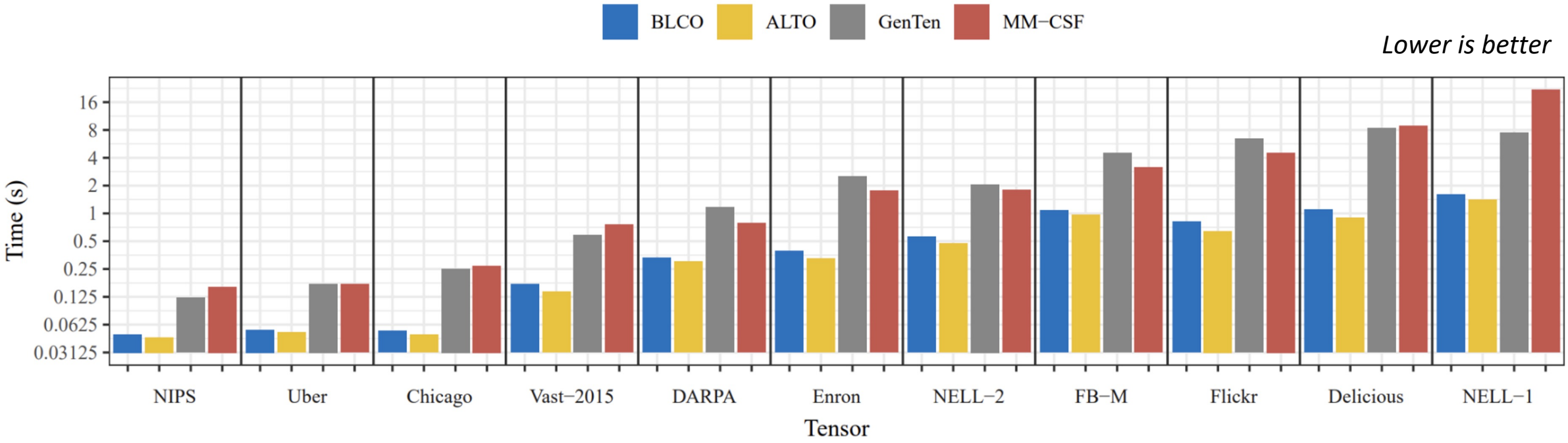


Performance

- MM-CSF is sensitive to data distribution, resulting in significant variation in performance across different modes (e.g., `Uber`, `DARPA`, `Enron`, and `FB-M`)

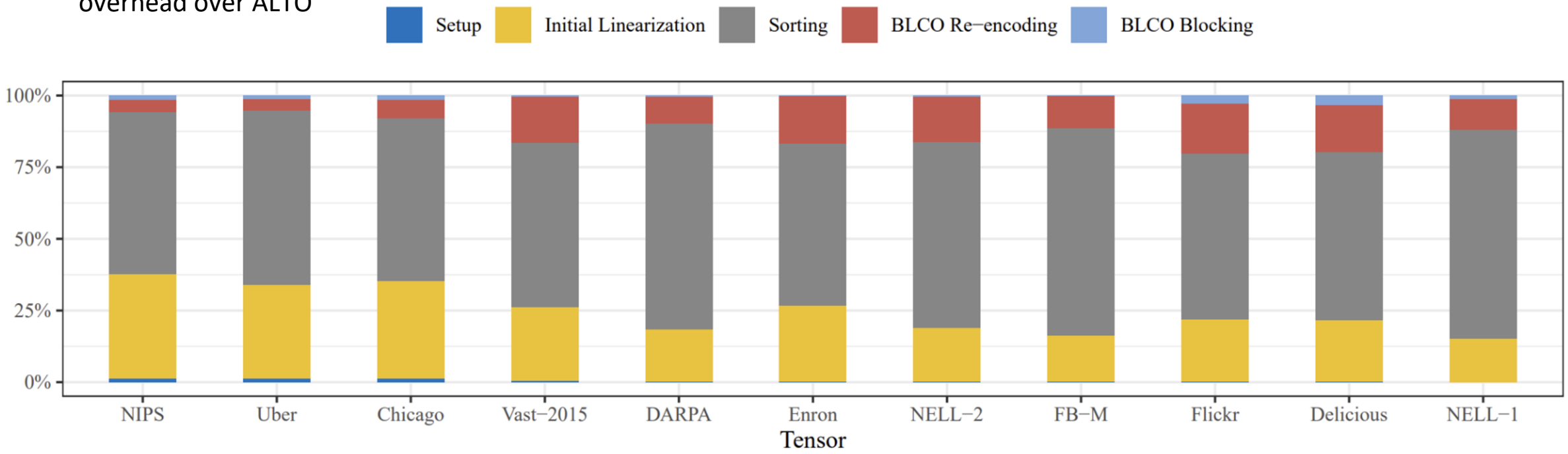


Format Construction Cost



Format Construction Cost

BLCO has minimal overhead over ALTO

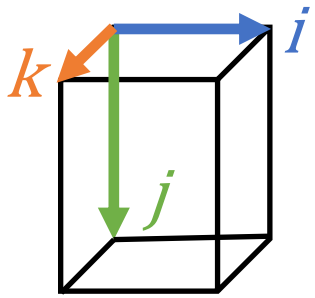


Q&A

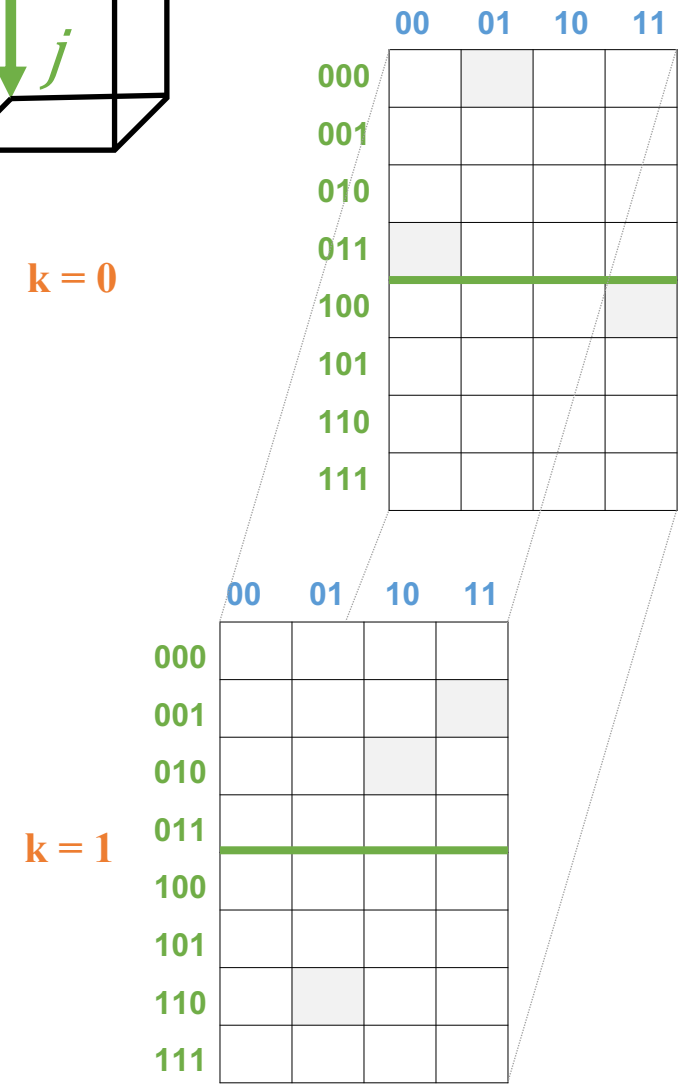
Backup Slides

ALTO

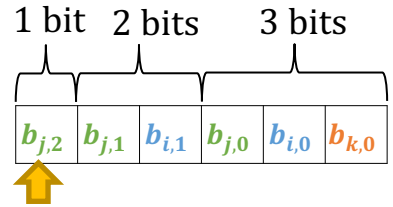
- The amount of information about the spatial position of a nonzero element decreases with each consecutive bit

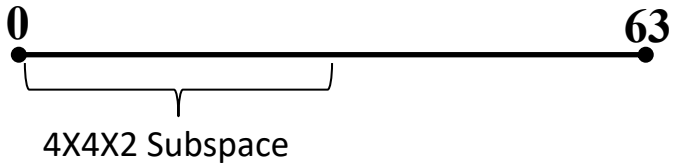


4x8x2 tensor



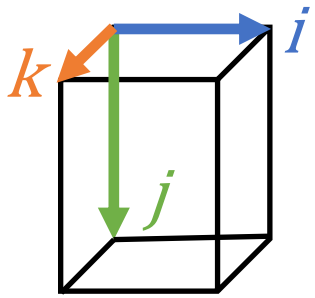
ALTO bit mask



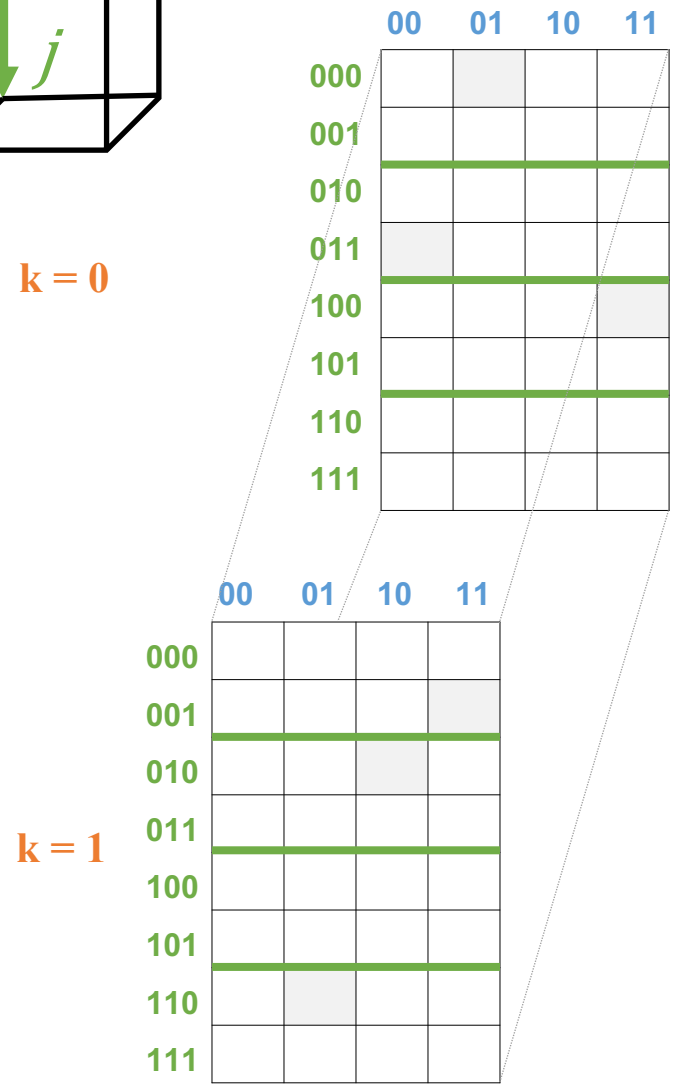


ALTO

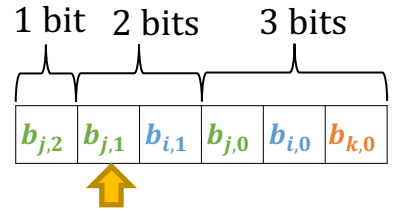
- The amount of information about the spatial position of a nonzero element decreases with each consecutive bit

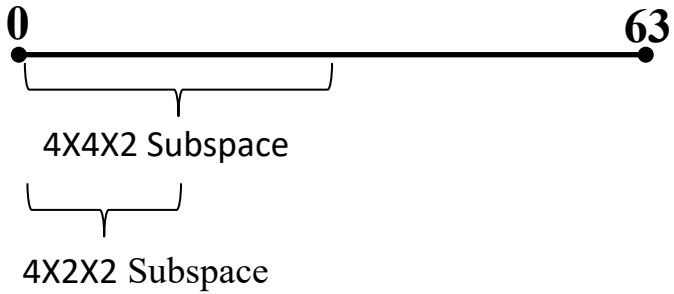


4x8x2 tensor



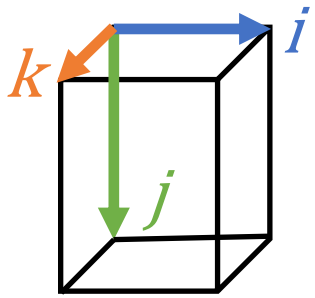
ALTO bit mask



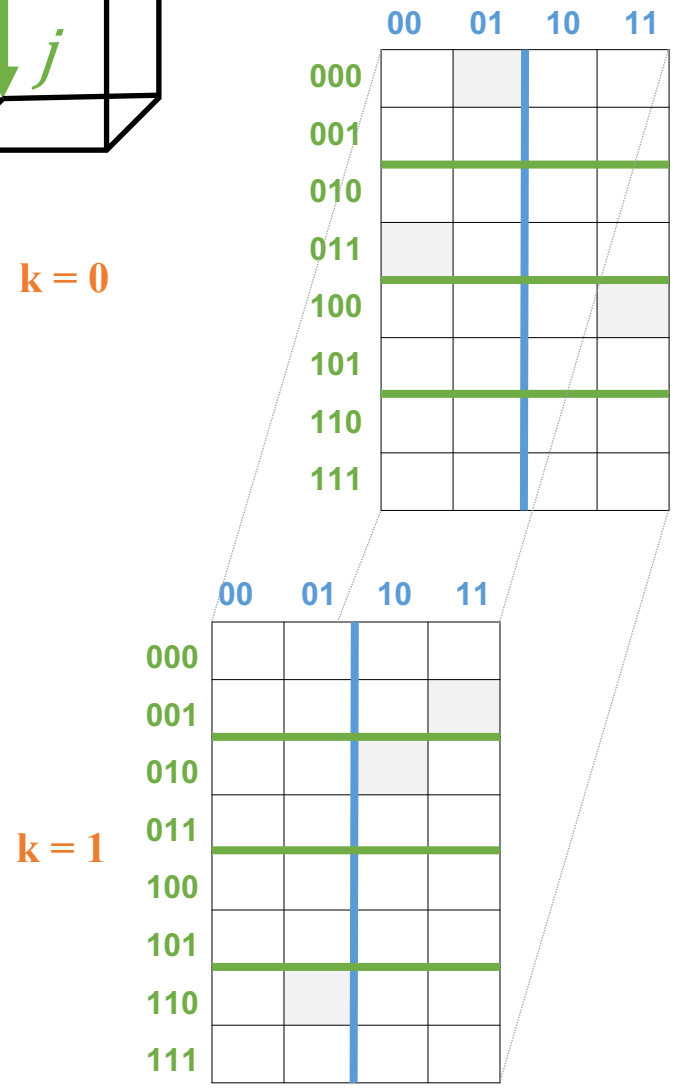


ALTO

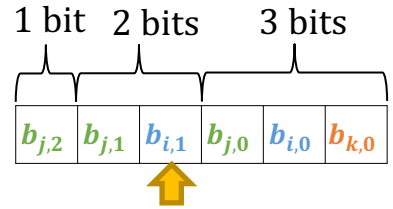
- The amount of information about the spatial position of a nonzero element decreases with each consecutive bit

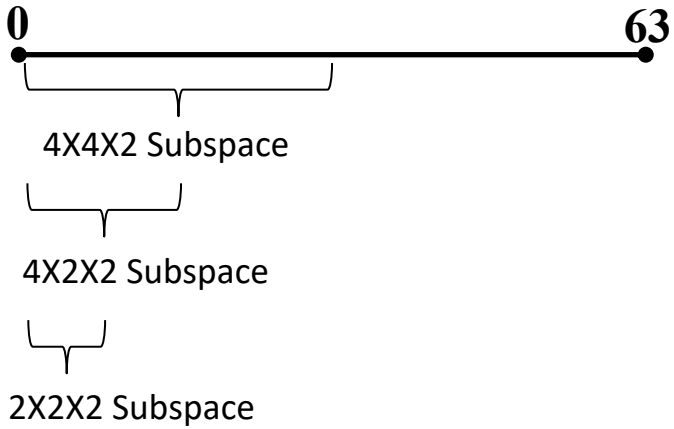


4x8x2 tensor



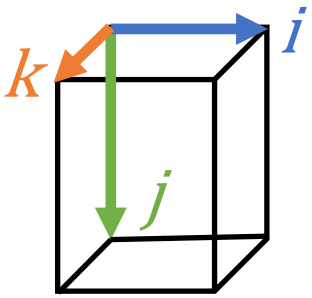
ALTO bit mask



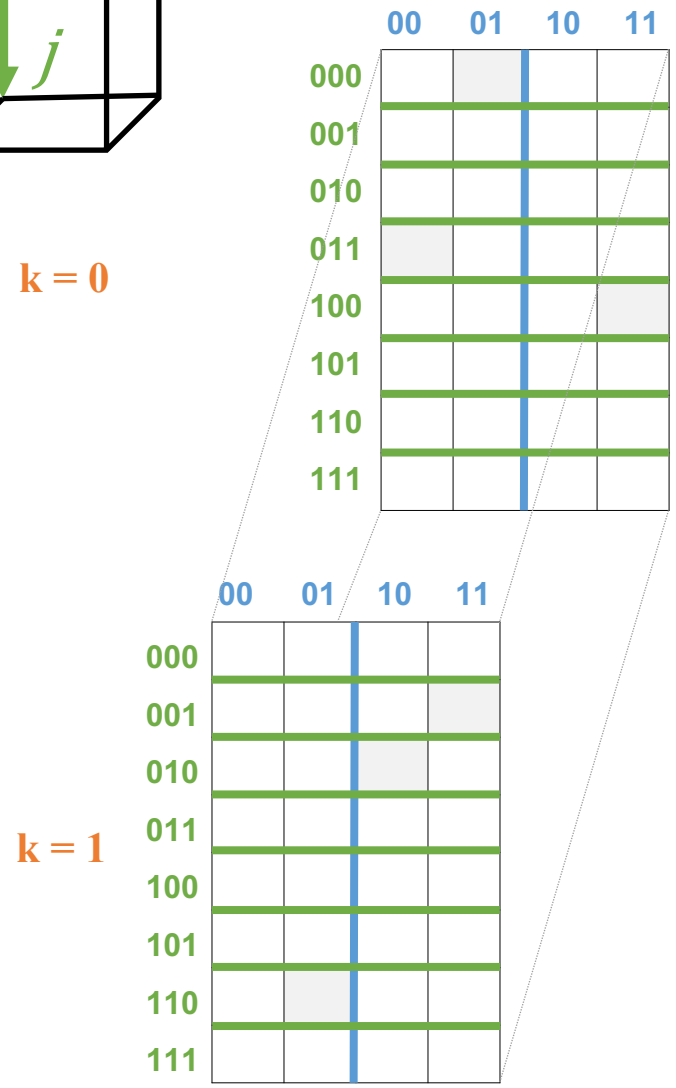


ALTO

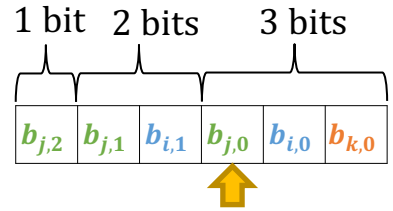
- The amount of information about the spatial position of a nonzero element decreases with each consecutive bit

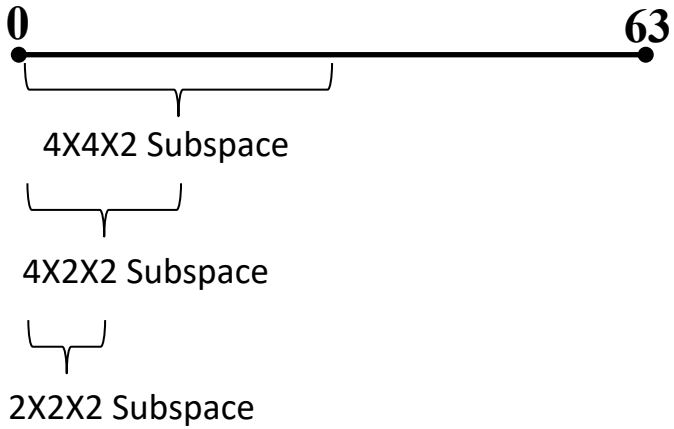


4x8x2 tensor



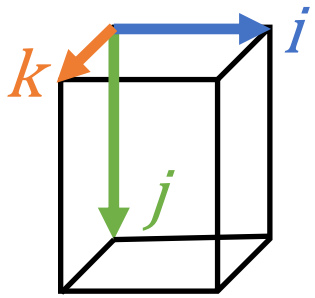
ALTO bit mask



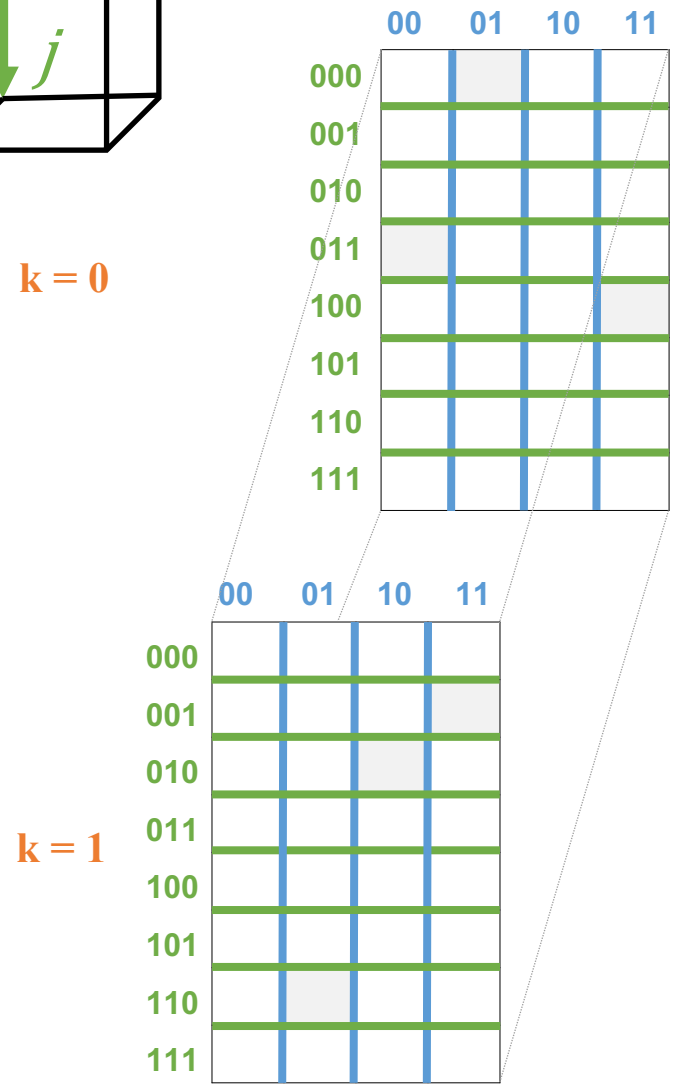


ALTO

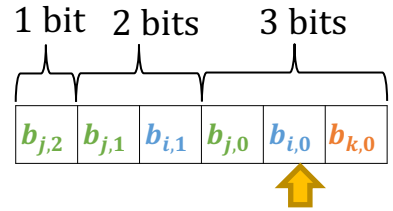
- The amount of information about the spatial position of a nonzero element decreases with each consecutive bit

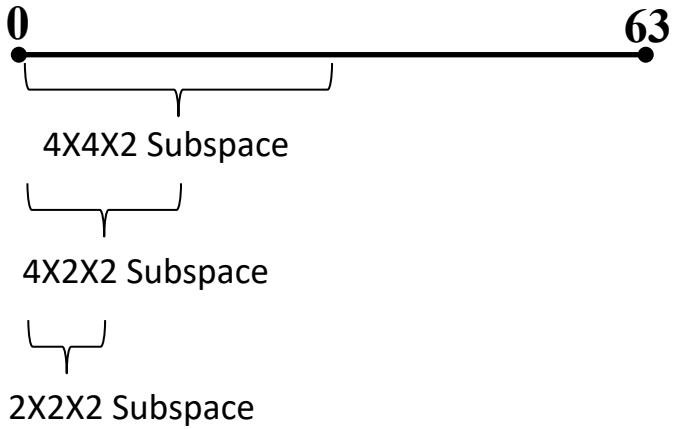


4x8x2 tensor



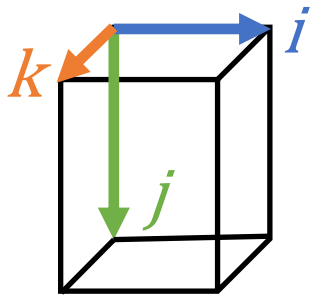
ALTO bit mask



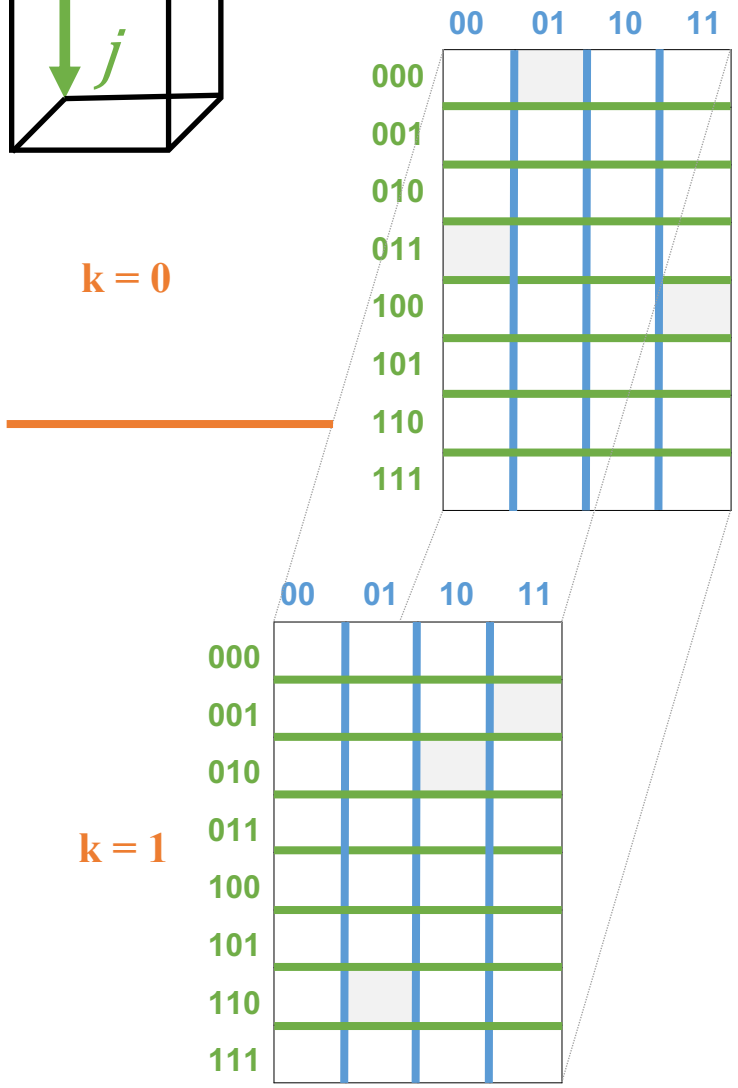


ALTO

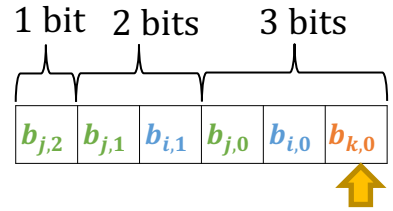
- This is equivalent to partitioning the multi-dimensional space along the longest mode first

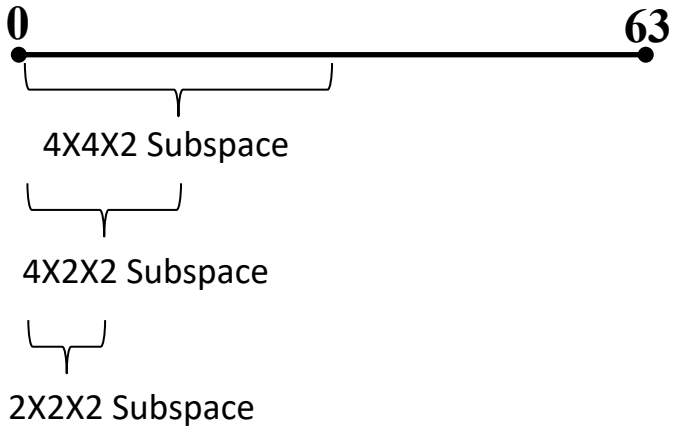


4x8x2 tensor



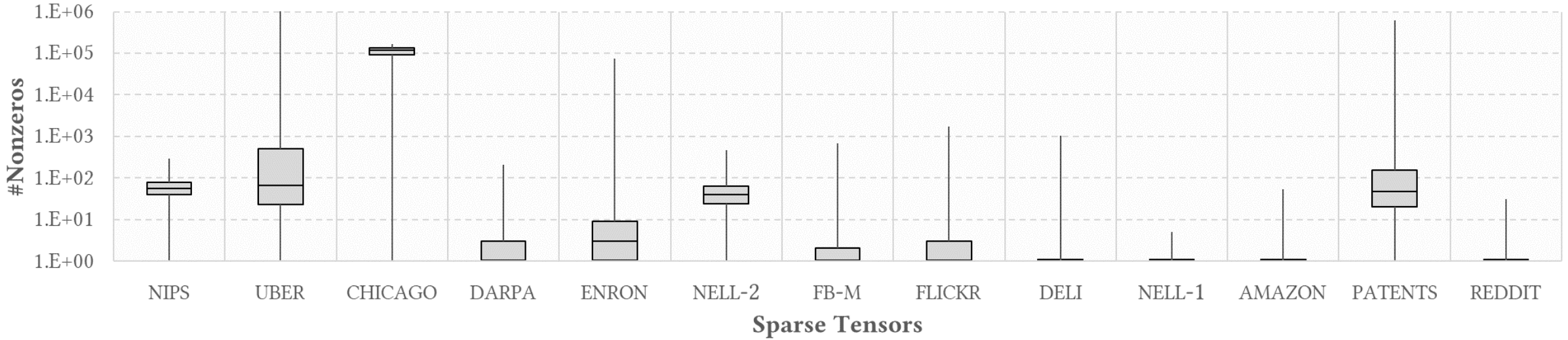
ALTO bit mask





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).