Sparse Symmetric Format for Tucker Decomposition

Shruti Shivakumar, Jiajia Li, Ramakrishnan Kannan, and Srinivas Aluru, *Fellow, IEEE*

***Abstract*—Tensor-based methods are receiving renewed attention in recent years due to their prevalence in diverse real-world applications. There is considerable literature on tensor representations and algorithms for tensor decompositions, both for dense and sparse tensors. Many applications in hypergraph analytics, machine learning, psychometry, and signal processing result in tensors that are both sparse and symmetric, making them an important class for further study. Similar to the critical Tensor Times Matrix chain operation (TTMc) in general sparse tensors, the Sparse Symmetric Tensor Times Same Matrix chain (S^3 TTMc) operation is compute and memory intensive due to high tensor order and the associated factorial explosion in the number of non-zeros. We present the novel Compressed Sparse Symmetric (CSS) format for sparse symmetric tensors, along with an efficient parallel algorithm for the S^3 TTMc operation. We theoretically establish that S^3 TTMc on CSS achieves a better memory versus run-time trade-off compared to state-of-the-art implementations, and visualize the variation of the performance gap over the parameter space. We demonstrate experimental findings that confirm these results and achieve up to $2.72\times$ speedup on synthetic and real datasets. The scaling of the algorithm on different test architectures is also showcased to highlight the effect of machine characteristics on algorithm performance.**

***Index Terms*—Compressed storage, sparse tensors, symmetric tensors, tensor times matrix chain.**

TABLE I
SYMBOLS LIST

Symbols	Description
$\mathcal{X}, \mathcal{Y}$	Sparse symmetric tensor
$\mathbf{X}$	Matricization of $\mathcal{X}$
$\mathbf{U}, \mathbf{Y}$	Dense matrices
$\mathbf{i}, \mathbf{j}$	Index tuple of a non-zero in $\mathcal{X}$. E.g., $\mathbf{i} = (i_1, \dots i_N)$
N	Tensor order
I	Mode size of $\mathcal{X}$
$nz(\mathcal{X})$	Non-zero index set of $\mathcal{X}$
$unz(\mathcal{X})$	IOU non-zero index set of $\mathcal{X}$
nnz	#Total non-zeros of $\mathcal{X}$
$unnz$	#Total IOU non-zeros of $\mathcal{X}$
l	Level in CSS
nnz_l	#Level-l nodes of CSS
nnz'_l	#Level-l nodes of CSF
$\mathcal{P}_{\mathbf{i}_l}$	Path from level-1 to l in CSS, $\mathcal{P}_{\mathbf{i}_l} = (i_1, i_2 \dots i_l) \subseteq \mathbf{i}$
$\mathcal{T}_i^{SG}$	Subgraph of the symbolic parent graph rooted at node i at level-$(N-1)$
$SP(\mathcal{P}_{\mathbf{i}_l})$	Set of l paths indicating the symbolic parents of node i_l at level l

I. INTRODUCTION

TENSORS are higher dimensional generalizations of matrices, and are used to represent multi-dimensional data. Symmetric tensors are an important class of tensors, arising in diverse fields such as psychometry, signal processing, machine learning, and hypergraph analytics [1], [2], [3], [4], [5], [6]. Estimating means of Gaussian graphical models and independent component analysis often utilize symmetric tensors for decompositions [3], [7], [8]. Hypergraphs, generalizations of graphs which allow edges to span multiple vertices, have become ubiquitous in understanding real world networks and multi-entity interactions [9], [10]. Affinity relations in a hypergraph can be represented as a high-order adjacency tensor which is sparse and symmetric [3], [11], [12], [13]. While mathematical research on symmetric tensors is longstanding [14], [15], [16], [17], emerging massive data in these applications has sparked the demand for scalable, efficient algorithms that utilize advances in numerical linear algebra, numerical optimization, as well as high performance computing. State-of-the-art tensor libraries [18], [19], [20] incorporate high performance tensor methods for general sparse tensors; however, to the best of our knowledge, they lack specialized algorithms for sparse tensors that are symmetric. Exploiting symmetry can lead to faster and memory-efficient algorithms than are possible for the general case.

An important mathematical technique in dimensionality reduction and hypergraph clustering is tensor decomposition, and we study the popular Tucker decomposition [21] in this work. A key kernel in symmetric Tucker decomposition is the *Sparse Symmetric Tensor Times Same Matrix chain* (S^3 TTMc) operation[1], a specialization of the fundamental tensor times matrix chain operation (TTMc) for symmetric tensors. In symmetric Tucker decomposition, instead of different matrix factors in the chain, the same matrix is repeatedly used.

In high-performance algorithms for sparse tensor decomposition, a balance between two desirable but competing goals is pursued - (i) compressed storage format to represent the sparse tensor, and (ii) efficient computation based on it to perform the decomposition. For general sparse tensors, the two goals could align well [18], [22], [23], [24] since when a sparse tensor is stored compactly, the memory footprint of the tensor - potentially a dominating factor of the memory traffic during computation - is reduced. Moreover, superior sparse tensor compression

Manuscript received 14 May 2022; revised 6 February 2023; accepted 20 March 2023. This research was supported in part by NVIDIA Corporation, and in part by the U.S. Department of Energy and Pacific Northwest National Laboratory under Grant 129254. Recommended for acceptance by B. Ucar. *(Corresponding author: Shruti Shivakumar.)*

Shruti Shivakumar and Srinivas Aluru are with the Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: sshivakumar9@gatech.edu; aluru@cc.gatech.edu).

Jiajia Li is with the North Carolina State University, Raleigh, NC 27695 USA (e-mail: jli49@wm.edu).

Ramakrishnan Kannan is with the Oak Ridge National Laboratory, Oak Ridge, TN 37830 USA (e-mail: kannanr@ornl.gov).

Digital Object Identifier 10.1109/TPDS.2023.3263124

[1]For notational convenience, we call S^3 TTMc in lieu of SSTTSMc.

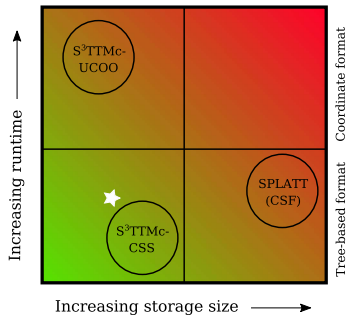

Fig. 1. Trade-off between run-time and storage size for proposed S^3 TTMc-CSS computation to different algorithms and storage formats.

can co-exist with better data locality. For example, a compact Compressed Sparse Row (CSR) representation of a sparse matrix demonstrates better data locality from the reuse of row indices. A similar phenomenon occurs in Compressed Sparse Fibers (CSF) [18] and Hierarchical COOrdinate (HiCOO) [22] formats from different perspectives.

For sparse symmetric tensors, efficient computation and compressed storage format are *contending characteristics*, especially for high dimensional data. Fig. 1 illustrates the trade-off between efficiently computing S^3 TTMc and the size of compressed storage data structures for symmetric tensors. The *Unique COOrdinate* (*UCOO*) representation achieves compact storage by saving the coordinates (indices) along with the value of each non-zero exactly once. However, the compression comes at the price of slow S^3 TTMc computation, as retrieving all index permutations of every non-zero increases redundant computation and limits parallelism. On the other hand, by saving all index permutations of each non-zero, we can overcome the computation challenges in $UCOO$. Thus, S^3 TTMc becomes a general TTMc where state-of-the-art high performance libraries, such as SPLATT [18] and ParTI! [19], can be leveraged. Unfortunately, this approach leads to $\mathcal{O}(N!)$ increase in storage, where N is the tensor order. With the pervasiveness of high order adjacency tensors representing massive hypergraphs, this memory overhead is formidable. For example, an order-14 sparse symmetric tensor generated from Amazon reviews [25] (Details in Section VI-B) causes $14! = 8.7 \times 10^{10}$ factor increase in storage to save all index permutations. Thus, one is forced to choose between efficient algorithms that use storage formats that cannot be sustained for high-order tensors ($SPLATT$), and efficient storage formats ($UCOO$) that do not permit efficient algorithms.

In our previous work [26], we proposed a *computation-aware compact storage format*, termed the Compressed Sparse Symmetric (CSS) format, which has storage requirement of the same order as *UCOO* while being able to perform S^3 TTMc more efficiently than state-of-the-art TTMc implementations, such as SPLATT, as shown in Fig. 1. We also presented a novel shared-memory parallel algorithm S^3 TTMc-CSS to efficiently perform S^3 TTMc using CSS.

In this work, we provide further insight into the construction and properties of the CSS, and inspect its space complexity bounded by 2^N, asymptotically smaller than $N!$. We investigate the cost model used to assess the performance of S^3 TTMc-CSS, and visualize the dependence of S^3 TTMc rank and tensor order on the performance improvement of S^3 TTMc-CSS over SPLATT. Finally, we demonstrate through experiments the performance of our parallel algorithm achieves the twin advantages illustrated in Fig. 1. The scaling performance of S^3 TTMc-CSS on different architectures helps understand the effect of cache sizes and memory bottlenecks on parallelism in S^3 TTMc-CSS.

Our main contributions are summarized as follows:

- We propose the novel computation-aware CSS structure for sparse symmetric tensors. (Section IV)
- We design and implement an efficient multi-core parallel S^3 TTMc algorithm, called S^3 TTMc-CSS, based on the CSS format. (Section V)
- We perform a thorough cost analysis of S^3 TTMc-CSS and outline a detailed comparison to baseline implementations and their formats. (Sections IV-A and V-E)
- Our S^3 TTMc-CSS bridges the gap between efficient S^3 TTMc computation and compact storage - S^3 TTMc-CSS is (i) up to 2.72× faster than SPLATT while requiring up to five orders of magnitude lesser memory, and (ii) at least two orders of magnitude faster than *S^3 TTMc-UCOO* while only requiring up to 6× more memory than *UCOO*. (Section VI)
- We analyze the scaling characteristics of S^3 TTMc-CSS on different test architectures using hardware counters and roofline model, and provide insight into scaling bottlenecks. (Section VI-D)
- We report the Tucker decomposition of an order-14 tensor from a real world hypergraph. (Section VI-G)

II. Background

A. Sparse Symmetric Tensors

An order-N symmetric tensor $\mathcal{X}$ has N modes or dimensions, and is a general tensor with the special property that non-zero values, $\mathcal{X}_{(i_1,i_2,\ldots i_N)}$ remains unchanged under any permutation of its indices. We observe sparse symmetric tensors in several contexts, one of which are those originating from hypergraphs. A N-uniform hypergraph on I vertices, where every hyperedge spanning multiple vertices has the same cardinality N, can be represented as an order-N adjacency tensor $\mathcal{X} \in \mathbb{R}^{I \times I \times \ldots I}$ That is, for every hyperedge $e = \{v_{i_1}, v_{i_2}, \ldots, v_{i_N}\} \in E$ with weight $w(e)$, $\mathcal{X}_{\sigma(\mathbf{i})} = w(e)$, where $\sigma(\mathbf{i})$ denotes any of the $N!$ permutations of the index tuple $\mathbf{i} = (i_1, \ldots, i_N)$. We refer to a non-zero $\mathcal{X}_{\mathbf{i}} \in unz(\mathcal{X})$ as an *index-ordered unique (IOU) non-zero* if the index set $\mathbf{i}$ is ordered $(i_1 < i_2 < \ldots < i_N)$. Thus, $nz(\mathcal{X})$ is divided into a set of equivalence classes, $unz(\mathcal{X})$ of size $unnz = \frac{nnz}{N!}$, under the permutation relation σ. The IOU non-zero $\mathcal{X}_{\mathbf{i}}$ represents an equivalence class containing all permutations $\sigma(\mathbf{i})$ of the index set $\mathbf{i}$. A subtensor of a tensor $\mathcal{X}$ is obtained by fixing indices along some dimensions while varying along the others, denoted by ':'. The matricization of symmetric tensor $\mathcal{X}$ along any mode flattens/unfolds the tensor into a matrix $\mathbf{X} \in \mathbb{R}^{I \times I^{N-1}}$, and is obtained by arranging the fibers of the tensor as columns of $\mathbf{X}$. Note that for symmetric

Algorithm 1. Symmetric HOOI for Tucker Decomposition

1: **Output**: Core tensor $\mathcal{C}$, and orthonormal matrix $\mathbf{U}$
2: **while** $\mathcal{C}$ not converged **do**
3: $\mathcal{Y} = \mathcal{X} \times_{-1} [\mathbf{U}]$ //S^3 TTMc
4: $\mathbf{U} \leftarrow R$ left singular vectors of matricized $\mathbf{Y}$
5: $\mathcal{C} = \mathcal{Y} \times_1 \mathbf{U}$
6: **end while**

tensors, the matricization along any tensor mode yields the same matrix, as the fibers remain the same along all modes. Following Kolda and Bader [21], we denote vectors using bold lowercase letter (e.g., **i**, **j**), matrices using bold uppercase letters (e.g., $\mathbf{U}$), and tensors using bold calligraphic letters (e.g., $\mathcal{X}$).

B. Sparse Tensor Formats

Multiple compressed data structures, with varying levels of compactness, have been proposed for general sparse tensors. Popular among these include COO [21], CSF [18], MM-CSF [24], F-COO [23], HiCOO [27] and ALTO [28] formats The simplest storage format for sparse tensors is the COOrdinate (COO) format. COO stores each non-zero as a tuple $(i_1, i_2, \ldots i_N; v)$, where $i_j, j = 1, \ldots, N$ is an index coordinate and v is the non-zero value. The Compressed Sparse Fiber (CSF) format is a higher order analogue to the Compressed Sparse Row (CSR) sparse matrix format. It is a compressed storage format structured as a tree, where every level corresponds to a tensor mode. Every non-zero in a sparse tensor is represented by a root to leaf path. The CSS format for storing IOU non-zeros of sparse symmetric tensors also has a tree structure inspired from the CSF format (Section IV)

C. Sparse Symmetric Tucker Decomposition

Symmetric Tucker decomposition of an order-N sparse symmetric tensor $\mathcal{X} \in \mathbb{R}^{I \times I \ldots I}$ finds a dense orthonormal matrix $\mathbf{U} \in \mathbb{R}^{I \times R}$ and an order-N dense symmetric core tensor $\mathcal{C} \in \mathbb{R}^{R \times R \ldots R}$ such that $\arg\min_{\mathcal{C},\mathbf{U}} ||\mathcal{X} - \mathcal{C} \times_1 \mathbf{U} \times_2 \mathbf{U} \ldots \times_N \mathbf{U}||$. In hypergraph clustering and dimensionality reduction, the matrix rank R corresponds to number of clusters and size of reduced space respectively, and is usually a small value.

We show the symmetric higher order iterations (HOOI) algorithm (Algorithm 1) by introducing the symmetry property into the popular HOOI approach to Tucker decomposition [21]. The operation in Line 2, *Sparse Symmetric Tensor Times Same Matrix chain* (S^3 TTMc), is observed to be computationally expensive in large data, and will be our focus.

$$\mathcal{Y} = \mathcal{X} \times_{-1} [\mathbf{U}^T] = \mathcal{X} \times_2 \mathbf{U}^T \times_3 \mathbf{U}^T \ldots \times_N \mathbf{U}^T \tag{1}$$

$$matricize(\mathcal{Y}_{k,:,\ldots,:}) = \mathbf{Y}(k,:) = \sum_{\substack{\mathcal{X}_\mathbf{i} \in nz(\mathcal{X}) \\ \mathbf{i}=(i_1,\ldots i_N) \\ i_1=k}} \mathcal{X}_\mathbf{i} \left\{ \bigotimes_{j=2}^{N} \mathbf{U}(i_j,:) \right\} \tag{2}$$

S^3 TTMc, given by (1), is represented by a sequence of tensor times matrix products (TTM) [29] on all but one mode of the symmetric tensor $\mathcal{X}$. TTM of an order-N symmetric tensor $\mathcal{X} \in \mathbb{R}^{I \times I \ldots I}$ with a dense matrix $\mathbf{U} \in \mathbb{R}^{I \times R}$ along mode n, denoted by $\mathcal{Z} = \mathcal{X} \times_n \mathbf{U}^T$, is defined for $r \in \{1, \ldots, R\}$ as $\mathcal{Z}_{i_1 \ldots i_{n-1} r i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I} \mathcal{X}_{i_1 \ldots i_{n-1} i_n i_{n+1} \ldots i_N} \mathbf{U}_{i_n r}$.

The Kronecker product of row vectors $\mathbf{u} \in \mathbb{R}^{1 \times m}$ and $\mathbf{v} \in \mathbb{R}^{1 \times n}$, denoted by $\mathbf{u} \otimes \mathbf{v} \in \mathbb{R}^{1 \times mn}$, is the vectorized outer product [21]. The sparsity of $\mathcal{X}$ favors rewriting S^3 TTMc using Kronecker products in (2), similar to the approach used in the work [29], [30], by only doing meaningful computation corresponding to its non-zeros (with index permutations), thus obtaining the matricized output, $\mathbf{Y}$.

D. Other Related Work

State-of-the-art sparse tensor CANDECOMP/PARAFAC (CP) decomposition [22], [24], [31] and Tucker decomposition [29], [32], [33] research targets general sparse tensors. Kaya et al. proposed a dimension tree data structure [34] to efficiently utilize intermediate results among TTM operations. There are optimized approaches designed for dense symmetric tensor methods [35], [36], [37], [38], [39], [40]. Our previous work [26] is the first to effectively utilize symmetry in sparse tensors for space- and performance-efficiency.

III. Baseline Implementations

S^3 TTMc performance has been evaluated using the three existing baselines described in our previous work [26] based on two general sparse tensor formats. Two S^3 TTMc implementations - *S^3 TTMc-UCOO* and Cyclops Tensor Framework (CTF) [20], [41] - use the 'unique COO' format or *UCOO* that stores only IOU non-zeros. The 'full CSF' format which stores all index permutations has been termed *FCSF* and is used by *SPLATT* [42] for S^3 TTMc computation.

S^3 TTMc-UCOO and CTF Consider an IOU non-zero $t = \mathcal{X}_{i_1,\ldots i_N}$. t contributes to $(N-1)!$ terms in the summation in (2) for rows $\mathbf{Y}(i_1,:), \mathbf{Y}(i_2,:), \ldots, \mathbf{Y}(i_N,:)$. For instance, in the example in Fig. 2, $t = \mathcal{X}_{1,2,3,9}$ contributes 6 terms to the summation for row $\mathbf{Y}(1,:)$

$$\begin{aligned}\mathcal{K}(2,3,9) = {} & \mathbf{U}(2,:) \otimes \mathbf{U}(3,:) \otimes \mathbf{U}(9,:) \\ & + \mathbf{U}(2,:) \otimes \mathbf{U}(9,:) \otimes \mathbf{U}(3,:) \\ & + \mathbf{U}(3,:) \otimes \mathbf{U}(2,:) \otimes \mathbf{U}(9,:) \\ & + \mathbf{U}(3,:) \otimes \mathbf{U}(9,:) \otimes \mathbf{U}(2,:) + \mathbf{U}(9,:) \\ & \otimes \mathbf{U}(2,:) \otimes \mathbf{U}(3,:) + \mathbf{U}(9,:) \otimes \mathbf{U}(3,:) \\ & \otimes \mathbf{U}(2,:) \\ \mathbf{Y}(1,:) \leftarrow {} & \mathbf{Y}(1,:) + \mathcal{X}_{1,2,3,9} \mathcal{K}(2,3,9)\end{aligned} \tag{3}$$

Thus, for indices tuple $\mathbf{i} = (i_1, \ldots i_N)$ of t, every permutation $\mathbf{j} = (j_1, \ldots j_N) = \sigma(\mathbf{i})$ corresponds to a term $t(\mathbf{U}(j_2,:) \otimes \ldots \otimes \mathbf{U}(j_N,:))$ in the summation in (2) for $\mathbf{Y}(j_1,:)$. From

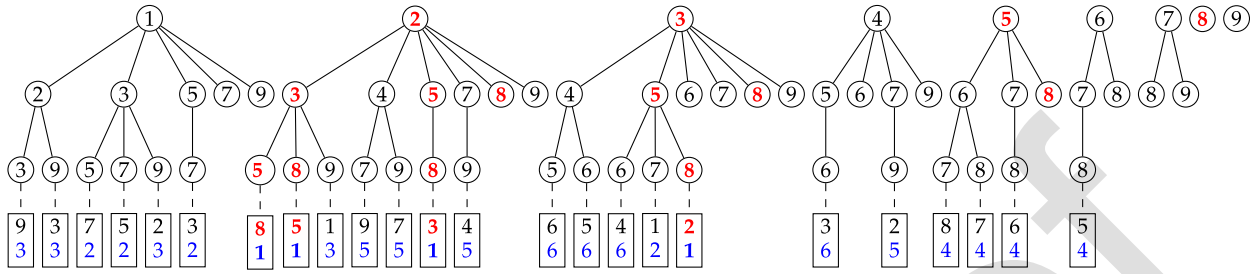

Fig. 2. CSS format for the sparse symmetric tensor in Table II. For the IOU non-zero $(2, 3, 5, 8)$ with value 1, the paths $\mathcal{P}_{i_l}$ corresponding to l-length subsequences, $1 \leq l < N$ of the IOU index set are highlighted in red. The left-out index along with the non-zero value (shown in blue) is stored by the leaf nodes at level $N-1$.

the discussion above, we can rewrite (2) as follows:

$$\mathbf{Y}(ind, :) = \sum_{\substack{\mathcal{X}_\mathbf{i} \in unz(\mathcal{X}) \\ ind \in \mathbf{i}}} \mathcal{X}_\mathbf{i} \sum_{\mathbf{j}=\sigma(\mathbf{i} \setminus ind)} \left\{ \bigotimes_{k=2}^{N} \mathbf{U}(j_k, :) \right\} \quad (4)$$

The *S^3 TTMc-UCOO* baseline iterates over only the IOU nonzeros of $\mathcal{X}$ in parallel by implementing (4). The *CTF* baseline computes S^3 TTMc uses the existing Cyclops Tensor Framework (CTF) [20] which also represents the sparse symmetric tensor using the *UCOO* format. A disadvantage of both baselines is the absence of *non-zero memoization*, i.e., memoizing intermediate permutation results among IOU non-zeros

SPLATT using FCSF format The third baseline directly uses SPLATT [18] - a fast state-of-the-art TTMc algorithm for sparse general tensors stored in the CSF format - for sparse symmetric tensors by storing all permutations of IOU non-zeros in the 'full' CSF (*FCSF*) format. Storing only the IOU non-zeros in the CSF format, while being a more compact representation than *UCOO*, suffers from two major drawbacks - (i) the factorial number of traversals required for S^3 TTMc computation due to the mode specificity of CSF format, and (ii) the absence of memoization between permutations of IOU non-zeros.

IV. CSS Structure

The CSS structure [26] is a compact storage format for sparse symmetric tensors that enables efficient S^3 TTMc computation. We include the introduction of CSS for completeness. It overcomes the memory explosion due to storing redundant symmetric information associated with the *FCSF* format by storing only the IOU non-zeros, which is sufficient as CSS can retrieve all non-zeros $\sigma(\mathcal{X}_i)$ of the equivalence class that are related to a given IOU nonzero $\mathcal{X}_i$ (Section II). The CSS format facilitates non-zero memoization of intermediate Kronecker product results among IOU non-zeros and within permutations of an IOU non-zero, thus outperforming all our baselines for high-order tensors. For a symmetric tensor $\mathcal{X} \in \mathbb{R}^{I \times I \times \ldots I}$ of order N, CSS is a forest with $N-1$ levels constructed from IOU non-zeros, $unz(\mathcal{X})$. A key advantage of using CSS is its computation-aware nature - by storing all ordered subsequences of IOU non-zeros, intermediate results in the S^3 TTMc computation can be easily memoized with minimal additional index information. The order-4 tensor in Table II consisting of 6 non-zeros can be represented by the CSS format shown in Fig. 2. The non-zero value (shown in blue in Fig. 2 and Table II) and

TABLE II
A Sparse Symmetric Tensor $\mathcal{X} \in \mathbb{R}^{10 \times 10 \times 10 \times 10}$ With 6 IOU Nonzeros

i_1	2	1	1	5	2	3
i_2	3	3	2	6	4	4
i_3	5	5	3	7	7	5
i_4	8	7	9	8	9	6
vals	1	2	3	4	5	6

the left-out index $i_N = \mathbf{i} \setminus \mathcal{P}_{\mathbf{i}_{N-1}}$ is stored at the leaf nodes at level $N-1$. Note that as any path $\mathcal{P}_{\mathbf{i}_l}$ is a l-length subsequence of the index tuple $\mathbf{i}$ of some IOU non-zero, each IOU non-zero corresponds to multiple paths $\mathcal{P}_{\mathbf{i}_l}$ to levels $l = 1, 2, \ldots N-1$. For a single IOU non-zero, the number of nodes at level l is $\binom{N}{l}$ for CSS. We need to save both pointers and indices for every level, which doubles the storage of CSS. The affiliated leaf level indices and non-zero values are stored in each of the N nodes at level $(N-1)$ generated by this non-zero. Thus, the space complexity for storing one IOU non-zero unz in CSS is given by $(2^{N+1}-4)\beta_{int} + N(\beta_{int} + \beta_{dbl})$ [26], and for storing $unnz$ non-zeros is given in Table III. CSS is constructed by recursing over all paths $\mathcal{P}_{\mathbf{i}_l}$ corresponding to non-empty subsequences of indices of IOU non-zeros as shown in Algorithm 2, which uses the following key property of CSS :

Property 1. Any path in the CSS forest exists only if it is a subsequence of the index tuple of at least one IOU non-zero. Moreover, the ordering of the node values in every path further ensures unique representation of permutations of subsets of all IOU non-zeros.

In the function `InsertIOU`, the vector argument $\mathbf{r}$ is constructed such that every node in $\mathcal{P}_{parent}$ is strictly smaller than any element of $\mathbf{r}$. As child nodes are selected from $\mathbf{r}$, Algorithm 2 maintains the ordering of CSS. The number of nodes for which `InsertIOU` is called with $|r| = N - l + 1$, $1 < l \leq N$ is given by $\sum_{k=0}^{l-1} \binom{l-1}{k}$. We use a binary search tree for inserting nodes in CSS in order to maintain the ordering of nodes while traversing depth-wise and level-wise through the CSS tree. Thus, summing up over all values of $|r|$ and over all IOU non-zeros $unnz$, we get the complexity of constructing CSS

$$\mathcal{O}\left(unnz \sum_{j=1}^{N} \sum_{k=0}^{j-1} \binom{j-1}{k} (N-j) \log I \right) = \mathcal{O}(2^N unnz \log I)$$

Lemma IV.1. There exists an injective mapping between a set of IOU non-zeros of an order-N sparse symmetric tensor and the CSS forest constructed from it.

TABLE III
SUMMARY OF SPACE AND COST COMPLEXITY ANALYSIS OF S^3 TTMC-CSS COMPARED TO BASELINE IMPLEMENTATIONS [26]

Algorithm(Format)	Tensor Storage Space (#Bytes)	Algorithm Cost Complexity (#Flops)
SPLATT(FCSF)	$2\beta_{int}\sum_{l=1}^{N-1} nnz'_l + N!\,unnz(\beta_{int}+\beta_{dbl})$	$\sum_{l=2}^{N-1} 2R^{N-l+1}nnz'_l + 2\cdot unnz\cdot N!\cdot R$
UCOO-N(UCOO)	$(N\beta_{int}+\beta_{dbl})unnz$	$2R^{N-1}N!\,unnz$
S^{3}TTMC-CSS(CSS)	$2\beta_{int}\sum_{l=1}^{N-1} nnz_l + unnz\cdot N(\beta_{int}+\beta_{dbl})$	$\sum_{l=2}^{N-1}(2l-1)R^l\binom{N}{l}unnz + 2\cdot unnz\cdot NR^{N-1}$

Algorithm 2: CSS Construction.

procedure InsertIOU$parent, \mathbf{r}, l, val$
▷ Insertion at level l of $\mathbf{r}$, where $\mathbf{r}_j < \mathbf{r}_{j+1}$ for $1 \le j < |\mathbf{r}|$
▷ Also, $1 \le |\mathbf{r}| \le N - l + 1$ for $l > 1$ and $|\mathbf{r}| = N$ for $l = 1$
  if $l == N$ **then**
    $child = node(val)$
    Add $child$ to children of $parent$ **return**
  end if
  for $j = 1, \ldots |\mathbf{r}|$ **do**
    $child = node(r_j)$
    Add $child$ to children of $parent$
    $\mathbf{c} = (r_{j+1}, r_{j+2}, \ldots, r_{|\mathbf{r}|})$
    INSERTIOU($child$, $\mathbf{c}$, $l+1$, val)
  end for
end procedure
for $\mathbf{i}, val \in unz(\mathcal{X})$ **do**
  INSERTIOU($root$, $\mathbf{i}$, 1, val)
end for

The CSS format compresses a sparse symmetric tensor from two perspectives: First, rather than explicitly storing all permutations of an IOU non-zero, we cover the permutations by shorter lengths of CSS tree paths. Second, the CSS structure naturally hides the path overlapping among the permutations of one IOU non-zero, as well as among permutations of different IOU non-zeros. The redundancy in the CSS format in terms of storing subsequences of index tuples of IOU non-zeros is a necessity for efficient S^3 TTMC computation (discussed in Section V).

A. Comparison to Baselines

UCOO Storing IOU non-zeros in *UCOO* is more compact than CSS, as can be seen in Table III. However, as we will demonstrate in Section VI, S^3 TTMC using the CSS format is more work efficient than using *UCOO*.

FCSF Unlike CSS, *FCSF* does not recognize the symmetry feature of a tensor - it stores all $\mathcal{O}(N!)$ permutations of every IOU non-zero - and hence has a space complexity given in Table III for $nnz = unnz \cdot N!$ non-zeros in $\mathcal{X}$. nnz'_l is the number of nodes at level l in the CSF format with $nnz'_1 < nnz'_2 < \ldots < nnz'_{N-1}$. This is in contrast to the the variation of the number of nodes nnz_l with CSS depth l wherein, owing to the combinatorial construction of the data structure, the nnz_l follows the trend of the binomial coefficient i.e., $nnz_1 < nnz_2 \ldots < nnz_{N/2} > \ldots > nnz_{N-1}$. While it is difficult to directly compare nnz_l and nnz'_l due to the significant overlapping among nodes, experimental results indicate significant space savings of CSS for both synthetic and real world datsets (Section VI).

V. S^3 TTMC COMPUTATION

S^3 TTMC-CSS uses the CSS format which inherently supports efficient memoization to compute S^3 TTMC for sparse symmetric tensors.

A. Formulation

(4) iterates over only the IOU nonzeros of $\mathcal{X}$, but still retains the arithmetic redundancies from (2) which can be eliminated by using the distributivity of Kronecker product. Consider the vector function $\mathcal{K}$

$$\mathcal{K}(\mathbf{i}) = \begin{cases} \mathbf{U}(i,:) & |\mathbf{i}| = 1 \\ \sum_{j \in i_1 \ldots i_l} \mathbf{U}(j,:) \otimes \mathcal{K}(\mathbf{i} \setminus j) & |\mathbf{i}| = l > 1 \end{cases} \tag{5}$$

Unrolling the recursion in (5) gives (6) i.e., $\mathcal{K}(\mathbf{i})$ is the sum of Kronecker products of rows of $\mathbf{U}$ for every permutation $\mathbf{j} = \sigma(\mathbf{i})$. Then, (4) can be succinctly factorized using (5), as seen in (7).

$$\mathcal{K}(\mathbf{i}) = \sum_{\mathbf{j}=\sigma(\mathbf{i})} \otimes_k \mathbf{U}(j_k,:) \tag{6}$$

$$\mathbf{Y}(ind,:) = \sum_{\substack{\mathcal{X}_\mathbf{i} \in unz(\mathcal{X}) \\ ind \in \mathbf{i}}} \mathcal{X}_\mathbf{i}\, \mathcal{K}(\mathbf{i} \setminus ind) \tag{7}$$

For example, (3) can be further factorized using the distributivity of Kronecker product according to (5) as

$$\mathcal{K}(2,3,9) = \mathbf{U}(2,:) \otimes \mathcal{K}(3,9) + \mathbf{U}(3,:) \otimes \mathcal{K}(2,9) + \mathbf{U}(9,:) \otimes \mathcal{K}(2,3) \tag{8}$$

The computation-aware property of the CSS format lends itself to using (7) to achieve both forms of non-zero memoization i.e., sharing intermediate $\mathcal{K}$ vectors across multiple IOU non-zeros as well as within permutations of a given IOU non-zero. The remainder of this section explores parallel approaches to computing (7).

B. Naive Approach

Consider node i_l in the CSS data structure at level l with path $\mathcal{P}_{\mathbf{i}_l} = (i_1, i_2, \ldots, i_l)$ storing $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})$. A key insight from (5) is that $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})$ depends only on $\mathcal{K}(\mathcal{P}^{-i_k}_{\mathbf{i}_l}), 1 \le k \le l$, where $\mathcal{K}(\mathcal{P}^{-i_k}_{\mathbf{i}_l})$ is the path of length $l-1$ with nodes given by $\mathbf{i}_l \setminus i_k$. This implies we need to aggregate results from nodes $\mathcal{P}^{-i_k}_{\mathbf{i}_l}(l-1), 1 \le k \le l$ at level $l-1$ in CSS format to compute $\mathcal{K}$ at node i_l. Such level-wise memoization naturally suits a

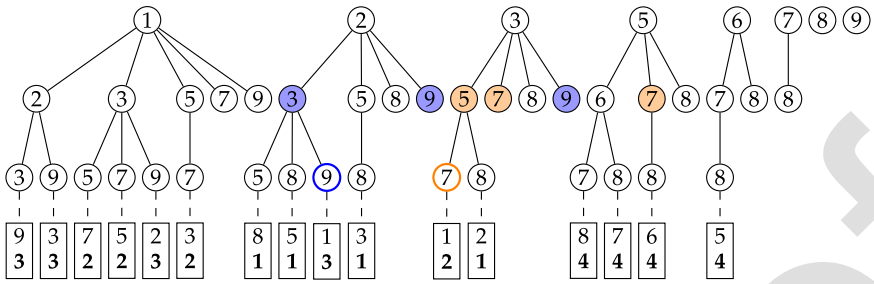

Fig. 3. Naive approach to computing $\mathcal{K}$ using the property that $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})$ depends only on $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l}^{-i_k}), 1 \leq k \leq l$, where $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l}^{-i_k})$ is the path of length $l-1$ with nodes given by $\mathbf{i}_l \setminus i_k$. $\mathcal{K}$ values for nodes at level 3 encircled in purple and yellow depend on the nodes in level 2 coloured in purple and yellow respectively.

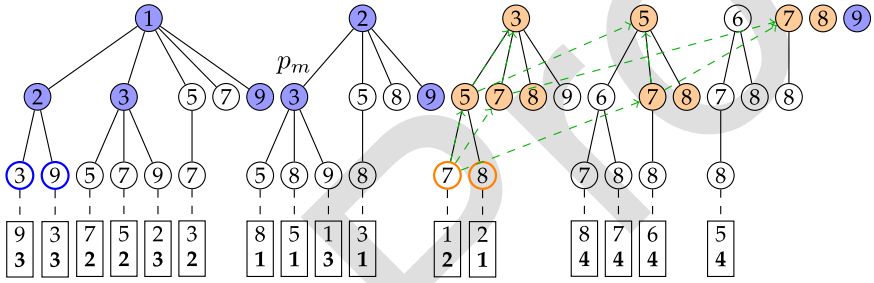

Fig. 4. Reducing memory overhead of naive approach by traversing CSS depth-wise. The encircled nodes in level 3 only depend on shaded ancestor nodes to their right, thus allowing us to discard previously memoized $\mathcal{K}$. Retrieving $\mathcal{K}$ ancestor nodes can be done by the symbolic parent graph which is a union of DAGs. One of the DAGs, indicated by red dashed edges and rooted at node 7 encircled in yellow, is shown in the figure. Node 3, annotated p_m, has the largest number of children among all nodes in level 2, and hence upper bounds the memory requirement for memoization.

breadth-first traversal of the CSS format, wherein we need to only store $\mathcal{K}$ results of level $l-1$ to compute $\mathcal{K}$ at level l. Fig. 3 shows a snapshot of the naive approach to computing $\mathcal{K}$ using the CSS format constructed from a subset of the non-zeros in Table II. Consider node 9 at level 3 on path $(2,3,9)$ encircled in purple storing the chained Kronecker result shown in (3). Note that $\mathcal{K}(2,3,9)$ depends on $\mathcal{K}(2,3), \mathcal{K}(2,9), \mathcal{K}(3,9)$ (8) which are stored in the nodes shaded in purple at level 2. However, such naive level-wise memoization is prohibitive due the binomial variation in the CSS level sizes (see Section IV-A) and the exponential increase in the size of $\mathcal{K}$ with increasing CSS depth. Moreover, while computing $\mathcal{K}$ for nodes at level l, the nodes at level $l-1$ that contribute to the Kronecker product are not consecutive in memory, and this step becomes expensive due to the random accesses in a large $\mathcal{K}$ array. The upper bound on the memory required - determined by the level l $(1 \leq l < N-2)$ with the largest $nnz_l R^l + nnz_{l+1} R^{l+1}$ - can be made tighter by considering the ordering of the nodes in CSS, as described in Section V-C. The S^3 TTMc-CSS algorithm (Algorithm 3) stores the minimum number of intermediate $\mathcal{K}(\mathcal{P})$ vectors needed to perform S^3 TTMc on the CSS tree while still maintaining computational efficiency.

C. Optimizations

Memoization overhead Consider the set of l paths $SP(\mathcal{P}_{\mathbf{i}_l}) = \{\mathcal{P}_{\mathbf{i}_l}^{-i_k} : \mathcal{P}_{\mathbf{i}_l}^{-i_k} = \mathcal{P}_{\mathbf{i}_l} \setminus i_k, 1 \leq k \leq l\}$ for some path $\mathcal{P}_{\mathbf{i}_l}$ in CSS. Then, $SP(\mathcal{P}_{\mathbf{i}_l})$ indicates the *symbolic parents* of node i_l at level l i.e for every $\mathcal{P}_{\mathbf{i}_l}^{-i_k} \in SP(\mathcal{P}_{\mathbf{i}_l})$, computing $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})$ at node i_l requires $\mathcal{K}$ results from node $\mathcal{P}_{\mathbf{i}_l}^{-i_k}(l-1)$. Moreover, the ordering of the CSS format ensures that every path in $SP(\mathcal{P}_{\mathbf{i}_l})$ is either a subpath of $\mathcal{P}_{\mathbf{i}_l}$ or, is to the right of $\mathcal{P}_{\mathbf{i}_l}$ i.e., for any $\mathcal{P}_{\mathbf{i}_l}^{-i_k} \in SP(\mathcal{P}_{\mathbf{i}_l})$, node at level $1 \leq l' \leq l-1, \mathcal{P}_{\mathbf{i}_l}^{-i_k}(l')$ is either $\mathcal{P}_{\mathbf{i}_l}(l')$ or to the right of $\mathcal{P}_{\mathbf{i}_l}(l')$. For example, in Fig. 4, for node 7 in level 3 given by the path $(3,5,7)$, $SP((3,5,7)) = \{(3,5),(3,7),(5,7)\}$. Note that the red directed edges from this node, which indicate the last node of the paths in $SP((3,5,7))$, are either on the path $(3,5,7)$ or to the right of it. This implies that while computing $\mathcal{K}$ for node 7, we can discard all the memoized results from level 2 that are to the left of $SP((3,5,7))$ i.e., results from all nodes coloured purple in level 2 can be discarded. More generally, for computing $\mathcal{K}(\mathcal{P}_{\mathbf{i}_{N-1}})$, we store $\mathcal{K}$ vector of only contributing paths in $SP(\mathcal{P}_{\mathbf{i}_l})$ for every level $2 \leq l \leq N-1$ and discard $\mathcal{K}$ vectors of paths that are to the left of $\mathcal{K}(\mathcal{P}_{\mathbf{i}_{N-1}})$. Given a set of IOU non-zeros, the size of memory that needs to be allocated for this memoization strategy is determined by the node p_m at level $N-2$ with the largest number of children. That is, the maximum number of $\mathcal{K}$ vectors that need to be stored before at least one of them is discarded is the sum of the number of $\mathcal{K}$ vectors required to compute $\mathcal{K}(\mathbf{j})$ for every child j of p_m.

Symbolic parent graph In order to efficiently retrieve the *symbolic parents* of a given node i.e., the nodes in the CSS data structure that contribute to the partial Kronecker product at any given node, we need an auxiliary data structure. To this end, we construct an auxiliary data structure called the *symbolic parent*

Algorithm 3: Rank-R S^3 TTMc-CSS.

Output: Matricized S^3 TTMc, $\mathbf{Y} \in \mathbb{R}^{I \times R^{N-1}}$
Construct symbolic parent graph SG
$p_m = \arg\max_{j \in nz_{N-2}} |child(j)|$
Allocate memoization workspace $WS = \sum_{l=1}^{N-2} R^l |\{n : n \in \mathcal{T}_j^{SG}, n \in nz_l \forall j \in child(p_m)\}|$
parfor $i_{N-1} = 1, \ldots nnz_{N-1}$ **do**
  while $u_l \in nodes(\mathcal{T}_{i_{N-1}}^{SG})$ at level l **do**
    if $l < 2$ **then**
      $\mathcal{K}(u_l) = \mathbf{U}(u_l)$
    else
      for $\mathcal{P}' \in SP(u_l)$ **do**
        $n = \mathcal{P}_{u_l} \setminus \mathcal{P}'$
        $\mathcal{K}(\mathcal{P}_{u_l}) \leftarrow \mathcal{K}(\mathcal{P}_{u_l}) + \mathbf{U}(n,:) \otimes \mathcal{K}(\mathcal{P}')$
      end for
    end if
  end while
end parfor
parfor $i_{N-1} = 1, \ldots nnz_{N-1}$ **do**
  while $ind \in$ nonzero array of node i_{N-1} **do**
    $\mathcal{P} = (i_1, \ldots i_{N-1})$
    AtomicAdd($\mathbf{Y}(ind,:), \mathcal{K}(\mathcal{P}) \cdot \mathcal{X}_{(\mathcal{P},ind)}$)
  end while
end parfor

graph $SG(V, E)$ from elements in $SP(\mathcal{P}_{\mathbf{i}_l})$ for every path $\mathbf{i}_l$ in CSS data structure, where

$$V = \bigcup_{l=1}^{N-1} V_l$$

$$E = \{(u, v) : u \in V_{j+1}, v \in V_j \cap \{\mathcal{P}'(j), \mathcal{P}' \in SP(\mathcal{P}_u)\}, \; 1 \le j < N-1\}$$

Here, V_l is the set of all nodes of the CSS tree at level l, and for every node u in level l, $deg(u) = l$. Moreover, $SG(V, E)$ is a union of nnz_{N-1} directed acyclic graphs, i.e., $SG = \bigcup_{i=1}^{nnz_{N-1}} \mathcal{T}_i^{SG}$, where $\mathcal{T}_i^{SG}$ is the DAG with node i as the first node in its topological sorting. For example, Fig. 4 shows $\mathcal{T}_7^{SG}$. The space complexity of this auxiliary graph is $S_{SG} = \beta_{int} \sum_{l=2}^{N-1} l \cdot nnz_l$. The time complexity of $SG(V, E)$ construction is $\mathcal{O}(|V| + |E|) = \mathcal{O}(\sum_{l=2}^{N-1} l \cdot nnz_l)$, which is smaller than the S^3 TTMc cost by $\mathcal{O}(R^l)$ per CSS level ((11)).

D. Parallelism

The construction of the $SG(V, E)$ indicates that we can distribute the DAGs $\mathcal{T}_i^{SG}, 1 \le i \le nnz_{N-1}$ between p threads, and compute the intermediate $\mathcal{K}(\mathcal{P})$ results using a parallel depth-wise traversal of CSS. Each thread allocates its own memoization workspace based on the node p_m identified for the set of DAGs $\mathcal{T}_i^{SG}$ assigned to it, as described in Section V-C. Since the $\mathcal{K}$ computations are independent between threads, there is no need for explicit synchonization to compute $\mathcal{K}$. Thus, the leaf nodes of CSS are distributed over p threads, each of which is responsible for computing $\mathcal{K}(\mathcal{P})$ for all of its leaf nodes, as shown in Algorithm 3. We can propagate intermediate Kronecker products in parallel to nodes down the levels of the CSS tree using the edges of SG without any synchronization overheads. However, at the leaf nodes at level $N-1$, aggregating results into the matricized output $\mathbf{Y}$ requires atomic operations as multiple nodes at level $N-1$ can index into the same row of $\mathbf{Y}$ S^3 TTMc-CSS can be directly used in Algorithm 1 to gain better performance for Tucker decomposition.

E. Cost Comparison With Baselines

We compare the number of floating point operations in S^3 TTMc-CSS (C^{CSS}) to *SPLATT* (C^{SPLATT}) and *S^3 TTMc-UCOO* (C^{UCOO}) [2]. We provide a cost model for each of the algorithms, and analyze their performance for rank R-S^3 TTMc.

Flop count of S^3 TTM c-CSS We assume $nnz_l \approx \binom{N}{l} unnz$ for $N-1 \ge l \ge \lceil \frac{N}{2} \rceil$. The assumption stems from two observations on the CSS structure - (i) the number of nodes at level l is proportional to the number of distinct subsequences of length l from the set of IOU non-zeros [26] , and (ii) for high-order tensors, frequency of overlap of subsequences between IOU non-zeros reduces deeper down the tree i.e., branching in the tree decreases deeper into the CSS structure. As level $\lceil N/2 \rceil$ has the largest number of nodes, we restrict our assumption to $N-1 \ge l \ge \lceil \frac{N}{2} \rceil$. The cost of computing $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})$ for any path $\mathcal{P}_{\mathbf{i}_l}$ is the sum of Kronecker products $\mathcal{K}(\mathcal{P}'_k) \otimes \mathbf{U}(i_k,:)$ for each path $\mathcal{P}'_k \in SP(\mathcal{P}_{\mathbf{i}_l}), 1 \le k \le l$. Moreover, as $\mathcal{K}(\mathcal{P}'_k) \in \mathbb{R}^{R^{l-1}}$, the cost of each Kronecker product computation is R^l. Thus, we get

$$\begin{aligned} c(l; N, R) &= nnz_l \cdot cost(\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})) \\ &= nnz_l \cdot \sum_{\substack{\mathcal{P}'_k \in SP(\mathcal{P}_{\mathbf{i}_l}) \\ 1 \le k \le l}} cost(\mathcal{K}(\mathcal{P}'_k) \otimes \mathbf{U}(i_k,:)) \\ &= nnz_l \underbrace{(2l-1)R^l}_{\substack{\#\text{flops to sum } l \text{ Kronecker product results,} \\ \text{with each product computed using vectors} \\ \text{of length } R^{l-1} \text{ and } R}} \end{aligned} \tag{9}$$

Note that for $l < \lceil \frac{N}{2} \rceil$, we have $nnz_l < nnz_{l+1}$ (Section IV), and thus $c(l) < c(l+1)$. We focus on the flop count at level $l \ge \lceil \frac{N}{2} \rceil$, as shown in (10). We then write the total flop count of S^3 TTMc computation as the sum of $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})$ computation cost $c(l)$ over all levels of the CSS tree, along with the cost of scaling the $\mathcal{K}$ vectors at level $N-1$ by the non-zero value, as shown in (11)

$$c(l; N, R) = (2l-1)R^l \binom{N}{l} unnz \tag{10}$$

[2]Since CTF targets sparse symmetric tensor contractions, a more general case compared to TTM, and *S^3 TTMc-UCOO* outperforms CTF in Section VI, we ignore its analysis.

$$C^{CSS} = \sum_{l=2}^{N-1} c(l;N,R) + \underbrace{2 \cdot unnz \cdot NR^{N-1}}_{\substack{\#\text{flops to compute (7) given} \\ \mathcal{K} \text{ at level } N-1}} \quad (11)$$

For $l \geq \lceil \frac{N}{2} \rceil$, we analyze the extrema of $c(l;N,R)$ to obtain the level l_0 for which C^{CSS} is dominated by $c(l_0;N,R)$. We have

$$\log c(l;N,R) = \log\left(R^l N!(2l-1)\right) - \log\left(\Gamma\left(l+1\right)\right) - \log\left(\Gamma\left(N-l+1\right)\right)$$

Since the gamma function is log-convex, $c(l;N,R)$ is a log-concave function. Moreover, the gradient is given by

$$\frac{d}{dl}\log c = \frac{2}{2l-1} + \log R - \psi(l+1) + \psi(N-l+1) \quad (12)$$

$$= \frac{2}{2l-1} + \log R + H_{N-l} - H_l \quad (13)$$

$$\left.\frac{d}{dl}\log c(l)\right|_{l=\lceil \frac{N}{2} \rceil} \geq \frac{2}{N} + \log R - \frac{2}{N+1} > 0 \quad (14)$$

$$\left.\frac{d}{dl}\log c(l)\right|_{l=N-1} = \log R - \left(H_{N-1} - \frac{2N-1}{2N-3}\right) \quad (15)$$

where $\psi(x)$ is the digamma function, and H_n is the n^{th} Harmonic number [43]. Thus, there exists a level l_0, with $[\log c(l)]'|_{l=l_0}$, whose contribution to the total computation cost of S^3 TTMc dominates. We know that H_n closely approximates the natural logarithm function i.e., values of the sequence $H_n - \log n$ decreases monotonically towards the Euler–Mascheroni constant γ. Thus, we can write

$$\lim_{N\to\infty} H_{N-1} - \frac{2N-1}{2N-3} = \gamma + \lim_{N\to\infty} \log\{N-1\} \to \infty \quad (16)$$

From (16), we can infer that as tensor order increases, there exists a level $N-1 > l_0 \geq \lceil \frac{N}{2} \rceil$ where $c(l;N,R)$ reaches a maxima, and dominates C^{CSS}. On the other hand, if (15) is non-negative, log-concavity of $c(l;N,R)$ implies that the cost contribution of the last level toward C^{CSS} is the largest.

Comparison to S^3 *TTMc-UCOO*: *S^3 TTM* c *-UCOO* implements (4) without any memoization of intermediate Kronecker products shared between IOU non-zeros. Its cost model is given by $C^{UCOO} = 2R^{N-1}N!unnz$. Irrespective of the level contributing the largest cost to C^{CSS}, it is clear that $\log\left(\frac{C^{CSS}}{C^{UCOO}}\right) < 0$. S^3 TTMc-CSS performs less work than *S^3 TTMc-UCOO* for high-order sparse symmetric tensors.

Comparison to SPLATT: The cost model of *SPLATT* in terms of the number of floating point operations is given by (17)

$$C^{SPLATT} = \sum_{l=2}^{N-1} 2R^{N-l+1} nnz'_l + 2 \cdot unnz \cdot N! \cdot R \quad (17)$$

The dominant term in (17) is the computation at level N [29] due to the growth of the factorial term with increasing tensor order. In order to compare cost of CSS with *SPLATT*, we look at the dominant costs of both algorithms. As the dominant cost for C^{CSS} depends on the sign of $(\log c(l))'$, we will consider both cases for the comparison. We demonstrate that for S^3 TTMc rank and tensor order chosen such that (15) is negative, we outperform the *FCSF* baseline.

Theorem V.1. S^3 TTMc-CSS outperforms *SPLATT* in terms of number of floating point operations for rank R-S^3 TTMc operation on order-N sparse symmetric tensors, if (15) is negative.

Proof. Based on the sign of (15), we compare the flop counts of S^3 TTMc-CSS and *SPLATT* baseline.

(15) is negative, implying that there exists a level $N-1 > l_0 \geq \lceil \frac{N}{2} \rceil$ such that $c(l_0;N,R)$ is the dominant term, i.e., $(\log c(l))' = 0$ at $l = l_0$. This ensures that R is non-negative

$$\left.\frac{d}{dl}\log c(l)\right|_{l=l_0} = 0 \quad (18)$$

$$\log R = H_{l_0} - H_{N-l_0} - \frac{2}{2l_0-1} = \sum_{k=N-l_0+1}^{l_0} \frac{1}{k} - \frac{2}{2l_0-1} \quad (19)$$

$$\geq \frac{2(2l_0-N)}{N+1} - \frac{2}{2l_0-1} \quad (20)$$

using AM-HM inequality. Requiring that $R > 1$ further restricts the feasible values of l_0 to $\frac{N}{2}+1 \leq l_0 < N-1$. Using the main condition for this case, i.e., $\log R < H_{N-1} - \frac{2N-1}{2N-3}$, we can write

$$\log\left(\frac{C^{CSS}}{C^{SPLATT}}\right) = \log \frac{(2l_0-1)R^{l_0}\binom{N}{l_0} unnz}{RN! unnz}$$

Moreover, we know that

$$\log n! \sim n\log n - n + \log\sqrt{2\pi n} \quad \text{(Stirling's approx)}$$

$$\lim_{n\to\infty}(H_n - \ln n) = \gamma \quad \text{(Euler-Mascheroni const, } \gamma \approx 0.57\text{)}$$

Substituting,

$$\log\left(\frac{C^{CSS}}{C^{SPLATT}}\right) < \log(2l_0-1) + (l_0-1)\left(\log(N-1) - \frac{2N-1}{2N-3}\right) + \left(l_0 - l_0\log l_0 - \log\sqrt{2\pi l_0}\right) + \left((N-l_0) - (N-l_0)\log(N-l_0) - \log\sqrt{2\pi(N-l_0)}\right)$$

$$= \log(2l_0-1) + (l_0-1)\log(N-1) - \left(l_0+\frac{1}{2}\right)\log l_0 - \left(N-l_0+\frac{1}{2}\right)\log(N-l_0) + \left(N - \frac{(l_0-1)(2N-1)}{2N-3} - \log 2\pi\right)$$

$$:= q(l_0, N) \quad (21)$$

TABLE IV
S^3 TTMc-CSS Outperforms *SPLATT* When its Tensor Order $N \geq N_{min}$ and Matrix Rank $R \leq R_{max}$

R_{max}	2	4	8	16
N_{min}	5	8	14	26

Inspecting $q(l_0, N)$, for $\lceil N/2 \rceil \leq l_0 < N-1$ and $N \to \infty$

$$\frac{\partial}{\partial l_0} q = \log\left(\frac{(N-1)(N-l_0)}{l_0}\right) + \frac{2}{2l_0-1} - \frac{2N-1}{2N-3} + \frac{l_0 - \frac{N}{2}}{(N-l_0)l_0} > 0$$

Thus, the maximum value of $q(., N)$ is at $l_0 = N-2$

$$q(N-2, N) = \log(2N-5) + (N-3)\log(N-1) - \left(N - \frac{3}{2}\right)\log(N-2) + \frac{4N-3}{2N-3} - C$$

Using L'Hôpital's rule

$$\lim_{N\to\infty} q(N-2) = \log 2 + 1 - \frac{1}{2}\log \lim_{N\to\infty}(N-2) + 2 - C < 0 \tag{22}$$

So, for large N

$$\log\left(\frac{C^{CSS}}{C^{SPLATT}}\right) < 0$$

(15) is non-negative, implying that the dominant term in C^{CSS} is $c(N-1; N, R)$. Then, for large N

$$\begin{aligned}
\log\left(\frac{C^{CSS}}{C^{SPLATT}}\right) &= \log\left(\frac{(2N-1)NR^{N-1}unnz}{RN!unnz}\right) \\
&= \log(2N-1) + (N-2)\log R + (N-1) \\
&\quad - (N-1)\log(N-1) - \frac{1}{2}\log(N-1) - \frac{1}{2}\log 2\pi \\
&> \log(2N-1) + (N-2)\left(\log(N-1) + \gamma - \frac{2N-1}{2N-3}\right) \\
&\quad - \left(N - \frac{1}{2}\right)\log(N-1) - \frac{1}{2}\log 2\pi \\
&= \log\left(\frac{2N-1}{N-1}\right) - \frac{1}{2}\log(N-1) \\
&\quad + (N-2)\left(\gamma - \frac{2N-1}{2N-3}\right) > 0
\end{aligned}$$

Thus, for S^3 TTMc, if rank and tensor order satisfy $\log R < H_{N-1} - \frac{2N-1}{2N-3}$, our algorithm is more cost-efficient than *SPLATT*. ■

Using (15), Table IV lists a range of (N, R) pairs using N_{min} and R_{max} values such that for tensors with order $N \geq N_{min}$, performance of S^3 TTMc-CSS is theoretically superior to SPLATT for rank R-S^3 TTMc when $R < R_{max}$.

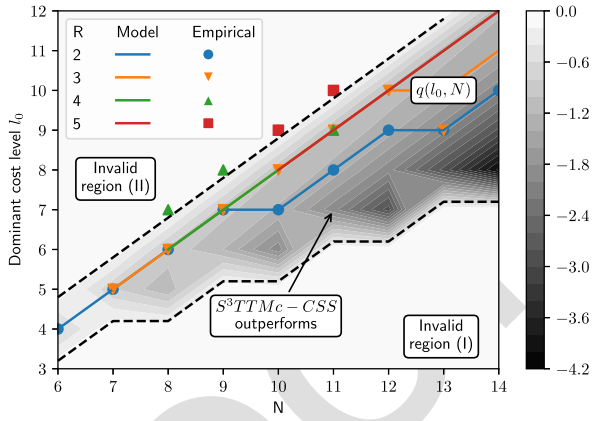

Fig. 5. Visualization of (N, l_0, R) space for the case where S^3 TTMc-CSS performs better than SPLATT (Analysis of Case 1 in Theorem V.1). For different values of $R < R_{max}(N)$, the level l_0 contributing the dominant cost is obtained theoretically as well as experimentally, and is visualized on the contour plot of $q(l_0, N)$.

F. Model Analysis

We can evaluate the performance gain of S^3 TTMc-CSS when $R < R_{max}$ for a given order-N tensor by analyzing $q(l_0, N)$ (21) as it serves as a proxy to the entire scaled $c(l_0; N, R)$. That it, more negative $q(l_0, N)$ indicates a larger cost-gap between S^3 TTMc-CSS and SPLATT. Moreover, by (15), as $c(l_0; N, R)$ is maximized at some $l_0 < N-1$, the domain of q is further restricted to $N-1 > l_0 \geq \lceil \frac{N}{2} \rceil$ at each N. Fig. 5 shows the density plot of $q(l_0, N)$, with the dashed lines indicating the boundary of its domain, and regions (I) and (II) not belonging to the domain of q.

Note that in the derivation of (21), decreasing $(\log c(l))'_{l=N-1}$ results in improving performance of S^3 TTMc-CSS over SPLATT. That is, S^3 TTMc rank R is preferred over R' if $l_0(R) < l_0(R')$ due to $q(., N)$ being an increasing function. We also see this fact reflected in Fig. 5 - for a given tensor order N, smaller values of R result in $c(l)$ maximizing at higher levels of the CSS forest, thereby decreasing $q(l_0, N)$ and increasing the performance gap between S^3 TTMc-CSS and SPLATT.

In order to examine the empirical confirmation of our model, we compute the level l_0 contributing the largest cost to S^3 TTMc computation using (19), and compare it to l_0 obtained empirically by directly evaluating $c(l; N, R)$ for datasets in Table VI. Fig. 5 plots theoretical and experimentally observed variation of $l_0(N)$ for different $R < R_{max}(N)$. For a given tensor order, we observe an exact agreement for smaller R, and a difference of at most one level for larger $R < R_{max}(N)$. Note that as $R \to R_{max}$, the maximum contributing level $l_0 \to N-2$ as (15) approaches zero. However, experimentally, the value of $q(N-2, N)$ is no longer negative for smaller tensor orders since the limit in (22) is no longer satisfied. Thus, we see that $l_0 = N-1$ experimentally for $R = R_{max}(N)$ for the smaller tensor orders.

VI. Experiments

A. Platform and Experimental Configurations

We conducted experiments using a single node on three different machines - (i) Summit supercomputer, each node containing

TABLE V
MACHINE CONFIGURATIONS

Model	AMD EPYC 7642	Intel Xeon E5-2650	IBM POWER9
Microarchitecture	Zen 2	Haswell	Power9
Frequency	2.3 GHz	2.3 GHz	2.3 GHz
Total hyperthreads	96	56	168
Total Cores	46 × 1 socket	14 × 2 sockets	21 × 2 sockets
LLC size	256 MiB	35 MiB	110 MiB
DRAM	512 GB	1 TB	2 TB

TABLE VI
DESCRIPTION OF SYMMETRIC SPARSE TENSORS AND IMPLEMENTATIONS THAT CAN SUCCESSFULLY COMPUTE S^3 TTMC WITHIN A REASONABLE TIME LIMIT ON ANDES SUPERCOMPUTER. AS CTF ALSO USES THE *UCOO* FORMAT FOR S^3 TTMC COMPUTATION, THE ABSENCE OF NON-ZERO MEMOIZATION RESULTS IN LACK OF SCALING TO LARGE AND HUGE DATASETS

Category	Tensor	Order	Dim	$unnz$	R	Implementations			
						S^{3}TTMC-UCOO	CTF (*UCOO*)	*SPLATT (FCSF)*	S^{3}TTMC-CSS
Small	S5	5	100	100K	3	5.935	13.669	**0.011**	0.043
	S6	6	100	10K	2	1.214	624.137	0.013	**0.008**
	S7	7	50	1K	3	29.617	699.046	0.024	**0.006**
Large	L6	6	400	1M	2	105.804	✗	0.842	**0.798**
	L7	7	400	1M	3	✗	✗	19.339	**13.170**
	L9	9	400	10K	5	✗	✗	96.735	**55.123**
	L10	10	400	1K	5	✗	✗	161.518	**76.224**
	L11	11	400	100	6	✗	✗	1193.156	**439.209**
	Walmart	8	15624	3082	4	✗	✗	1.345	**0.672**
Huge	H8	8	400	1M	4	✗	✗	✗	**175.407**
	H12	12	400	10K	3	✗	✗	✗	**131.118**
	Amazon	14	12981	2131	2	✗	✗	✗	**9.400**

2TB DDR4 RAM and two POWER9 sockets with 21 cores each, (ii) Andes supercomputer, each node containing 1TB RAM and two Intel Xeon E5-2695 processors with 14 cores each, and (iii) POD Computing node with 512GB RAM and 2 AMD EPYC processors each with 23 cores, as detailed in Table V. The code is written in C++ and parallelized using OpenMP, configured to double-precision floating point arithmetic and 64-bit unsigned integers. It is compiled with GCC 8.3.0 and Netlib LAPACK 3.8.0 for linear algebra routines.

B. Datasets

We tested ten synthetic and two real-world symmetric tensors used in [26] of varying orders and number of IOU non-zeros, as shown in Table VI. The *Walmart* and *Amazon* datasets are constructed from real-world hypergraphs, and have been used to evaluate the performance of hypergraph clustering algorithms. *Walmart*, an order-8 symmetric tensor [44], is generated from a hypergraph representing Walmart's user relations where hyperedges are co-purchased products. *Amazon*, an order-14 symmetric tensor [25], is extracted from a user-review hypergraph with product reviews as hyperedges, and nodes are labelled with product categories. Only one CSF tree and the fastest leaf-to-root TTMc algorithm outlined in the work [29] are used from SPLATT owing to the symmetry feature. CTF [20], [41] is run with LAPACK 3.8.0 [45] and HPTT 1.0.5 [46] functionalities. The twelve datasets have been divided into three categories - *small*, *large*, and *huge* - depending on the ability of each implementation to compute S^3 TTMC successfully within a reasonable time limit. The choice of R for rank-R S^3 TTMC on each dataset is determined by our cost analysis in Section V-E. Among all datasets in *small* and *large* categories, except for dataset S5, R is set to the largest R_{max}. We chose $R = 3$ for S5 with the purpose to demonstrate a case where *SPLATT* outperforms S^3 TTMC-CSS and verify Theorem V.1.

C. Overall Performance

We compare the performance of our S^3 TTMC-CSS with *SPLATT*, *S^3 TTMC-UCOO* and *CTF* for all three categories of datasets on Andes.

Small datasets Fig. 7(a) demonstrates the trade-off in the runtime of the S^3 TTMC operations and the size of the symmetric tensor storage format observed in each of the implementations. The size of the symmetric tensors chosen is small enough to ensure that all baseline implementations run successfully without timing out. We observe three groups in Fig. 7(a) - (i) *S^3 TTMC-UCOO* and CTF implementations, which are the most compact but exhibit high runtimes; (ii) *SPLATT* using *FCSF*, which has a massive memory overhead but is significantly faster than *S^3 TTMC-UCOO* and CTF using *UCOO*; and (iii) S^3 TTMC-CSS, which is comparable to *UCOO* in terms of storage requirement while still being faster than *SPLATT*, *S^3 TTMC-UCOO* and CTF. Fig. 7(a) aligns perfectly with Fig. 1 and shows that S^3 TTMC-CSS does an effective trade-off between storage and efficient computation. S^3 TTMC-CSS is faster than *SPLATT* for S6 and S7 datasets, as the rank R chosen satisfies Theorem V.1. However, as we have chosen $R > \exp(H_4 - 1.285)$ ((15)), S^3 TTMC-CSS performs more work than *SPLATT* and is hence slower.

Large and huge datasets The lack of scalability of both *S^3 TTMC-UCOO* and CTF with increasing tensor order in Fig.

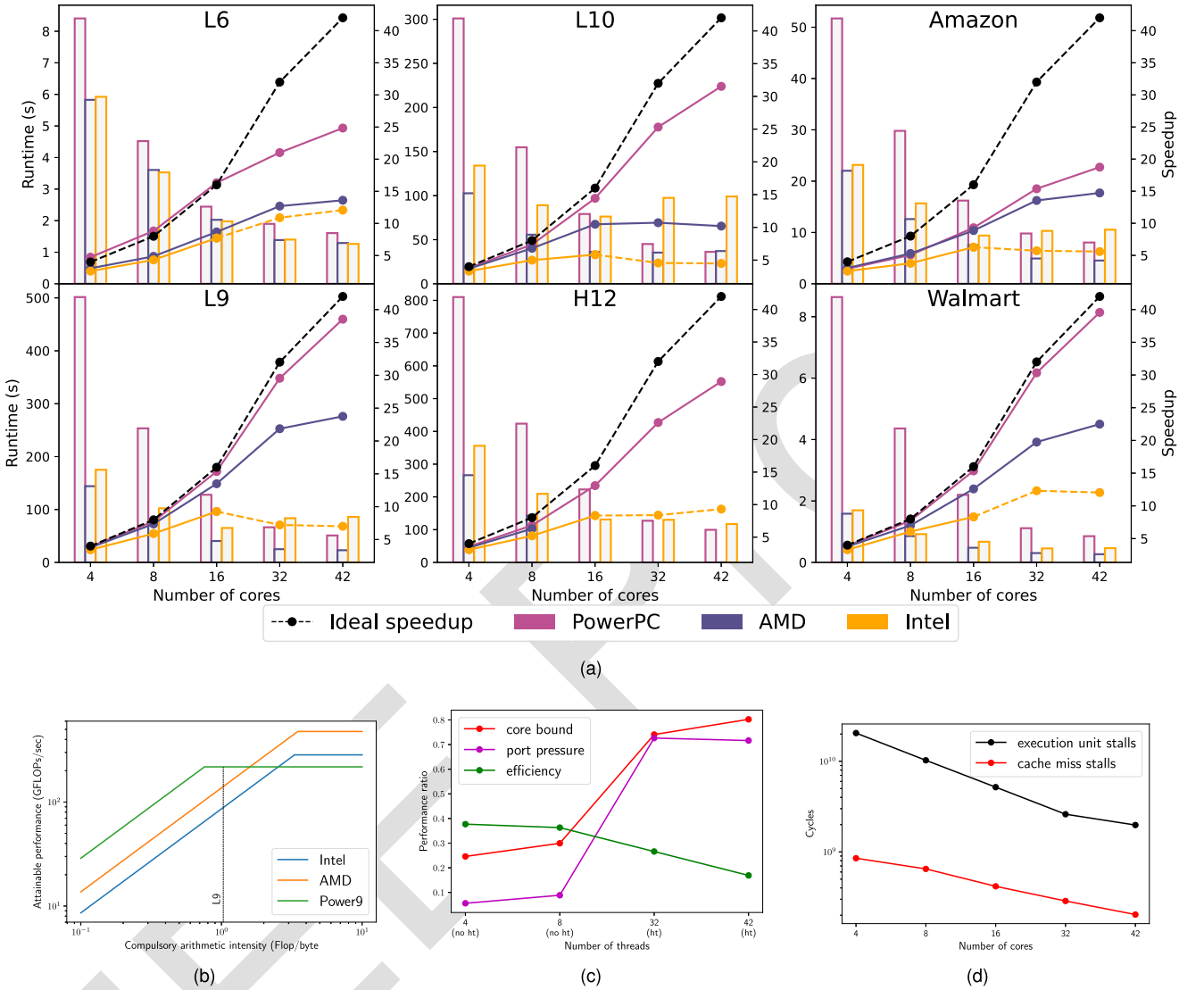

Fig. 6. (a) Scaling performance for representative datasets across different architectures. Hyperthreading has been enabled for Intel processor runs (indicated by dashed lines) as the machine has only 28 cores available. (b) Naïve roofline model to understand the superior speedup of S^3 TTMc-CSS on Power9 when compared to Intel and AMD using L9 as representative dataset. S^3 TTMc-CSS is bandwidth-bound on Intel (without hyperthreading) and AMD processors, but compute-bound on Power9. (c) Hyperthreading performance of representative L9 dataset on Intel processor (d) Hardware counters on Power9 to compare completion stalls on execution units with cache miss stalls for S^3 TTMc-CSS run on L9 dataset.

7(a) makes them an infeasible baseline for comparison on *large* datasets. While CTF runs out of memory for all datasets, *S^3 TTMc-UCOO* does not complete in reasonable time for order-7 tensors and higher, as can be seen in Fig. 7(b). For the best thread configuration, we achieve up to 2.72× speedup over *SPLATT*, the fastest state-of-the-art baseline (details in Section III). Moreover, our algorithm is always faster than *SPLATT* for *large* symmetric tensors. For *huge* datasets, SPLATT runs out of memory, and only S^3 TTMc-CSS can be used to compute S^3 TTMc. H8 dataset has the largest memory requirement of all datasets, with CSS storage requirement of 1.66GB.

D. Performance Across Architectures

Fig. 6(a) demonstrates the scaling performance of S^3 TTMc-CSS on POWER9, AMD EPYC, and Intel Xeon processors, for the large and huge datasets in Table VI.

Performance impact of hyperthreading Note that the performance of S^3 TTMc-CSS decreases on the Intel processor for 32 and 42 cores due to hyperthreading, since every node on Andes machine has 28 cores in total. Fig. 6(c) compares the hyperthreading performance metrics for S^3 TTMc-CSS on L9 dataset. Using two threads per core instead of one shifts the S^3 TTMc-CSS from memory-bound to core-bound, which makes sense as execution units are a shared resource between threads per core. Moreover, owing to small number of execution gaps in the pipeline, the penalty due to increased resource contention with hyperthreading outweighs time saved by utilizing idle resources. The utilization levels of the core's execution units is measured through efficiency i.e., the ratio of the number of instructions retired and the total number of micro-operation slots per core. Decreasing efficiency coincides with increasing port pressure, which is the fraction of cycles the CPU performance was limited due to core computation issues like low Instruction

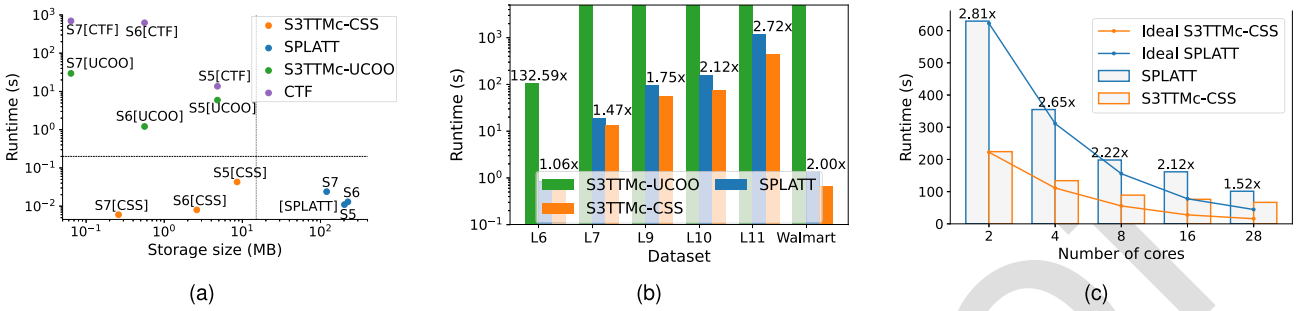

Fig. 7. Comparison of overall performance of S^3 TTMc-CSS with *SPLATT*, *S^3 TTMc-UCOO* and CTF for datasets in Table VI across different architectures. (a) Storage-runtime trade off of all algorithms for *small* tensors. The grouping of runtimes and storage size is similar to Fig. 1. (b) S^3 TTMc runtime for *large* tensors. *S^3 TTMc-UCOO* completes only for dataset L6. The speed-up of S^3 TTMc-CSS over each of the baselines is indicated over the corresponding bar. (c) Multithreading scalability of S^3 TTMc-CSS compared to *SPLATT* for order-10 tensor L10. The speed-up of CSS over *SPLATT* is specified above the blue bars. Ideal runtime of S^3 TTMc-CSS and *SPLATT* for p cores is the ratio of the sequential runtime to the number of cores, p.

Level Parallelism(ILP) or contention on multiply units. This is to be expected due the large number of multiplication operations in the Kronecker product computations in S^3 TTMc-CSS.

Note that on AMD and Power9 processors, SMT is not enabled. The algorithm is faster on AMD EPYC processors than Power9 and Intel Xeon because of the following factors - (i) the memory-intensive workload resulting in worse performance on NUMA systems like Power9, (ii) AMD having faster processor than Power9, and higher peak performance than Intel Xeon, and (iii) AMD having lower last-level cache latency than Intel due to the fragmented L3 cache. On the other hand, S^3 TTMc-CSS shows best scalability on POWER9 processors since POWER9 offers much higher memory bandwidth than AMD and Xeon processors, which serves to reduce the memory bottleneck in S^3 TTMc-CSS. While the loosely coupled parallel execution in S^3 TTMc-CSS is usually bandwidth-bound, it is compute-bound on Power9 processor, as can be seen for the representative L9 dataset in Fig. 6(b) and (d). This implies that as the size of available cache increases with the number of cores, low DRAM access latency associated with intermediate Kronecker product prefetching on Power9 results in fewer cache misses, and thus we see better scalability for S^3 TTMc-CSS on Power9 processor than Intel or AMD. We also observe better than ideal speedup on POWER9 for the smallest large dataset considered - L6 - until 16 cores. The overhead associated with forking and synchronization outweighs the parallelism gains for larger number of cores as the dataset is small enough to fit entirely in the cache, and any further increase in cache size does not improve performance.

E. Thread Scalability

We analyze thread scalability of S^3 TTMc-CSS and *SPLATT* for a representative order-10 tensor, L10. Note that we have excluded *S^3 TTMc-UCOO* and CTF as they do not run to completion for any of the *large* datasets, owing to their lack of scalability with increasing tensor order. *SPLATT* and S^3 TTMc-CSS show similar scalability due to the similarities in tree layout of the data structure, and its depth wise traversal to compute S^3 TTMc, as can be seen in Fig. 7(c). Though S^3 TTMc-CSS is faster than *SPLATT*, the scaling becomes poor for higher order tensors using more cores, as can be seen in Fig. 7(c), because of the size of memory allocated per thread increases, resulting in degrading data locality and cache behavior. Another effect of increasing thread memory is the worsening data locality in the Kronecker product computations. Though the memoization strategy described in Algorithm 3 minimizes the number of stored partial Kronecker products, the overhead due to random accesses in the intermediate workspace still remains, and manifests as suboptimal scaling.

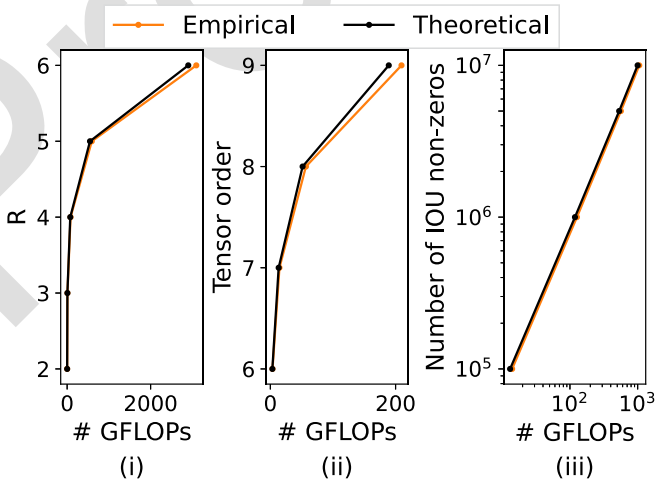

Fig. 8. Effect of (i) matrix rank R on S^3 TTMc-CSS for order-11 tensor L11, (ii) tensor order N for rank 2-S^3 TTMc on tensors having 1M IOU non-zeros, and (iii) number of IOU non-zeros in order-7 tensor on S^3 TTMc-CSS with $R = 3$.

F. Parameter Sweep

We analyze the behavior of S^3 TTMc-CSS as a function of three key parameters affecting the S^3 TTMc operation - (i) rank R chosen for S^3 TTMc, (ii) tensor order, and (iii) number of IOU non-zeros in the tensor. We compare the number of floating point operations performed by S^3 TTMc-CSS with the theoretical number of FLOPs defined in (11). Note that empirical observations closely follow our cost model in Section V-E.

Fig. 8(i) shows the behavior of the S^3 TTMc operation with varying matrix rank for order-11 tensor, L11. From (11), we see

that C^{CSS} grows exponentially with R for a given symmetric tensor, and we correspondingly observe nonlinear scaling in Fig. 8(i). This is different from the close to linear scalability of general sparse TTMc [32]. Similarly, a non-linear growth in the number of FLOPs is observed with increasing tensor order (Fig. 8(ii)). We have chosen $R = 2$ - which is R_{max} for order-6 tensors (Table IV) - as it satisfies Theorem V.1 for all tensor orders considered in the figure. It is difficult to estimate the ideal scaling trend with both R and N from (11) due to the dependence on the level of the CSS tree that dominates the cost of C^{CSS}. For the number of IOU non-zeros $unnz$, (11) indicates linear scaling, the same as observed in Fig. 8(iii) for rank-3 S^3 TTMc on order-7 tensor. With increasing $unnz$, we also observe non-linear growth in storage size, similar to *FCSF*; though size of *UCOO* increases linearly. There is, however, less significant change in the size of CSS format with increasing mode size I due to the inherent sparsity of the tensor.

G. Symmetric Tucker Decomposition

We compute symmetric Tucker decomposition on the largest real-world dataset in Table VI, the Amazon dataset. As all the three baselines run out of memory, we use S^3 TTMc-CSS to compute S^3 TTMc for the decomposition (Line 2 in Algorithm 1). We are able to run rank-2 symmetric Tucker decomposition for the order-14 Amazon dataset in ~ 1.78hrs, which shows the applicability of this work in the analysis of real world hypergraphs. S^3 TTMc-CSS provides a practical framework to use symmetric Tucker decomposition in a hierarchical clustering approach to hypergraph analytics.

VII. Conclusion

This work focuses on the S^3 TTMc operation - the specialization of the frequently used tensor times matrix chain operation to symmetric tensors - used in symmetric tensor decomposition. We propose a computation-aware storage format CSS designed for symmetric tensors, with which we can efficiently perform S^3 TTMc in parallel. We underscore the utility of our algorithm by demonstrating a better trade-off between memory and run-time over SPLATT, S^3 TTMc-UCOO, and CTF for multiple synthetic and real-world datasets. In the future, we intend to explore the applicability of CSS to other tensor operations like the Matricized Tensor Times Khatri-Rao Product (MTTKRP) and decompositions like tensor train. We plan to improve the access patterns of intermediate results while executing S^3 TTMc-CSS in parallel, adopt operation memoization, and apply our efficient S^3 TTMc operation to hypergraph analytics.

Acknowledgments

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility. We would like to thank AMD for the donation of critical hardware and support resources from its HPC Fund that made this work possible. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

References

[1] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, Sep. 1966. [Online]. Available: https://link.springer.com/article/10.1007/BF02289464

[2] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3551–3582, Jul. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7891546/

[3] A. Anandkumar, D. Hsu, S. M. Kakade, and M. Telgarsky, "Tensor decompositions for learning latent variable models," Tech. Rep., 2014.

[4] D. Ghoshdastidar and A. I. E. In, "A provable generalized tensor spectral method for uniform hypergraph partitioning Ambedkar Dukkipati," Tech. Rep.

[5] D. Ghoshdastidar and A. Dukkipati, "Consistency of spectral partitioning of uniform hypergraphs under planted partition model," Tech. Rep.

[6] Z. T. Ke, F. Shi, and D. Xia, "Community detection for hypergraph networks via regularized tensor power iteration," Tech. Rep., 2020.

[7] J. Anderson, M. Belkin, N. Goyal, L. Rademacher, and J. Voss, "The more, the merrier: The blessing of dimensionality for learning large Gaussian mixtures," in *Proc. 27th Conf. Learn. Theory*, 2014, pp. 1135–1164. [Online]. Available: http://proceedings.mlr.press/v35/anderson14.html

[8] N. Goyal, S. Vempala, and Y. Xiao, "Fourier PCA and robust tensor decomposition," in *Proc. Annu. ACM Symp. Theory Comput.*, 2013, pp. 584–593, *arXiv:1306.5825*.

[9] E. Estrada and J. A. Rodriguez-Velazquez, "Complex networks as hypergraphs," *Physica A: Stat. Mechanics Appl.*, vol. 364, pp. 581–594, May 2005, *arXiv:physics/0505137*, doi: 10.1016/j.physa.2005.12.002.

[10] S. R. Gallagher, M. Dombrower, and D. S. Goldberg, "Using 2-node hypergraph clustering coefficients to analyze disease-gene networks," in *Proc. 5th ACM Conf. Bioinf., Comput. Biol., Health Inform.*, New York, New York, USA, 2014, pp. 647–648. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2649387.2660817

[11] D. Ghoshdastidar and A. Dukkipati, "Uniform hypergraph partitioning: Provable tensor methods and sampling techniques," *J. Mach. Learn. Res.*, vol. 18, no. 50, pp. 1–41, 2017. [Online]. Available: http://jmlr.org/papers/v18/16--100.html

[12] A. Shashua, R. Zass, and T. Hazan, "Multi-way clustering using super-symmetric non-negative tensor factorization," in *Eur. Conf. Comput. Vis.*, 2005, pp. 595–608.

[13] D. Zhou, J. Huang, and B. Schölkopf, "Learning with Hypergraphs: Clustering, classification, and embedding," Tech. Rep.

[14] S. Sherman and T. G. Kolda, "Estimating higher-order moments using symmetric tensor decomposition," Tech. Rep.

[15] T. G. Kolda, "Numerical optimization for symmetric tensor decomposition," Oct. 2014, *arXiv:1410.4536*, doi: 10.1007/s10107–015-0895-0.

[16] J. Brachat, P. Comon, B. Mourrain, and E. Tsigaridas, "Symmetric tensor decomposition," in *Proc. Eur. Signal Process. Conf.*, 2009, pp. 525–529, *arXiv:0901.3706*.

[17] P. Comon, G. Golub, L.-H. Lim, and B. Mourrain, "Symmetric tensors and symmetric tensor rank," Feb. 2008, *arXiv:0802.1681*.

[18] S. Smith, N. Ravindran, N. Sidiropoulos, and G. Karypis, "SPLATT: Efficient and parallel sparse tensor-matrix multiplication," in *Proc. IEEE 29th Int. Parallel Distrib. Process. Symp.*, 2015, pp. 61–70.

[19] J. Li, Y. Ma, and R. Vuduc, "ParTI! : A parallel tensor infrastructure for multicore CPUs and GPUs," Oct. 2018. [Online]. Available: http://parti-project.org

[20] E. Solomonik, D. Matthews, J. Hammond, and J. Demmel, "Cyclops tensor framework: Reducing communication and eliminating load imbalance in massively parallel contractions," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, 2013, pp. 813–824. [Online]. Available: https://ieeexplore.ieee.org/document/6569864/

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

[22] J. Li, J. Sun, and R. Vuduc, "HiCOO: Hierarchical storage of sparse tensors," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2018, pp. 238–252. [Online]. Available: https://ieeexplore.ieee.org/document/8665782/
[23] B. Liu, C. Wen, A. D. Sarwate, and M. M. Dehnavi, "A unified optimization approach for sparse tensor operations on GPUs," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2017, pp. 47–57.
[24] I. Nisa, J. Li, A. Sukumaran-Rajam, P. S. Rawat, S. Krishnamoorthy, and P. Sadayappan, "An efficient mixed-mode representation of sparse tensors," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2019, pp. 49:1–49:25. [Online]. Available: http://doi.acm.org/10.1145/3295500.3356216
[25] J. Ni, J. Li, and J. McAuley, "Justifying recommendations using distantly-labeled reviews and fine-grained aspects," in *Proc. Conf. Empir. Methods Natural Lang. Process. 9th Int. Joint Conf. Natural Lang. Process.*, 2019, pp. 188–197. [Online]. Available: https://www.aclweb.org/anthology/D19--1018
[26] S. Shivakumar, J. Li, R. Kannan, and S. Aluru, "Efficient parallel sparse symmetric tucker decomposition for high-order tensors," in *Proc. SIAM Conf. Appl. Comput. Discrete Algorithms*, 2021, pp. 193–204. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611976830.18
[27] J. Li, J. Sun, and R. Vuduc, "HiCOO: Hierarchical storage of sparse tensors," in *Proc. ACM/IEEE Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Dallas, TX, USA, 2018, pp. 238–252.
[28] A. E. Helal et al., "ALTO: Adaptive linearized storage of sparse tensors," in *Proc. Int. Conf. Supercomput.*, 2021, pp. 404–416.
[29] S. Smith and G. Karypis, "Accelerating the tucker decomposition with compressed sparse tensors," in *Proc. Eur. Conf. Parallel Process.*, 2017, pp. 653–668.
[30] O. Kaya and B. Ucar, "High performance parallel algorithms for the tucker decomposition of sparse tensors," in *Proc. 45th Int. Conf. Parallel Process.*, 2016, pp. 103–112. [Online]. Available: https://ieeexplore.ieee.org/document/7573808/
[31] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "SPLATT: Efficient and parallel sparse tensor-matrix multiplication," in *Proc. IEEE 29th Int. Parallel Distrib. Process. Symp.*, 2015, pp. 61–70.
[32] Y. Ma, J. Li, X. Wu, C. Yan, J. Sun, and R. Vuduc, "Optimizing sparse tensor times matrix on GPUs," *J. Parallel Distrib. Comput.*, vol. 129, no. c, pp. 99–109, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731518305161
[33] S. Smith and G. Karypis, "Tensor-matrix products with a compressed sparse tensor," in *Proc. ACM 5th Workshop Irregular Appl. Architectures Algorithms*, New York, New York, USA, 2015, pp. 1–7. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2833179.2833183
[34] O. Kaya and B. Uçar, "Parallel candecomp/parafac decomposition of sparse tensors using dimension trees," *SIAM J. Sci. Comput.*, vol. 40, no. 1, pp. C99–C130, 2018.
[35] J. Cambré, L. De Lathauwer, and B. De Moor, "Best rank-(R, R, R) super-symmetric tensor approximation - A continuous-time approach," in *Proc. IEEE Signal Process. Workshop Higher-Order Statist.*, 1999, pp. 242–246.
[36] G. Ballard, T. Kolda, and T. Plantenga, "Efficiently computing tensor eigenvalues on a GPU," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, 2011, pp. 1340–1348. [Online]. Available: https://ieeexplore.ieee.org/document/6008988/
[37] M. D. Schatz, T. M. Low, R. A. van de Geijn, and T. G. Kolda, "Exploiting symmetry in tensors for high performance: Multiplication with symmetric tensors," *SIAM J. Sci. Comput.*, vol. 36, no. 5, pp. C453–C479, 2014. [Online]. Available: http://epubs.siam.org/doi/10.1137/130907215
[38] E. Solomonik, "Provably efficient algorithms for numerical tensor algebra," Ph.D. dissertation, EECS Dept., Univ. California, Berkeley, Sep. 2014.
[39] E. Solomonik, J. Demmel, and T. Hoefler, "Communication lower bounds for tensor contraction algorithms," ETH Zürich, Tech. Rep., 2015.
[40] M. Baskaran, B. Meister, R. Lethin, and J. Cai, "Optimization of symmetric tensor computations," in *Proc. IEEE Conf. High Perform. Extreme Comput., Waltham, MA, USA*, 2015, pp. 1–7.
[41] E. Solomonik and T. Hoefler, "Sparse tensor algebra as a parallel programming model," 2015. [Online]. Available: https://github.com/solomonik/ctf
[42] S. Smith and G. Karypis, "SPLATT: The surprisingly parallel spArse tensor toolkit," 2016. [Online]. Available: http://cs.umn.edu/splatt/
[43] E. W. Weisstein, "Harmonic number,".
[44] I. Amburg, N. Veldt, and A. Benson, "Clustering in graphs and hypergraphs with categorical edge labels," in *Proc. Web Conf.*, 2020, pp. 706–717. [Online]. Available: https://dl.acm.org/doi/10.1145/3366423.3380152
[45] E. Anderson et al., *LAPACK Users' Guide*, Philadelphia, PA, USA: SIAM, 1999.
[46] P. Springer, T. Su, and P. Bientinesi, "HPTT: A high-performance tensor transposition C++ library," in *Proc. 4th ACM SIGPLAN Int. Workshop Libraries, Lang., Compilers Array Program.*, 2017, pp. 56–62.

Shruti Shivakumar received the BTech degree from the Indian Institute of Technology, Madras. She is currently working towrad the PhD degree with the School of Computational Science and Engineering, Georgia Institute of Technology. Her research is with the intersection of high performance computing and numerical linear algebra, with a focus on parallel algorithms for sparse tensor algebra.

Jiajia Li received the PhD degree from the Georgia Institute of Technology and Institute of Computing Technology, Chinese Academy of Sciences. She is an assistant professor with the North Carolina State University. Her research emphasizes on high performance sparse tensor algorithms. She was a research scientist with the Pacific Northwest National Laboratory.

Ramakrishnan Kannan is the group leader for discrete algorithms with Oak Ridge National Laboratory. His research expertise is in distributed machine learning and graph algorithms on HPC platforms and their application to scientific data with a focus on accelerating scientific discovery by reducing computation time from weeks to seconds. He was the lead for DSNAPSHOT for a COVID-19 project, which is a finalist for the esteemed Association of Computing Machinery's Gordon Bell Award in 2021, and listing Summit in 3rd place on Graph500 benchmark using the fewest resources; this is the first time an OLCF system has ranked in Graph500. With more than 24 patents issued in USPTO, he was an IBM master inventor.

Srinivas Aluru (Fellow, IEEE) is an executive director of the Institute for Data Engineering and Science (IDEaS) and professor with the School of Computational Science and Engineering, Georgia Institute of Technology. He co-leads the NSF South Big Data Regional Innovation Hub which nurtures big data partnerships between organizations in the 16 Southern States and Washington D.C., and the NSF Transdisciplinary Research Institute for Advancing Data Science. He conducts research in high performance computing, data science, bioinformatics and systems biology, combinatorial scientific computing, and applied algorithms. He was a chair of the ACM Special Interest Group on Bioinformatics, Computational Biology and Biomedical Informatics (SIGBIO; 2015-2021) and is the current editor-in-chief of the *IEEE Transactions on Computational Biology and Bioinformatics*. He is a recipient of the NSF CAREER award, IBM faculty award, Swarnajayanti Fellowship from the Government of India, the John. V. Atanasoff Discovery Award from Iowa State University, and the Outstanding Senior Faculty Research award, the Dean's award for faculty excellence, and the Outstanding Research Program Development Award at Georgia Tech. He received the IEEE Computer Society Meritorious Service and Golden Core awards, and is a Fellow of the AAAS, ACM, and SIAM.