Hashed Coordinate Sparse Tensor Storage with MATLAB

MeiLi Charles

Advisor: Michael Berry

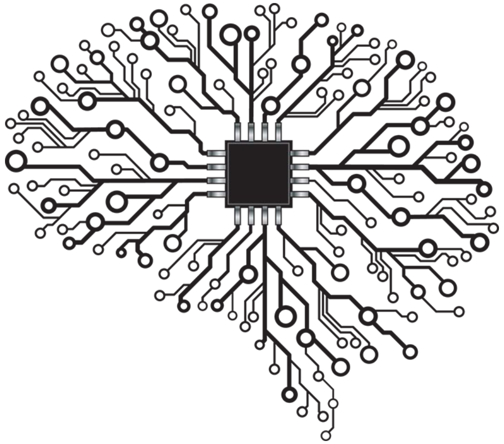
THE UNIVERSITY OF TENNESSEE KNOXVILLE

Outline

1. Background and Motivation
2. Existing Storage Formats
3. Hashed Coordinate (HaCOO) format
4. Evaluation
5. Textual Analysis Application
6. Conclusions and Future Goals
7. Q&A

Motivation

- Many important application domains produce and manipulate massive amounts of high-dimensional data
- This data can also be sparse



Source: Laukemann, 2021

Motivation

- For sparse tensors, we want to reduce storage requirements and eliminate meaningless computations on 0 values.

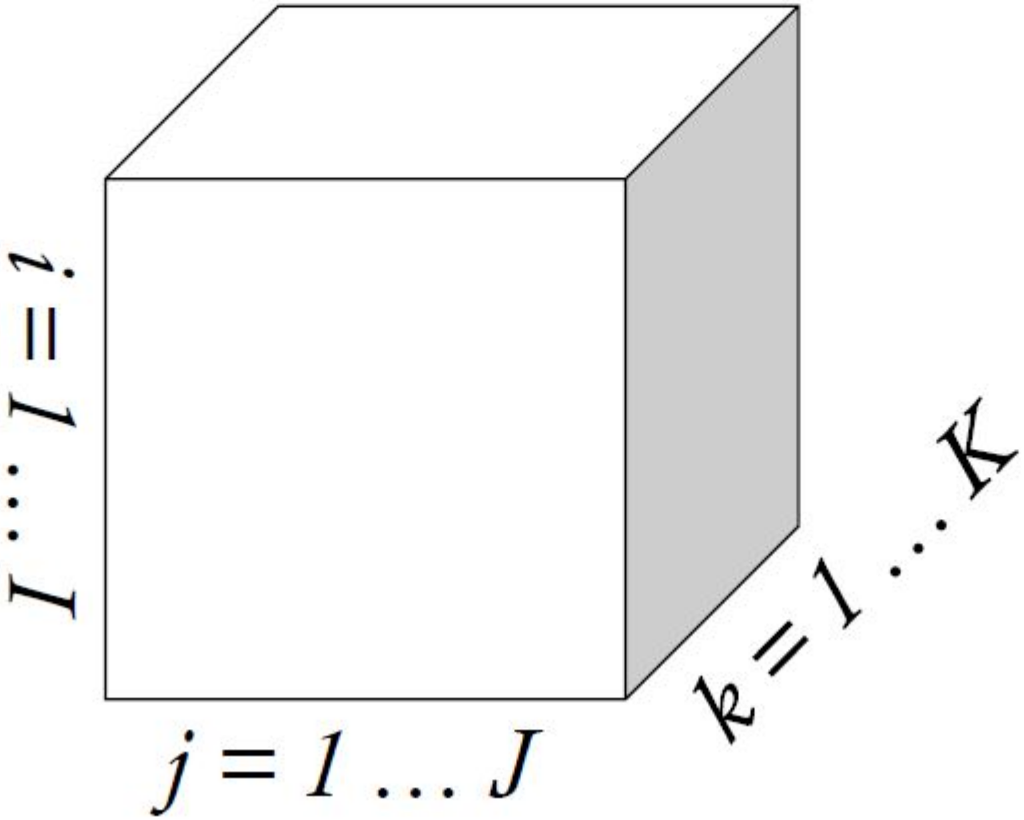
- Challenge: How do we effectively store and compute using high dimensional data?

Background

- A tensor is an *n*-way array

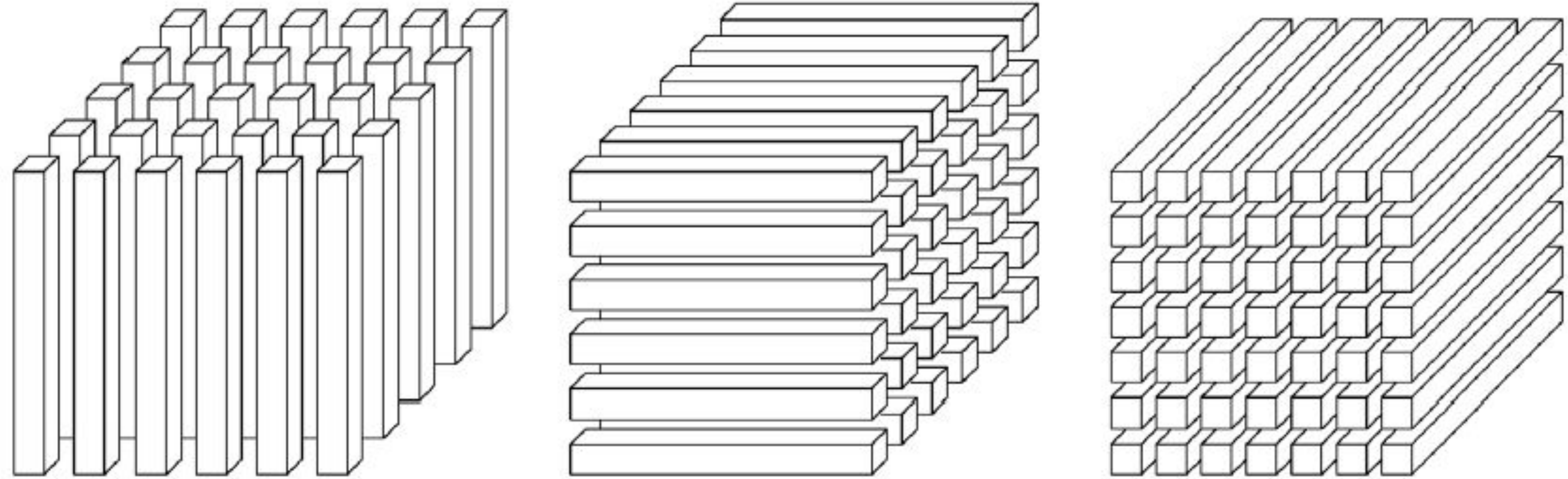
- Each way is referred to as a *mode*

- The number of modes determines a tensor's *order*



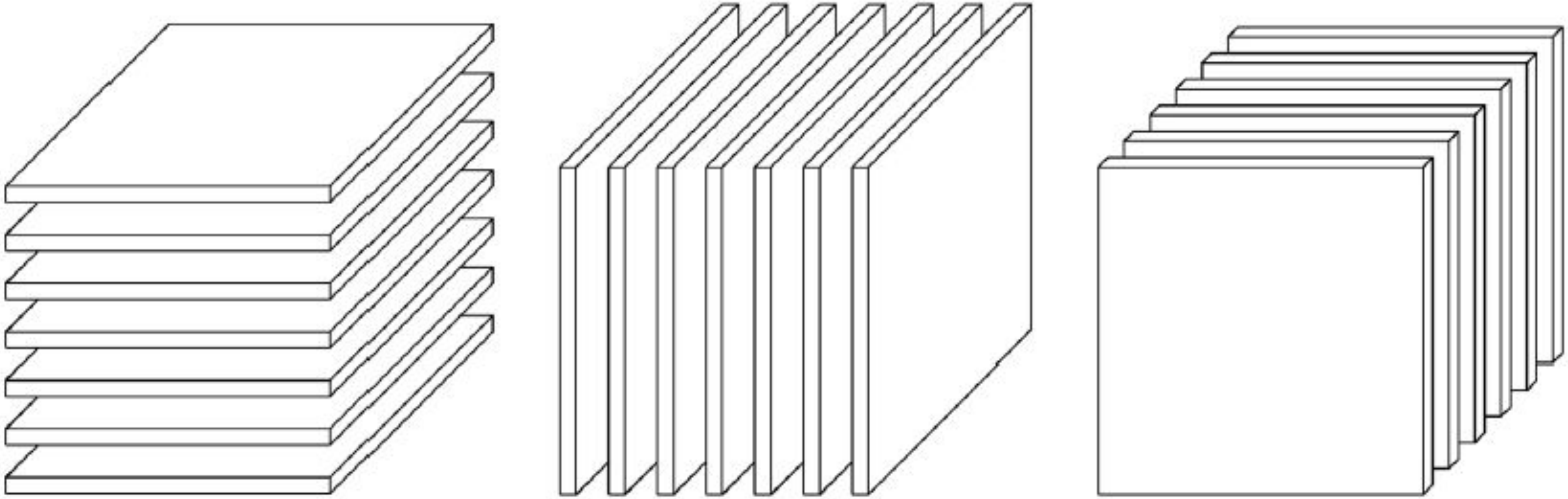
A third-order tensor

Background



(a) Mode-1 (column) fibers: $\mathbf{x}_{:jk}$ (b) Mode-2 (row) fibers: $\mathbf{x}_{i:k}$ (c) Mode-3 (tube) fibers: $\mathbf{x}_{ij:}$

Background



(a) Horizontal slices: $\mathbf{X}_{i::}$ (b) Lateral slices: $\mathbf{X}_{:j:}$ (c) Frontal slices: $\mathbf{X}_{::k}$ (or $\mathbf{X}_k$)

Background

- A tensor can be unfolded, or *matricized* along any of its modes
- $\boldsymbol{\mathcal{X}}_{(n)}$ is a matricized tensor unfolded along the n^{th} mode and is composed of mode-*n* fibers.

Background

Frontal slices:

$$\mathcal{X}_1 = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \quad \mathcal{X}_2 = \begin{bmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{bmatrix}$$

$$\mathcal{X}_{(1)} = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}, \quad \mathcal{X}_{(2)} = \begin{bmatrix} 1 & 2 & 3 & 7 & 8 & 9 \\ 4 & 5 & 6 & 10 & 11 & 12 \end{bmatrix}$$

Background

$$\mathcal{X}_1 = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \quad \mathcal{X}_2 = \begin{bmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{bmatrix}$$

$$\mathcal{X}_{(3)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix}$$

Background

The *Kronecker product* of matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$ is denoted by $\mathbf{A} \otimes \mathbf{B}$, which results in a (IJ) x (KL) matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \dots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \dots & a_{IJ}\mathbf{B} \end{bmatrix}$$

$$= [\mathbf{a}_1 \otimes \mathbf{b}_1 \quad \mathbf{a}_1 \otimes \mathbf{b}_2 \quad \mathbf{a}_1 \otimes \mathbf{b}_3 \quad \dots \quad a_J \otimes \mathbf{b}_{L-1} \quad a_J \otimes \mathbf{b}_L]$$

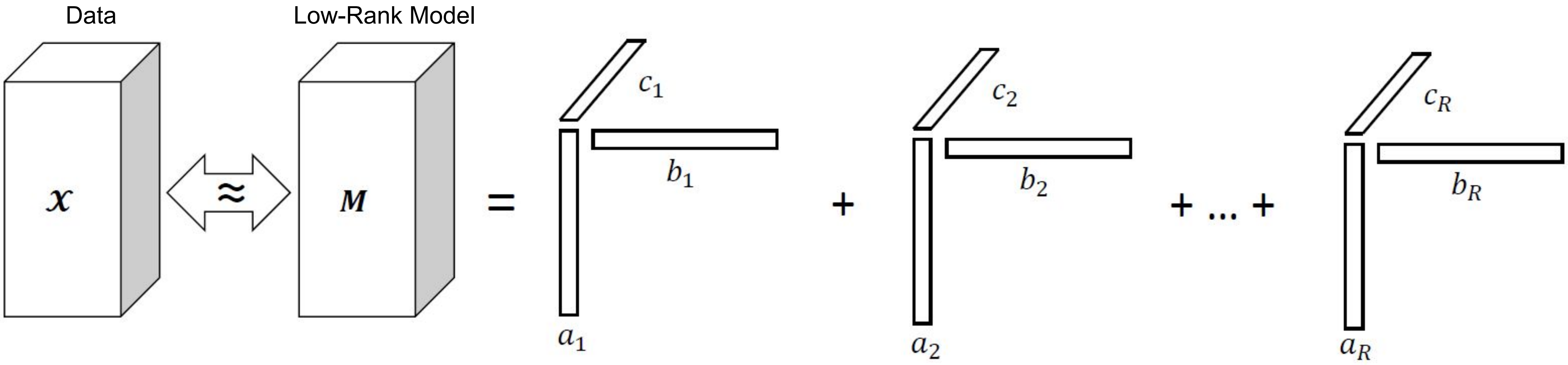
Background

The *Khatri-Rao product* is defined in terms of the Kronecker product. The Khatri-Rao product of matrices $\mathbf{A}^{I \times J}$ and $\mathbf{B}^{M \times J}$ is denoted by $\mathbf{A} \odot \mathbf{B}$, where the resulting matrix is (IM) x J.

$$\mathbf{A} \odot \mathbf{B} = [a_1 \otimes b_1, a_2 \otimes b_2, \ldots, a_n \otimes b_n]$$

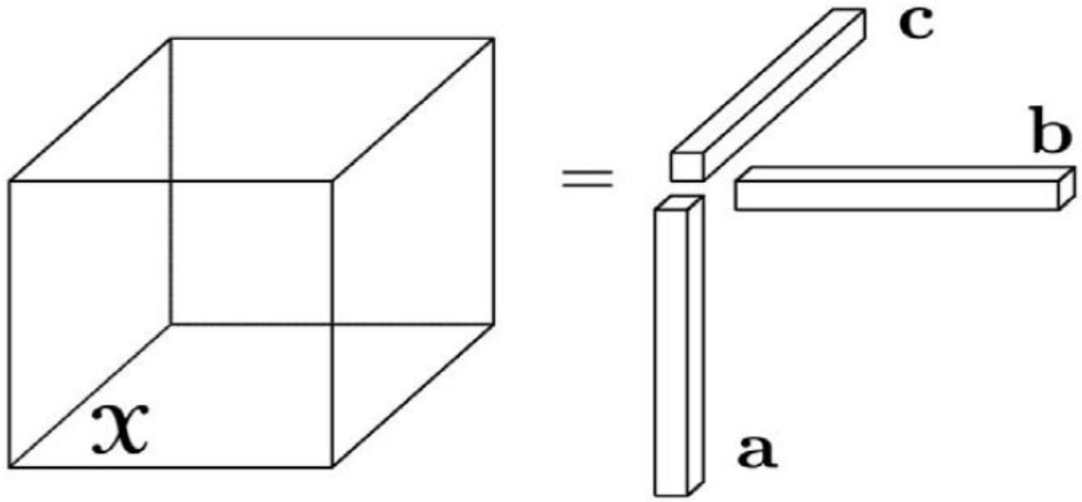
Canonical Polyadic Decomposition (CPD)

- Goal: approximate a mode-*N* tensor X as the sum of *R* rank-1 tensors



Canonical Polyadic Decomposition (CPD)

- An *n*-way tensor is rank-1 if it can be written as the outer product of *n* vectors



Canonical Polyadic Decomposition (CPD)

For the **3-way** case…

- Given 3 vectors:

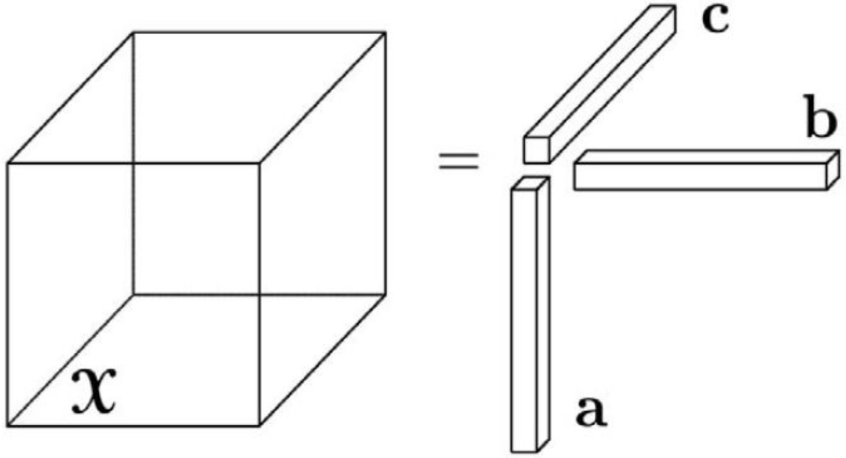
$$\mathbf{a} \in \mathbb{R}^m, \mathbf{b} \in \mathbb{R}^n, \mathbf{c} \in \mathbb{R}^p$$

- Their **outer product** is:

$$\boldsymbol{\mathcal{X}} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \in \mathbb{R}^{m \times n \times p}$$

- Each entry is given by:

$$x(i, j, k) = a(i)b(j)c(k)$$



Canonical Polyadic Decomposition (CPD)

Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ be a three-way tensor.

Compute a CP decomposition with *R* components that best approximates $\boldsymbol{\mathcal{X}}$, i.e., to find:

$$\min_{\hat{\mathcal{X}}} \|\mathcal{X} - \hat{\mathcal{X}}\| \quad \text{with} \quad \hat{\mathcal{X}} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r = \boxed{[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!].}$$

Factor matrices

Canonical Polyadic Decomposition (CPD)

Decompose 3-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ into 3 factor matrices:

$$A \in \mathbb{R}^{I \times R}, B \in \mathbb{R}^{J \times R}, C \in \mathbb{R}^{K \times R}$$

The following produces approximate matricizations of the original tensor:

$$\mathbf{X}_{(1)} \approx \mathbf{A}(\mathbf{C} \odot \mathbf{B})^{\mathsf{T}},$$
$$\mathbf{X}_{(2)} \approx \mathbf{B}(\mathbf{C} \odot \mathbf{A})^{\mathsf{T}},$$
$$\mathbf{X}_{(3)} \approx \mathbf{C}(\mathbf{B} \odot \mathbf{A})^{\mathsf{T}}.$$

Canonical Polyadic Decomposition (CPD)

To compute CP, we will use the Alternating Least Squares (ALS) method.

Method:

- Fix matrix **B** and **C** and solve for matrix **A**
- Fix **A** and **C** to solve for **B**
- Fix **A** and **B** to solve for **C**

Canonical Polyadic Decomposition (CPD)

For example:

Matrices **B** and **C** are fixed, solve for **A**.

$$\mathbf{A} = \min_{\mathbf{A}} \| \boldsymbol{\mathcal{X}}_{(1)} - \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T \|_F^2$$

- The optimal solution is given by:

$$\mathbf{A} = \boldsymbol{\mathcal{X}}_{(1)}[(\mathbf{C} \odot \mathbf{B})^T]^\dagger$$

Canonical Polyadic Decomposition (CPD)

- The following form is preferred:

$$\mathbf{A} = \boldsymbol{\mathcal{X}}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{\dagger}$$

- Only need to calculate the pseudoinverse of an R x R matrix, instead of a (JK) x R matrix

Canonical Polyadic Decomposition (CPD)

CP-ALS:

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

repeat

$$\mathbf{A} = \mathcal{X}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^\dagger$$

$$\mathbf{B} = \mathcal{X}_{(2)}(\mathbf{C} \odot \mathbf{A})(\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A})^\dagger$$

$$\mathbf{C} = \mathcal{X}_{(3)}(\mathbf{B} \odot \mathbf{A})(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A})^\dagger$$

until maximum number of iterations are reached or error less than ε

end

MTTKRP

Algorithm 1: MTTKRP via Sparse Tensor-Vector products [23]

Input: $indI[M]$, $indJ[M]$, $indK[M]$, $vals[M]$ dense matrices $B^{J \times R}$, $C^{K \times R}$

Output: dense matrix $\mathbf{M}^{I \times R}$

for f=0 *to* F **do**

    for z=0 *to* M **do**

        $t[z] = vals[z] * \mathbf{B}(indJ[z], f) * \mathbf{C}(indK[z], f)$;

    end

    for z=0 *to* M **do**

        $\mathbf{M}(indI[z], f) = \mathbf{M}(indI[z], f) + t[z]$;

    end

end

Source: [11]

Datasets

- Retrieved from FROSTT, Formidable Repository of Open Sparse Tensors and Tools

- Publicly available collection of sparse tensor datasets to facilitate reproducible results

Datasets

- **Uber** tensor: data on Uber pickups in New York City
- **NELL-2** tensor: a smaller version of NELL-1, which is pulled from the Never Ending Language Learner knowledge base, part of a machine learning project from Carnegie Mellon University.
- **Enron** tensor: email data that was publicly released during an investigation by the Federal Energy Regulatory Commission

Datasets

- **Chicago** tensor: crime reports in the city of Chicago
- **Nips** tensor: publications from the NeurIPS Conference on Neural Information Processing Systems.
- **LBNL** tensor: anonymized internal network traffic from Lawrence Berkeley National Laboratory (LBNL).

Datasets

tensor	M	dimensions	storage
uber	3.3M	$183 \times 24 \times 1.1K \times 1.7K$	52.9 MB
nell-2	76.9M	$12.1K \times 9.2K \times 28.8K$	1.51 GB
enron	54.2M	$6K \times 5.7K \times 1.2K$	1.2 GB
chicago	5.3M	$6.2K \times 24 \times 77 \times 32$	80 MB
nips	3.1M	$2.5K \times 2.9K \times 14K \times 17$	58.9 MB
lbnl	1.7M	$1.6K \times 4.2K \times 1.6K \times 4.2K \times 868.1K$	55.1 MB

Sparse Tensor Storage Formats

- Can be roughly grouped into list, block, or tree structures
- Dates range from 2009-2021

Sparse Tensor Storage Formats

- Coordinate (COO) format
  - Kolda and Bader, 2009
  - Stores elements in a list
  - "Standard" sparse tensor storage format
  - Sorted or unsorted

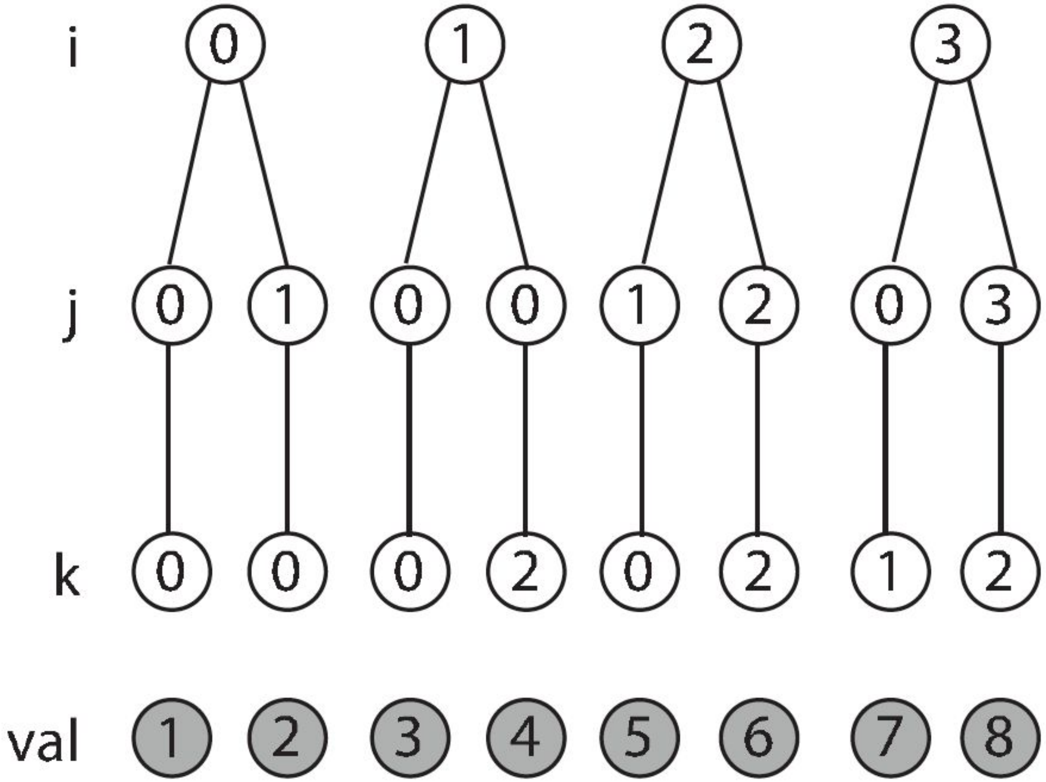
i	j	k	val
0	0	0	1
0	1	0	2
1	0	0	3
1	0	2	4
2	1	0	5
2	2	2	6
3	0	1	7
3	3	2	8

Sparse Tensor Storage Formats

- Compressed Sparse Fiber (CSF) format
  - Smith and Karypis, 2015
  - Tree-based format
  - Mode-specific

SPLATT

(The Surprisingly ParalleL spArse Tensor Toolkit)



Sparse Tensor Storage Formats

- Hierarchical Coordinate (HiCOO) format
  - Li et al., 2018
  - Block-based format
  - Smaller space to search within blocks
  - Smaller indices means less bits to store

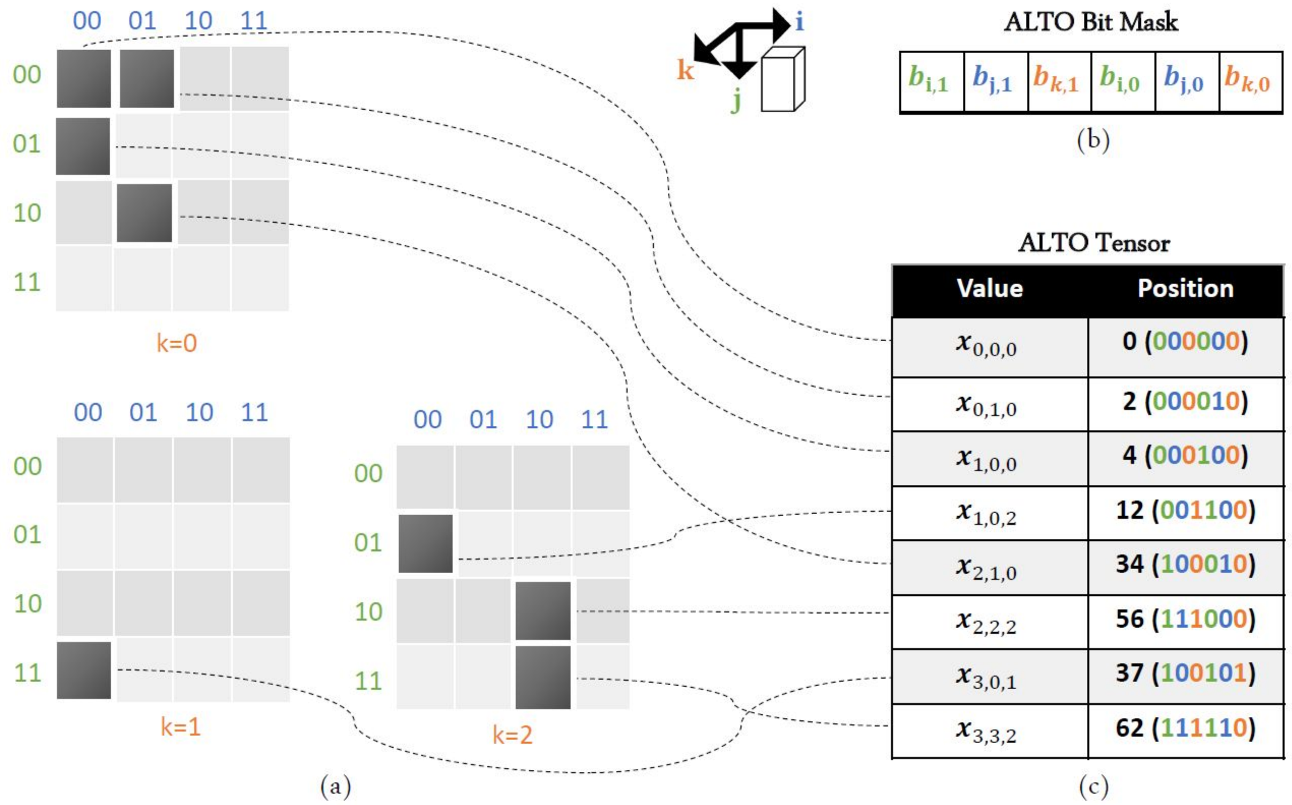
ParTI!
(A Parallel Tensor Infrastructure!)

	bptr	bi	bj	bk	ei	ej	ek	**val**
	0	0	0	0	0	0	0	**1**
B1					0	1	0	**2**
					1	0	0	**3**
B2	3	0	0	1	1	0	2	**4**
B3	4	1	0	0	2	1	0	**5**
					3	0	1	**7**
B4	6	1	1	1	2	2	2	**6**
					3	3	2	**8**

Sparse Tensor Storage Formats

- Adaptive Linearized Storage of Sparse Tensors (ALTO)
  - Helal et. al, 2021
  - Uses bit encoding to order indexes along a line

ALTO library



Why another format?

- Many existing formats:
  - rely on spatial locality of non-zero elements (HiCOO, CSF)
  - do not have a method to insert new tensor values (ALTO, CSF, HiCOO)

Why MATLAB?

- Tensor Toolbox
- Common application that is easily accessible

Hashed Coordinate Format (HaCOO)

- Developed by Robert Lowe, et. al. (2021)

- Separate chaining hash table to store tensor indices and values
  - Uses a low-collision hash function to map indexes into slots within the table
  - Amortized O(1) insertion, update, and retrieval

HaCOO format

i	j	k	val
0	0	0	1
0	1	0	2
1	0	0	3
1	0	2	4
2	1	0	5
2	2	2	6
3	0	1	7
3	3	2	8

COO

Convert each index to Morton code →

morton	val
000000	1
000010	2
100001	3
001010	4
001010	5
111000	6
001101	7
111011	8

Apply Jenkins One-at-a-Time Hash →

morton	val	key
000000	1	1
111000	6	2
		3
		4
		5
		6
		7
111011	8	8
001010 001010	4 5	9
		10
		11
001101	7	12
		13
		14
000010 100001	2 3	15
		16

HaCOO

Hashed Coordinate Format (HaCOO)

- Determine hash values

Algorithm 3: Hash Values

Input: number of buckets in hash table *nbuckets*

Output: sx, sy, sz, $mask$

$bits = \log_2 (nbuckets)$

$sx = \text{ceil}(bits\ /\ 8) - 1$

$sy = 4 * sx\text{-}\ 1$

$sz = \text{ceil}(bits\ /\ 2) - 1$

$mask = nbuckets\text{-}1$

Hashed Coordinate Format (HaCOO)

- Jenkins One-at-a-Time Hash

Algorithm 4: Hash Algorithm

Input: list of non-zero integers $index$

Output: morton value m, hash key k

$m = \text{morton}(index)$

$hash = hash + hash \ll sx$

$hash = hash \oplus hash \gg sy$

$hash = hash + hash \ll sz$

$k = hash \;\&\; mask$

Hashed Coordinate Format (HaCOO)

Example:

idx	morton	step 2	step 3	step 4	k
0, 0, 0	000000	0000000	0000000	000000000	0
0, 1, 0	000010	0000100	0000110	000011110	14
1, 0, 0	000001	0000010	0000011	000001111	15
1, 0, 2	100001	1000010	1100011	111101111	15
2, 1, 0	001010	0010100	0011110	010010110	6
2, 2, 2	111000	1110000	1001000	101101000	8
3, 0, 1	001101	0011010	0010111	001110011	3
3, 3, 2	111011	1110110	1001101	110000001	1

Hashed Coordinate Format (HaCOO)

Tensor	Collision Rate	Mean Probe Depth	Median Probe Depth	Mode Probe Depth	Max Probe Depth
uber	16.43%	1.20	1	1	7
nell-2	26.45%	1.36	1	1	11
enron	37.53%	1.60	1	1	37
chicago	57.27%	25.95	2	1	28
nips	77.31%	4.41	4	4	29
lbnl	89.39%	9.43	1	1	2994

Hashing statistics using the original hash function.

Modified Hashing

- Can the hash function be improved?
- Original algorithm:
  - convert tensor index to Morton code, apply Jenkins hash
- Modified algorithm:
  - concatenate the tensor index, apply Jenkins hash

Modified Hashing

Tensor	Collision Rate	Mean Probe Depth	Median Probe Depth	Mode Probe Depth	Max Probe Depth
uber	17.17%	1.21	1	1	7
nell-2	23.87%	1.31	1	1	9
enron	17.74%	1.22	1	1	8
chicago	26.23%	1.36	1	1	8
nips	16.40%	1.20	1	1	7
lbnl	17.13%	1.21	1	1	7

Results using the modified hash.

HaCOO MATLAB Class

- Create, manipulate, and perform CP decomposition on HaCOO format tensors

- The *htensor* class:
  - hash table, represented as a cell array
  - individual cells contain a matrix of index-value tuples that have hashed into that bucket
  - locations of non-empty buckets
  - modes, number of nonzero elements, maximum chain length, hash parameters, etc.

HaCOO MATLAB Class

- Methods to set/get/search for a tensor index, extract all indexes/values, rehash

- Various class constructors:
  - create a blank HaCOO tensor
  - create a HaCOO tensor from COO tensor or text file

Additional functions to read and write HaCOO tensors from a *.mat* file.

Evaluations

- How do we evaluate HaCOO vs COO in MATLAB?
- Method: simulate “online updates” by inserting elements
- FROSTT tensors are in COO format, pre-sorted and verified to have no duplicate entries

Evaluations

Solution: randomly shuffle the rows of the COO tensor and insert nonzeros one at a time

- Accumulate the time required to insert an element
- Compare wall-clock and CPU time

Building a new COO tensor took days!

- Limit the number of inserted elements to the first *n* entries

Evaluations

Setup:

- Apple MacBook Pro (late 2013)
- 2.4 GHz Dual-Core Intel Core i5 processor
- 128 GB and 4 GB of 1600MHz DDR3L onboard memory

MATLAB functions:

- *tic/toc* to measure elapsed time
- *cputime* to measure total CPU time (summed across threads)

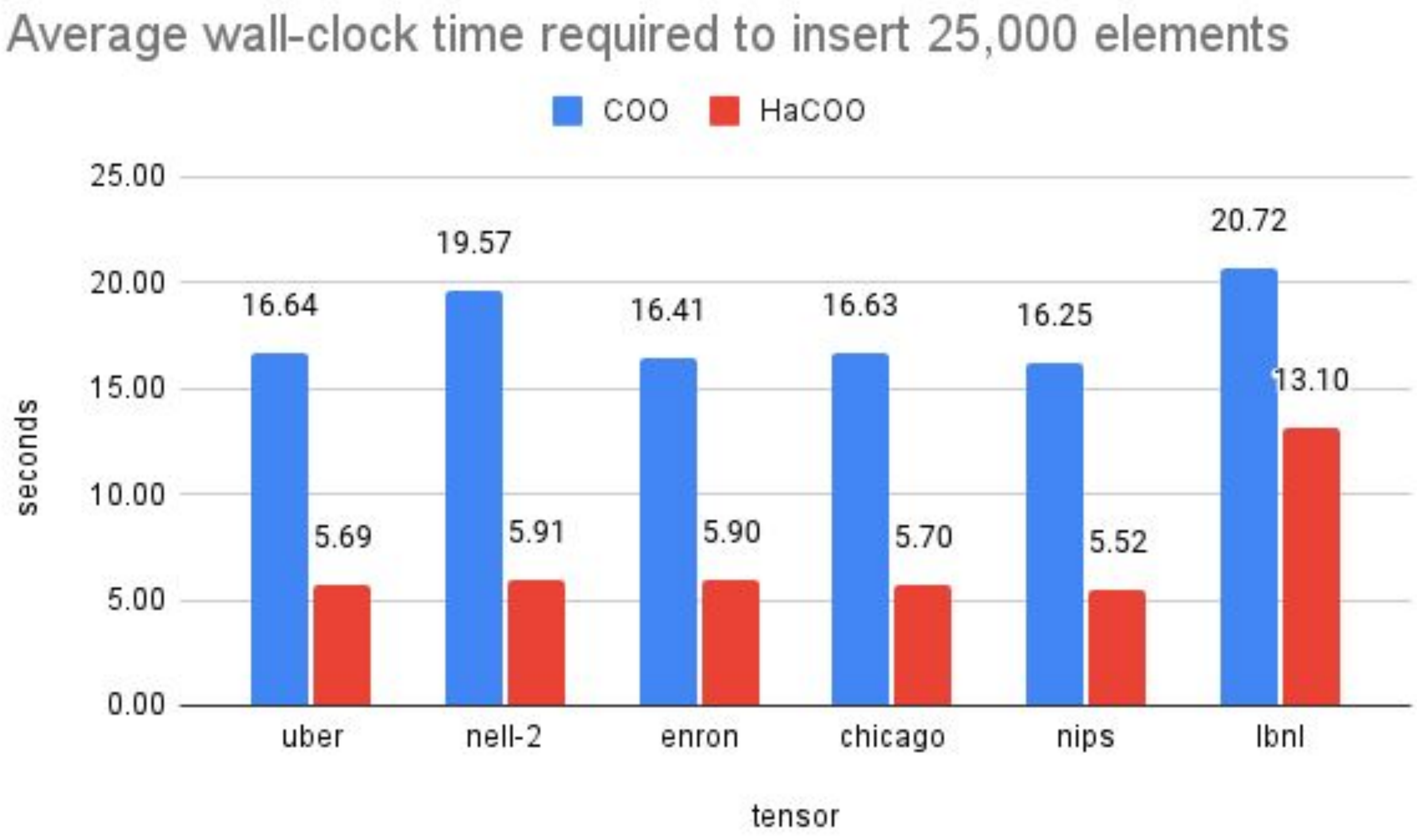
Reported times are averaged over 10 trials

Updating FROSTT Tensors

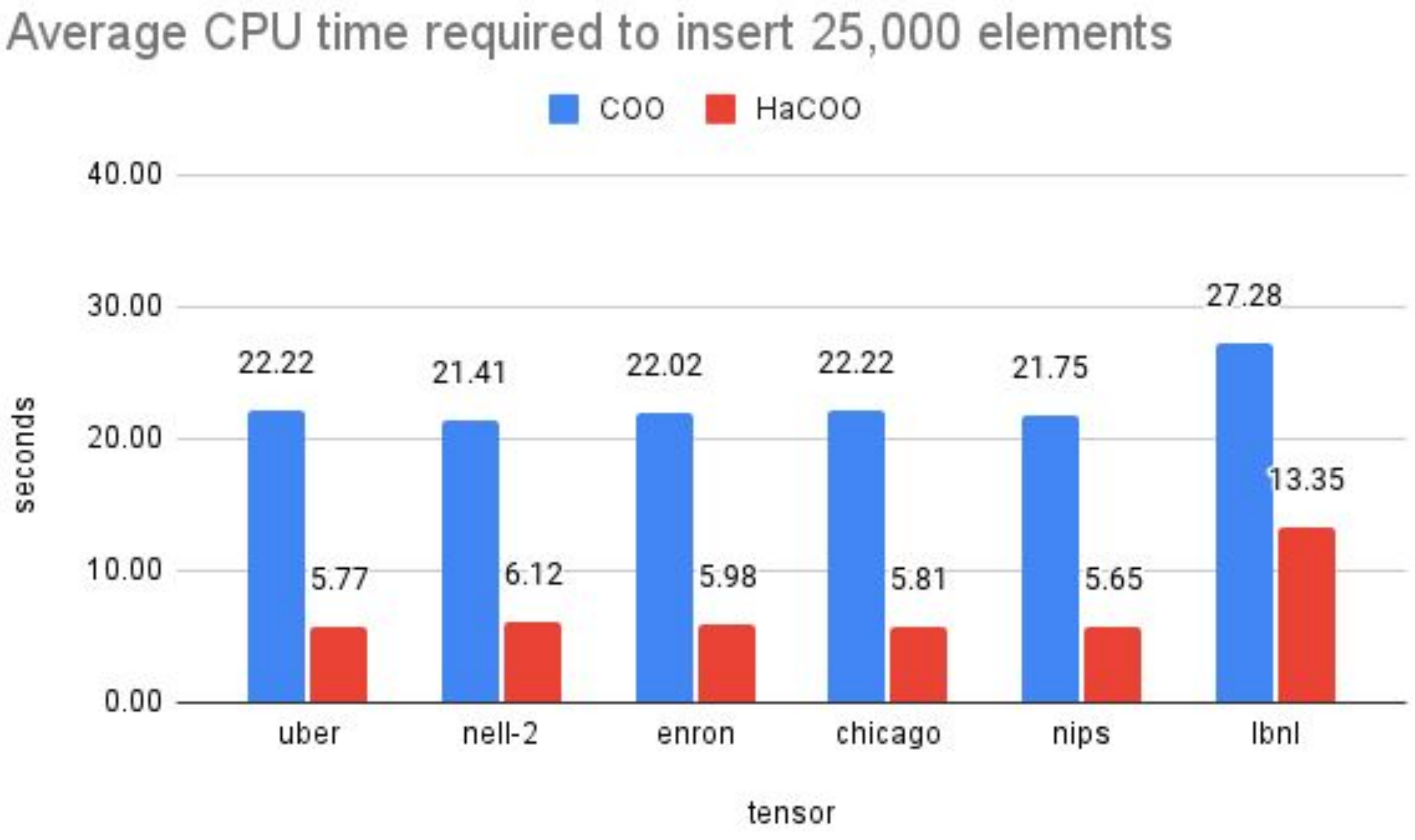
- HaCOO format began to consistently outperform COO once the number of elements inserted reached 25,000.

- Inserting 100,000 random elements from the Uber tensor using HaCOO format yielded around 91-93% reduction in both cumulative wall-clock and CPU time compared to COO format

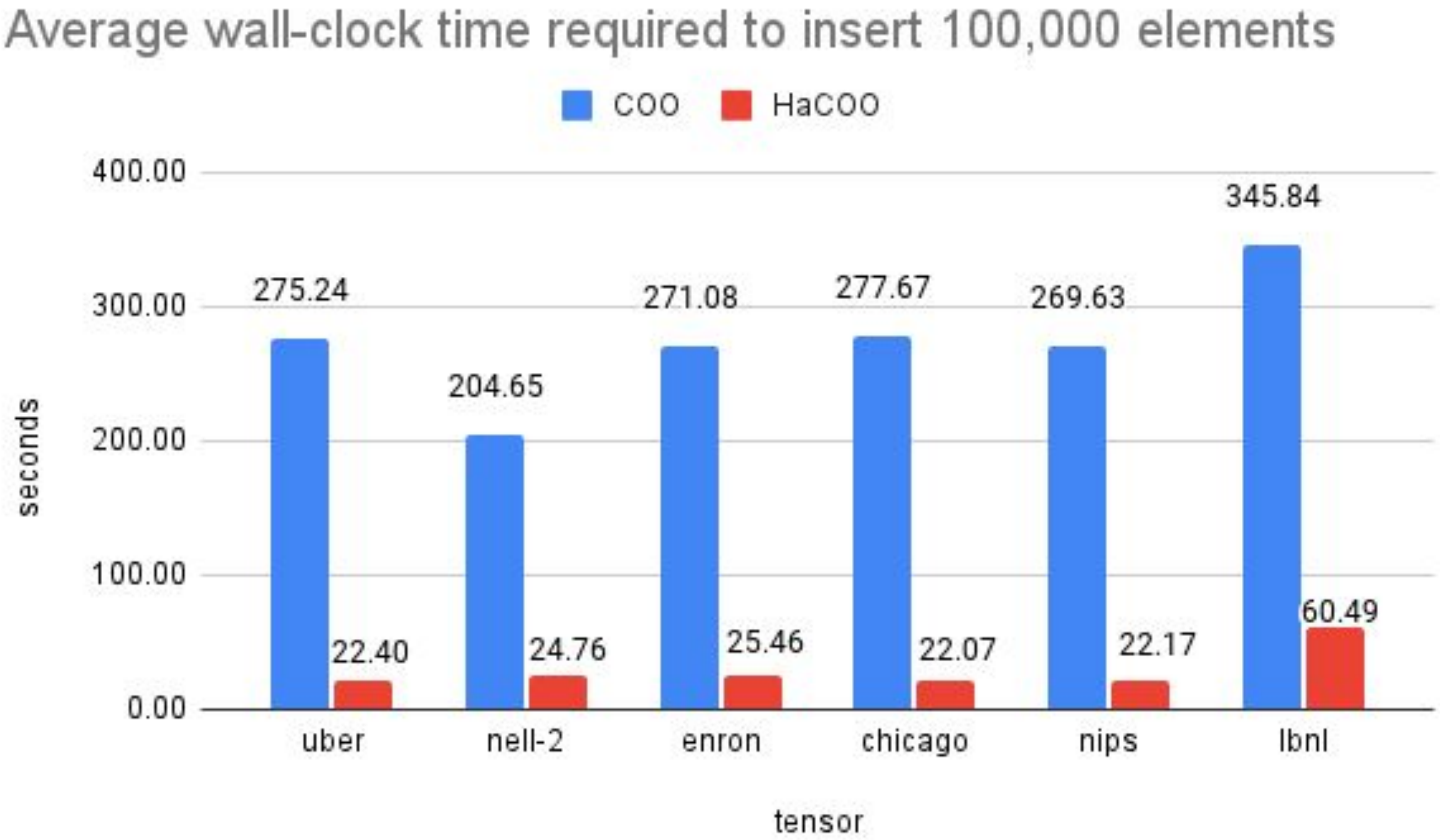
Updating FROSTT Tensors



Updating FROSTT Tensors

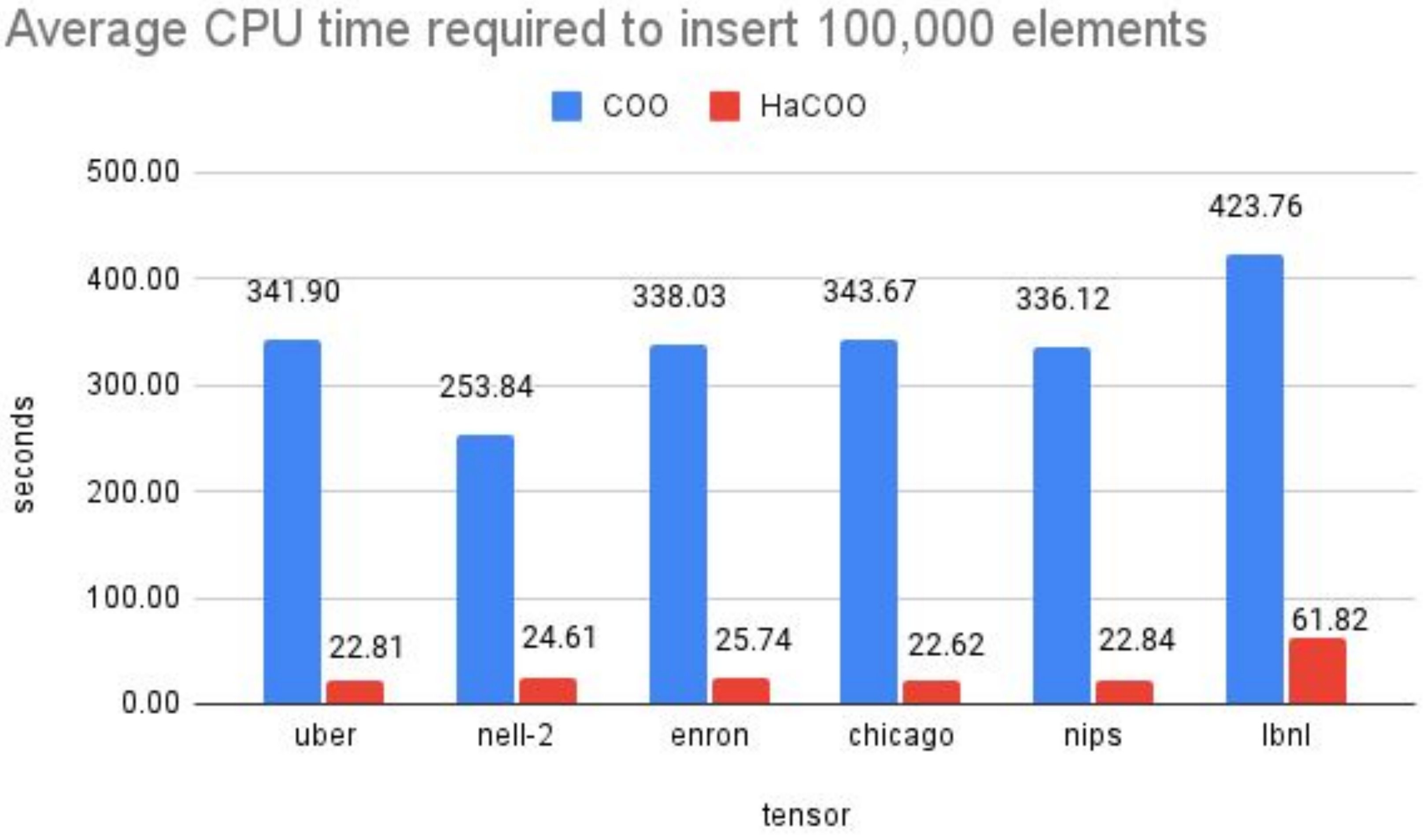


Updating FROSTT Tensors

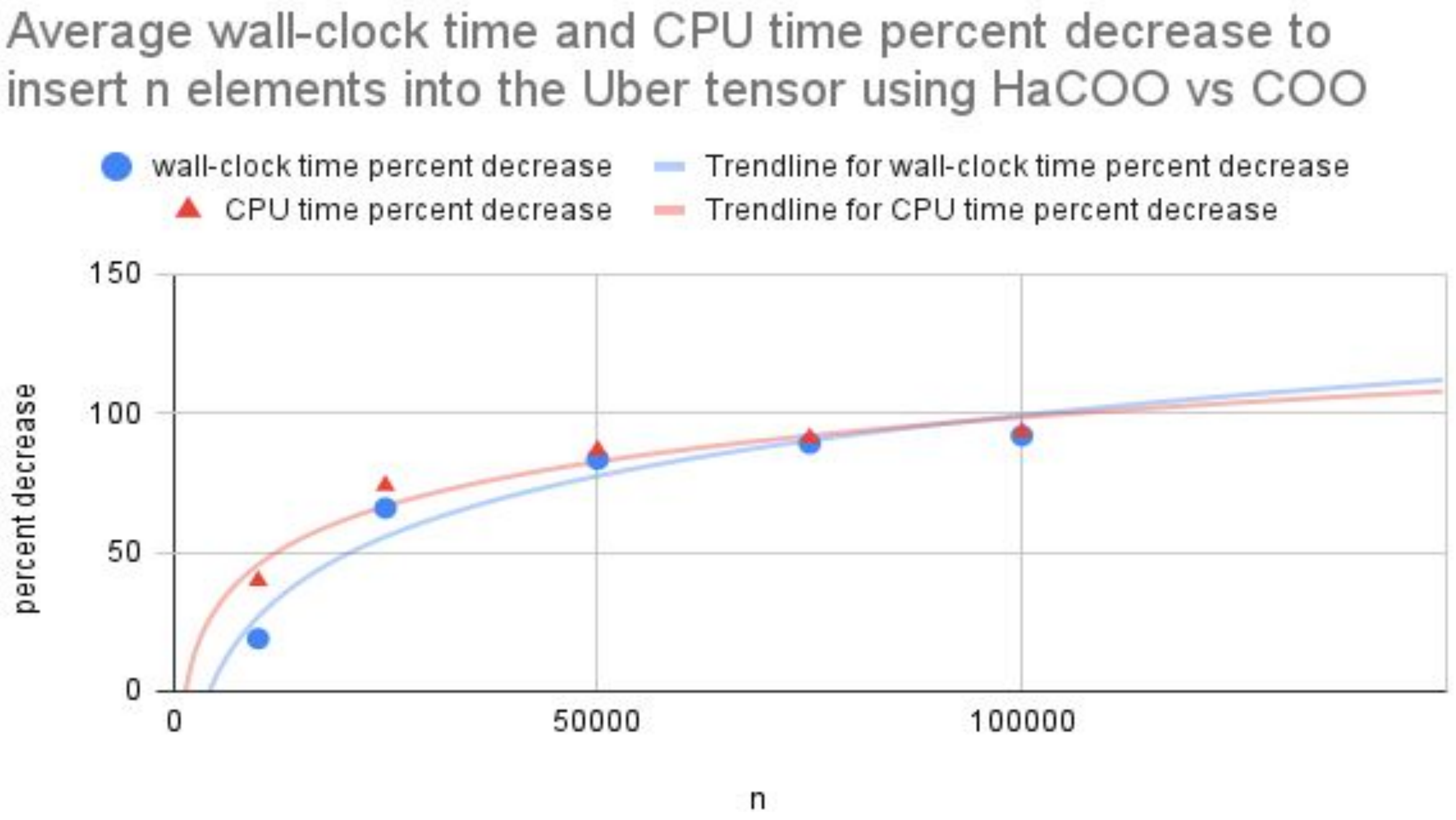


THE UNIVERSITY OF TENNESSEE KNOXVILLE

Updating FROSTT Tensors



Evaluations



Evaluations - MTTRKP

- MTTRKP is typically the main bottleneck of CP decomposition

$$A = \mathcal{X}_{(1)}(C \odot B)(C^T C * B^T B)^\dagger$$

- HaCOO method:
  - extract all elements from nonempty cells from the hash table

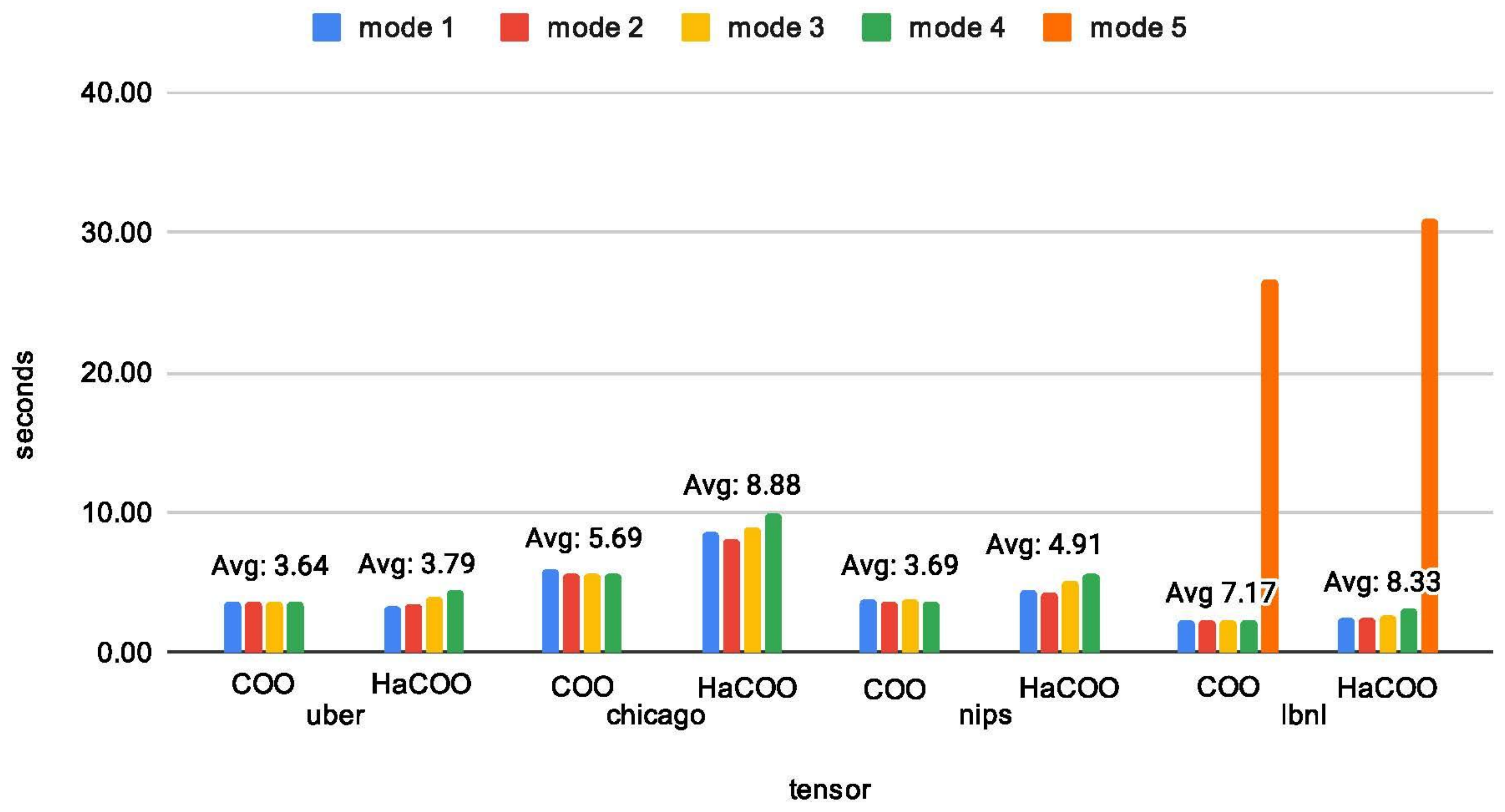
- Time and report averages to compute MTTKRP over every mode

Evaluations - MTTRKP

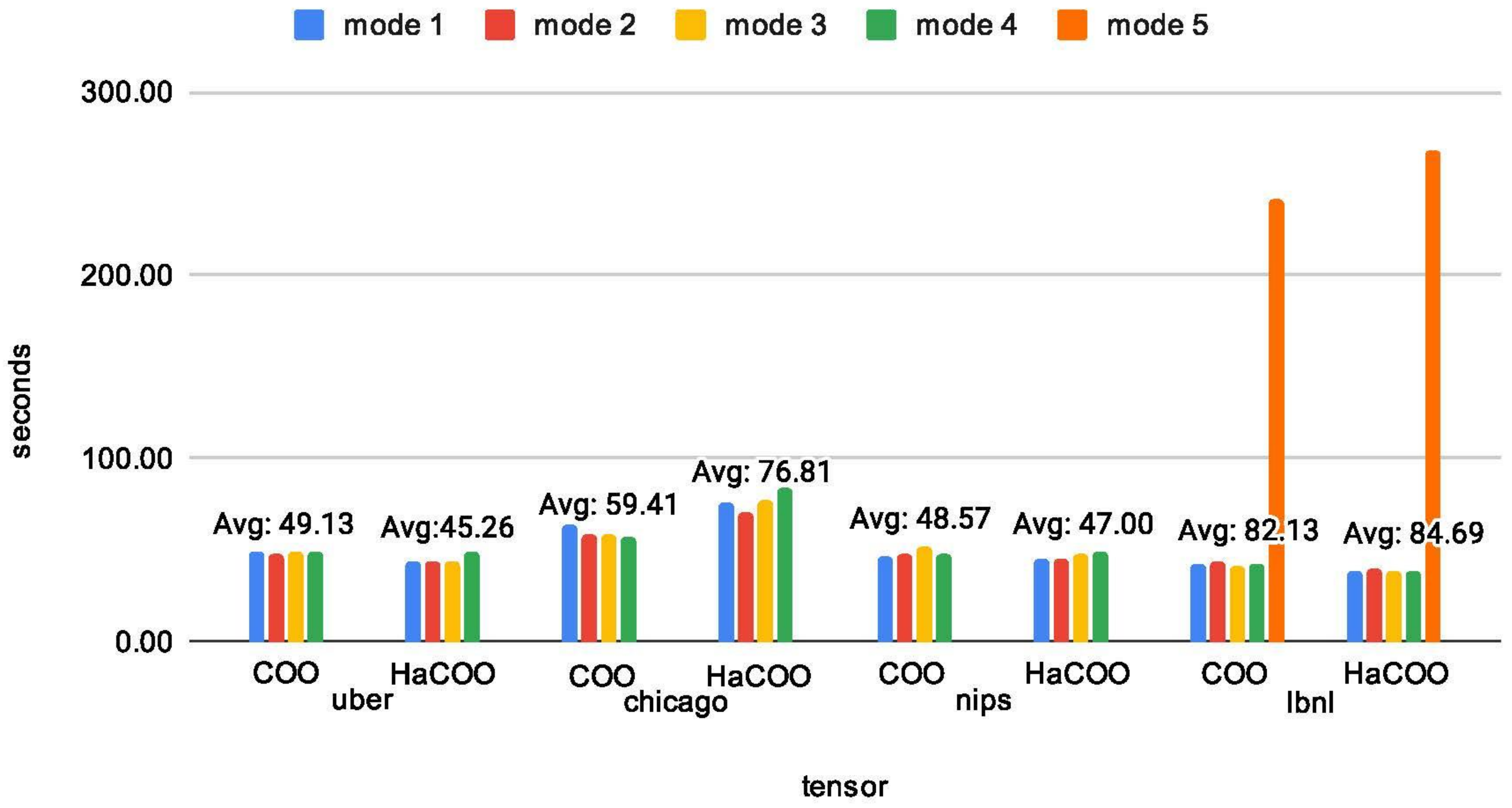
- On average, HaCOO's current MTTRKP method incurs around 26.78% increase in time

- Largest increase observed was over mode 4 of the Chicago tensor
  - 77.88% increase in time to complete
  - Maximum difference in elapsed time over any mode was slightly over 4 seconds

Average wall-clock time required to calculate MTTKRP over all modes



Average CPU time required to calculate MTTKRP over all modes



Text Analysis Application

- Textual Influence model by Lowe (2018)
- Goal:
  - Use sparse tensor decomposition to measure the weight of influence a written document exerts on a target work

First step is to convert all documents into tensors.

Representing Documents as Tensors

Sample document:

The cat jumped on the couch. He yawned and stretched. Then he fell asleep.

Index vocabulary:

1	the	7	yawned
2	cat	8	and
3	jumped	9	stretched
4	on	10	then
5	couch	11	fell
6	he	12	asleep

Representing Documents as Tensors

Counting *n*-grams using a sliding window

Representing Documents as Tensors

The cat jumped on the couch. He yawned and stretched. Then he fell asleep.

1	the	7	yawned
2	cat	8	and
3	jumped	9	stretched
4	on	10	then
5	couch	11	fell
6	he	12	asleep

1, 2, 3	the cat jumped	6, 7, 8	he yawned and
2, 3, 4	cat jumped on	7, 8, 9	yawned and stretched
3, 4, 1	jumped on the	8, 9, 10	and stretched then
4, 1, 5	on the couch	9, 10, 6	stretched then he
1, 5, 6	the couch he	10, 6, 11	then he fell
5, 6, 7	couch he yawned	6, 11, 12	he fell asleep

List of n-grams with corresponding indices

Representing Documents as Tensors

- Unsorted document tensor
- None of the n-grams repeated, so values are 1
- Not all possible *n*-grams will appear, so these tensors are sparse

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

HaCOO vs COO

- A document tensor's modes grows with the size of the vocabulary
  - COO must spend an increasing amount of time searching if the *n*-gram/index already exists
  - Additional time to do an in-order insert
- HaCOO can spend a constant amount of time to insert

Conference Corpus

5 papers on handwritten digit recognition

2 papers on unrelated topics

45,152 words total

5,236 unique words

Num	Document Information
1	Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In Proc. DMKD 2003, pages 211. ACM Press, 2003.
2	Andreas Schlapbach and Horst Bunke. Using hmm based recognizers for writer identication and verication. In Proc. FHR 2004, pages 167172. IEEE, 2004.
3	Yusuke Manabe and Basabi Chakraborty. Identity detection from on-line handwriting time series. In Proc. SMCia 2008, pages 365370. IEEE, 2008.
4	Sami Gazzah and Najoua Ben Amara. Arabic handwriting texture analysis for writer identication using the dwt-lifting scheme. In Proc. ICDAR 2007, pages 11331137. IEEE, 2007.
5	Kolda, Tamara Gibson. Multilinear operators for higher-order decompositions. 2006
6	Blei, David M and Ng, Andrew Y and Jordan, Michael I. Latent dirichlet allocation. 2007
7	Serfas, Doug. Dynamic Biometric Recognition of Handwritten Digits Using Symbolic Aggregate Approximation. Proceedings of the ACM Southeast Conference 2017

Shakespeare Corpus

7 works by William Shakespeare

181,760 words total

15,203 unique words

Num	Document Information
1	"Hamlet, Prince of Denmark by William Shakespeare." Project Gutenberg, Nov. 1998, www.gutenberg.org/ebooks/1524. Accessed 10 July 2023.
2	"Julius Caesar by William Shakespeare." Project Gutenberg, Nov. 1998, www.gutenberg.org/ebooks/1522. Accessed 10 July 2023.
3	"Macbeth by William Shakespeare." Project Gutenberg, Nov. 1998, www.gutenberg.org/ebooks/1533. Accessed 10 July 2023.
4	"A Midsummer Night's Dream by William Shakespeare." Project Gutenberg, Nov. 1998, www.gutenberg.org/ebooks/1514. Accessed 10 July 2023.
5	"Othello, the Moor of Venice by William Shakespeare." Project Gutenberg, Nov. 1998, www.gutenberg.org/ebooks/1531. Accessed 10 July 2023.
6	"The Tragedy of Romeo and Juliet by William Shakespeare." Project Gutenberg, Nov. 1997, www.gutenberg.org/ebooks/1112. Accessed 10 July 2023.
7	"Twelfth Night; Or, What You Will by William Shakespeare." Project Gutenberg, Nov. 1998, www.gutenberg.org/ebooks/1526. Accessed 10 July 2023.

Setup

MATLAB scripts to build a vocabulary and build document tensors (Appendix B)

Time how long to build and decompose all document tensors using CP-ALS (50 components)

- constrained and unconstrained vocabularies

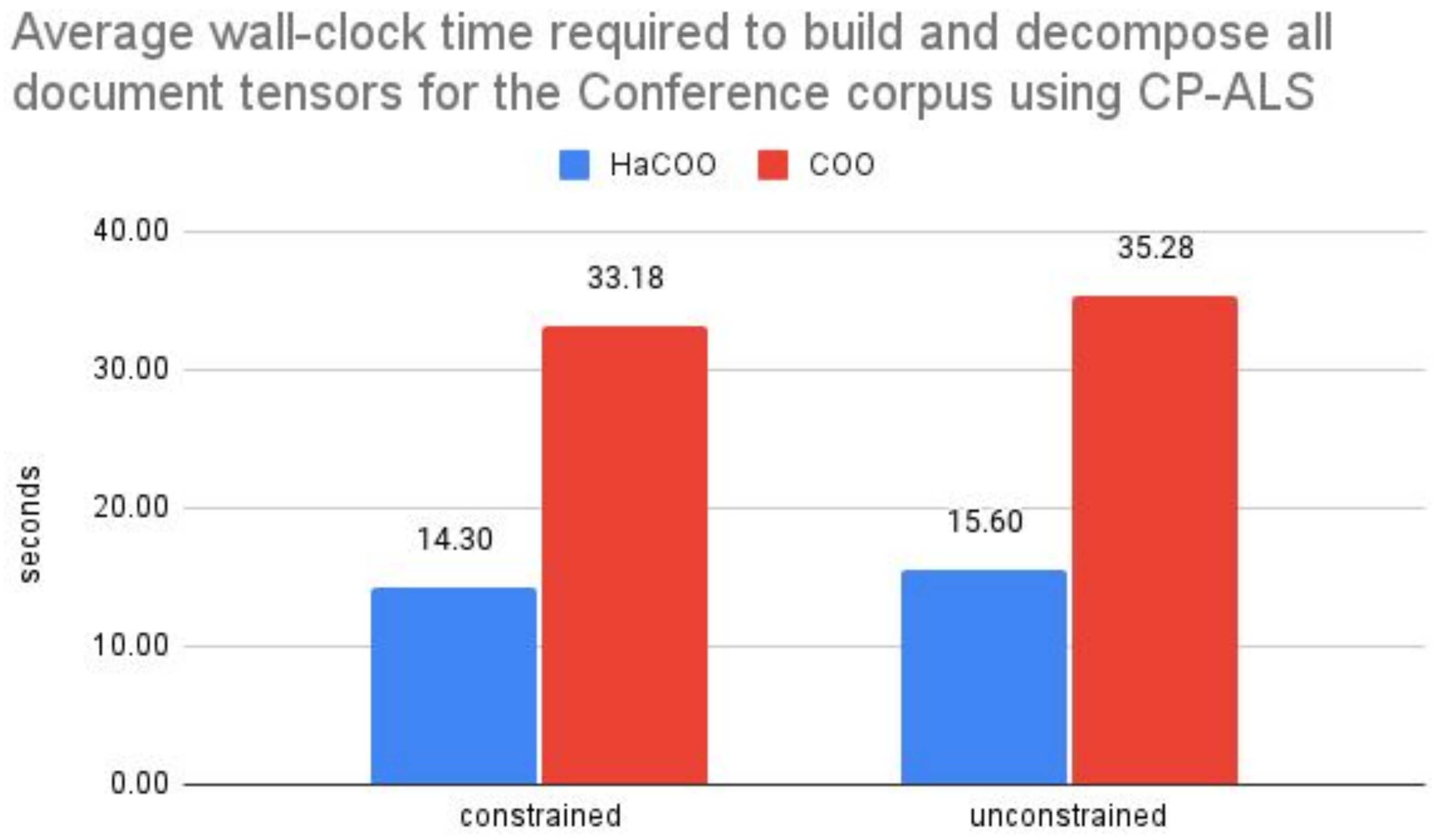
HaCOO: Initial number of buckets was specified to be 1,048,576, or 2^{20}

Results

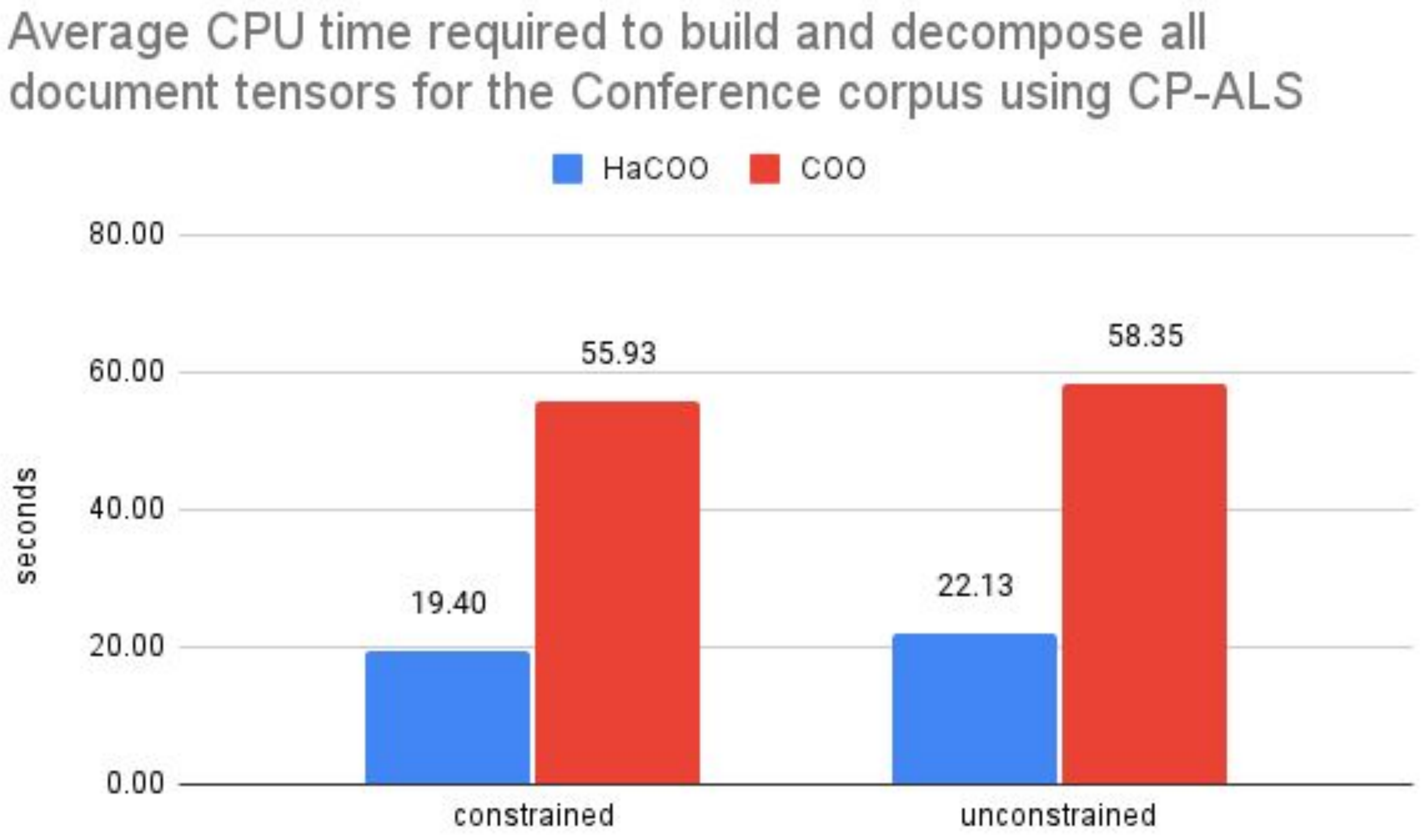
- Conference corpus
  - 44-49% reduction in both wall-clock and CPU time for both the constrained and unconstrained cases

- Shakespeare corpus
  - Constrained:
    - ~14% decrease in wall-clock time
    - ~32% decrease in CPU time
  - Unconstrained:
    - ~72% decrease in wall-clock time
    - ~78% decrease in CPU time

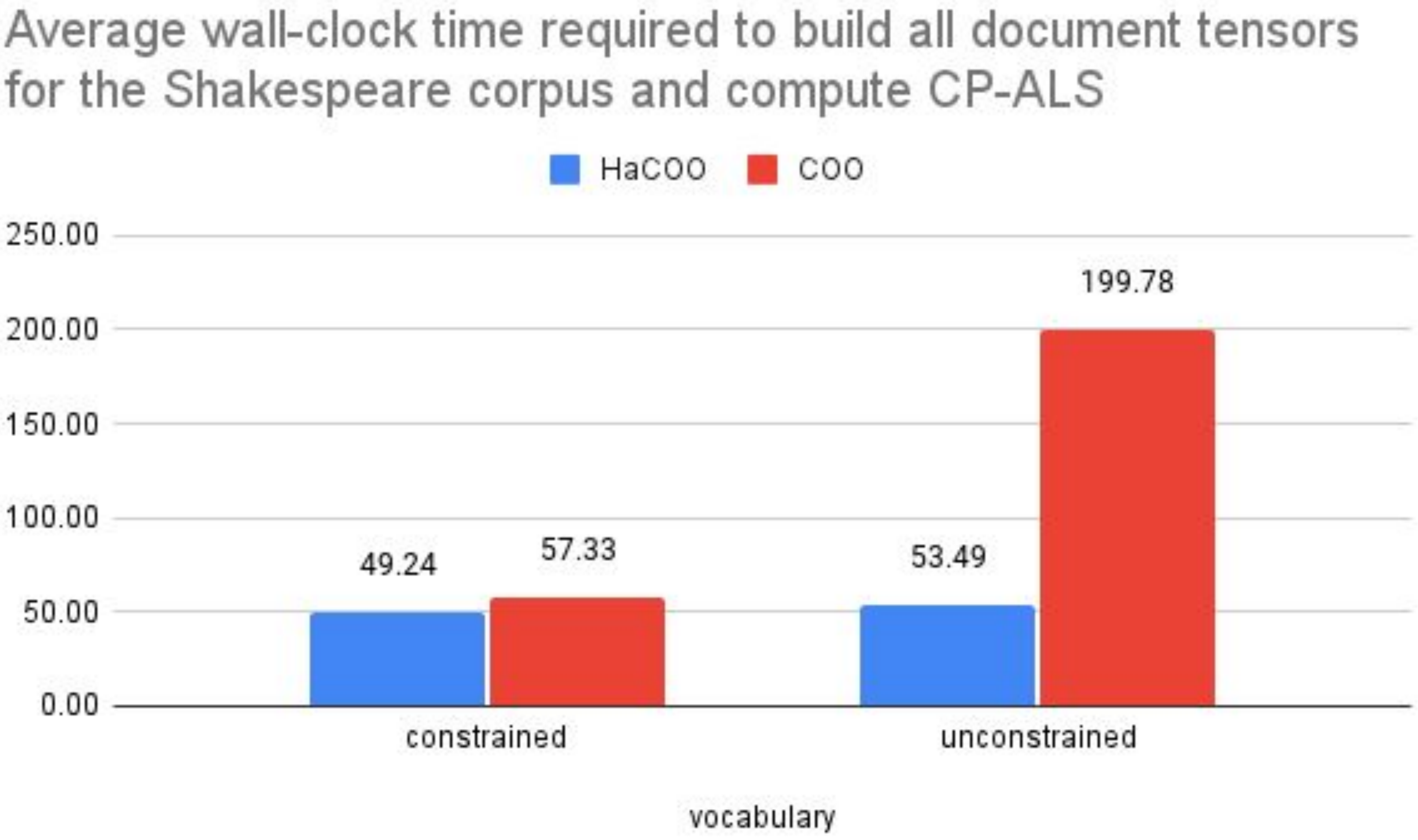
Results - Conference Corpus



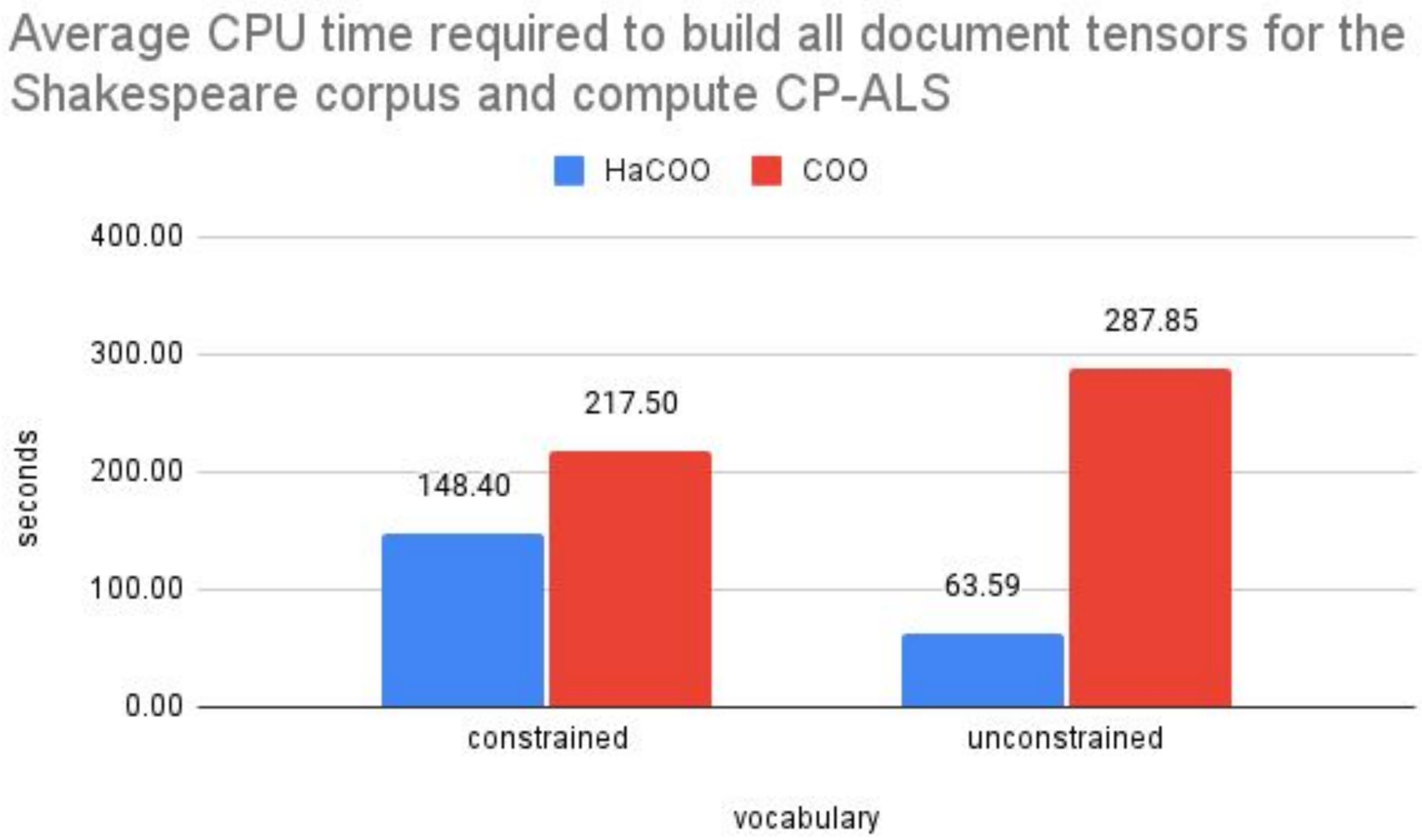
Results - Conference Corpus



Results - Shakespeare Corpus



Results - Shakespeare Corpus



Conclusions

- How to store large, sparse, high-dimensional data?
- Many common sparse tensor storage formats do not allow tensor updates

- HaCOO format benefits:
  - constant time insertion and retrieval
  - MATLAB class to interface with Tensor Toolbox for additional tensor operations without requiring additional hardware or environment setup

Conclusions

- HaCOO outperformed COO format in terms of tensor updates once the number of elements reached a specific threshold.
- CP-ALS was comparable, due to HaCOO's MTTRKP operation incurs a small amount of overhead from extracting tensor elements from the hash table

Future Goals

- MATLAB code clean-up
- What tensor properties contribute to a higher collision rate?
- Further improve hash function
- Workshop on Sparse Tensor Computations
  - University of Illinois Urbana-Champaign
  - October 2023
- Journal article to ACM-TOMS

Q&A

Thank you for your time!

References

1. Choi, Jee, et al. “Blocking Optimization Strategies for Sparse Tensor Computation.” SIAM, July 2017, chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://users.wfu.edu/ballard/SIAM-AN17/choi.pdf.
2. Hastad, J. (1990). Tensor rank is np-complete. Journal of Algorithms, 11(4):644-654.
3. Helal, et al. ALTO: Adaptive Linearized Storage of Sparse Tensors. June 2021, https://doi.org/10.1145/3447818.3461703. Accessed 19 July 2023.
4. Kolda, Tamara G., and Brett W. Bader. “Tensor Decompositions and Applications.” *SIAM Review*, Society for Industrial and Applied Mathematics, 2009, https://epubs.siam.org/doi/10.1137/07070111X.
5. Li, Jiajia, et al. “HICOO: Hierarchical Storage of Sparse Tensors.” *IEEE Xplore*, Nov. 2018, https://ieeexplore.ieee.org/abstract/document/8665782.

References

6. Lowe, Robert, and Michael V. Berry. Using Non-Negative Tensor Decomposition for Unsupervised Textual Influence Modeling. Sept. 2019, https://doi.org/10.1007/978-3-030-22475-2_4. Accessed 19 July 2023.
7. Lowe, Robert, et al. "Hashed-Coordinate Storage of Sparse Tensors." *Hashed-Coordinate Storage of Sparse Tensors*, Nov. 2021, https://sc21.supercomputing.org/proceedings/tech_poster/tech_poster_pages/rpost108.html.
8. "MacBook pro (Retina, 13-Inch, Late 2013) - Technical Specifications." Apple.com, 2013, support.apple.com/kb/sp691?locale=en_US. Accessed 17 July 2023.
9. "Measure the Performance of Your Code." Mathworks.com, The MathWorks, Inc., www.mathworks.com/help/matlab/matlab_prog/measure-performance-of-your-program.html. Accessed 17 July 2023.

References

10. Smith, Shaden, et al. “FROSTT: The Formidable Repository of Open Sparse Tensors and Tools.” Frostt.io, 2017, frostt.io/. Accessed 17 July 2023.
11. Smith, Shaden, et al. “SPLATT: Efficient and Parallel Sparse Tensor-Matrix Multiplication.” IEEE Xplore, May 2015, https://ieeexplore.ieee.org/abstract/document/7161496/?casa_token=19Se-TWP4iAAAAAA%3AnWsOkFJqOzg-clXzamHuUTZ_DnMGqVwdGBSEoEyKGSF2L0_CaZNnqCxsKEjaxBUvCwP80nhV6w.
12. “Tensor Toolbox for MATLAB.” Tensortoolbox.org, 2023, www.tensortoolbox.org/index.html. Accessed 18 July 2023.