UNIVERSITY OF OREGON

Efficient, Out-of-Memory Sparse MTTKRP on Massively Parallel Architectures

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], Jee Whan Choi[1]

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

February 25th @ SIAM PP ‘22

Outline

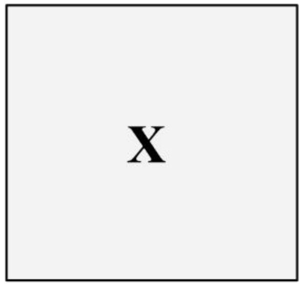
- Introduction to sparse tensors
- Sparse tensor kernel: MTTKRP in CPD-ALS
- Prior state of the art approaches
- Our approach
  - Tensor data format
  - Parallel algorithm
  - Evaluations
- Closing thoughts

Tensors

Vector

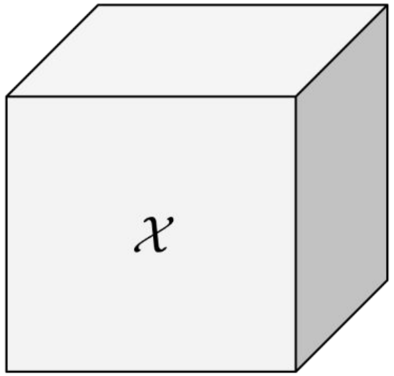
$\mathbf{x} \in \mathbb{R}^{I}$

Matrix



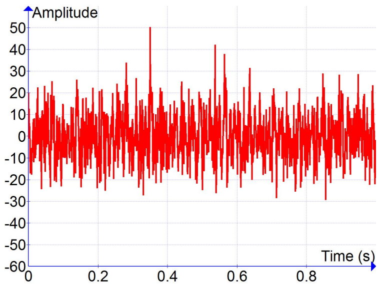
$\mathbf{X} \in \mathbb{R}^{I \times J}$

Tensor



$\mathcal{X} \in \mathbb{R}^{I \times J \times K}$

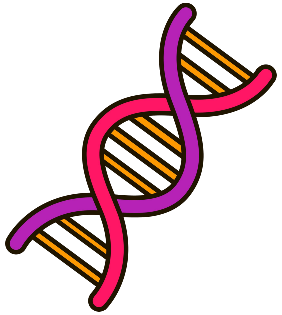
Real-World Tensors



Signal processing

Recommendation systems

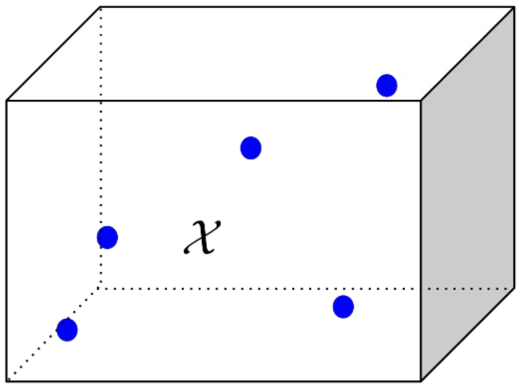
Product reviews



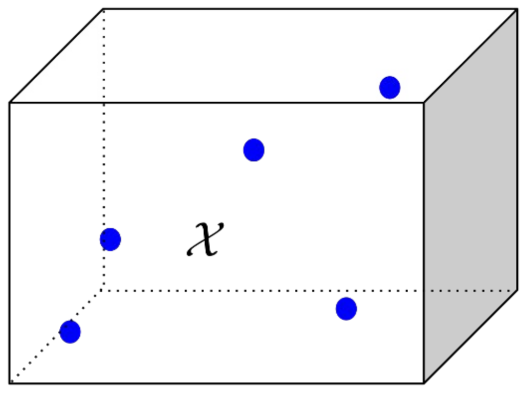
Electronic health records

Law enforcement data

Sparse Tensors

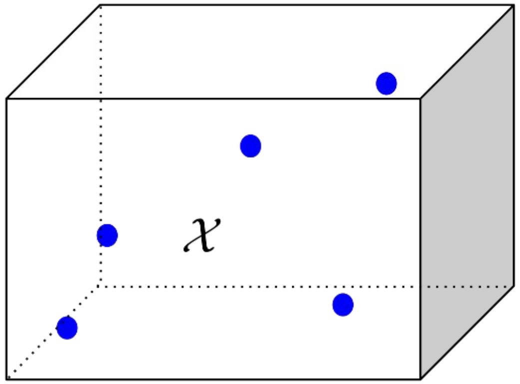


Sparse Tensors



i	j	k	v
1	4	3	5.2
4	5	2	8.7
1	1	3	0.3
5	3	2	4.6
4	1	2	1.9

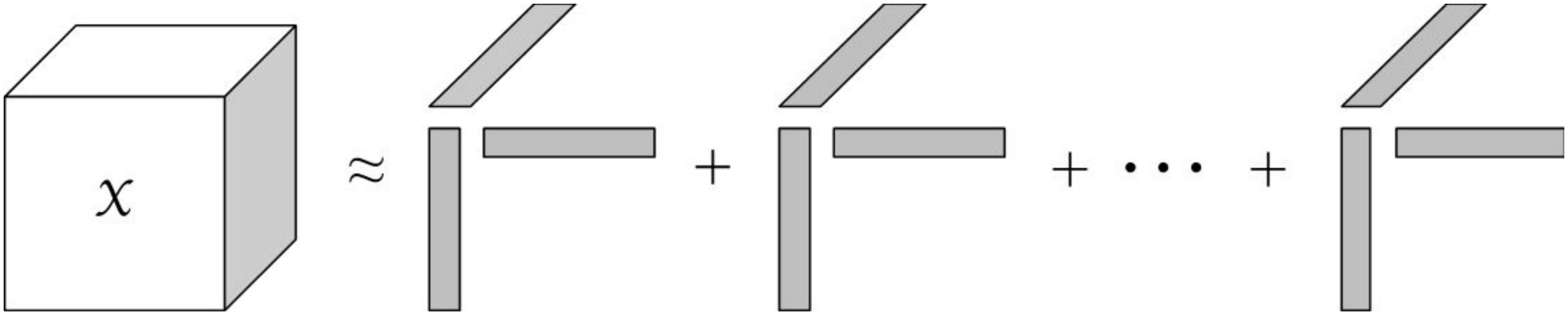
Sparse Tensors



Even harder to optimize than sparse matrix kernels:

- Curse of dimensionality
- Algorithm scalability
- Complex data compression

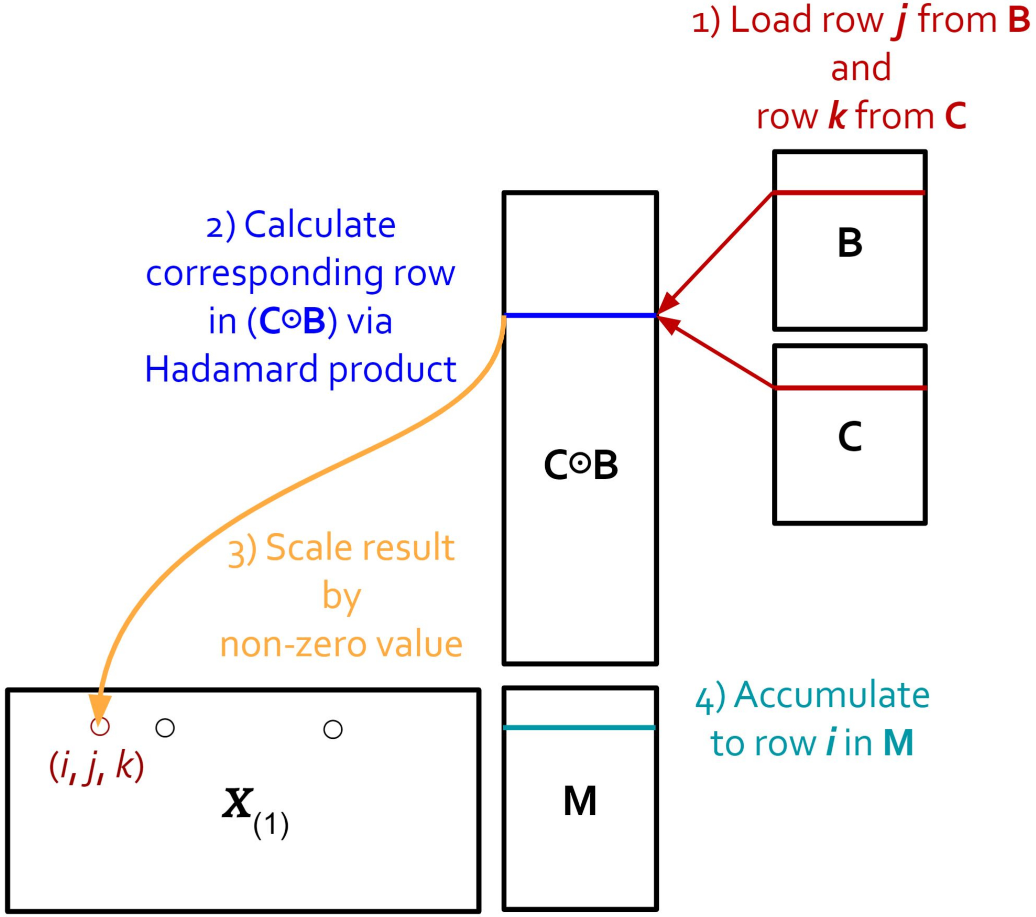
Tensor Decomposition



Canonical Polyadic Decomposition (CPD)

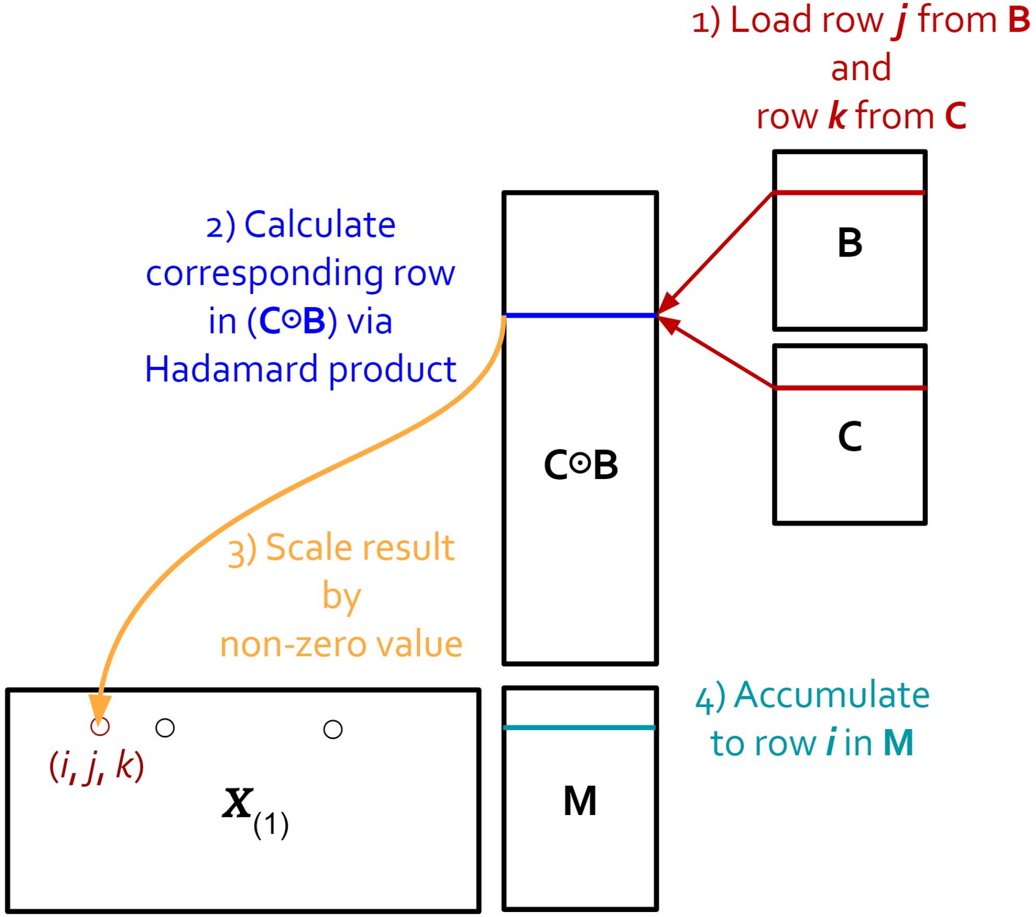
MTTKRP

- Bottleneck in well known CPD-ALS algorithm
- $M \leftarrow X_{(1)} C \odot B$



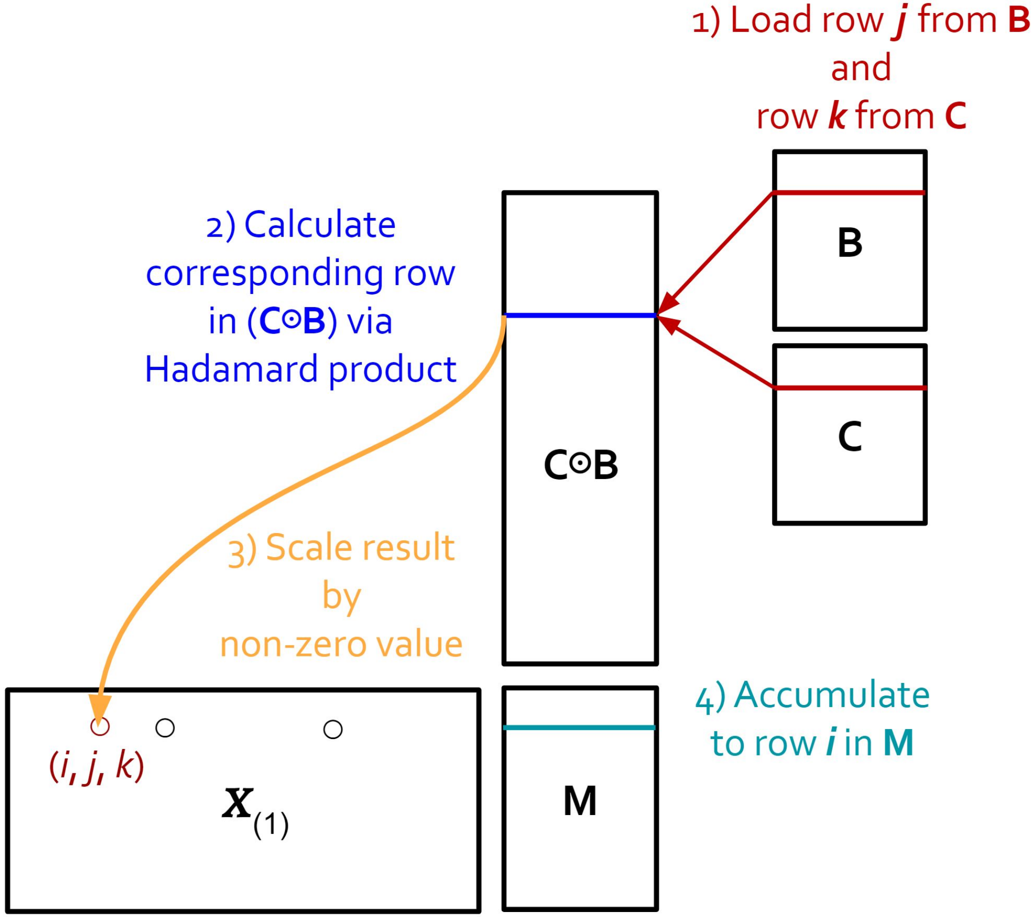
MTTKRP

- Challenges in optimizing MTTKRP:



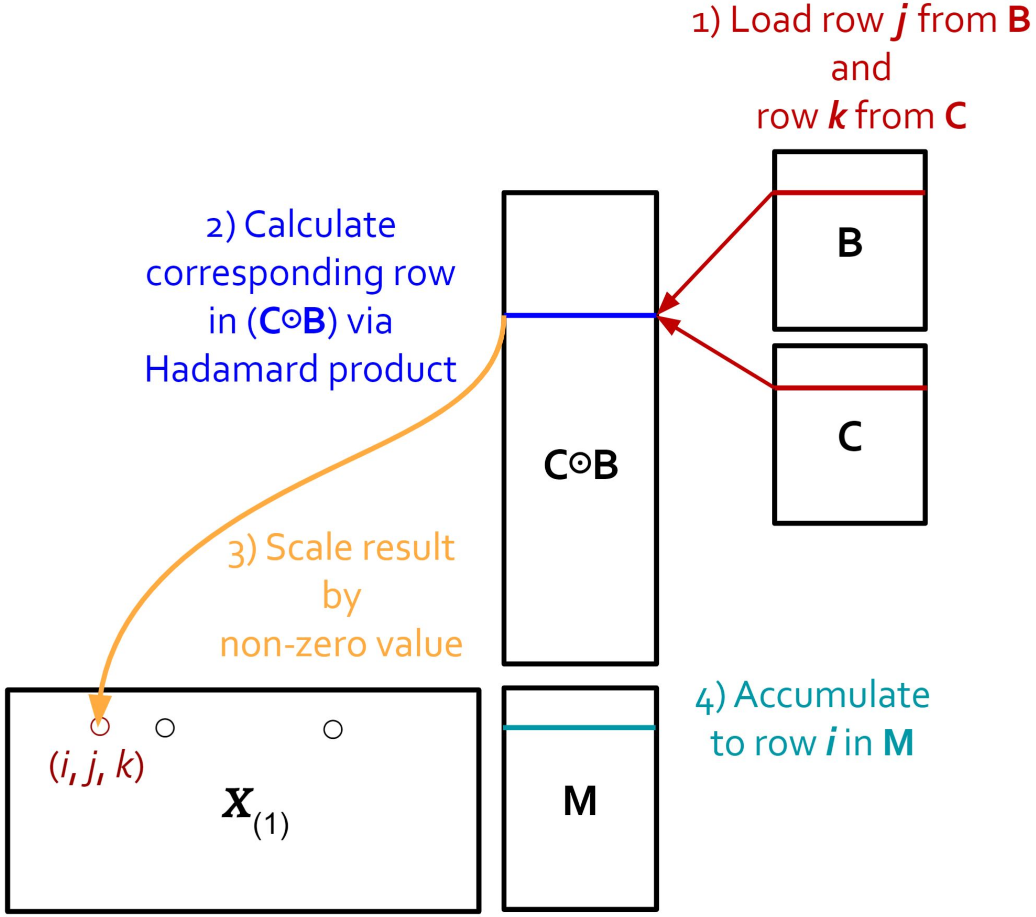
MTTKRP

- Challenges in optimizing MTTKRP:
  - **Memory-intensive** kernel



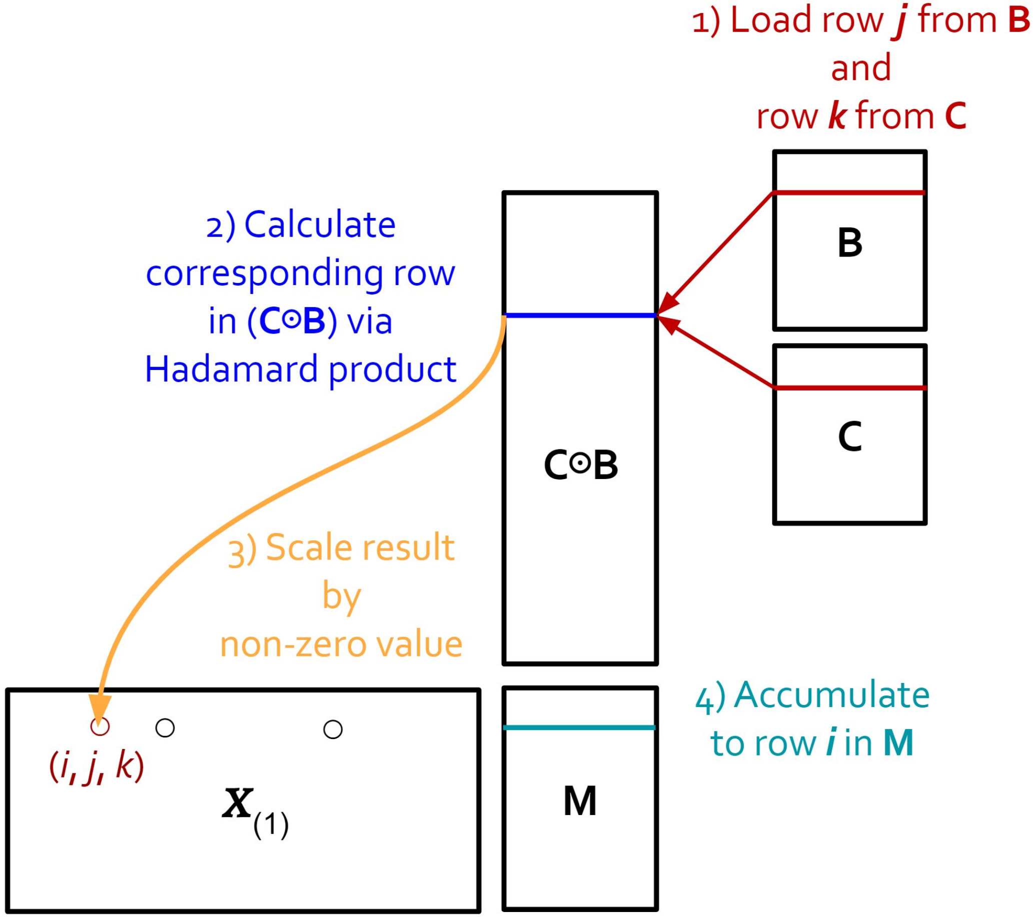
MTTKRP

- Challenges in optimizing MTTKRP:
  - **Memory-intensive** kernel
  - **Complex data locality** nature



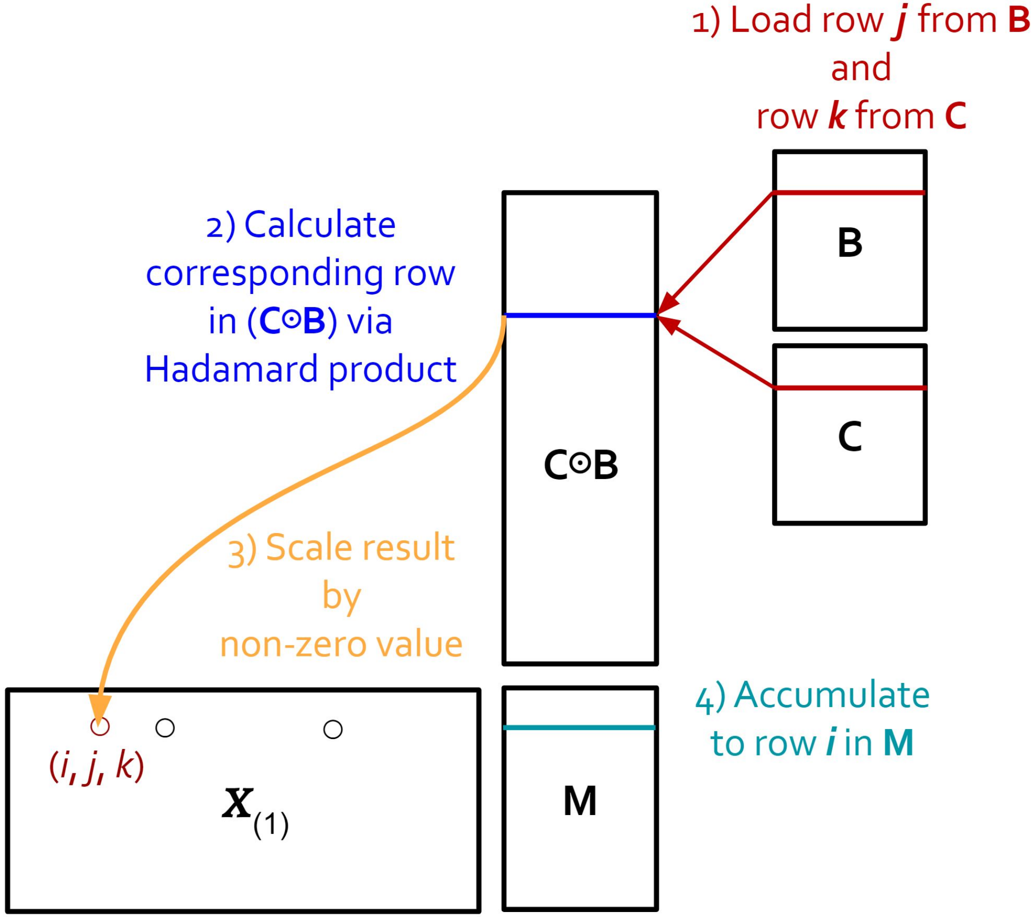
MTTKRP

- Challenges in optimizing MTTKRP:
  - **Memory-intensive** kernel
  - **Complex data locality** nature
  - **Synchronization overhead** when updating factor matrices



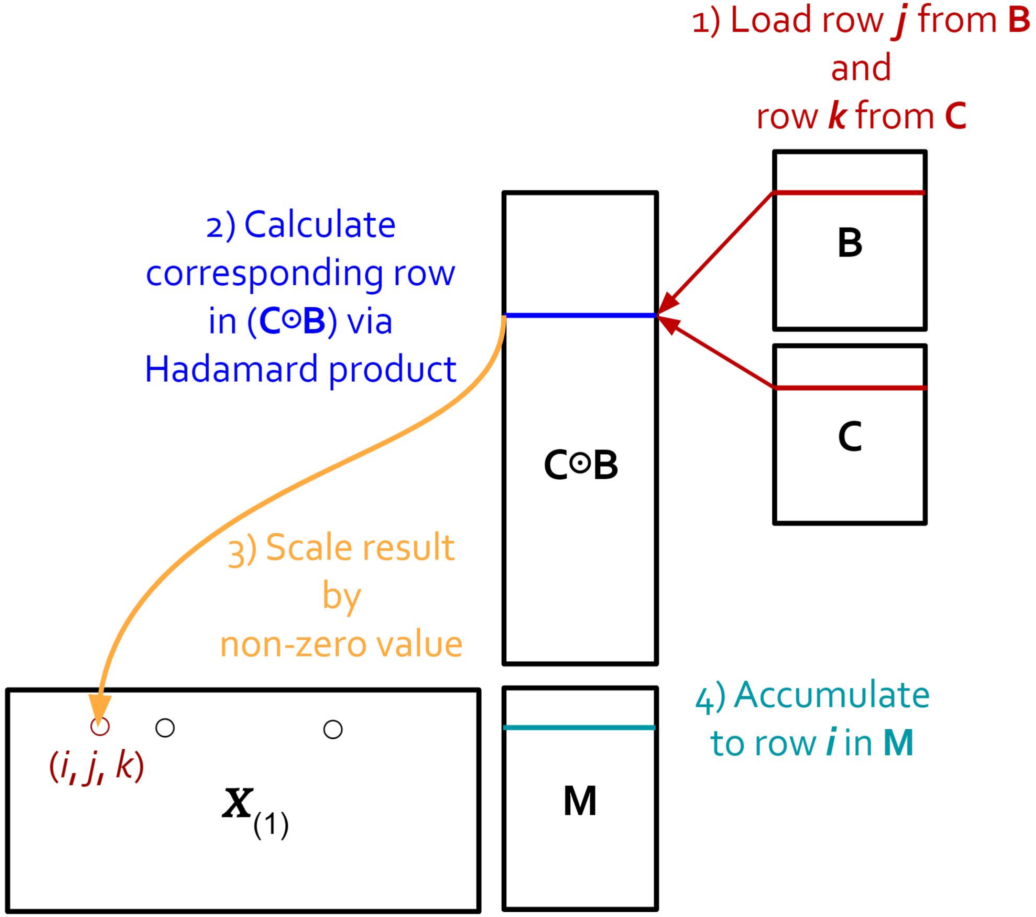
MTTKRP

- Challenges in optimizing MTTKRP:
  - **Memory-intensive** kernel
  - **Complex data locality** nature
  - **Synchronization overhead** when updating factor matrices
  - **Load imbalance** across threads



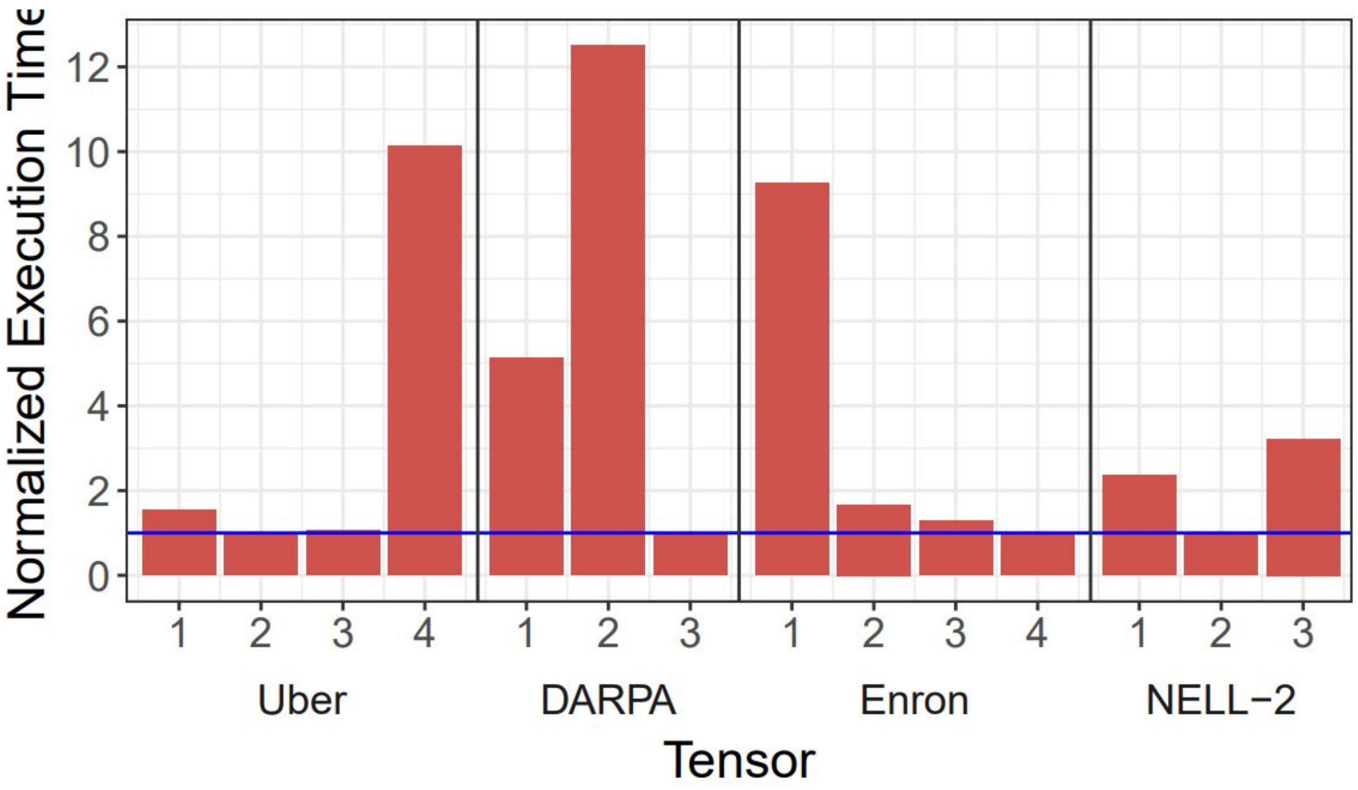
MTTKRP

- Challenges in optimizing MTTKRP:
  - **Memory-intensive** kernel
  - **Complex data locality** nature
  - **Synchronization overhead** when updating factor matrices
  - **Load imbalance** across threads
  - **Limited memory** on GPU accelerators



State-of-the-Art Performance Issues

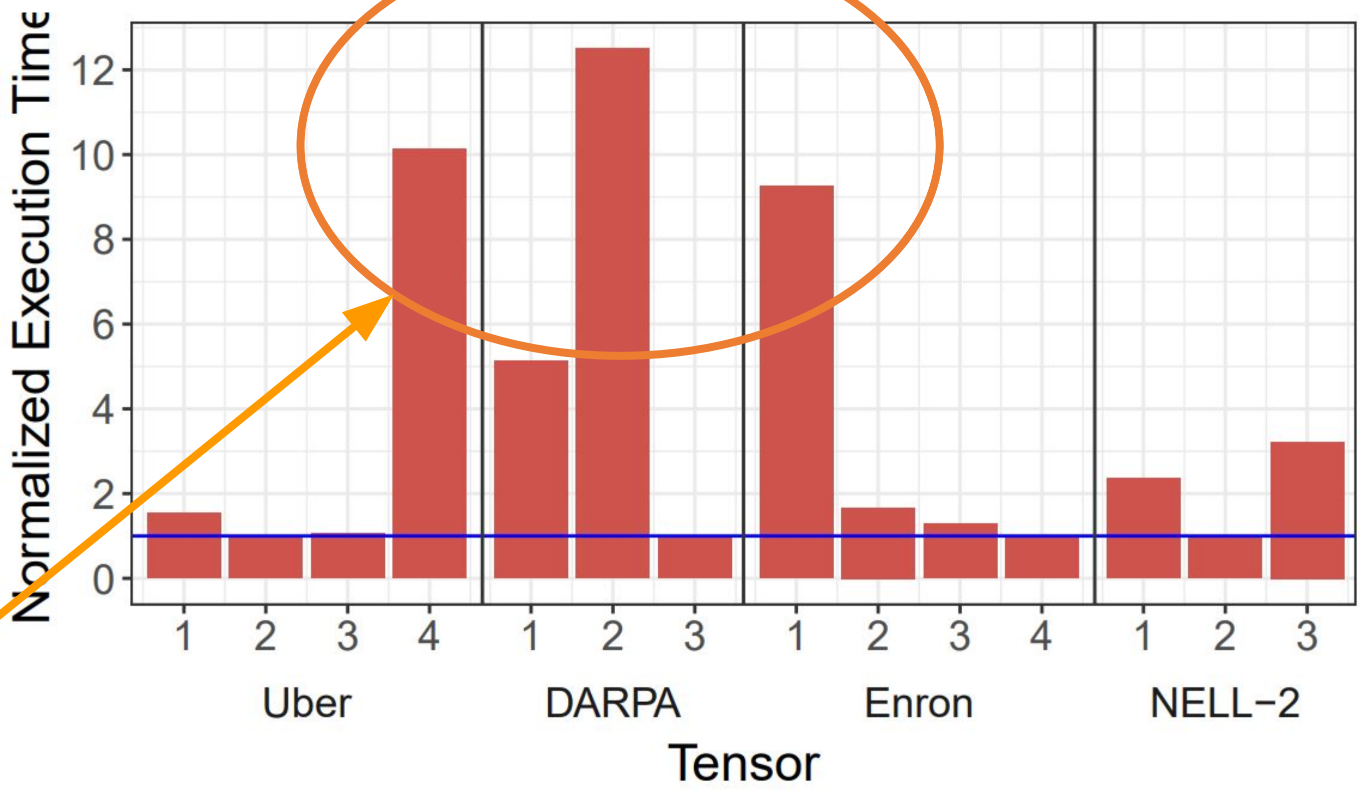
- Setup:
  - Individual mode MTTKRP on MM-CSF framework
  - Normalized against best performing mode
  - Blue line is baseline
  - NVIDIA A100
- Higher is worse



State-of-the-Art Performance Issues

- Setup:
  - Individual mode MTTKRP on MM-CSF framework
  - Normalized against best performing mode
  - Blue line is baseline
  - NVIDIA A100
- Higher is worse

Over an *order of magnitude* worse depending on the mode. All modes require the same number of FLOPs



State-of-the-Art Approaches

Name	Data Format	Load Balance	Synchronization	Mode Orientation	Data Compression	Mode Algorithms
F-COO [Liu et al. 2017]	List-based	Optimal	Reductions in shared memory	Mode-specific	Multiple copies required	1
B-CSF [Nisa et. al, 2019]	Tree-based	Improved	Minimized by data structure	Mode-specific	Multiple copies required	N
MM-CSF [Nisa et. al, 2019]	Tree-based	Improved	Minimized by data structure	Mode-specific	Single copy	N

State-of-the-Art Approaches

Name	Data Format	Load Balance	Synchronization	Mode Orientation	Data Compression	Mode Algorithms
F-COO [Liu et al. 2017]	List-based	Optimal	Reductions in shared memory	Mode-specific	Multiple copies required	1
B-CSF [Nisa et. al, 2019]	Tree-based	Improved	Minimized by data structure	Mode-specific	Multiple copies required	N
MM-CSF [Nisa et. al, 2019]	Tree-based	Improved	Minimized by data structure	Mode-specific	Single copy	N

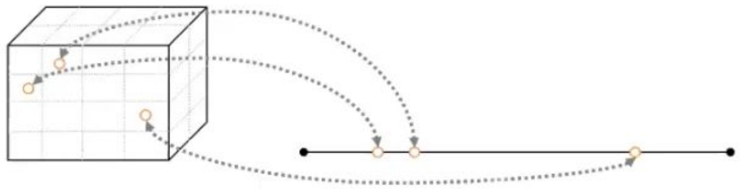
→ State of the art approaches have tradeoffs that severely impact performance

Blocked Linearized CoOrdinates (BLCO)

- A list-based yet highly compressed and massively parallel tensor format

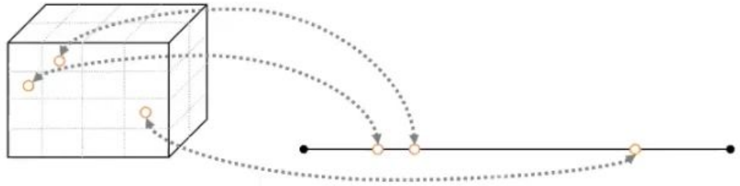
Blocked Linearized CoOrdinates (BLCO)

- A list-based yet highly compressed and massively parallel tensor format
- **Index linearization** maps multidimensional coordinates into linear space



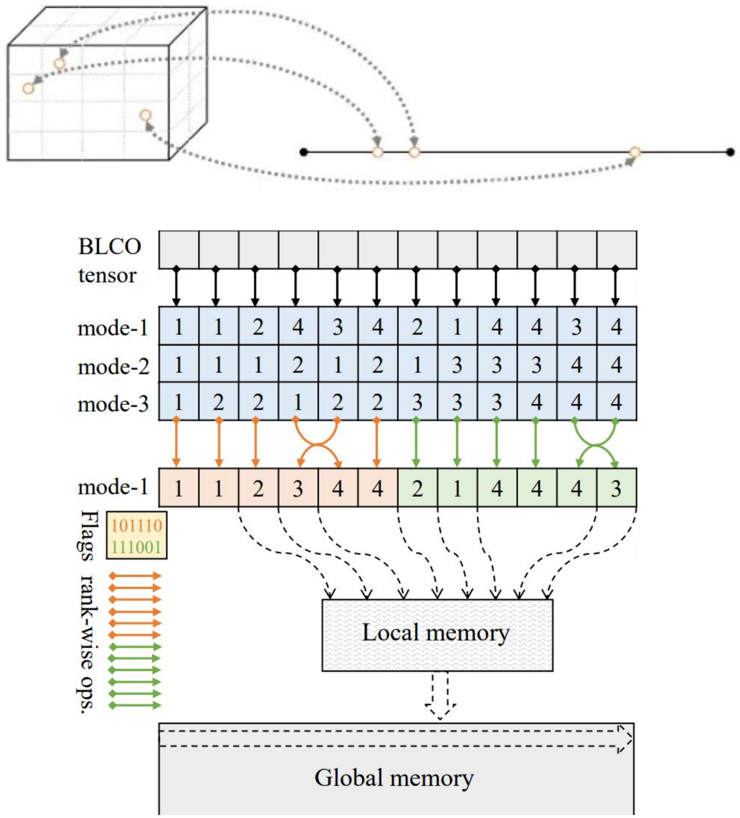
Blocked Linearized CoOrdinates (BLCO)

- A list-based yet highly compressed and massively parallel tensor format
- **Index linearization** maps multidimensional coordinates into linear space
- **Adaptive blocking** blocks the tensor to meet target GPU resource constraints

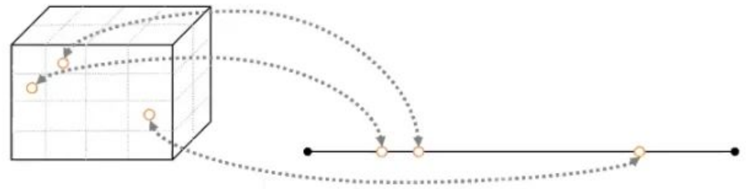


Blocked Linearized CoOrdinates (BLCO)

- A list-based yet highly compressed and massively parallel tensor format
- **Index linearization** maps multidimensional coordinates into linear space
- **Adaptive blocking** blocks the tensor to meet target GPU resource constraints
- **Hierarchical conflict resolution** resolves synchronization conflicts at multiple levels of the memory hierarchy

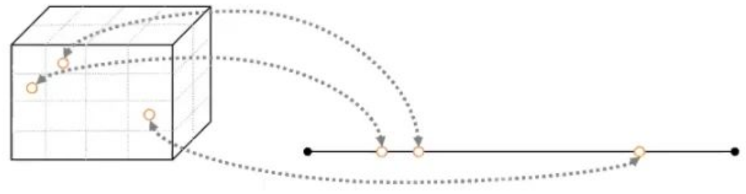


Index Linearization



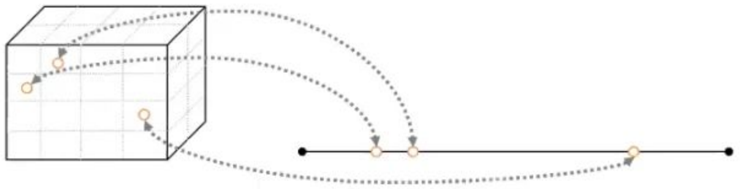
- Element indices are linearized in one-dimensional space

Index Linearization



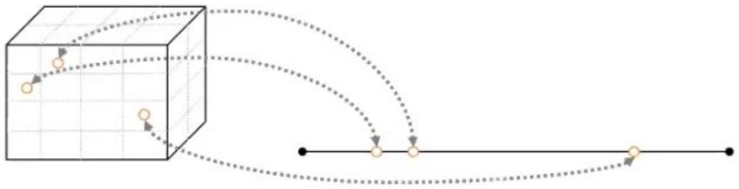
- Element indices are linearized in one-dimensional space
  - Highly compressed representation

Index Linearization



- Element indices are linearized in one-dimensional space
  - Highly compressed representation
  - More effective use of memory bandwidth

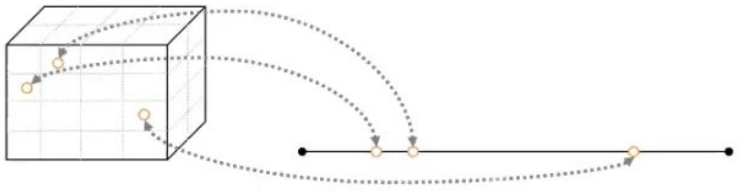
Index Linearization



- Element indices are linearized in one-dimensional space
  - Highly compressed representation
  - More effective use of memory bandwidth
  - Allows for fine-grained parallelism (load balance)

Index Linearization

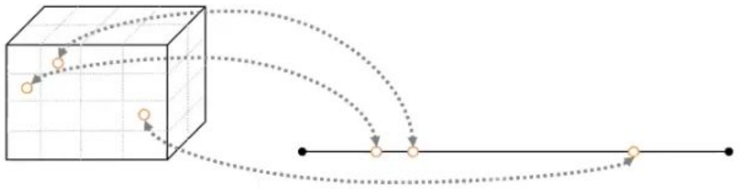
- Element indices are linearized in one-dimensional space
  - Highly compressed representation
  - More effective use of memory bandwidth
  - Allows for fine-grained parallelism (load balance)
- Elements are linearized and ordered based on ALTO [Helal et. al, 2021]



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

Index Linearization

- Element indices are linearized in one-dimensional space
  - Highly compressed representation
  - More effective use of memory bandwidth
  - Allows for fine-grained parallelism (load balance)
- Elements are linearized and ordered based on ALTO [Helal et. al, 2021]
- Indices are then re-linearized
  - Allows for fast delinearization on GPUs

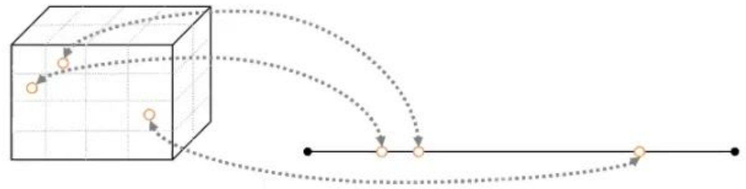


l		v
0	$(000000)_2$	1.0
4	$(000100)_2$	2.0
5	$(000101)_2$	4.0
10	$(001010)_2$	8.0
12	$(001100)_2$	6.0
15	$(001111)_2$	9.0
33	$(100001)_2$	5.0
48	$(110000)_2$	3.0
57	$(111001)_2$	10.0
61	$(111101)_2$	11.0
62	$(111110)_2$	7.0
63	$(111111)_2$	12.0

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

Adaptive Blocking

- Linearization may exceed integer width (i.e. 64 bits)

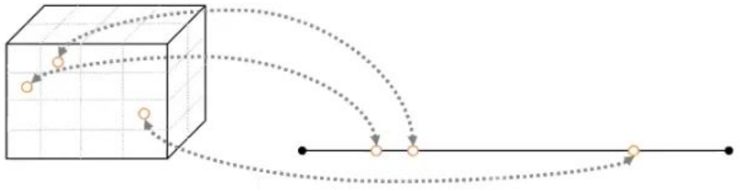


l		v
0	$(000000)_2$	1.0
4	$(000100)_2$	2.0
5	$(000101)_2$	4.0
10	$(001010)_2$	8.0
12	$(001100)_2$	6.0
15	$(001111)_2$	9.0
33	$(100001)_2$	5.0
48	$(110000)_2$	3.0
57	$(111001)_2$	10.0
61	$(111101)_2$	11.0
62	$(111110)_2$	7.0
63	$(111111)_2$	12.0

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

Adaptive Blocking

- Linearization may exceed integer width (i.e. 64 bits)
- Block the tensor based on the excessive bits

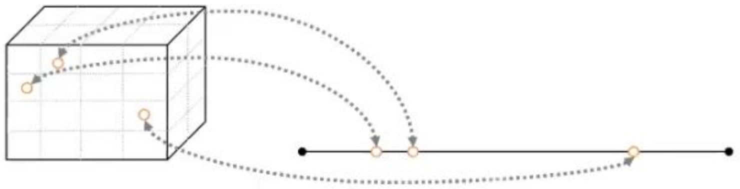


l		v
0	$(000000)_2$	1.0
4	$(000100)_2$	2.0
5	$(000101)_2$	4.0
10	$(001010)_2$	8.0
12	$(001100)_2$	6.0
15	$(001111)_2$	9.0
33	$(100001)_2$	5.0
48	$(110000)_2$	3.0
57	$(111001)_2$	10.0
61	$(111101)_2$	11.0
62	$(111110)_2$	7.0
63	$(111111)_2$	12.0

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

Adaptive Blocking

- Linearization may exceed integer width (i.e. 64 bits)
- Block the tensor based on the excessive bits
  - Mode-agnostic approach

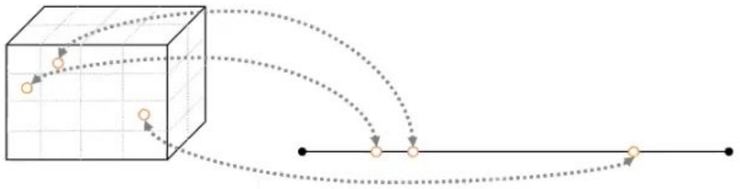


l		v
0	$(000000)_2$	1.0
4	$(000100)_2$	2.0
5	$(000101)_2$	4.0
10	$(001010)_2$	8.0
12	$(001100)_2$	6.0
15	$(001111)_2$	9.0
33	$(100001)_2$	5.0
48	$(110000)_2$	3.0
57	$(111001)_2$	10.0
61	$(111101)_2$	11.0
62	$(111110)_2$	7.0
63	$(111111)_2$	12.0

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

Adaptive Blocking

- Linearization may exceed integer width (i.e. 64 bits)
- Block the tensor based on the excessive bits
    - Mode-agnostic approach
    - Adds minimal memory overhead

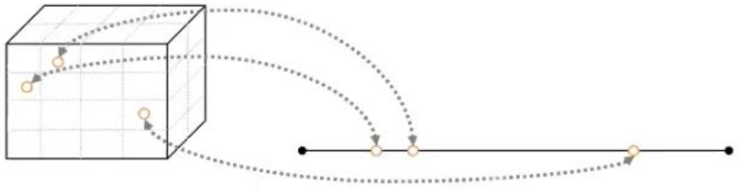


l		v
0	$(000000)_2$	1.0
4	$(000100)_2$	2.0
5	$(000101)_2$	4.0
10	$(001010)_2$	8.0
12	$(001100)_2$	6.0
15	$(001111)_2$	9.0
33	$(100001)_2$	5.0
48	$(110000)_2$	3.0
57	$(111001)_2$	10.0
61	$(111101)_2$	11.0
62	$(111110)_2$	7.0
63	$(111111)_2$	12.0

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

Adaptive Blocking

- Linearization may exceed integer width (i.e. 64 bits)
- Block the tensor based on the excessive bits
  - Mode-agnostic approach
  - Adds minimal memory overhead
  - Blocks can be streamed for *out-of-memory computation*

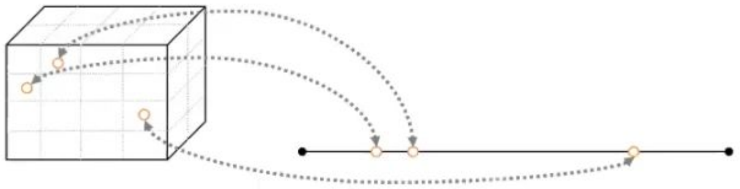


l		v
0	$(000000)_2$	1.0
4	$(000100)_2$	2.0
5	$(000101)_2$	4.0
10	$(001010)_2$	8.0
12	$(001100)_2$	6.0
15	$(001111)_2$	9.0
33	$(100001)_2$	5.0
48	$(110000)_2$	3.0
57	$(111001)_2$	10.0
61	$(111101)_2$	11.0
62	$(111110)_2$	7.0
63	$(111111)_2$	12.0

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

Adaptive Blocking

- Linearization may exceed integer width (i.e. 64 bits)
- Block the tensor based on the excessive bits
  - Mode-agnostic approach
  - Adds minimal memory overhead
  - Blocks can be streamed for *out-of-memory computation*
  - Leverage list-based storage to expose fine grained parallel processing…



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

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

Hierarchical Conflict Resolution

- Synchronization overhead is expensive for list-based formats
    - Especially true on GPUs

Hierarchical Conflict Resolution

- Synchronization overhead is expensive for list-based formats
  - Especially true on GPUs
- Leverage multiple memory levels hierarchically to minimize atomics

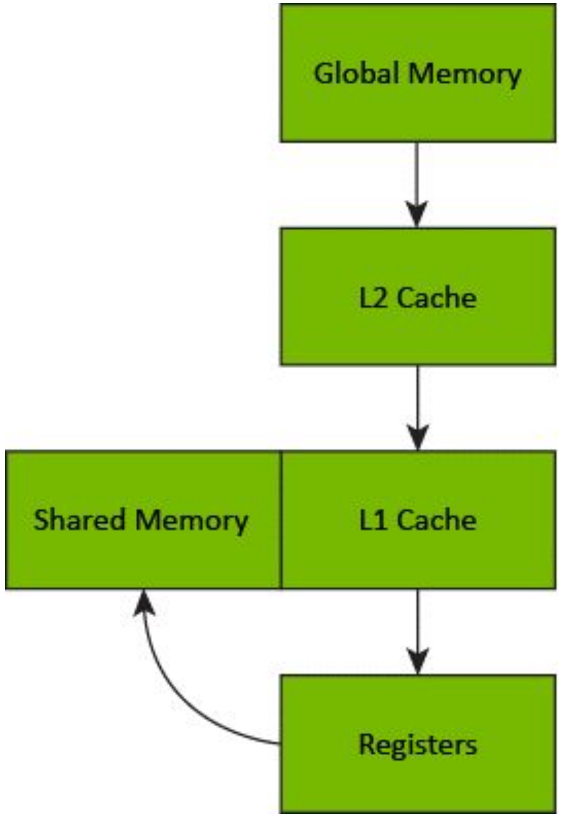


Figure source: https://developer.nvidia.com/blog/controlling-data-movement-to-boost-performance-on-ampere-architecture/

Hierarchical Conflict Resolution

- Synchronization overhead is expensive for list-based formats
  - Especially true on GPUs
- Leverage multiple memory levels hierarchically to minimize atomics
  - Register memory: segmented scan

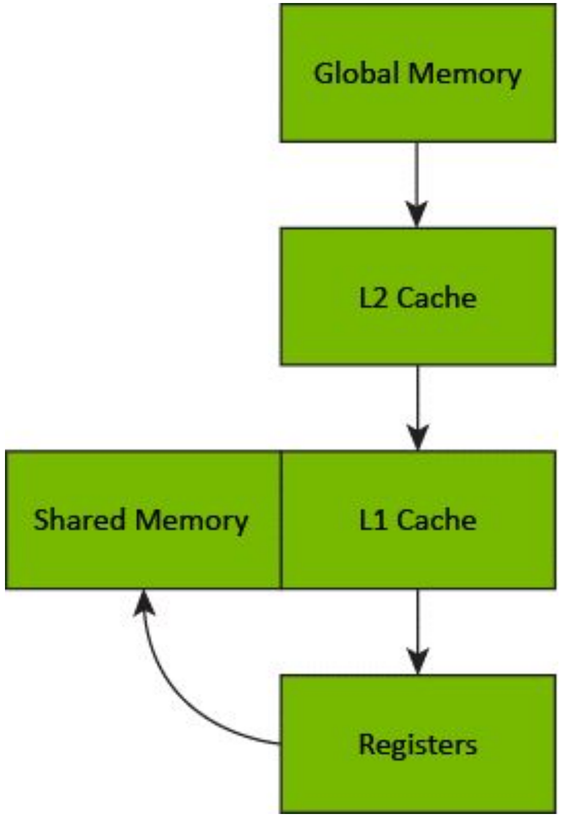


Figure source: https://developer.nvidia.com/blog/controlling-data-movement-to-boost-performance-on-ampere-architecture/

Hierarchical Conflict Resolution

- Synchronization overhead is expensive for list-based formats
  - Especially true on GPUs
- Leverage multiple memory levels hierarchically to minimize atomics
  - Register memory: segmented scan
  - Shared memory: software cache

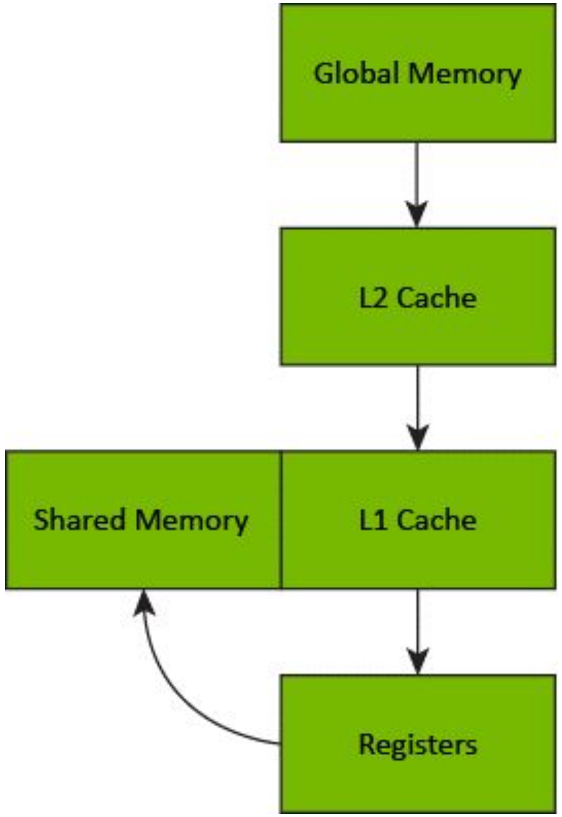


Figure source: https://developer.nvidia.com/blog/controlling-data-movement-to-boost-performance-on-ampere-architecture/

Hierarchical Conflict Resolution

- Synchronization overhead is expensive for list-based formats
  - Especially true on GPUs
- Leverage multiple memory levels hierarchically to minimize atomics
  - Register memory: segmented scan
  - Shared memory: software cache
  - Global memory: pull-based reduction

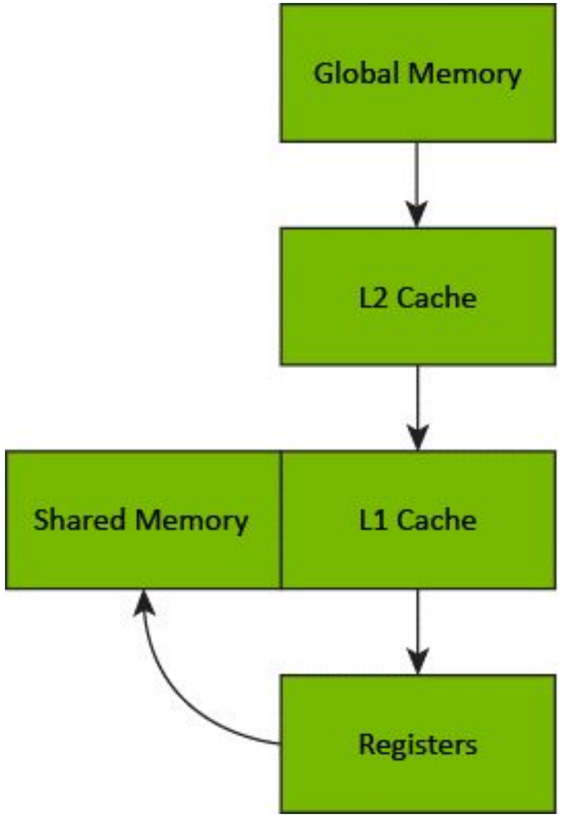


Figure source: https://developer.nvidia.com/blog/controlling-data-movement-to-boost-performance-on-ampere-architecture/

State-of-the-Art Approaches

Name	Data Format	Load Balance	Synchronization	Mode Orientation	Data Compression	Mode Algorithms
F-COO [Liu et al. 2017]	List-based	Optimal	Reductions in shared memory	Mode-specific	Multiple copies required	1
B-CSF [Nisa et. al, 2019]	Tree-based	Improved	Minimized by data structure	Mode-specific	Multiple copies required	N
MM-CSF [Nisa et. al, 2019]	Tree-based	Improved	Minimized by data structure	Mode-specific	Single copy	N
BLCO [this work]	**List-based**	**Optimal**	**Reductions at multiple memory levels**	**Mode-agnostic**	**Single copy, streamable**	**1**

Evaluation Platform

GPUS:

- NVIDIA A100
- NVIDIA V100

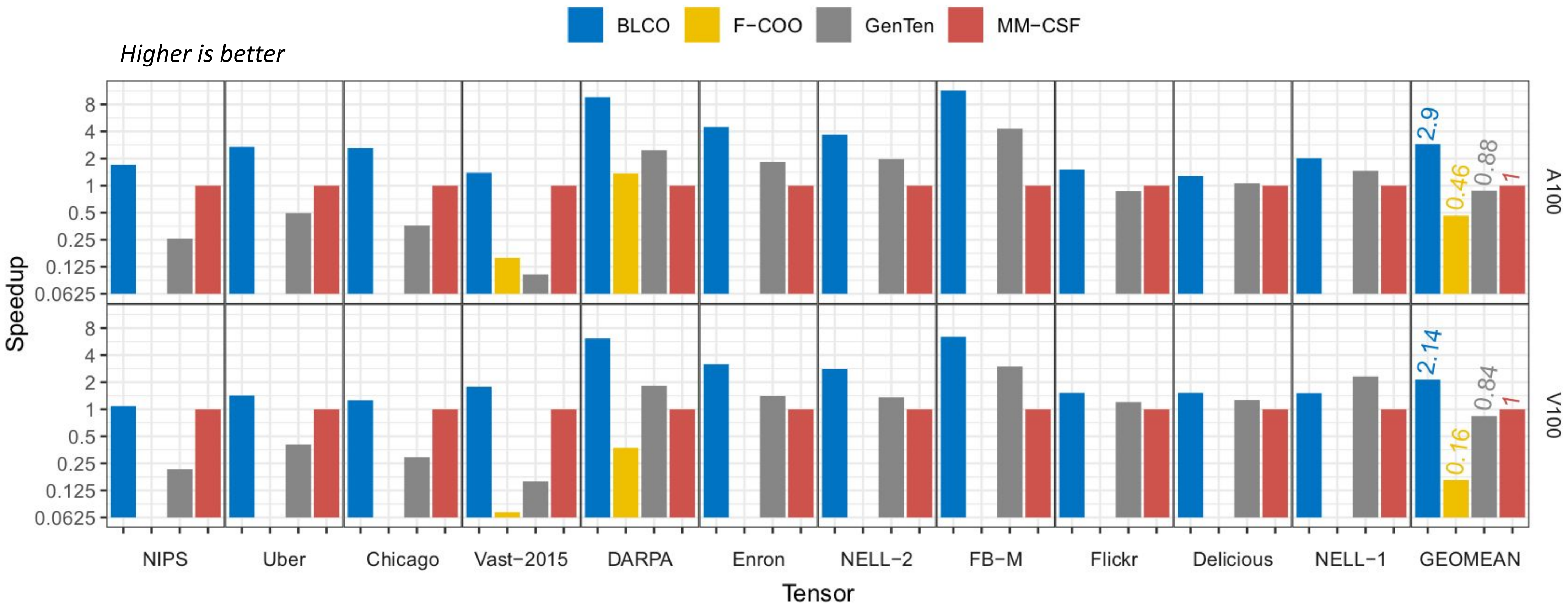
Considered frameworks:

- MM-CSF
- F-COO
- Genten
- BLCO

Benchmark datasets: 11 real-world tensors from FROSTT

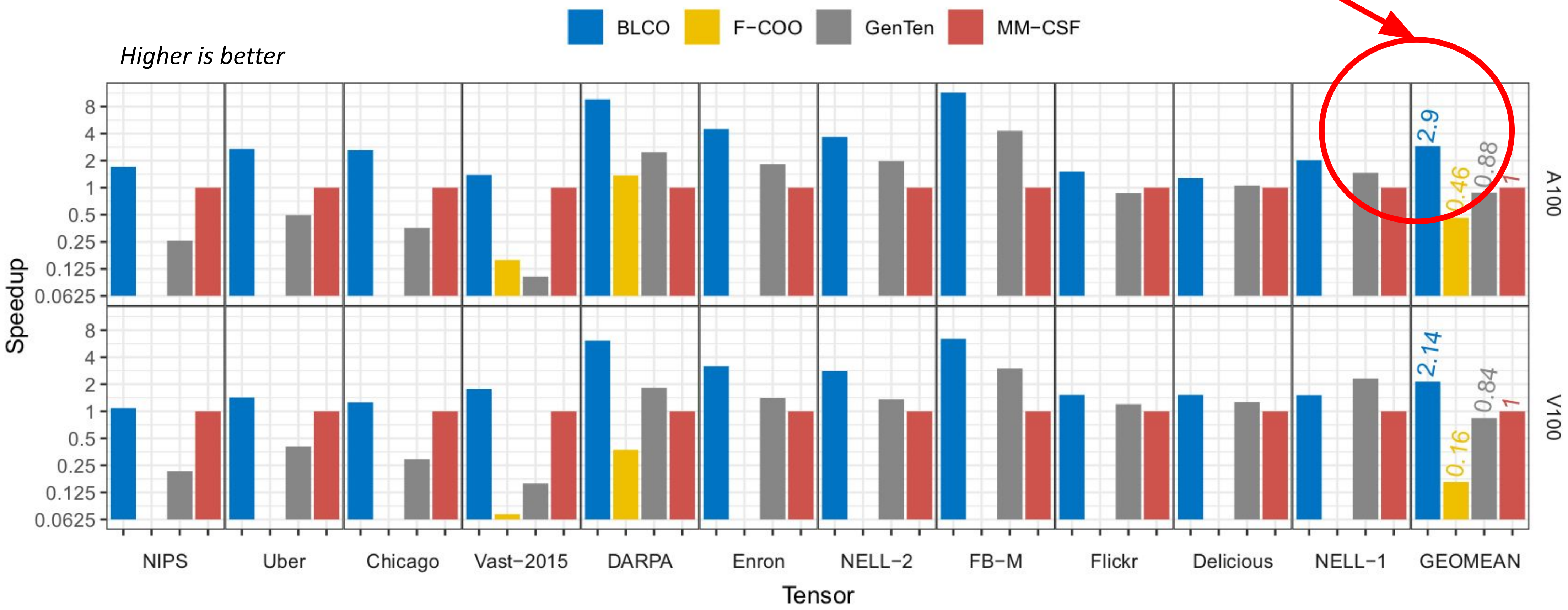
Compilation: double-precision / 64-bit data types

Framework Comparison



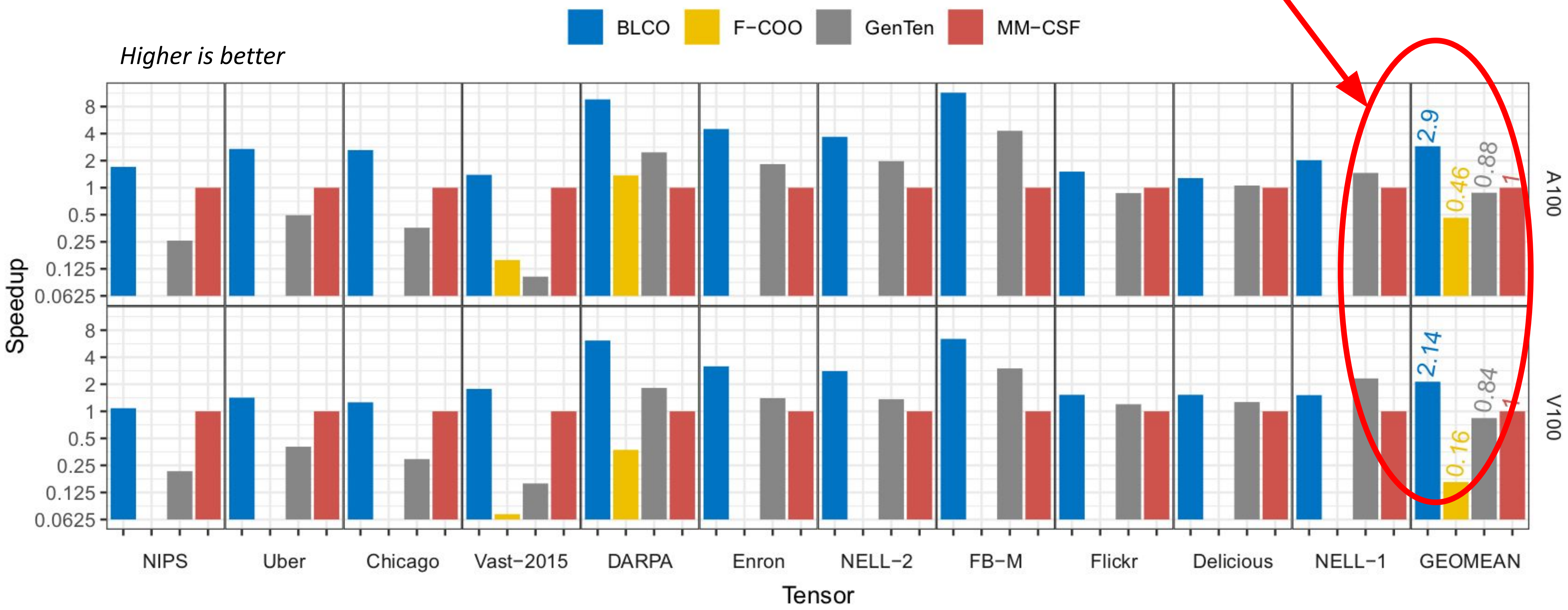
Framework Comparison

Up to 2.90x speedup over prior state of the art

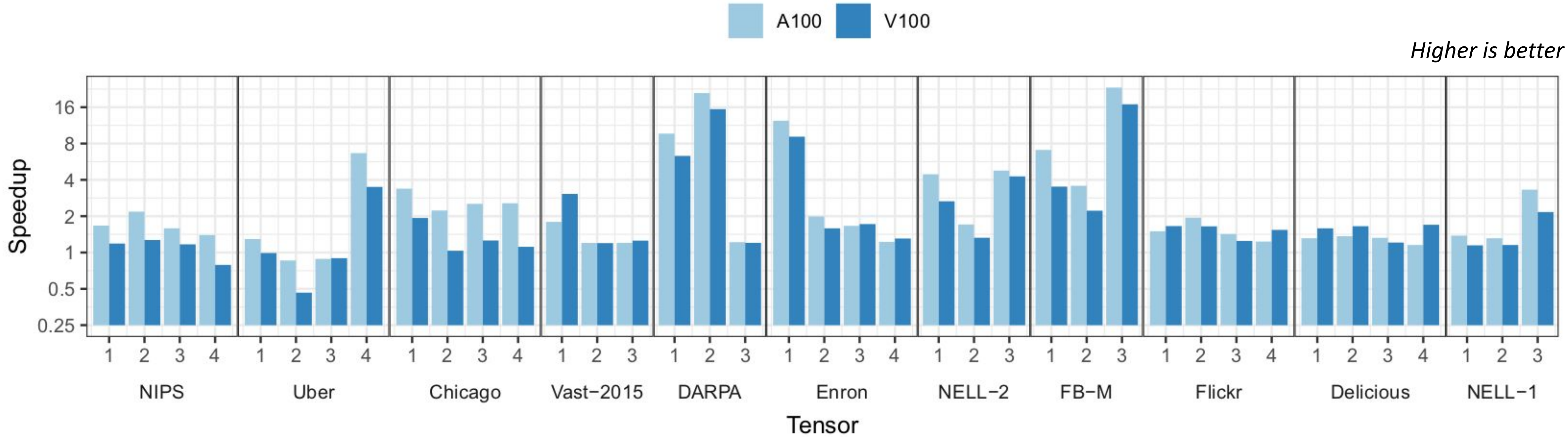


Framework Comparison

Speedup observed across different architectures



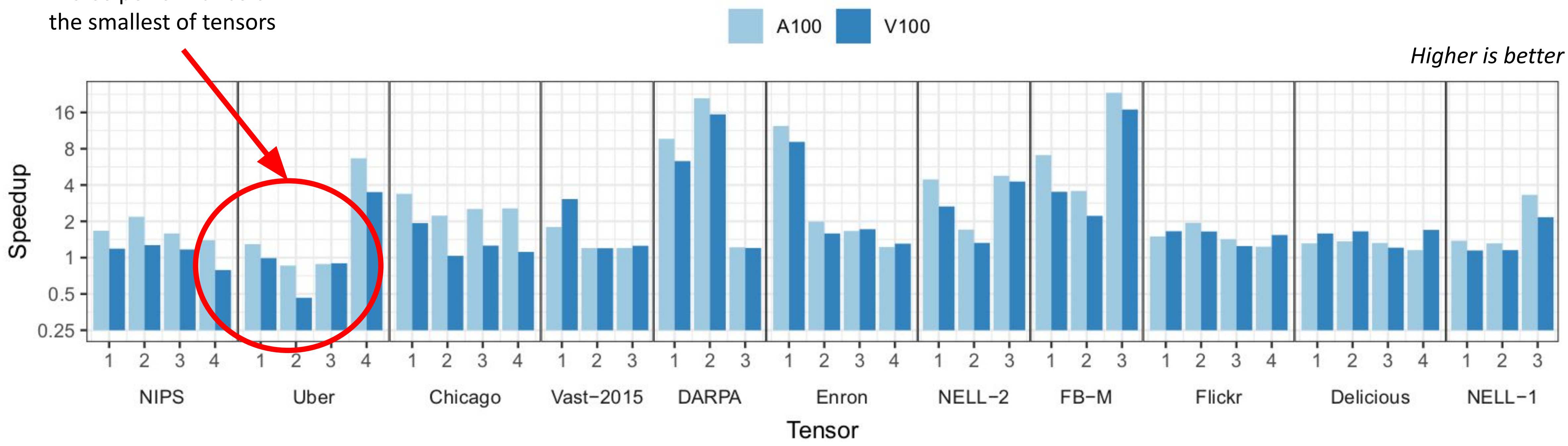
State-of-the-Art Comparison



State-of-the-Art Comparison

Worse performance on
the smallest of tensors

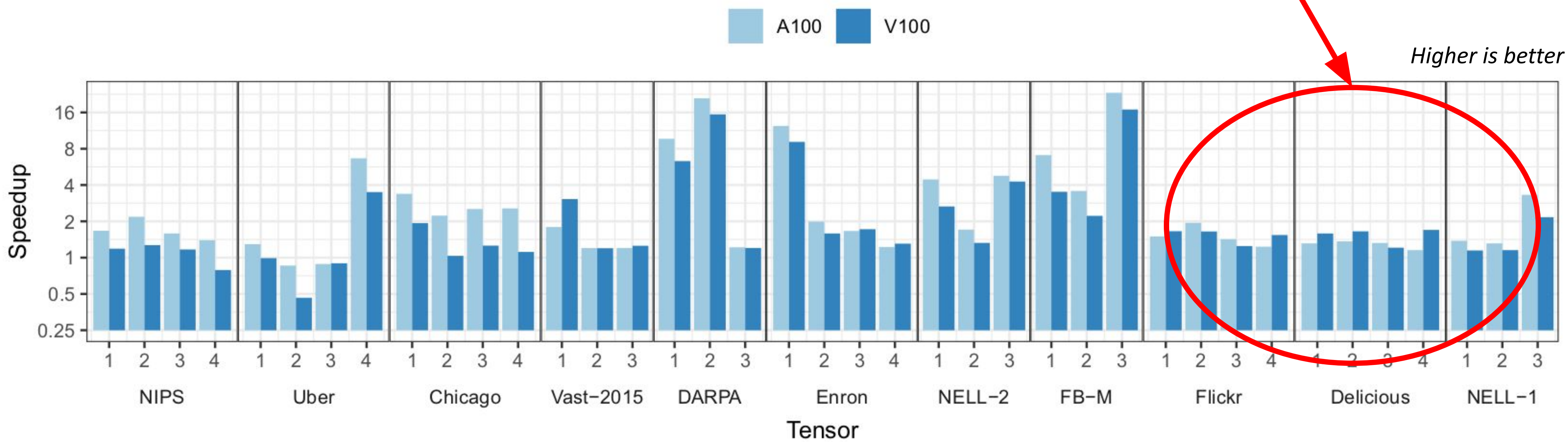
Higher is better



State-of-the-Art Comparison

Consistently better performance
on even the largest of datasets

Higher is better



Memory Traffic Analysis

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)

Memory Traffic Analysis

MM-CSF achieves lower memory traffic by being more compressed overall

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)

Memory Traffic Analysis

BLCO achieves higher memory throughput with hierarchical conflict resolution (using every level of memory)

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)

Memory Traffic Analysis

BLCO also has less performance irregularity across the different modes

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)

Closing Thoughts

- Tensors are becoming prevalent for data analysis
- Efficient tensor formats and algorithms are needed to maximize hardware utilization and throughput
- Our BLCO format outperforms state-of-the-art by up to 33.35x (avg 2.90x) through novel data compression and massively parallel computation
- Future work includes exploring heterogeneous shared-memory computation and extension to distributed-memory platforms

Thank you!