SIAM J. SCI. COMPUT. © 2024 Society for Industrial and Applied Mathematics
Vol. 46, No. 2, pp. A1186–A1213

# PARALLEL RANDOMIZED TUCKER DECOMPOSITION ALGORITHMS[*]

RACHEL MINSTER[†], ZITONG LI[‡], AND GREY BALLARD[†]

**Abstract.** The Tucker tensor decomposition is a natural extension of the singular value decomposition (SVD) to multiway data. We propose to accelerate Tucker tensor decomposition algorithms by using randomization and parallelization. We present two algorithms that scale to large data and many processors, significantly reduce both computation and communication cost compared to previous deterministic and randomized approaches, and obtain nearly the same approximation errors. The key idea in our algorithms is to perform randomized sketches with Kronecker-structured random matrices, which reduces computation compared to unstructured matrices and can be implemented using a fundamental tensor computational kernel. We provide probabilistic error analysis of our algorithms and implement a new parallel algorithm for the structured randomized sketch. Our experimental results demonstrate that our combination of randomization and parallelization achieves accurate Tucker decompositions much faster than alternative approaches. We observe up to a 16X speedup over the fastest deterministic parallel implementation on 3D simulation data.

**Key words.** Tucker decompositions, tensors, randomized algorithms, parallel algorithms, low-rank, multilinear algebra

**MSC codes.** 15A69, 15A18, 15B52, 65F99, 68W10, 68W15, 68W20

**DOI.** 10.1137/22M1540363

## 1. Introduction.

Tucker decompositions are low-rank tensor approximations capable of approximating multidimensional data with large compression rates while maintaining high accuracy. Large scale multidimensional data arises from many applications such as simulations of partial differential equations, data mining, facial recognition, and imaging. Processing these data requires computationally efficient methods. Randomized algorithms have been used to efficiently compute Tucker decompositions in works such as [1, 4, 9, 11, 25, 27, 30, 33], but the growing size of data is outpacing even randomized algorithms. Scaling these methods to handle large data calls for efficient parallelization. Many high-performance implementations of deterministic algorithms have been developed for Tucker decompositions [3, 7, 21, 12, 24]. We develop both sequential and parallel randomized algorithms that efficiently compute Tucker decompositions of large-scale multidimensional data by reducing both computation and communication compared to previous work.

As we review in section 2, there are two dominant computational kernels to computing a Tucker decomposition: computing matrix singular value decompositions (SVD) and computing tensor-times-matrix (TTM) products. For the deterministic algorithms HOSVD [14] (Algorithm 2.1) and STHOSVD [29] (Algorithm 2.2), computing the SVD is the typical bottleneck, and various methods trade off accuracy for reduced computational complexity. Our goal is to reduce the complexity of the SVD computation via randomization and remove it as the dominant cost without sacrificing too much accuracy. Existing randomized Tucker approaches, discussed in

[*]Submitted to the journal's Numerical Algorithms for Scientific Computing section December 12, 2022; accepted for publication (in revised form) November 27, 2023; published electronically April 2, 2024.
https://doi.org/10.1137/22M1540363
**Funding:** This work is supported by the National Science Foundation under grant CCF-1942892.
[†]Wake Forest University, Winston-Salem, NC 27109 USA (minsterr@wfu.edu, ballard@wfu.edu).
[‡]University of California Irvine, Irvine, CA 92697 USA (zitongl5@uci.edu).

section 3, apply low-rank matrix approximation algorithms in place of matrix SVDs. These matrix algorithms include randomized range finder [17] (see Algorithm 2.3), which computes part of the low-rank approximation and involves a slight overestimate of the target rank, or randomized SVD [17] (RandSVD: see Algorithm 2.4), which involves a second pass over the data to obtain the final approximation with the exact target rank.

We propose two randomized algorithms in section 4, one based on HOSVD and one based on STHOSVD, which have comparable accuracies and running times. In our algorithms, we use the randomized range finder approach with Kronecker-structured random matrices, which reduces the computational complexity of the sketch compared to previous randomized approaches. As a significant added practical benefit, the Kronecker structure reduces the amount of random number generation compared to unstructured random matrices such as Gaussian. Furthermore, we propose a deterministic truncation of the resulting core (with overestimated ranks) in order to achieve the exact target ranks, obtaining the same effect as RandSVD-based approaches at much lower cost. We show that our HOSVD-based algorithm can be as computationally efficient as our STHOSVD-based algorithm by employing a dimension tree optimization to avoid recomputation across sketches using memoization.

To accompany our algorithms, we develop probabilistic error guarantees in section 5 for a randomized matrix algorithm using a Kronecker product of random matrices. We use the matrix results to obtain theoretical guarantees for our Tucker algorithms. Our bounds differ from previous results by accounting for the Kronecker structure and rank truncation in our algorithms and by reducing the amplification factors.

In section 6, we describe the parallelization of our proposed algorithms for distributed memory using the TuckerMPI library [3], allowing us to scale the algorithms to large datasets that cannot be processed on a single server. While previous work has combined randomization and parallelization, our implementation is the first to parallelize the randomized sketch, which significantly reduces the computational cost. Moreover, in exploiting the Kronecker structure of our sketch, we implement a new parallel algorithm that communicates less data than the algorithm used by TuckerMPI and in fact minimizes interprocessor communication for the computation [2].

Our experimental results are presented in section 7. We validate the error guarantees of section 5 and show empirically that our structured random matrices are just as accurate as standard Gaussian random matrices. Using synthetic data as well as two large simulation datasets, we demonstrate that our parallel randomized algorithms given in section 6 scale well to thousands of cores and outperform alternative deterministic and randomized algorithms, achieving speedups of up to 16× over the state-of-the-art implementation of the best deterministic algorithm.

## 2. Background.

We first review the relevant background on tensors and randomized algorithms for matrices. For more details on tensors, see [22], and for more details on randomized algorithms, see [17].

### 2.1. Tensor notation and operations.

A tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ is a $d$-way array. We can unfold a $d$-mode tensor along each of its modes, or dimensions; the mode-$j$ unfolding, denoted $\mathbf{X}_{(j)}$, is a matrix with columns formed as the mode-$j$ fibers of the tensor. Let $\|\cdot\|$ denote the tensor norm, which generalizes the matrix Frobenius norm. Since the following products will be frequently used to describe the sizes and ranks of a tensor, we define the following notations: $n^{\circledast} = \prod_{k=1}^{d} n_k$, $n_i^{\circleddash\!\!\!\!<} = \prod_{k=1}^{i-1} n_k$, $n_i^{>} = \prod_{k=i+1}^{d} n_k$, $n_i^{\oslash} = \prod_{k\neq i} n_k$.

One key operation for tensors is the TTM product. A tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ is multiplied along mode $j$ by a matrix $\mathbf{A} \in \mathbb{R}^{m\times n_j}$, denoted by $\boldsymbol{\mathcal{X}} \times_j \mathbf{A}$, to obtain a tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{n_1\times\cdots\times n_{j-1}\times m\times n_{j+1}\times\cdots\times n_d}$. This product can also be expressed in terms of its mode-$j$ unfolding as $\mathbf{Y}_{(j)} = \mathbf{A}\mathbf{X}_{(j)}$. We can also multiply a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ by up to $d$ matrices $\mathbf{A}_j \in \mathbb{R}^{m_j\times n_j}$, $j = 1,\ldots,d$, across distinct modes to obtain $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_1 \mathbf{A}_1 \times_2 \mathbf{A}_2 \times\cdots\times_d \mathbf{A}_d \in \mathbb{R}^{m_1\times\cdots\times m_d}$. We call this product a multi-TTM; it is also known as a multilinear multiplication. If unfolded along mode $j$, we have $\mathbf{Y}_{(j)} = \mathbf{A}_j\mathbf{X}_{(j)}\left(\mathbf{A}_d\otimes\cdots\otimes\mathbf{A}_{j+1}\otimes\mathbf{A}_{j-1}\otimes\cdots\otimes\mathbf{A}_1\right)^{\top}$, where $\otimes$ is the matrix Kronecker product.

**2.2. Tucker decomposition.** The Tucker decomposition of a given tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ of multirank $\mathbf{r} = (r_1,\ldots,r_d)$, where $r_j = \operatorname{rank}(\mathbf{X}_{(j)})$ for each $j$, represents $\boldsymbol{\mathcal{X}}$ as the product of a core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1\times\cdots\times r_d}$ and $d$ factor matrices $\mathbf{U}_j \in \mathbb{R}^{n_j\times r_j}$ such that $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U}_1 \times\cdots\times_d \mathbf{U}_d$. We can also obtain a low-rank approximation to $\boldsymbol{\mathcal{X}}$ in the Tucker form by taking the target rank $(r_1,\ldots,r_d)$, or size of the core tensor $\boldsymbol{\mathcal{G}}$, to be less than the ranks of the unfoldings in each mode.

*Higher-order SVD (HOSVD) and Sequentially Truncated HOSVD.* Two algorithms that compute low-rank Tucker decompositions of tensors are the higher-order SVD (HOSVD) [14] and sequentially truncated HOSVD (STHOSVD) [29]. The HOSVD algorithm forms each factor matrix $\mathbf{U}_j$ from the first $r_j$ left singular vectors of the mode unfolding $\mathbf{X}_{(j)}$, and once all the factor matrices are computed, computes the core tensor as $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_1 \mathbf{U}_1^{\top} \times\cdots\times_d \mathbf{U}_d^{\top}$ (see Algorithm 2.1).

The STHOSVD algorithm is similar to HOSVD, but instead of handling all modes independently, it processes the modes in a predetermined sequence. After the first factor matrix is computed from the first $r_j$ left singular vectors of $\mathbf{X}_{(j)}$, we truncate in that mode by computing a partially truncated core tensor via a TTM with the factor matrix, $\boldsymbol{\mathcal{G}} \times_j \mathbf{U}_j^{\top}$. We then use the partially truncated core $\boldsymbol{\mathcal{G}}$ for the next mode instead of the full tensor $\boldsymbol{\mathcal{X}}$, as shown in Algorithm 2.2.

**Algorithm 2.1** HOSVD [14].

```
1: function $[\boldsymbol{\mathcal{G}}, \{\mathbf{U}_j\}] = \text{HOSVD}(\boldsymbol{\mathcal{X}}, \mathbf{r})$
2:   for $j = 1:d$ do
3:     $\mathbf{U}_j$ = first $r_j$ left sing. vecs. of $\mathbf{X}_{(j)}$
4:   end for
5:   $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_1 \mathbf{U}_1^{\top} \times\cdots\times_d \mathbf{U}_d^{\top}$
6: end function
```

**Algorithm 2.2** STHOSVD [29].

```
1: function $[\boldsymbol{\mathcal{G}}, \{\mathbf{U}_j\}] = \text{STHOSVD}(\boldsymbol{\mathcal{X}}, \mathbf{r})$
2:   $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}}$
3:   for $j = 1:d$ do
4:     $\mathbf{U}_j$ = first $r_j$ left singular vecs. of $\mathbf{G}_{(j)}$
5:     $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{G}} \times_j \mathbf{U}_j^{\top}$
6:   end for
7: end function
```

**Algorithm 2.3** Randomized Range Finder [17].

```
1: function $\mathbf{Q} =$ RandRangeFinder($\mathbf{M}, \mathbf{\Omega}$)
2:     $\mathbf{Y} = \mathbf{M\Omega}$
3:     Compute thin QR $\mathbf{Y} = \mathbf{QR}$
4: end function
```

**Algorithm 2.4** Randomized SVD [17].

```
1: function $[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] =$ RandSVD($\mathbf{M}, r, \mathbf{\Omega}$)
2:     $\mathbf{Q} =$ RandRangeFinder($\mathbf{M}, \mathbf{\Omega}$)
3:     $\mathbf{B} = \mathbf{Q}^\top \mathbf{M}$
4:     Compute thin SVD $\mathbf{B} = \hat{\mathbf{U}} \mathbf{\Sigma} \mathbf{V}^\top$
5:     $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}(:, 1:r)$
6:     Truncate $\mathbf{\Sigma} = (1:r, 1:r)$, $\mathbf{V} = \mathbf{V}(:, 1:r)$
7: end function
```

**2.3. Randomized matrix algorithms.** The algorithm, made popular by [17] and shown in Algorithm 2.3, efficiently computes a low-rank representation of a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$. Given a target rank $r$ and oversampling parameter $p$, we multiply $\mathbf{M}$ by a random matrix $\mathbf{\Omega} \in \mathbb{R}^{n \times \ell}$ with $\ell = r + p$ such that $\ell < m$, to form $\mathbf{Y} \in \mathbb{R}^{m \times \ell}$, a matrix made up of random linear combinations of the columns of $\mathbf{M}$. We then compute a thin QR decomposition of $\mathbf{Y}$ to obtain a matrix $\mathbf{Q} \in \mathbb{R}^{m \times \ell}$ whose range is a good estimate of the range of $\mathbf{M}$. Projecting $\mathbf{M}$ onto the range of $\mathbf{Q}$ gives us the low-rank approximation $\mathbf{M} \approx \mathbf{QQ}^\top \mathbf{M}$. We can choose any random distribution for random matrix $\mathbf{\Omega}$. In this paper, we will consider both subsampled random Hadamard transform (SRHT) and standard Gaussian matrices.

Note that the resulting approximation from Algorithm 2.3 is actually rank-$\ell$. If we seek a rank-$r$ approximation, further truncation is necessary. One way of truncating is to take a thin SVD of $\mathbf{Q}^\top \mathbf{M}$, which is the process taken in the randomized SVD algorithm in [17], reproduced in Algorithm 2.4. We will adapt this truncation method in the algorithms developed in later sections.

## 3. Related work.

Our work builds on three different categories of previous work, namely, randomized algorithms for Tucker decompositions, probabilistic analysis of randomized algorithms, and parallel algorithms for tensor computations.

*Randomized algorithms.* There has been much previous work on randomized algorithms for Tucker decompositions; a good survey of this work can be found in [1]. The basic algorithms for randomized HOSVD and randomized STHOSVD are proposed in [33], while later work improves on the algorithms in various ways. One important distinction among randomized algorithms is the rank of the output approximation. In [30, 33], the approximation has rank $\ell = \mathbf{r} + p$ as the (Algorithm 2.3) is used without additional truncation. Other algorithms, such as those presented in [9, 27] do not oversample at all, limiting the potential accuracy of their methods. In [11], the authors use the, but truncate by only taking the first $r_j$ columns of each factor matrix. The randomized SVD algorithm (Algorithm 2.4) can be applied instead to both oversample and more accurately obtain the desired rank-$\mathbf{r}$ approximation, which is done in [4, 25]. Our approach is most similar to the randomized SVD approach, but we apply it in a holistic manner, as discussed in section 4.1.

Another common improvement to the basic randomized algorithms comes from exploiting structure in the random matrices used to reduce storage and/or computational costs, as well as the number of random entries generated. Khatri–Rao products of random matrices are used in [9, 27], compact random matrices are employed in [4], and Kronecker products of random matrices are used in [10, 11]. Our work is most similar to [11] as we also employ Kronecker products, but our algorithms improve upon those in [11] by truncating to the desired target rank in a more accurate manner. Kronecker product structure has also been exploited in other tensor decompositions besides Tucker decompositions: in [5, 19], Kronecker products of random matrices are used to accelerate algorithms for CP decompositions; while in [13], Kronecker product structure was exploited in the context of the tensor-train decomposition. We also discuss how to implement our algorithms on distributed systems and provide improved probabilistic analysis.

*Error analysis.* To accompany the discussed randomized algorithms, other work has developed probabilistic error analysis. Analysis for the standard version of randomized HOSVD is presented in [16, 25], and for the standard randomized STHOSVD algorithm in [9, 25] for Khatri–Rao products of Gaussian matrices and dense Gaussian random matrices, respectively. Previous error analysis has been done for a randomized STHOSVD algorithm employing Kronecker products of subsampled randomized Fourier transform (SRFT) matrices in [11], but we make several improvements on this work. Our error bound, for our algorithms with Kronecker products of the real-valued equivalent of SRFTs, i.e., SRHT matrices, has an improved error constant.

*Parallel algorithms.* Our parallel algorithms and implementation, described in section 6, are built upon the foundation of TuckerMPI [3] and its improvements [24]. TuckerMPI is a C++/MPI library that implements the STHOSVD algorithm to compute Tucker decompositions of large dense tensors that are distributed across machines. It implements many other utilities such as file I/O and subroutines such as parallel TTM, that we use in our algorithms and experiments. Other parallel implementations of Tucker algorithms have been developed for both dense [7] and sparse [21] tensors. The work most similar to ours combines parallelism and randomization to compute Tucker decompositions of dense tensors [12]. The approach taken by Choi, Liu, and Chakaravarthy [12] is to employ STHOSVD (Algorithm 2.2) and compute the SVD of $\mathbf{G}_{(j)}$ by computing its Gram matrix in parallel and then sequentially applying randomized SVD (Algorithm 2.4) to the Gram matrix. Our approach differs in that we parallelize the randomized algorithm and avoid the Gram matrix computation; we provide a more detailed comparison in section 6.4.

We propose a novel implementation of a parallel algorithm for the multi-TTM computation in section 6.1. Communication lower bounds for this computation and theoretical algorithms that achieve those lower bounds are presented in [2]. The multi-TTM algorithm that we present in this paper can be seen as a specialization of [2, Algorithm 8.1], but our implementation is novel. We also highlight previous optimizations of tensor computations using dimension trees, a memoization technique. First introduced in [26] in the context of computing gradients of the CP decomposition optimization problem, dimension trees have also been used for Tucker decompositions. For example, the higher-order orthogonal iteration (e.g., [15]) benefits from storing and reusing intermediate quantities across tensor modes as demonstrated in [20]. We use the dimension tree approach in a different context in one of our randomized algorithms; this process is described in section 4.4.

## 4. Sequential algorithms.

We present our novel sequential algorithms before discussing how they may be implemented in parallel. There are several different variations of algorithms we will present, and we provide a hierarchy diagram for how they relate in Figure 1. The first optimizations we show, using sequential truncation and randomization, have been developed in previous work. We then progress to using a Kronecker product of random matrices within the randomized algorithms, and then finally reusing Kronecker factors in the HOSVD case. The two most efficient algorithms we present are in the leftmost boxes of the two main subtrees: randomized STHOSVD with Kronecker products (Algorithm 4.3) and randomized HOSVD with reused Kronecker factors (Algorithm 4.4).

In all the algorithms presented in this section, we employ a holistic truncation approach. Given a tensor $\boldsymbol{\mathcal{X}}$, target rank $\mathbf{r}=(r_1,\ldots,r_d)$, and oversampling parameter $p$, and letting $\ell_j = r_j + p$ for $j = 1,\ldots,d$, we first apply the randomized range finder algorithm (Algorithm 2.3) to each mode unfolding, and obtain an initial core $\hat{\boldsymbol{\mathcal{G}}} \in \mathbb{R}^{\ell_1\times\cdots\times\ell_d}$. We then apply the truncation phase of Algorithm 2.4 by computing a deterministic STHOSVD of $\hat{\boldsymbol{\mathcal{G}}}$ such that $\hat{\boldsymbol{\mathcal{G}}} \approx \boldsymbol{\mathcal{G}} \times_1 \mathbf{V}_1 \times \cdots \times_d \mathbf{V}_d$. The rank-$\mathbf{r}$ representation of $\boldsymbol{\mathcal{X}}$ is then $\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{G}} \times_1 \hat{\mathbf{U}}_1\mathbf{V}_1 \times \cdots \times_d \hat{\mathbf{U}}_d\mathbf{V}_d$.

### 4.1. Randomized HOSVD/STHOSVD.

The first algorithms we present are the basic form of randomized algorithms on which we improve throughout the paper. Algorithms 4.1 and 4.2 are similar to other randomized projection algorithms for Tucker decompositions found in [1, 25] except for the truncation approach. Instead of directly applying the randomized SVD algorithm (Algorithm 2.4) to each mode

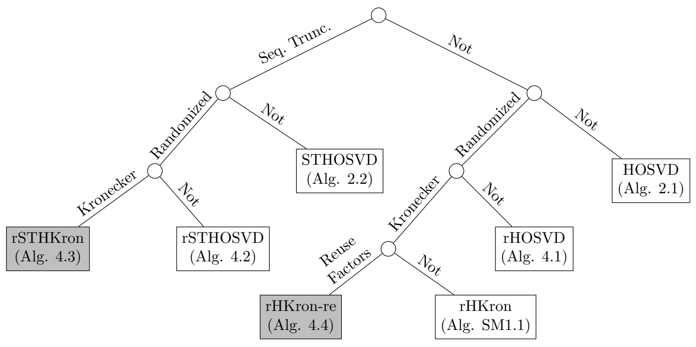


FIG. 1. *Hierarchy of algorithms, with most efficient algorithms highlighted in gray.*

**Algorithm 4.1** Randomized HOSVD.

```
1: function $[\boldsymbol{\mathcal{G}},\{\mathbf{U}_j\}] = \mathrm{rHOSVD}(\boldsymbol{\mathcal{X}},\mathbf{r},p)$
2:   for $j=1:d$ do
3:     Draw $\mathbf{\Omega}\in\mathbb{R}^{n_j^{\oslash}\times\ell_j}$
4:     $\hat{\mathbf{U}}_j = \mathrm{RandRangeFinder}(\mathbf{X}_{(j)},\mathbf{\Omega})$
5:   end for
6:   $\hat{\boldsymbol{\mathcal{G}}} = \boldsymbol{\mathcal{X}} \times_1 \hat{\mathbf{U}}_1^\top \times\cdots\times_d \hat{\mathbf{U}}_d^\top$
7:   $[\boldsymbol{\mathcal{G}},\{\mathbf{V}_j\}] = \mathrm{STHOSVD}(\hat{\boldsymbol{\mathcal{G}}},\mathbf{r})$
8:   $\mathbf{U}_j = \hat{\mathbf{U}}_j\mathbf{V}_j$ for $j=1,\ldots,d$
9: end function
```

**Algorithm 4.2** Randomized STHOSVD.

```
1: function $[\boldsymbol{\mathcal{G}}, \{\mathbf{U}_j\}] = $ rSTHOSVD($\boldsymbol{\mathcal{X}}, \mathbf{r}, p$)
2:   $\hat{\boldsymbol{\mathcal{G}}} = \boldsymbol{\mathcal{X}}$
3:   for $j = 1 : d$ do
4:     Draw $\boldsymbol{\Omega} \in \mathbb{R}^{r_j^{\otimes} n_j^{\otimes} \times \ell_j}$
5:     $\hat{\mathbf{U}}_j = $ RandRangeFinder$(\hat{\mathbf{G}}_{(j)}, \boldsymbol{\Omega})$
6:     $\hat{\boldsymbol{\mathcal{G}}} = \hat{\boldsymbol{\mathcal{G}}} \times_j \hat{\mathbf{U}}_j^\top$
7:   end for
8:   $[\boldsymbol{\mathcal{G}}, \{\mathbf{V}_j\}] = $ STHOSVD$(\hat{\boldsymbol{\mathcal{G}}}, \mathbf{r})$
9:   $\mathbf{U}_j = \hat{\mathbf{U}}_j \mathbf{V}_j$ for $j = 1, \dots, d$
10: end function
```

**Algorithm 4.3** Randomized STHOSVD with Kronecker product.

```
1: function $[\boldsymbol{\mathcal{G}}, \{\mathbf{U}_j\}] = $ rSTHKron($\boldsymbol{\mathcal{X}}, \mathbf{r}, p$)
2:   $\hat{\boldsymbol{\mathcal{G}}} = \boldsymbol{\mathcal{X}}$
3:   Compute matrix of subranks $\mathbf{S}$
4:   for $j = 1 : d$ do
5:     Draw $d - 1$ random matrices $\boldsymbol{\Phi}_{j,k} \in \mathbb{R}^{s_{j,k} \times \ell_k}$ for $k < j$ and $\boldsymbol{\Phi}_{j,k} \in \mathbb{R}^{s_{j,k} \times n_k}$ for
       $k > j$
6:     $\boldsymbol{\mathcal{Y}} \leftarrow \hat{\boldsymbol{\mathcal{G}}} \times_1 \boldsymbol{\Phi}_{j,1} \times \cdots \times_{j-1} \boldsymbol{\Phi}_{j,j-1} \times_{j+1} \boldsymbol{\Phi}_{j,j+1} \times \cdots \times_d \boldsymbol{\Phi}_{j,d}$
7:     Compute thin QR $\mathbf{Y}_{(j)} = \hat{\mathbf{U}}_j \mathbf{R}$
8:     $\hat{\boldsymbol{\mathcal{G}}} = \hat{\boldsymbol{\mathcal{G}}} \times_j \hat{\mathbf{U}}_j^\mathsf{T}$
9:   end for
10:  $[\boldsymbol{\mathcal{G}}, \{\mathbf{V}_j\}] = $ STHOSVD$(\hat{\boldsymbol{\mathcal{G}}}, \mathbf{r})$
11:  $\mathbf{U}_j = \hat{\mathbf{U}}_j \mathbf{V}_j$ for $j = 1, \dots, d$
12: end function
```

unfolding, we take the holistic approach described above. We can use this technique both with HOSVD, shown in Algorithm 4.1, and STHOSVD, shown in Algorithm 4.2.

**4.2. Randomized HOSVD/STHOSVD with Kronecker product.** Our main algorithm combines the HOSVD/STHOSVD algorithms with a special case of the randomized range finder algorithm used on each mode unfolding. Within the randomized range finder, we will represent the random matrix $\boldsymbol{\Omega}$ as a Kronecker product of random matrices each with a small number of columns instead of a single large $\boldsymbol{\Omega}$. This allows us both to employ a multi-TTM operation instead of matrix multiplication, reducing the computational complexity, and to exploit properties of tall-and-skinny matrices in our parallel algorithms. Specifically, for a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, rank $\mathbf{r} = (r_1, \dots, r_d)$, and oversampling parameter $p$ with $\ell_j = r_j + p$, define $\boldsymbol{\Omega}_j \in \mathbb{R}^{n_j^{\oslash} \times \ell_j}$ as$\boldsymbol{\Omega}_j = (\boldsymbol{\Phi}_{j,d} \otimes \cdots \otimes \boldsymbol{\Phi}_{j,j+1} \otimes \boldsymbol{\Phi}_{j,j-1} \otimes \cdots \otimes \boldsymbol{\Phi}_{j,1})^\mathsf{T}$ with $\boldsymbol{\Phi}_{j,k} \in \mathbb{R}^{s_{j,k} \times n_k}$ a random matrix from some distribution (e.g., Gaussian, SRHT, etc.), and $\mathbf{S} \in \mathbb{N}^{d \times d}$ a matrix of subranks. We define $\mathbf{S}$ to have entries $s_{j,k}$ the $k$th subrank for mode $j$, where $k \neq j$, and diagonal entries $s_{j,j} = 1$ for $j = 1, \dots, d$, such that the row products $\prod_{k=1}^{d} s_{j,k} = \ell_j$ for $j = 1, \dots, d$. We summarize the steps for our algorithm in STHOSVD form in Algorithm 4.3 (rSTHKron), and include an HOSVD version in Algorithm SM2.1. Note that lines 5 to 7 in Algorithm 4.3 consist of applying randomized range finder to the current mode unfolding using the Kronecker product as our random matrix.

*Choosing subranks.* The restriction on the subranks $\mathbf{S}$ is that $\prod_{k\neq j}^{d} s_{j,k} = \ell_j = r_j + p$ for each row $j = 1, \dots d$. In practice, we can actually choose the subranks so that $\prod_{k\neq j}^{d} s_{j,k} \geq \ell_j$. Satisfying this looser condition means we are essentially increasing the oversampling we are already doing through parameter $p$. This frees us to use heuristics to choose the subranks. We choose each row of $\mathbf{S}$ to be composed of $d-1$ integer factors of $\ell_j$. In the case that $\ell_j$ does not have exactly $d-1$ integer factors, we adjust the oversampling parameter until we can obtain the correct number of factors.

**4.3. Randomized HOSVD with Kronecker factor reuse.** An additional adaptation we make to reduce the number of random values generated and computation is to reuse the components of the Kronecker product $\mathbf{\Omega}$. Instead of generating $d-1$ random matrices for each mode as in Algorithm 4.3, we generate $d$ random matrices $\{\mathbf{\Phi}_j\}$ once, and use different combinations of $d-1$ of those same matrices in each mode. This approach is summarized in Algorithm 4.4. One benefit of this approach is that we generate significantly fewer random entries. We can also, as will be addressed in section 6, implement dimension trees to reduce computational cost. This variation only works for the HOSVD approach as the size of each $\mathbf{\Phi}_j$ remains the same, while it would change after each mode in an STHOSVD approach.

*Choosing subranks.* Note that in this case we compute only one vector of subranks $\mathbf{s} \in \mathbb{N}^d$, instead of a matrix as in Algorithm 4.3. We compute these subranks heuristically as well, and in this case in a straightforward manner, deriving the formula $s_i = \lceil (\prod_{j=1}^{d} \ell_j)^{\frac{1}{d-1}} / \ell_i \rceil$, from the conditions $\mathbf{s} \in \mathbb{N}^d$ and $\prod_{k=1}^{d-1} s_k \geq \ell_j$ for $j = 1, \dots, d$. This formula, while more straightforward, is more constricting than the method we use to compute subranks for Algorithm 4.3 because $s_j \ell_j$ is approximately fixed for each mode $j$. This can create problems when computing with tensors with skewed modes and ranks, so we recommend not reusing Kronecker factors in these cases.

**4.4. Dimension tree optimization.** In line 5 of Algorithm 4.4, we perform a multi-TTM to compute the sketch tensor $\boldsymbol{\mathcal{Y}}$ for each mode, resulting in $d$ multi-TTM products. Notice, however, that the $d-1$ random matrices that we use in each multi-TTM are drawn from the same set of $d$ random matrices $\{\mathbf{\Phi}_1, \dots, \mathbf{\Phi}_d\}$. Thus a significant number of computations in each multi-TTM are repeated and can be reused. The dimension tree concept, which has been employed to reduce computational complexity in other algorithms [7, 20], also applies in this situation.

**Algorithm 4.4** Randomized HOSVD with Kronecker product reusing factors.

```
 1: function $[\boldsymbol{\mathcal{G}}, \{\mathbf{U}_j\}] = \text{rHKron}(\boldsymbol{\mathcal{X}}, \mathbf{r}, p)$
 2:     Compute subranks $\mathbf{s}$
 3:     Draw $d$ random matrices $\mathbf{\Phi}_k \in \mathbb{R}^{s_k \times n_k}$ for $k = 1, \dots, d$
 4:     for $j = 1 : d$ do
 5:         $\boldsymbol{\mathcal{Y}} \leftarrow \boldsymbol{\mathcal{X}} \times_1 \mathbf{\Phi}_1 \times \cdots \times_{j-1} \mathbf{\Phi}_{j-1} \times_{j+1} \mathbf{\Phi}_{j+1} \times \cdots \times_d \mathbf{\Phi}_d$
 6:         Compute thin QR $\mathbf{Y}_{(j)} = \mathbf{U}_j \mathbf{R}$
 7:     end for
 8:     $\hat{\boldsymbol{\mathcal{G}}} = \boldsymbol{\mathcal{X}} \times_1 \hat{\mathbf{U}}_1^\top \times \cdots \times_d \hat{\mathbf{U}}_d^\top$
 9:     $[\boldsymbol{\mathcal{G}}, \{\mathbf{V}_j\}] = \text{STHOSVD}(\hat{\boldsymbol{\mathcal{G}}}, \mathbf{r})$
10:     $\mathbf{U}_j = \hat{\mathbf{U}}_j \mathbf{V}_j$ for $j = 1, \dots, d$
11: end function
```

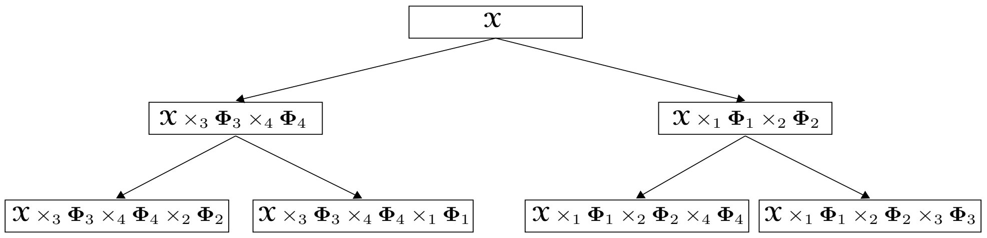

$\mathcal{X}$
$\mathcal{X} \times_3 \mathbf{\Phi}_3 \times_4 \mathbf{\Phi}_4$
$\mathcal{X} \times_1 \mathbf{\Phi}_1 \times_2 \mathbf{\Phi}_2$
$\mathcal{X} \times_3 \mathbf{\Phi}_3 \times_4 \mathbf{\Phi}_4 \times_2 \mathbf{\Phi}_2$
$\mathcal{X} \times_3 \mathbf{\Phi}_3 \times_4 \mathbf{\Phi}_4 \times_1 \mathbf{\Phi}_1$
$\mathcal{X} \times_1 \mathbf{\Phi}_1 \times_2 \mathbf{\Phi}_2 \times_4 \mathbf{\Phi}_4$
$\mathcal{X} \times_1 \mathbf{\Phi}_1 \times_2 \mathbf{\Phi}_2 \times_3 \mathbf{\Phi}_3$

FIG. 2. *Dimension tree for computing the sketches of a* 4*-way tensor* $\boldsymbol{\mathcal{X}}$ *in Algorithm* 4.4.

TABLE 1
*Complexity reduction examples that can be achieved using dimension trees.*

| | **With dimTree** | **Without dimTree** |
|---|---|---|
| $\boldsymbol{d=3}$ | $2(2rn^3+3r^2n^2)$ | $2(3rn^3+3r^2n^2)$ |
| $\boldsymbol{d=4}$ | $2(2rn^4+2r^2n^3+4r^3n^2)$ | $2(4rn^4+4r^2n^3+4r^3n^2)$ |
| $\boldsymbol{d=5}$ | $2(2rn^5+2r^2n^4+3r^3n^3+5r^4n^2)$ | $2(5rn^5+5r^2n^4+5r^3n^3+5r^4n^2)$ |

In our implementation, we use a binary tree with $d$ leaf nodes for a $d$-way tensor. For example, given a 4-way tensor $\boldsymbol{\mathcal{X}}$ and four random matrices $\{\mathbf{\Phi}_1, \mathbf{\Phi}_2, \mathbf{\Phi}_3, \mathbf{\Phi}_4\}$, the 4 multi-TTM operations that would be carried out without a dimension tree are shown in the four leaf nodes of Figure 2. The dimension tree provides an efficient way to perform the computations shared between each pair of adjacent leaf nodes and store the results in memory to reduce computation. For a $d$-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n\times\cdots\times n}$ and $d$ random matrices $\{\mathbf{\Phi}_j\}$ each of size $r \times n$, the cost of using dimension trees to sketch every unfolding of $\boldsymbol{\mathcal{X}}$ is shown in Table 1. When $r \ll n$, both costs are approximated by the first terms in the summations. As $d$ increases, the cost reduction that comes from using a dimension tree increases proportionally to $d/2$.

**4.5. Computational complexity.** To analyze the computational cost of our algorithms, we consider the notationally simpler case with a $d$-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n\times\cdots\times n}$, target rank $(r,\dots,r)$, and oversampling parameter $p$, letting $\ell = r + p$. We will also let the subranks $\{s_{j,k}\}$ and $\{s_k\}$ all be the same value, which we denote as $s = \ell^{d-1}$. Assume $s < r \ll n$.

There are two dominant costs for each algorithm presented: computing an SVD for each mode, and forming the core tensor via a multi-TTM or a series of TTMs (in the STHOSVD case). We show the leading terms of both dominant steps for the standard algorithms, Algorithms 2.1 and 2.2, and compare to those for Algorithms 4.1 to 4.4 in Table 2. More details on the cost analysis for Algorithms 2.1 and 2.2 can be found in [29], while more details on the analysis for Algorithms 4.1 and 4.2 can be found in [25]. Based on the terms shown in Table 2, we advocate the use of either Algorithm 4.3 or Algorithm 4.4, and we show more detailed analysis on these two algorithms in section SM2.

**4.6. Comparison with previous work.** We compare our sequential algorithms with previous approaches, in particular those based on the use of randomized SVD [25] and Kronecker-structured random matrices [11]. As summarized in [1], randomization can be used in multiple ways to compute Tucker approximations of tensors. The most similar approaches to Algorithms 4.1 and 4.2 are [25, Algs. 3.1 and 3.2]. These algorithms replace the deterministic SVD within Algorithms 2.1 and 2.2 with randomized SVD (Algorithm 2.4). The randomized SVD requires computing

Table 2
*Computational cost of sequential algorithms.*

| | **Leading term** | |
|---|---|---|
| **Algorithm** | **SVD** | **TTM** |
| HOSVD (2.1) | $dn^{d+1}$ | $2rn^d$ |
| STHOSVD (2.2) | $n^{d+1}$ | $2rn^d$ |
| rHOSVD (4.1) | $2d\ell n^d$ | $2\ell n^d$ |
| rSTHOSVD (4.2) | $2\ell n^d$ | $2\ell n^d$ |
| rSTHOSVDkron (4.3) | $2\ell^{\frac{1}{d-1}} n^d$ | $2\ell n^d$ |
| rHOSVDkronreuse (4.4) | $4\ell^{\frac{1}{d-1}} n^d$ | $2\ell n^d$ |

the thin SVD of the projection of the approximate column space; it increases the computational cost compared to randomized range finder by a constant factor greater than 2. Because Algorithms 4.1 and 4.2 use randomized range finder (Algorithm 2.3), they involve only one operation with the input (the random sketch) at the expense of working with the oversampled rank $\ell$ rather than the target rank $\mathbf{r}$ until the final core truncation step. The oversampled ranks are only slightly larger than the target ranks, so Algorithms 4.1 and 4.2 are computationally cheaper than [25, Algorithms 3.1 and 3.2].

The Kronecker-structured random sketches of Algorithms 4.3 and 4.4 are similar to [11, Algorithm 3.1]. This algorithm uses an SRFT to sketch each mode's matricization within STHOSVD. Besides using a complex-valued random matrix, the key difference with Algorithm 4.3 is the truncation strategy: after computing the thin QR decomposition of the sketched matrix with $\ell$ columns, all but the first $r$ columns are truncated. As we demonstrate in section 7.1, our truncation strategy using a deterministic STHOSVD of the core tensor computes a more accurate approximation.

To the best of our knowledge, reusing Kronecker factors and exploiting the possible memoization of temporary quantities as presented in subsections 4.3 and 4.4 have not been considered before.

## 5. Error analysis.

We now present error guarantees for Algorithms 4.3 and 4.4. Let $\boldsymbol{\mathcal{T}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U}_1 \times \cdots \times_d \mathbf{U}_d$ be the approximation from either algorithm, and $\hat{\boldsymbol{\mathcal{T}}} = \hat{\boldsymbol{\mathcal{G}}} \times_1 \hat{\mathbf{U}}_1 \times \cdots \times_d \hat{\mathbf{U}}_d$ be the intermediate rank-$\ell$ approximation. The overall form of the error is

$$\varepsilon_{\text{total}} = \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{T}}\| \leq \|\boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{T}}}\| + \|\hat{\boldsymbol{\mathcal{T}}} - \boldsymbol{\mathcal{T}}\| = \varepsilon_{\text{rand}} + \varepsilon_{\text{core}},$$

where $\varepsilon_{\text{rand}}$ represents the error from forming the rank-$\ell$ approximation, and $\varepsilon_{\text{core}}$ represents the error in truncating the approximation to rank $\mathbf{r}$. The component $\varepsilon_{\text{core}}$ is equivalent to the error in computing the STHOSVD of $\hat{\boldsymbol{\mathcal{G}}}$, which we can see from

$$\varepsilon_{\text{core}} = \left\| \left(\hat{\boldsymbol{\mathcal{G}}} - \boldsymbol{\mathcal{G}} \times_1 \mathbf{V}_1 \times \cdots \times_d \mathbf{V}_d\right) \times_1 \hat{\mathbf{U}}_1 \times \cdots \times_d \hat{\mathbf{U}}_d \right\| = \|\hat{\boldsymbol{\mathcal{G}}} - \boldsymbol{\mathcal{G}} \times_1 \mathbf{V}_1 \times \cdots \times_d \mathbf{V}_d\|,$$

where the second equality follows from the orthonormality of $\{\hat{\mathbf{U}}_j\}$. We can then apply the error bound for STHOSVD [29, Theorem 6.5] to the initial core $\hat{\boldsymbol{\mathcal{G}}}$, i.e, $\varepsilon_{\text{core}}^2 \leq \sum_{j=1}^{d} \sum_{i=r_j+1}^{\ell_j} \sigma_i^2 \left(\hat{\mathbf{G}}_{(j)}\right)$, where $\sigma_i(\mathbf{A})$ denotes the $i$th singular value of $\mathbf{A}$.

Then, as $\hat{\boldsymbol{\mathcal{G}}}$ is a random quantity, we need to relate the singular values of $\hat{\mathbf{G}}_{(j)}$ to the singular values of $\mathbf{X}_{(j)}$ to get a deterministic upper bound. Because matrices $\{\hat{\mathbf{U}}_j\}$ are orthonormal, the modewise singular values of $\hat{\boldsymbol{\mathcal{G}}}$ cannot be larger than those of $\boldsymbol{\mathcal{X}}$. A formal proof of this fact can be found in section SM3. Thus, we have

$$\varepsilon_{\text{core}}^2 \leq \sum_{j=1}^{d} \sum_{i=r_j+1}^{\ell_j} \sigma_i^2\left(\mathbf{X}_{(j)}\right). \tag{5.1}$$

The rest of this section considers the component $\varepsilon_{\text{rand}}$. Starting with an error bound for randomized range finder (Algorithm 2.3) using a Kronecker product of SRHT matrices, we extend the results to an error bound for our HOSVD-type algorithm (Algorithm 4.4) and discuss how to adapt the proof for our STHOSVD-type algorithm (Algorithm 4.3).

## 5.1. Matrix bound.

*Random matrix.* Let $n = \prod_{j=1}^{q} n_j$ and $\ell = \prod_{j=1}^{q} s_j$. We will consider a Kronecker product of SRHT matrices of the form

$$\mathbf{\Omega} = \mathbf{D}(\mathbf{H}_1 \otimes \mathbf{H}_2 \otimes \cdots \otimes \mathbf{H}_q) \in \mathbb{R}^{n \times \ell}, \tag{5.2}$$

where $\mathbf{D} \in \mathbb{R}^{n\times n}$ is a diagonal matrix with independent and identically distributed (i.i.d.) entries from the Rademacher distribution, i.e., either 1 or $-1$ with equal probability, and for each $j = 1,\dots,q$, $\mathbf{H}_j \in \mathbb{R}^{n_j \times s_j}$ is formed from $s_j$ randomly sampled columns of an $n_j \times n_j$ Walsh–Hadamard matrix scaled by $\frac{1}{\sqrt{n_j}}$. Due to this scaling, $\mathbf{\Omega}$ is orthonormal. Also note that the $\{\mathbf{H}_j\}$ matrices are generated independently.

*Notation.* We now introduce the setup and notation for our main theorem. Following the notation of [17], let $\mathbf{X} = \mathbf{U\Sigma V}^\top$ be the SVD of matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ with $m \leq n$. Fix target rank $r$, and partition the SVD as

$$\mathbf{X} = \mathbf{U}\begin{bmatrix} \mathbf{\Sigma}_1 & \\ & \mathbf{\Sigma}_2 \end{bmatrix}\begin{bmatrix} \mathbf{V}_1^\top \\ \mathbf{V}_2^\top \end{bmatrix} \tag{5.3}$$

with $\mathbf{\Sigma}_1 \in \mathbb{R}^{r\times r}, \mathbf{\Sigma}_2 \in \mathbb{R}^{(m-r)\times(m-r)}, \mathbf{V}_1 \in \mathbb{R}^{n\times r}$, and $\mathbf{V}_2 \in \mathbb{R}^{n\times(m-r)}$ such that $\mathbf{V}_1$ and $\mathbf{V}_2$ have orthonormal columns. Now let $\mathbf{\Omega} \in \mathbb{R}^{n\times\ell}$ be the random matrix defined in (5.2), and define

$$\mathbf{\Omega}_1 = \mathbf{V}_1^\top \mathbf{\Omega}, \quad \mathbf{\Omega}_2 = \mathbf{V}_2^\top \mathbf{\Omega}. \tag{5.4}$$

We are interested in bounding the largest singular values of $\mathbf{\Omega}_1^\dagger$ and $\mathbf{\Omega}_2$. We will use the orthonormality of $\mathbf{\Omega}_2$ to bound its largest singular values, but the argument is more complicated for $\mathbf{\Omega}_1$. Here we will adapt the approach in [31], allowing for the application to a Kronecker product of independent SRHT matrices. This bound is stated in Lemma 5.1, which we will then use to prove our approximation error bound. We prove Lemma 5.1 in Appendix A.

LEMMA 5.1. *Let $\mathbf{\Omega}_1 \in \mathbb{R}^{r\times\ell}$ be as defined in* (5.4) *with $\mathbf{\Omega} \in \mathbb{R}^{n\times\ell}$ the Kronecker product of $q$ SRHT matrices as defined in* (5.2)*, where $n = \prod_{j=1}^{q} n_j$ and $\ell = \prod_{j=1}^{q} s_j$. If there exist real numbers $\alpha, \beta > 1$ that satisfy*

$$\min_k\{s_k\} \geq \frac{\alpha^2\beta}{(\alpha-1)^2}(r^2+r), \tag{5.5}$$

*then $\frac{1}{\sigma_{\min}^2(\mathbf{\Omega}_1)} \leq \frac{\alpha n}{\ell}$ with probability at least $1 - \frac{1}{\beta}$.*

*Remark* 5.2. The bound in Lemma 5.1 contains a factor of $n$, the number of rows of Kronecker product $\mathbf{\Omega}$. This factor is not present in singular value bounds for Gaussian random matrices, and is a consequence of using SRHT matrices, as seen in bounds involving a single SRHT matrix in [28]. In the empirical results shown in section 7.1, we see that using SRHT matrices produces similar accuracy to Gaussian matrices; we anticipate that future work in random matrix theory will be able to improve this factor.

THEOREM 5.3. *Let $\hat{\mathbf{X}} = \mathbf{Q}\mathbf{Q}^\top\mathbf{X}$ be the approximation given by the randomized range finder of matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ with target rank $r$, oversampling parameter $p$ such that $\ell = r + p \le \min\{m,n\}$, and random sampling matrix $\boldsymbol{\Omega}$ as defined in* (5.2)*. If there exist real numbers $\alpha, \beta > 1$ that satisfy* (5.5)*, then with probability at least $1 - \frac{1}{\beta}$,*

$$\|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top\mathbf{X}\|_F \le \left(\left(1 + \frac{\alpha n}{\ell}\right) \sum_{i=r+1}^{\min\{m,n\}} \sigma_i^2(\mathbf{X})\right)^{1/2}.$$

*Proof.* Immediately from [17, Theorem 9.1] and recalling the partitioning from (5.3) and (5.4), we have

$$\begin{aligned} \|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top\mathbf{X}\|_F^2 = \|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top)\mathbf{X}\|_F^2 &\le \|\boldsymbol{\Sigma}_2\|_F^2 + \|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\|_F^2 \\ &\le \left(1 + \|\boldsymbol{\Omega}_2\|_2^2\|\boldsymbol{\Omega}_1^\dagger\|_2^2\right)\|\boldsymbol{\Sigma}_2\|_F^2. \end{aligned} \tag{5.6}$$

We can apply the singular value bounds from Lemma 5.1 to obtain the bounds for $\|\boldsymbol{\Omega}_1^\dagger\|_2^2$. With probability at least $1 - \frac{1}{\beta}$, $\|\boldsymbol{\Omega}_1^\dagger\|_2^2 = \frac{1}{\sigma_{\min}^2(\boldsymbol{\Omega}_1)} \le \frac{\alpha n}{\ell}$. For $\|\boldsymbol{\Omega}_2\|_2^2$, we use properties of both $\mathbf{V}_2$ and $\boldsymbol{\Omega}$: $\|\boldsymbol{\Omega}_2\|_2 = \|\mathbf{V}_2^\top\boldsymbol{\Omega}\|_2 \le \|\boldsymbol{\Omega}\|_2 = 1$, as $\mathbf{V}_2^\top$ has orthonormal rows, and $\boldsymbol{\Omega}$ has orthonormal columns as the Kronecker product of matrices with orthonormal columns. Combining these bounds, we obtain $\|\boldsymbol{\Omega}_2\|_2^2\|\boldsymbol{\Omega}_1^\dagger\|_2^2 \le \frac{\alpha n}{\ell}$ with probability at least $1 - \frac{1}{\beta}$. From (5.6), we now have $\|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top\mathbf{X}\|_F^2 \le \left(1 + \frac{\alpha n}{\ell}\right)\|\boldsymbol{\Sigma}_2\|_F^2 = \left(1 + \frac{\alpha n}{\ell}\right)\sum_{i=r+1}^{\min\{m,n\}} \sigma_i^2(\mathbf{X})$, and the result follows. ∎

This error bound can also be expressed in the form $\varepsilon_{\text{rand}} \le \sqrt{1 + \frac{\alpha n}{\ell}}\varepsilon_{\text{best}}$, where $\varepsilon_{\text{best}}$ is the best possible error in a rank-$r$ approximation.

**5.2. Tensor bound.** To generalize the result from Theorem 5.3 to higher dimensions, we first need a result that expresses the error in a Tucker decomposition in terms of the error in each mode.

LEMMA 5.4 ([29, Theorem 5.1]). *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ and let $\mathbf{P}_j \in \mathbb{R}^{n_j\times n_j}$ for $j = 1,\ldots,d$ be a sequence of orthogonal projectors. Then*

$$\left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \bigtimes_{j=1}^{d} \mathbf{P}_j\right\|^2 = \sum_{j=1}^{d} \left\|\boldsymbol{\mathcal{X}} \bigtimes_{i=1}^{j-1} \mathbf{P}_i \times_j (\mathbf{I} - \mathbf{P}_j)\right\|^2 \le \sum_{j=1}^{d} \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_j \mathbf{P}_j\|^2.$$

Recall the notation $n_j^{\oslash} = \prod_{k\ne j}^{d} n_k$. We present our main error bound result in Theorem 5.5, which we frame as the error bound for Algorithm 4.4. This result can be adapted to also apply to Algorithm 4.3 by following similar techniques to [25, Theorem 3.2]. We include the details for completeness in section SM4.

THEOREM 5.5. *Let $\boldsymbol{\mathcal{T}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U}_1 \times \cdots \times_d \mathbf{U}_d$ be the approximation given by Algorithm 4.4 to $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ with target rank $\mathbf{r} = (r_1,\ldots,r_d)$ and oversampling parameter $p$. Let $\ell_j = r_j + p$ for $j = 1,\ldots,d$. Then, if there exist sequences $\{\alpha_j\}_{j=1}^d$ and $\{\beta_j\}_{j=1}^d$ satisfying* (5.5) *for each $\{\ell_j\}_{j=1}^d$, then the following bound holds with probability at least $1 - \sum_{j=1}^d \frac{1}{\beta_j}$:*

$$\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{T}}\| \le \left(\sum_{j=1}^{d}\left(1 + \frac{\alpha_j n_j^{\oslash}}{\ell_j}\right)\sum_{i=r_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)})\right)^{1/2} + \left(\sum_{j=1}^{d}\sum_{i=r_j+1}^{\ell_j} \sigma_i^2\left(\mathbf{X}_{(j)}\right)\right)^{1/2}.$$

*Proof.* Using (5.1) to bound $\varepsilon_{\text{core}}$, we need only to bound $\varepsilon_{\text{rand}}$. From Lemma 5.4, we have for $\hat{\boldsymbol{\mathcal{T}}} = \hat{\boldsymbol{\mathcal{G}}} \times_1 \hat{\mathbf{U}}_1 \times \cdots \times_d \hat{\mathbf{U}}_d$ computed as the intermediate rank-$\ell$ approximation,

$$\|\boldsymbol{\mathcal{X}}-\hat{\boldsymbol{\mathcal{T}}}\|^2 = \left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \bigtimes_{j=1}^{d} \hat{\mathbf{U}}_j\hat{\mathbf{U}}_j^\top\right\|^2 \leq \sum_{j=1}^{d} \|\boldsymbol{\mathcal{X}}-\boldsymbol{\mathcal{X}}\times_j \hat{\mathbf{U}}_j\hat{\mathbf{U}}_j^\top\|^2 = \sum_{j=1}^{d} \|\mathbf{X}_{(j)}-\hat{\mathbf{U}}_j\hat{\mathbf{U}}_j^\top\mathbf{X}_{(j)}\|_F^2,$$

where the last equality comes from unfolding the tensor along the $j$th mode for $j = 1,\dots d$. We now apply the matrix bound from Theorem 5.3 on each term in this sum, which gives $\|\mathbf{X}_{(j)} - \hat{\mathbf{U}}_j\hat{\mathbf{U}}_j^\top\mathbf{X}_{(j)}\|_F^2 \leq \left(1 + \frac{\alpha_j n_j^{\oslash}}{\ell_j}\right)\sum_{i=r_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)})$, except with failure probability at most $\frac{1}{\beta_j}$. Then the failure probability for the entire sum is the union of all $d$ failure probabilities for each mode, which is bounded above by the sum of those probabilities by the union bound. Thus,

$$\|\boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{T}}}\|^2 \leq \sum_{j=1}^{d} \|\mathbf{X}_{(j)} - \hat{\mathbf{U}}_j\hat{\mathbf{U}}_j^\top\mathbf{X}_{(j)}\|_F^2 \leq \sum_{j=1}^{d}\left(1 + \frac{\alpha_j n_j^{\oslash}}{\ell_j}\right)\sum_{i=r_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)}), \tag{5.7}$$

except with probability at most $\sum_{j=1}^{d}\frac{1}{\beta_j}$. Then, taking square roots gives $\varepsilon_{\text{rand}}$.

Combining (5.7) and (5.1), the total error in approximation is

$$\varepsilon_{\text{total}} \leq \left(\sum_{j=1}^{d}\left(1 + \frac{\alpha_j n_j^{\oslash}}{\ell_j}\right)\sum_{i=r_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)})\right)^{1/2} + \left(\sum_{j=1}^{d}\sum_{i=r_j+1}^{\ell_j} \sigma_i^2\left(\mathbf{X}_{(j)}\right)\right)^{1/2}$$

with probability at least $1-\sum_{j=1}^{d}\frac{1}{\beta_j}$. ∎

We consider this result pessimistic due to the factor of $n_j^{\oslash}$ that comes directly from our analysis of SRHT matrices, but does not appear in our accuracy experiments in section 7.1. We anticipate that this factor could be improved in future analysis. Also note that the result of Theorem 5.5 differs from [11, Theorem 4.2] in two main ways, and our constant in $\varepsilon_{\text{rand}}$ is smaller as we divide by $\ell_j$ for each $j$, and we use a different diagonal matrix that is not a Kronecker product to ensure we have independence where needed.

## 6. Parallel algorithms.

We design and develop parallel implementations for all the randomized algorithms listed in Table 2 using C++/MPI. Our implementations are based on TuckerMPI [3], which uses the STHOSVD algorithm. Similarly to TuckerMPI, our implementations leverage distributed-memory clusters to efficiently compute the Tucker decomposition of large multidimensional datasets. We employ the following data distribution scheme, proposed in [3], in our implementations. To distribute a $d$-way input tensor, the processors are organized in a $d$-way processor grid denoted by $\boldsymbol{\mathcal{P}}$, the dimensions of which are user determined. We use the function ProcID($\boldsymbol{\mathcal{P}}$) within our parallel pseudocode to specify a processor's multidimensional index or rank within the processor grid. Each processor owns a subtensor of the input tensor. For example, for an $8\times6\times2$ tensor and a $2\times3\times1$ processor grid, each processor owns a $4 \times 2 \times 2$ subtensor. All matrices involved in our algorithms are stored redundantly by every processor. Communication occurs via collective operations across fibers and slices of the processor grid. In mode $j$, the processor with index $(p_1,\dots,p_d)$ is part of fiber $\boldsymbol{\mathcal{F}} = \boldsymbol{\mathcal{P}}(p_1,\dots,p_{j-1},:,p_{j+1}\dots,p_d)$ and slice $\boldsymbol{\mathcal{S}} = \boldsymbol{\mathcal{P}}(:,\dots,:,p_j,:,\dots,:)$.

In addition, in the following algorithms, bars over letters denote local data owned by the current processor.

In the following sections we describe two optimizations for computing the sketch tensor via multi-TTMs as well as parallel implementations of the two algorithms from section 4 with the smallest computational cost, Algorithms 4.3 and 4.4. We also compare our implementations with previous work from [12].

**6.1. All-at-once multi-TTM.** The multi-TTM operation is one of the most expensive kernels of our algorithms and appears twice; we compute a multi-TTM both to form the sketch tensor $\boldsymbol{\mathcal{Y}}$ and to form the core tensor $\hat{\boldsymbol{\mathcal{G}}}$ (e.g,. in line 5 of Algorithm 4.4). More specifically, in both places, we multiply the tensor with matrices on all but its $j$th mode. In this section we introduce a new parallel algorithm that can optimize this operation.

A parallel implementation of a single TTM is proposed in [3]. One way to implement the multi-TTM is to simply perform this existing TTM algorithm multiple times, as shown in Algorithm 6.1; we call this approach the in-sequence multi-TTM or IS-mTTM. In this in-sequence approach, a reduce-scatter is performed at the end of each TTM operation, reducing the amount of data each processor owns so that the computation cost in the next TTM is also reduced. Generally, this approach performs additional communications to obtain lower computational cost. However, depending on the size of the local tensor, the reduction in computational cost may not justify the increased communication cost.

Our algorithm is shown in Algorithm 6.2, which we call the all-at-once multi-TTM or AAO-mTTM. In this approach, we avoid communication until all matrices have been multiplied with the local tensor. This strategy reduces communication by increasing the cost of storage and computation.

**Algorithm 6.1** IS-mTTM.

```
1: function $\hat{\boldsymbol{\mathcal{Y}}} =$ IS-mTTM($\bar{\boldsymbol{\mathcal{X}}}, j, \{\mathbf{M}_j\}, \boldsymbol{\mathcal{P}}$)
2:     $(p_1, \ldots, p_d) = \text{procID}(\boldsymbol{\mathcal{P}})$
3:     $\bar{\boldsymbol{\mathcal{Y}}} = \bar{\boldsymbol{\mathcal{X}}}$
4:     for $i = 1 : d$ and $i \neq j$ do
5:         $\boldsymbol{\mathcal{F}} = \boldsymbol{\mathcal{P}}(p_1, \ldots, p_{i-1}, :, p_{i+1}, \ldots, p_d)$
6:         $\bar{\mathbf{T}}_{(i)} = \bar{\mathbf{M}}_i \bar{\mathbf{Y}}_{(i)}$
7:         $\bar{\mathbf{Y}}_{(i)}$=Reduce-Scatter($\bar{\mathbf{T}}_{(i)}, \boldsymbol{\mathcal{F}}$)
8:     end for
9: end function
```

**Algorithm 6.2** AAO-mTTM.

```
1: function $\hat{\boldsymbol{\mathcal{Y}}} =$ AAO-mTTM($\bar{\boldsymbol{\mathcal{X}}}, j, \{\mathbf{M}_j\}, \boldsymbol{\mathcal{P}}$)
2:     $(p_1, \ldots, p_d) = \text{procID}(\boldsymbol{\mathcal{P}})$
3:     $\bar{\boldsymbol{\mathcal{T}}} = \bar{\boldsymbol{\mathcal{X}}}$
4:     for $i = 1 : d$ and $i \neq j$ do
5:         $\bar{\mathbf{T}}_{(i)} = \bar{\mathbf{M}}_i \bar{\mathbf{T}}_{(i)}$
6:     end for
7:     $\boldsymbol{\mathcal{S}} = \boldsymbol{\mathcal{P}}(:, \ldots, :, p_j, :, \ldots, :)$
8:     $\bar{\mathbf{Y}}_{(j)}$ = Reduce-Scatter($\bar{\mathbf{T}}_{(j)}, \boldsymbol{\mathcal{S}}$)
9: end function
```

The most significant difference between Algorithms 6.1 and 6.2 is that at the end of any iteration $i$ of the in-sequence approach, we form the intermediate result $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_1 \mathbf{M}_1 \times \cdots \times_i \mathbf{M}_i$; in the all-at-once approach, each processor stores a contribution to $\boldsymbol{\mathcal{Y}}$ until all iterations are completed and the all-reduce at the end of the algorithm forms the final result.

**6.1.1. Cost analysis.** To simplify the notation, we assume that the input tensor is a $d$-way cubic tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n\times\cdots\times n}$, that the processor tensor is size $q$ in each mode ($q^d = P$ processors in total), and that the input matrices $\{\mathbf{M}_i\}_{i=1}^d$ are of the same size $s \times n$ with $s < n$. With this notation, we analyze the per-processor computation and communication costs of performing the multi-TTM $\boldsymbol{\mathcal{X}} \times_1 \mathbf{M}_1 \times \cdots \times_{j-1} \mathbf{M}_{j-1} \times_{j+1} \mathbf{M}_{j+1} \times \cdots \times_d \mathbf{M}_d$ using Algorithm 6.2 and compare it with that of using Algorithm 6.1.

The computational cost of Algorithm 6.1 can be written as

$$C_{\text{in-sequence}} = 2\left(\frac{sn^d}{q^d} + \frac{s^2n^{d-1}}{q^d} + \cdots + \frac{s^dn}{q^d}\right) = 2\left(\sum_{i=1}^{d} \frac{s^in^{d-i+1}}{q^d}\right), \tag{6.1}$$

and the computational cost of Algorithm 6.2 can be written as

$$C_{\text{all-at-once}} = 2\left(\frac{sn^d}{q^d} + \frac{s^2n^{d-1}}{q^{d-1}} + \cdots + \frac{s^dn}{q}\right) = 2\left(\sum_{i=1}^{d} \frac{s^in^{d-i+1}}{q^{d-i+1}}\right). \tag{6.2}$$

The $i$th terms of the summations in both (6.2) and (6.1) represent the cost of multiplying the local factor matrix $\bar{\mathbf{M}}_i$ of size $s \times \frac{n}{q}$ and the $i$th mode unfolding of tensor $\bar{\boldsymbol{\mathcal{Y}}} = \bar{\boldsymbol{\mathcal{X}}} \times_1 \bar{\mathbf{M}}_1 \times \cdots \times_{i-1} \bar{\mathbf{M}}_{i-1}$. In Algorithm 6.1, due to the reduce-scatter at each iteration, $\bar{\mathbf{Y}}_{(i)}$ is of size $\frac{n}{q} \times \frac{s^{i-1}n^{d-i}}{q^{d-1}}$. In Algorithm 6.2, $\bar{\mathbf{M}}_i$ is of the same size. However, since the reduction is delayed until the last mode, $\bar{\mathbf{Y}}_{(i)}$ is of size $\frac{n}{q} \times \frac{s^{i-1}n^{d-i}}{q^{d-i}}$. Comparing (6.1) and (6.2), it is easy to see that the computational cost of an AAO-mTTM is always higher than that of an IS-mTTM because each of the summands in (6.2) is at least as large as the corresponding term in (6.1). This increase can be small, however, in certain cases. Note that the two series of summands are geometric and have the same leading term. For (6.1), the ratio of the series is $\frac{s}{n}$ while the ratio of the series in (6.2) is $\frac{sq}{n}$. As the subranks become smaller compared to the tensor dimensions (i.e., $s \ll n$), the sums of the two series get closer to their first terms and thus their difference becomes smaller.

Using the same notations and the $\alpha$-$\beta$-$\gamma$ model [8], we can express the communication cost of Algorithms 6.1 and 6.2. In Algorithm 6.2, there is only one communication step at the end where all $P$ processors communicate their local tensor $\bar{\boldsymbol{\mathcal{T}}}$. $\bar{\boldsymbol{\mathcal{T}}}$ is a $d$-way tensor with size $s$ for all of its modes except for the $j$th mode which has size $\frac{n}{q}$. Therefore, the communication cost for each processor is $\alpha\mathcal{O}(\log P) + \beta\mathcal{O}\left(\frac{s^{d-1}n}{q}\right)$. The communication cost of Algorithm 6.1 is more complicated because there are $d-1$ communication steps and each one involves $\bar{\mathbf{T}}_{(i)}$, which is changing in size. At the $i$th iteration of the for loop, $\bar{\mathbf{T}}_{(i)}$ has size $s \times \frac{s^{i-1}n^{d-i}}{q^{d-1}}$. Therefore, the communication cost can be written as $\alpha\mathcal{O}(d\log q) + \beta\mathcal{O}\left(\sum_{i=1}^{d-1} \frac{s^in^{d-i}}{q^{d-1}}\right) = \alpha\mathcal{O}(\log P) + \beta\mathcal{O}\left(\frac{sn^{d-1}}{q^{d-1}}\right)$. We obtain the right-hand side of the equation using the assumption that $s \ll n$ so that the summation is approximated by its first summand. When $sq < n$, the communication cost of AAO-mTTM is smaller. As the ratio $\frac{n}{sq}$ increases, the benefits of using AAO-mTTM become more substantial. A summary of these costs is presented in Table 3.

TABLE 3

*Comparison of computation and communication cost per processor. More details can be found in section* SM5.

| **Algorithm** | **Form {U} Comp cost** | **Form {U} Comm cost** | **Form $\mathcal{G}$ Comp cost** | **Form $\mathcal{G}$ Comm cost** |
|---|---|---|---|---|
| STHOSVD [3] | $\frac{n^{d+1}}{P}$ | $\alpha\mathcal{O}(dP^{1/d}) + \beta\mathcal{O}(\frac{n^d}{P})$ | $2\frac{rn^d}{P}$ | $\alpha\mathcal{O}(\log P) + \beta\mathcal{O}(\frac{rn^{d-1}}{P^{1-1/d}})$ |
| [12] | $\frac{n^{d+1}}{P}$ | $\alpha\mathcal{O}(dP) + \beta\mathcal{O}(\frac{n^d}{P})$ | $2\frac{rn^d}{P}$ | - |
| Algorithm 6.4 | $2\frac{r^{1/(d-1)}n^d}{P}$ | $\alpha\mathcal{O}(d\log P) + \beta\mathcal{O}(\frac{drn}{P^{1/d}})$ | $2\frac{rn^d}{P}$ | $\alpha\mathcal{O}(\log P) + \beta\mathcal{O}(\frac{rn^{d-1}}{P^{1-1/d}})$ |
| Algorithm 6.5 | $4\frac{r^{1/(d-1)}n^d}{P}$ | $\alpha\mathcal{O}(d\log P) + \beta\mathcal{O}(\frac{drn}{P^{1/d}})$ | $2\frac{rn^d}{P}$ | $\alpha\mathcal{O}(\log P) + \beta\mathcal{O}(\frac{rn^{d-1}}{P^{1-1/d}})$ |

**Algorithm 6.3** All modes multi-TTM using the dimension tree optimization and AAO-mTTM.

```
 1: function $\{\boldsymbol{\mathcal{Y}}_j\}$ =ALL-MODES-MULTI-TTM($\bar{\boldsymbol{\mathcal{X}}}$, $\{\bar{\mathbf{\Phi}}_j\}$, $\mathbf{m}$, $\mathbf{n}$, $\boldsymbol{\mathcal{P}}$)
 2:   $(p_1, \dots, p_d) = \text{procID}(\boldsymbol{\mathcal{P}})$
 3:   $\bar{\boldsymbol{\mathcal{Y}}} = \bar{\boldsymbol{\mathcal{X}}}$
 4:   for $i \in \mathbf{n}$ do
 5:     $\bar{\mathbf{Y}}_{(i)} = \bar{\mathbf{\Phi}}_i \bar{\mathbf{Y}}_{(i)}$
 6:   end for
 7:   if $\mathbf{m}$ only contains 1 integer, $j$ then
 8:     $\boldsymbol{\mathcal{S}} = \boldsymbol{\mathcal{P}}(:, \dots, p_j, \dots, :)$
 9:     $\bar{\boldsymbol{\mathcal{Y}}}_j$ = REDUCE-SCATTER($\bar{\boldsymbol{\mathcal{Y}}}$, $\boldsymbol{\mathcal{S}}$)
10:   else
11:     split $\mathbf{m}$ in equal half $\mathbf{m}_1$ and $\mathbf{m}_2$
12:     ALL-MODES-MULTI-TTM($\bar{\boldsymbol{\mathcal{Y}}}$, $\{\bar{\mathbf{\Phi}}_j\}$, $\mathbf{m}_1$, $\mathbf{m}_2$, $\boldsymbol{\mathcal{P}}$)
13:     ALL-MODES-MULTI-TTM($\bar{\boldsymbol{\mathcal{Y}}}$, $\{\bar{\mathbf{\Phi}}_j\}$, $\mathbf{m}_2$, $\mathbf{m}_1$, $\boldsymbol{\mathcal{P}}$)
14:   end if
15: end function
```

**6.2. Dimension tree optimization.** In section 4.4, we discuss how dimension trees can be used to make the randomized sketches less expensive for Algorithm 4.4. In Algorithm 6.3, we present an implementation using AAO-mTTM. Here, $\mathbf{m}$ and $\mathbf{n}$ are sets of integers in the range $[1, d]$. This algorithm returns $\boldsymbol{\mathcal{Y}}_j$, the sketch of $\mathbf{X}_{(j)}$ in tensor format, for all integers $j \in [1, d]$. Since dimension trees are a tool for reusing only local intermediate results, Algorithm 6.3 can also be modified to use IS-mTTM.

**6.3. Principal algorithms.** We now combine the discussed multi-TTM approaches and the dimension tree optimization within parallel implementations of our two best algorithms, Algorithms 4.3 and 4.4. The pseudocode is provided in Algorithms 6.4 and 6.5, respectively. Note that for both of these algorithms, when the size of core $\hat{\boldsymbol{\mathcal{G}}}$ is large, it is better to perform the IS-mTTM as described in Algorithm 6.1 to produce a distributed $\hat{\boldsymbol{\mathcal{G}}}$ so that the parallel STHOSVD can be used to reduce cost. When $\hat{\boldsymbol{\mathcal{G}}}$ is very small, performing STHOSVD in parallel can be counterproductive as the communication cost will dominate; it is better in this case to perform an all-gather among all processors after the IS-mTTM, producing $\hat{\boldsymbol{\mathcal{G}}}$ redundantly on every processor and then use the sequential STHOSVD.

**Algorithm 6.4** Parallel algorithm for Algorithm 4.3.

```
1: function rSTHKron($\boldsymbol{\mathcal{X}}$, $\mathbf{r}$, $p$, $\boldsymbol{\mathcal{P}}$) ▷ $\boldsymbol{\mathcal{X}}$ is distributed, the local part of $\boldsymbol{\mathcal{X}}$ is denoted as $\bar{\boldsymbol{\mathcal{X}}}$
2:   $\hat{\boldsymbol{\mathcal{G}}} = \boldsymbol{\mathcal{X}}$
3:   Redundantly compute matrix of subranks $\mathbf{S}$
4:   for $i = 1{:}d$ do
5:     for $j = 1{:}d$ and $j \neq i$ do
6:       Redundantly draw $d-1$ random matrices $\mathbf{\Phi}_{i,j} \in \mathbb{R}^{n_j \times s_{i,j}}$
7:     end for
8:     $\hat{\boldsymbol{\mathcal{Y}}} \leftarrow$ AAO-mTTM($\hat{\boldsymbol{\mathcal{G}}}$, $i$, $\{\mathbf{\Phi}_{i,1} \dots \mathbf{\Phi}_{i,d}\}$, $\boldsymbol{\mathcal{P}}$)   ▷ Can also use IS-mTTM
9:     $\boldsymbol{\mathcal{Y}} =$ All-Gather($\hat{\boldsymbol{\mathcal{Y}}}, \boldsymbol{\mathcal{P}}$)
10:    $\hat{\mathbf{U}}_i =$ QR($\mathbf{Y}_{(i)}$)   ▷ QR is serial as every processor owns global $\boldsymbol{\mathcal{Y}}$
11:    $\hat{\boldsymbol{\mathcal{G}}} \leftarrow$ TTM($\hat{\boldsymbol{\mathcal{G}}}$, $\hat{\mathbf{U}}_i^{\mathsf{T}}$, $i$)   ▷ For a single TTM we use the implementation proposed in [3]
12:  end for
13:  $[\boldsymbol{\mathcal{G}}, \{\mathbf{V}_i\}] =$ STHOSVD($\hat{\boldsymbol{\mathcal{G}}}, \mathbf{r}$)
14:  for $i = 1, \dots, d$ do
15:    $\mathbf{U}_i = \hat{\mathbf{U}}_i \mathbf{V}_i$   ▷ Computed with local matrix multiplication
16:  end for
17: end function
```

**Algorithm 6.5** Parallel algorithm for Algorithm 4.4 with AAO-mTTM and dimension trees.

```
1: function rHKron-re($\boldsymbol{\mathcal{X}}$, $\mathbf{r}$, $p$, $\boldsymbol{\mathcal{P}}$)
2:   Compute subranks $\mathbf{s}$
3:   for $i = 1{:}d$ do
4:     Redundantly draw $d-1$ random matrices $\mathbf{\Phi}_i \in \mathbb{R}^{s_i \times n_i}$
5:   end for
6:   $\{\boldsymbol{\mathcal{Y}}^{(1)}, \dots, \boldsymbol{\mathcal{Y}}^{(d)}\} =$ All-Modes-Multi-TTM($\bar{\boldsymbol{\mathcal{X}}}, \{\mathbf{\Phi}_1, \dots, \mathbf{\Phi}_d\}, \{1, \dots, d\}, \emptyset, \boldsymbol{\mathcal{P}}$)
7:   for $i = 1{:}d$ do
8:     $\hat{\mathbf{U}}_i =$QR$\mathbf{Y}_{(i)}^{(i)}$   ▷ Serial QR decomposition of the mode $i$ unfolding of $\boldsymbol{\mathcal{Y}}^{(i)}$
9:   end for
10:  $\hat{\boldsymbol{\mathcal{G}}} =$ IS-mTTM($\bar{\boldsymbol{\mathcal{X}}}, \emptyset, \{\hat{\mathbf{U}}_1^{\mathsf{T}}, \dots, \hat{\mathbf{U}}_d^{\mathsf{T}}\}, \boldsymbol{\mathcal{P}}$)
11:  $[\boldsymbol{\mathcal{G}}, \{\mathbf{V}_j\}] =$ STHOSVD($\hat{\boldsymbol{\mathcal{G}}}, \mathbf{r}$)
12:  for $i = 1, \dots, d$ do
13:    $\mathbf{U}_i = \hat{\mathbf{U}}_i \mathbf{V}_i$   ▷ Computed with local matrix multiplication
14:  end for
15: end function
```

**6.4. Comparison to previous work.** We compare our parallel algorithms with previous approaches, namely, the parallel STHOSVD algorithm in [3] and the approach from [12]. Assume that the input tensor is a $d$-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n \times \cdots \times n}$ with rank $(r, r, \dots, r)$ and that each mode of the $d$-way processor tensor has size $q$. We also assume that $s < r < l \ll n$, where $s$ is the subrank for each mode and $\ell = r + p$ with $p$ the oversampling parameter. Here $s = \ell^{1/(d-1)} \approx r^{1/(d-1)}$. In [12], Choi, Liu, and Chakaravarthy proposed a data distribution scheme and a tensor matricization strategy that reduces the communication costs of the Gram and TTM kernels. More specifically, in this new method, before the Gram computation, communication is performed every other mode to redistribute the tensor unfolding from a 2-dimensional distribution to a block-column 1-dimensional distribution, which can avoid the communication cost of the later TTM operations required to form the core tensor. This method is shown to achieve speedup over the parallel implementation proposed in [3]. However, these optimizations are not suitable for our randomized algorithms for the following reason. Instead of computing the Gram matrix of each mode unfolding, we compute each mode sketch with a multi-TTM. Since each sketch is much smaller at the end of the multi-TTM, it is beneficial to delay communicating the sketches as much as possible before all multi-TTM's are completed. However, the redistribution proposed by [12] requires every processor to communicate the entire uncompressed local tensor before the computation. While [12] also uses randomization to reduce computation, their focus is on reducing the cost of computing the eigenvectors of the Gram matrix, which is achieved by using a modified randomized SVD to replace the eigendecomposition.

To illustrate the benefit of using Kronecker-structured random matrices, we also compare Algorithms 6.4 and 6.5 with the parallel version of Algorithm 4.2, which uses dense Gaussian random matrices. The parallelized Algorithm 4.2 is very similar to Algorithm 6.4 with the only difference being that we use the parallel multi-TTM to apply the Kronecker-structured random matrices to the input tensor while in parallelized Algorithm 4.2 we use a single parallel TTM operation to apply each of the dense Gaussian random matrices.

## 7. Experimental results.

We now demonstrate the numerical benefits of our algorithms by considering the accuracy of our sequential algorithms in section 7.1, the performance of the multi-TTM and dimension tree optimizations in subsections 7.2 and 7.3, respectively, and the performance of the parallel implementations of our randomized algorithms on synthetic data in section 7.4 and on two real datasets in subsections 7.5 and 7.6.

*Computing platform.* The results shown in section 7.1 are generated by running MATLAB implementations of the sequential algorithms on a single node server. The experiments on parallel performance of our C++/MPI implementation (sections 7.2 to 7.6) are run on the Andes cluster at Oak Ridge Leadership Computing Facility. The system consists of 704 compute nodes with 2 AMD EPYC 7302 16-core CPUs and 256 GB of RAM. We directly call OpenBLAS and the Netlib implementation of LAPACK for local linear algebra kernels, which are the only available libraries on Andes.

### 7.1. Accuracy results.

We present an experiment on a synthetic tensor that demonstrates the accuracy of our algorithms compared to existing deterministic and randomized algorithms. In this experiment, we use both SRHT and Gaussian random matrices to compare numerical accuracy to the theoretical results we derived in section 5.

We construct a synthetic 3-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{500\times 500\times 500}$ by forming a (super-) diagonal tensor with geometrically decreasing entries and multiplying that tensor by a random orthogonal matrix along each mode. We set the largest entry of the original tensor to be 1 and choose the rate at which the core entries decrease to be 0.4 so that the 40th entry is approximately machine precision. In our experiment, we compress this tensor to rank $(10, 10, 10)$ using an oversampling parameter of $p = 5$. We show

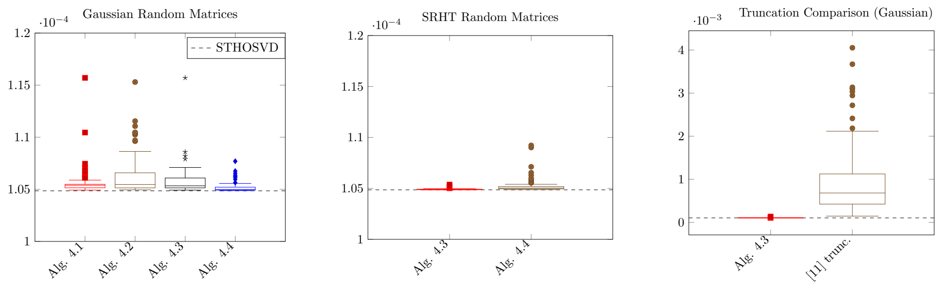


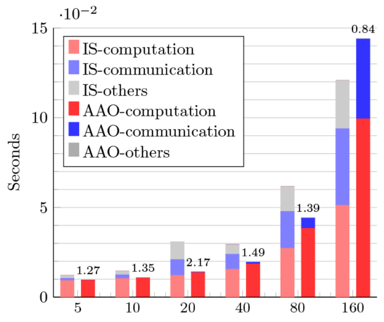
FIG. 3. *Boxplots of relative errors for our randomized algorithms using Gaussian random matrices (left) and SRHT random matrices (center) compared to the relative error for STHOSVD for the synthetic tensor with geometrically decreasing values. We also compare the truncation methods we use to those used in* [11] *(right). Note: color appears only in the online article.*

boxplots of the relative error over 100 trials in Figure 3, comparing results from all our algorithms (Algorithms 4.1 to 4.4) using Gaussian random matrices as well as our principal algorithms (Algorithms 4.3 and 4.4) using SRHT random matrices. We also compare our truncation strategy to the strategy in [11], and compare the relative error from all randomized algorithms to the relative error obtained from deterministic STHOSVD, Algorithm 2.2.

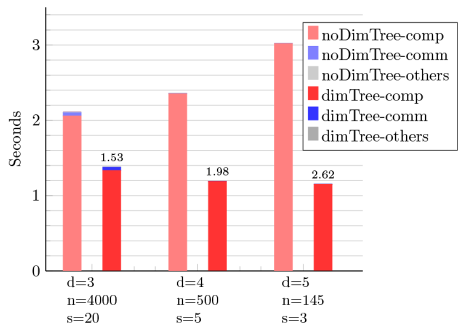
In Figure 3, we see that the relative errors for all our randomized algorithms deviate from the deterministic relative error by at most 10% for the given rank (the medians are within 1%). Also, the relative errors for each trial are very close together with each algorithm, as even the outliers are within the same order of magnitude, with a standard deviation of $1.9 \times 10^{-5}$. Regardless of random matrix distribution or whether we reuse $\mathbf{\Phi}_j$ matrices or generate new matrices for each mode, we do not lose significant accuracy compared to either the deterministic or standard randomized approaches. We can also see that the truncation strategy we employ in Algorithms 4.1 to 4.4 is much more consistently accurate than the strategy in [11].

We show an additional accuracy experiment in section SM4.1. Overall, we see comparable relative error for Algorithms 4.3 and 4.4 to both deterministic STHOSVD and existing randomized approaches. While our theoretical results (Theorem 5.5) apply only to SRHT matrices, these experimental results suggest that Gaussian matrices perform comparably in terms of accuracy. We note also that efficient application of Hadamard matrices may require padding to powers-of-two dimensions, which can partially offset the computational cost savings from using the Hadamard structure. Our parallel performance experiments use Gaussian matrices exclusively.

**7.2. In-sequence TTM versus all-at-once TTM.** To compare the performance of the two TTM approaches we discuss in section 6.1, we conduct an experiment performing an IS-mTTM and an AAO-mTTM of a 3-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{800\times800\times800}$ with matrices $\mathbf{U}, \mathbf{V}, \mathbf{W} \in \mathbb{R}^{s\times 800}$ with varying $s$. For this experiment, we use 64 cores (2 nodes) arranged in a $4 \times 4 \times 4$ processor grid. The results of this experiment are shown in Figure 4(a), where we can see that when the matrices have relatively few rows (when $sq \ll n$), the AAO-mTTM is much more communication-efficient. We observe speedups ranging from 27% to over 2×. This fits our prediction in section 6.1. However, when $\frac{n}{sq}$ is very large (i.e., the $s = 5$ case), both algorithms are cheap and communication is not as dominant. When $\frac{n}{sq}$ is small, AAO-mTTM has increased computational cost, and also loses its advantage in communication cost. In this case, IS-mTTM is preferred to AAO-mTTM. For this reason, we think the all-at-once



(a) Comparing the performance of all-at-once multi-TTM and in-sequence multi-TTM by multiplying a $800 \times 800 \times 800$ synthetic tensor with an $s \times 800$ matrix on all modes ($s \in [5, 160]$). The processor grid used is $4 \times 4 \times 4$. The labels on top of bars indicate the overall speedup achieved by all-at-once multi-TTM compared to in-sequence multi-TTM.



(b) Performance gain by using the dimension tree optimization for Alg. 4.4. We compute the sketch of a cubic synthetic $n$-way tensor where each mode has the same size $n$, resulting in a cubic $d$-way core tensor with each mode having the same size $s$. The labels on top of the bars show speedup gained by using dimension trees.

FIG. 4. *Performance benefits of our multi-TTM and dimension tree optimizations. Note: color appears only in the online article.*

optimization is more suitable for the sketching phase where the random matrices tend to have very few rows.

Note that the gray bars in Figure 4(a) represent overhead costs. For IS-mTTM, this overhead mainly comes from reorganizing the data in memory before and after communication (MPI collectives). Since AAO-mTTM avoids those communication steps, it also avoids those reorganizing costs. For higher dimensions, the benefits of AAO-mTTM still depend on $\frac{n}{sq}$ being large. If this ratio is fixed, AAO-mTTM will continue to outperform IS-mTTM.

**7.3. Dimension tree optimization.** To demonstrate the benefits of using dimension trees, we run Algorithm 4.4 with and without dimension trees on synthetic tensors with an increasing number of modes such that the total size of the input tensor and its rank are kept close to constant. We benchmark the time it takes for both methods to apply the random matrices $\{\Phi\}$ to the input tensor and present the results in Figure 4(b). This computation corresponds to line 5 of Algorithm 4.4. Since the communication and overhead costs are low, we see that the practical speedup from using dimension trees aligns closely with the theoretical prediction, with a computational reduction of $d/2$ as described in section 4.4.

**7.4. Strong scaling on synthetic data.** In this experiment we benchmark four variations of our randomized algorithms and STHOSVD as a baseline, scaling from 2 nodes (64 cores) to 32 nodes (1024 cores) on a fixed problem size. The input tensor is a $410 \times 410 \times 410 \times 410$ single-precision synthetic tensor ($\approx 113$ GB) constructed from multiplying a $20 \times 20 \times 20 \times 20$ randomly generated core with four $410 \times 20$ random matrices. No noise is added to this synthetic tensor so it is exactly low rank. This size is close to the largest tensor we can fit in the memory of 2 nodes, which makes the timing results more consistent and less influenced by noise in the system. Any order of modes used to compute the multi-TTM will not affect the performance as the tensor size and rank are the same across modes. All six algorithms are

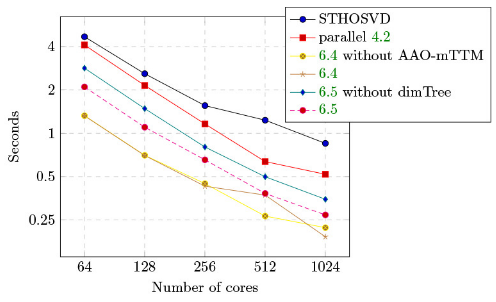


Fig. 5. *Strong scaling of different algorithm variants. Note: color appears only in the online article.*

given target ranks (20, 20, 20, 20) and the randomized ones use oversampling parameter $p = 3$.

The results are presented in Figure 5, where we see that all six algorithms presented scale well to 1024 cores. All randomized algorithms outperform the deterministic STHOSVD, and the randomized algorithms that use Kronecker-structured random matrices outperform the parallel version of Algorithm 4.2. A noticeable speedup of 3–4× is achieved by using Algorithm 6.4 compared to STHOSVD. The second-best algorithm is Algorithm 6.5 with the dimension tree optimization achieving 2–3× speedup. Note that the use of AAO-mTTM provided little improvement over IS-mTTM in this case, because the compression ratio is very high so the multi-TTMs are not bottlenecks. We can also see that the dimension tree optimization does provide noticeable and consistent speedup for Algorithm 6.5.

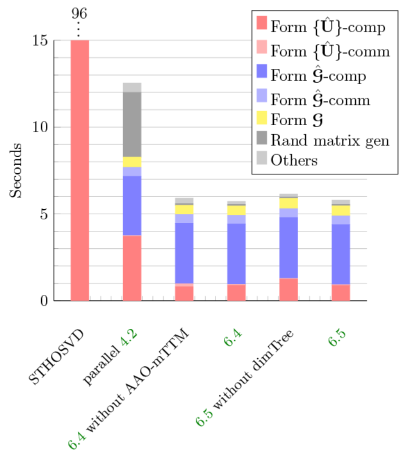
**7.5. Miranda dataset.** The Miranda dataset [6, 32] contains 3-dimensional simulation data of the density ratios of a nonreacting flow of viscous/diffusive fluids. This dataset is $3072 \times 3072 \times 3072$. Its values are in single precision and range between 1 and 3. We show visualizations for this dataset in section SM6.1.

In this experiment, we compare five variations of our randomized algorithms as well as the deterministic STHOSVD algorithm using the Miranda dataset. We use 4 nodes (128 cores) organized as a $1 \times 8 \times 16$ tensor. The target rank we choose for this run is (502, 504, 361), which corresponds to a $10^{-2}$ reconstruction error (estimated using precomputed singular values of the unfolding of the data tensor for each mode). The subrank matrix we used for Algorithm 6.4 is

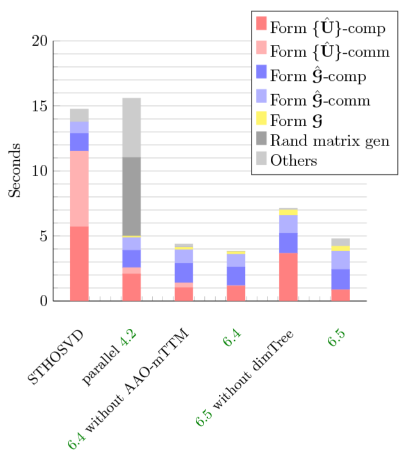
$$\begin{bmatrix} 1 & 39 & 13 \\ 30 & 1 & 17 \\ 6 & 61 & 1 \end{bmatrix}.$$

For Algorithm 6.5, the subrank vector we used is (20, 20, 26). The relative error achieved by STHOSVD is 0.0094, while the relative errors of the randomized algorithms Algorithms 4.2, 6.4, and 6.5 are 0.0234, 0.0194, and 0.0189, respectively, which are all within 2.5× the deterministic error.

The performance results are recorded in Figure 6(a), and we visualize reconstructed tensors from Algorithms 2.2, 4.2, and 6.5 in section SM6.1. First, we note that STHOSVD is particularly slow when compared to the randomized algorithms, in this case partly because the Miranda dataset has fewer modes and each mode is large. As a result, the Gram matrices are large and the eigendecompositions of those Gram matrices are expensive. Moreover, the eigendecompositions are carried out redundantly on every processor due to the TuckerMPI assumption of small individual mode



(a) Comparing the performance of all algorithms on Miranda dataset .



(b) Comparing the performance of all algorithms on SP dataset.

FIG. 6. *Performance breakdown of all algorithms on our two real datasets. Note: color appears only in the online article.*

dimensions. (More details on the parallel implementation of STHOSVD can be found in [3]). The randomized algorithms, on the other hand, can avoid this expensive step completely, and we see up to a 16× speedup, comparing Algorithm 6.4 to STHOSVD. In the next section, we compare the performance of all the algorithms again with a higher-order tensor, where each mode is relatively small. In that case, the Gram matrices are smaller and the eigendecompositions are cheaper, so the deterministic algorithm appears more competitive.

Also note that random number generation (forming the random matrices $\{\boldsymbol{\Omega}\}$) in Algorithm 4.2 takes up a large percentage of the total time. These results demonstrate the benefits of generating fewer random numbers by using Kronecker-structured random matrices. Now, comparing the computation cost (red bar) of Algorithm 4.2 with that of the following algorithms to the right, we can see that using Kronecker-structured random matrices further reduces the computation cost of forming the factor matrices as we have predicted. Among the algorithms that use Kronecker-structured random matrices, Algorithms 6.4 and 6.5 achieve the best performance. Comparing multi-TTM methods, we see that using either IS-mTTM or AAO-mTTM in Algorithm 6.4 results in very similar performance. Although AAO-mTTM achieved a 3× speedup in the communication cost of forming the factor matrices (pink bar) over IS-mTTM, the absolute speedup is not significant because the communication cost using IS-mTTM is already very small. This is mostly due to the subranks being very small compared to the size of the input tensor. Finally, we note that with our optimizations, applying the factor matrices to truncate the input tensor (the blue and light blue bars) now becomes the bottleneck of the algorithm.

**7.6. Stats-Planar dataset.** The Stats-Planar dataset is generated from the simulation of a statistically stationary planar (SP) methane-air flame [23]. The data have dimensions $500 \times 500 \times 500 \times 11 \times 400$ with the first 3 modes representing a 3-dimensional spatial grid, the 4th mode representing 11 variables, and the 5th mode representing time steps. These data have been used in previous studies such as [3] to demonstrate the effectiveness of Tucker decomposition algorithms. In this work, we use the single-precision-max-normalized version of this dataset. We visualize the 250th slice for each of the first three modes of the SP tensor in Figure SM4.

Similarly to the experiment on the Miranda dataset, we compare five variations of the randomized algorithm and the deterministic STHOSVD algorithm using 1024 cores (32 nodes). The target rank we use is (31, 38, 35, 6, 11). The subrank vector we used for Algorithm 6.5 is $\begin{bmatrix} 2 & 2 & 2 & 3 & 4 \end{bmatrix}$ and the subrank matrix used for Algorithm 6.4 is

$$\begin{bmatrix} 0 & 3 & 3 & 2 & 2 \\ 3 & 0 & 4 & 2 & 2 \\ 2 & 5 & 0 & 2 & 2 \\ 2 & 2 & 2 & 0 & 2 \\ 2 & 2 & 2 & 2 & 0 \end{bmatrix}.$$

The relative error achieved by STHOSVD on this dataset is 0.0028, while the relative errors of the randomized algorithms Algorithms 4.2, 6.4, and 6.5 are 0.0050, 0.0079, and 0.0079, respectively, which are all within $3\times$ of the deterministic error.

The performance results are shown in Figure 6(b). The speedup of the randomized algorithms appears less dramatic compared to the results from the Miranda dataset, which is mainly due to the difference in dimensions of these two datasets. Recall that compared to the Miranda dataset, the SP tensor is of higher order but has a smaller size in each mode. As a result, the sequential eigendecomposition in the STHOSVD algorithm is no longer as expensive. We also see that Algorithms 6.4 and 6.5 are still the best-performing algorithms. Algorithm 4.2 suffers from slow random number generation, similarly to the experiments on the Miranda dataset. Finally, forming the factor matrices is no longer a bottleneck for tensors with more cubical dimensions. In this case, accelerating the computations applying these factor matrices to derive the core tensor will become a more important issue.

## 8. Conclusions and future work.

We develop new randomized algorithms using a Kronecker product of random matrices that significantly decrease the computational cost in computing a Tucker decomposition. By accelerating the sketching step using Kronecker products, we remove the SVD as the dominant computational bottleneck. Our algorithms also reduce the number of random entries generated, which, as shown in our experimental results, could result in significant savings in the runtime compared to standard randomized algorithms. As the SVD step is no longer the dominant computation, future directions include accelerating the TTM computation, the other dominant portion of computing a Tucker decomposition, perhaps through a one-pass approach similar to [27]. We develop probabilistic error bounds for our algorithms using SRHT matrices as they generalize to Kronecker products better than Gaussian matrices. The empirical results comparing Gaussian and SRHT matrices show that the error incurred from using SRHT matrices is not any worse than using Gaussian matrices. Our theoretical bounds are pessimistic in comparison, so there is room for improvement in the analysis, another potential future direction. We implement our new randomized algorithms in parallel, developing a new algorithm that parallelizes the most expensive SVD component. Previous approaches such as [12] parallelize other components, leaving the most expensive part to be computed locally. Overall, we show in this work that choosing a random matrix that fits the structure of our problem is beneficial. The dense Gaussian matrix typically used in RandSVD in particular is not required, and performance is greatly improved by exploiting appropriate structure.

## Appendix A. Proof of Lemma 5.1.

*Proof.* Let $\mathbf{W} = \mathbf{V}_1^\top \in \mathbb{R}^{r\times n}$ for notational simplicity, and recall that $\mathbf{\Omega} = \mathbf{D}(\mathbf{H}_1 \otimes \cdots \otimes \mathbf{H}_q)$ is the Kronecker product of independent SRHT matrices, where $\mathbf{D} \in \mathbb{R}^{n\times n}$ and $\mathbf{H}_j \in \mathbb{R}^{n_j \times s_j}$ for every $j$, given $n = \prod_{j=1}^q n_j$ and $\ell = \prod_{j=1}^q s_j$. Define $\mathbf{G} = (\boldsymbol{W\Omega})(\boldsymbol{W\Omega})^\top \in \mathbb{R}^{r\times r}$. Note that $\boldsymbol{W\Omega}$ is equivalent to $\mathbf{\Omega}_1 \in \mathbb{R}^{r\times \ell}$.

Our approach will focus on the elementwise representation of $\mathbf{G}$ and will be composed of 3 main steps: first, we will express the elements of $\mathbf{G}$ in terms of two summands $\mathbf{M}$ and $\mathbf{N}$ that can be bounded more easily; second, we will obtain a deterministic bound for $\|\mathbf{M}\|_2$ and then bound $\mathbb{E}[N_{ij}^2]$, which is the bulk of the proof; and third, we use our result from the previous step in conjunction with Markov's inequality to obtain a concentration inequality for $\|\mathbf{N}\|_2$. Combining all these pieces will then give us the desired bound. Note that our three main steps follow the approach of [31], which analyzes the case where $\mathbf{\Omega}$ is a single SRHT matrix.

Define $\mathbf{H} = \mathbf{H}_1 \otimes \mathbf{H}_2 \otimes \cdots \otimes \mathbf{H}_q$, letting the Kronecker product of subsampled Hadamard matrices be $\mathbf{H}$ for ease of notation. Then we have another way to express $\mathbf{G}$ as

$$\mathbf{G} = (\boldsymbol{W\Omega})(\boldsymbol{W\Omega})^\top = \mathbf{WDHH}^\top\mathbf{DW}^\top = \mathbf{WDFDW}^\top, \tag{A.1}$$

letting $\mathbf{F} = \mathbf{HH}^\top$. There are some important properties of $\mathbf{D}$ and $\mathbf{F}$ we will need, which we now explore. The diagonal matrix $\mathbf{D}$ has i.i.d. entries drawn from the Rademacher distribution so that $\mathbb{E}D_a = 0$ and $D_a^2 = 1$ for $a = 1,\ldots,n$.

Each element of $\mathbf{F}$ can be written as a product of the entries of individual Gram matrices $\mathbf{H}_j\mathbf{H}_j^\top$. Specifically, $F_{ab} = (\mathbf{H}_1\mathbf{H}_1^\top)_{i_1j_1}(\mathbf{H}_2\mathbf{H}_2^\top)_{i_2j_2}\cdots(\mathbf{H}_q\mathbf{H}_q^\top)_{i_qj_q}$ with $a$ the linear index with respect to $i_1,\ldots,i_q$ and $b$ the linear index with respect to $j_1,\ldots,j_q$. This representation allows us to break dependent expressions down into their independent parts, as each $\mathbf{H}_i$ is independent from $\mathbf{H}_j$ when $i \neq j$. We can then write the expectation of $F_{ab}$ as

$$\mathbb{E}F_{ab} = \mathbb{E}(\mathbf{H}_1\mathbf{H}_1^\top)_{i_1j_1}\mathbb{E}(\mathbf{H}_2\mathbf{H}_2^\top)_{i_2j_2}\cdots\mathbb{E}(\mathbf{H}_q\mathbf{H}_q^\top)_{i_qj_q}. \tag{A.2}$$

Note that the expectation on the left is taken over all the $\sum_{j=1}^q s_j$ samples from the Hadamard factor matrices. From [31, eq. 38], we have that $\mathbb{E}(\mathbf{H}_k\mathbf{H}_k^\top)_{i_kj_k} = 0$ for $i_k \neq j_k$. If $a \neq b$, then $i_k \neq j_k$ for at least one $k$. Combining this and (A.2), we can say that $\mathbb{E}F_{ab} = 0$ for $a \neq b$. Now consider the case where $a = b$. From [31, eq. 37], we have that $(\mathbf{H}_k\mathbf{H}_k^\top)_{i_ki_k} = s_k/n_k$ deterministically. Then,

$$F_{aa} = \prod_{k=1}^q \frac{s_k}{n_k} = \frac{\ell}{n}. \tag{A.3}$$

The last piece we will need is $\mathbb{E}[F_{ab}^2]$. From [31, eq. 39], $\mathbb{E}[(\mathbf{H}_k\mathbf{H}_k^\top)_{i_kj_k}^2] = s_k/n_k^2$ for $i_k \neq j_k$, and $\mathbb{E}[(\mathbf{H}_k\mathbf{H}_k^\top)_{i_ki_k}^2] = s_k^2/n_k^2$ from above. If $a \neq b$, then $i_{k'} \neq j_{k'}$ for at least one $k'$. Combining this and (A.2),

$$\mathbb{E}[F_{ab}^2] \leq \frac{s_{k'}}{n_{k'}^2}\prod_{\substack{k=1\\k\neq k'}}^q \frac{s_k^2}{n_k^2} = \frac{\ell^2}{s_{k'}n^2} \leq \frac{\ell^2}{\min_k\{s_k\}n^2} = \frac{\ell^2}{s_k^* n^2}, \tag{A.4}$$

letting $s_k^* = \min_k\{s_k\}$.

With all these pieces in mind, we begin the main steps of the proof, which follow [31]. We start with an elementwise representation of $\mathbf{G}$, using the form in (A.1), with

$$\begin{aligned}G_{ij} &= \sum_{a,b=1}^{n} W_{ia}D_aF_{ab}D_bW_{jb}\\ &= \sum_{a=1}^{n} W_{ia}W_{ja}D_a^2F_{aa} + \sum_{a=1}^{n} W_{ia}D_a \sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}D_bF_{ab}\end{aligned}$$

for $1 \le i,j \le r$. Consider the first term, where we have isolated the case $a = b$. As the rows of $\mathbf{W}$ are orthonormal, $D_a^2 = 1$ and $F_{aa} = \ell/n$, $G_{ij}$ can be written as $G_{ij} = \frac{\ell}{n}\delta_{ij} + \sum_{a=1}^{n} W_{ia}D_a \sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}D_bF_{ab}$, where $\delta_{ij}$ is the Kronecker delta which is 1 when $i=j$ and 0 otherwise. Defining $\mathbf{M}\in\mathbb{R}^{r\times r}$ to be $\frac{\ell}{n}\mathbf{I}$, and letting $\mathbf{N}\in\mathbb{R}^{r\times r}$ be the matrix with entries $N_{ij} = \sum_{a=1}^{n} W_{ia}D_a \sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}D_bF_{ab}$, we have $\mathbf{G}=\mathbf{M}+\mathbf{N}$.

We immediately have $\|\mathbf{M}\|_2 = \ell/n$. Bounding $\|\mathbf{N}\|$ is trickier; our approach will be to use the fact $\mathbb{E}\|\mathbf{N}\|_2^2 \le \mathbb{E}\|\mathbf{N}\|_F^2 = \sum_{i,j=1}^{r}\mathbb{E}[N_{ij}^2]$ and first bound $\mathbb{E}[N_{ij}^2]$. We start by expanding the product

(A.5)

$$\begin{aligned}\mathbb{E}[N_{ij}^2] &= \mathbb{E}\left(\sum_{a=1}^{n} W_{ia}D_a \sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}D_bF_{ab}\right)\left(\sum_{c=1}^{n} W_{ic}D_c \sum_{\substack{f=1\\ f\neq c}}^{n} W_{jf}D_fF_{cf}\right)\\ &= \mathbb{E}\sum_{a=1}^{n} W_{ia}^2\left(\sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}D_bF_{ab}\right)^2 + \mathbb{E}\sum_{\substack{a,c=1\\ a\neq c}}^{n} W_{ia}W_{ic}D_aD_c \sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}D_bF_{ab} \sum_{\substack{f=1\\ f\neq c}}^{n} W_{jf}D_fF_{cf},\end{aligned}$$

which we have separated into the terms where $a = c$ and $a \neq c$. Consider the first term of (A.5), where $a = c$. As $\mathbf{W}$ is a deterministic matrix, the expectation only affects the terms with $D_b$ and $F_{ab}$, so we have

$$\mathbb{E}\sum_{a=1}^{n} W_{ia}^2\left(\sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}D_bF_{ab}\right)^2 = \sum_{a=1}^{n} W_{ia}^2\,\mathbb{E}\left(\sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}D_bF_{ab}\right)^2. \tag{A.6}$$

Now consider the expectation portion of (A.6) for a fixed $1 \le a \le n$. We can expand this product and distribute the expectation as

$$\begin{aligned}\mathbb{E}\left(\sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}D_bF_{ab}\right)^2 &= \mathbb{E}\sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}D_bF_{ab} \sum_{\substack{f=1\\ f\neq a}}^{n} W_{jf}D_fF_{af}\\ = \sum_{\substack{b,f=1\\ b,f\neq a}}^{n} W_{jb}W_{jf}\mathbb{E}[D_bD_fF_{ab}F_{af}] &= \sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}^2\mathbb{E}[F_{ab}^2] + \sum_{\substack{b,f=1\\ b,f\neq a\\ b\neq f}}^{n} W_{jb}W_{jf}\mathbb{E}[D_bD_fF_{ab}F_{af}],\end{aligned}$$

where the last equality separates terms into where $b=f$ and where $b\neq f$, respectively. From (A.4) and as the rows of $\mathbf{W}$ are normalized, we can write the term where $b = f$ as $\sum_{\substack{b=1\\ b\neq a}}^{n} W_{jb}^2\mathbb{E}[F_{ab}^2] \le \sum_{b=1}^{n} W_{jb}^2 \frac{\ell^2}{s_k^* n^2} = \frac{\ell^2}{s_k^* n^2}$. The term where $b \neq f$ can be written as $\sum_{\substack{b,f=1\\ b,f\neq a\\ b\neq f}}^{n} W_{jb}W_{jf}\mathbb{E}[D_bD_fF_{ab}F_{af}] = \sum_{\substack{b,f=1\\ b,f\neq a\\ b\neq f}}^{n} W_{jb}W_{jf}\mathbb{E}D_b\mathbb{E}D_f\mathbb{E}[F_{ab}F_{af}] = 0$

from $\mathbb{E}D_b = 0$. These two results can be combined into the expectation portion of (A.6) to obtain $\sum_{a=1}^n W_{ia}^2 \mathbb{E}\left(\sum_{\substack{b=1\\b\neq a}}^n W_{jb}D_bF_{ab}\right)^2 \leq \sum_{a=1}^n W_{ia}^2 \frac{\ell^2}{s_k^* n^2} = \frac{\ell^2}{s_k^* n^2}$.

We now focus on the second term of (A.5), where $a \neq c$. We split up both of the last two sums to extract the terms where $b = c$ and where $f = a$, giving

$$\begin{aligned}
&\mathbb{E}\sum_{\substack{a,c=1\\a\neq c}}^n W_{ia}W_{ic}D_aD_c \sum_{\substack{b=1\\b\neq a}}^n W_{jb}D_bF_{ab} \sum_{\substack{f=1\\f\neq c}}^n W_{jf}D_fF_{cf}\\
&\quad = \mathbb{E}\sum_{\substack{a,c=1\\a\neq c}}^n W_{ia}W_{ic}D_aD_c \left(W_{jc}D_cF_{ac} + \sum_{\substack{b=1\\a\neq b\neq c}}^n W_{jb}D_bF_{ab}\right)\left(W_{ja}D_aF_{ca} + \sum_{\substack{f=1\\a\neq f\neq c}}^n W_{jf}D_fF_{cf}\right).
\end{aligned}$$

We expand this product into four terms we can bound separately, as

$$\mathbb{E}\sum_{\substack{a,c=1\\a\neq c}}^n W_{ia}W_{ic}D_aD_c \left(W_{jc}D_cF_{ac} + \sum_{\substack{b=1\\a\neq b\neq c}}^n W_{jb}D_bF_{ab}\right)\left(W_{ja}D_aF_{ca} + \sum_{\substack{f=1\\a\neq f\neq c}}^n W_{jf}D_fF_{cf}\right)$$

$$= \mathbb{E}\sum_{\substack{a,c=1\\a\neq c}}^n W_{ia}W_{ic}D_aD_cW_{jc}W_{ja}D_cD_aF_{ac}F_{ca} \tag{A.7a}$$

$$+ \mathbb{E}\sum_{\substack{a,c=1\\a\neq c}}^n W_{ia}W_{ic}D_aD_c \sum_{\substack{b=1\\a\neq b\neq c}}^n W_{jb}D_bF_{ab} \sum_{\substack{f=1\\a\neq f\neq c}}^n W_{jf}D_fF_{cf} \tag{A.7b}$$

$$+ \mathbb{E}\sum_{\substack{a,c=1\\a\neq c}}^n W_{ia}W_{ic}D_aD_cW_{jc}D_cF_{ac} \sum_{\substack{f=1\\a\neq f\neq c}}^n W_{jf}D_fF_{cf} \tag{A.7c}$$

$$+ \mathbb{E}\sum_{\substack{a,c=1\\a\neq c}}^n W_{ia}W_{ic}D_aD_cW_{ja}D_aF_{ca} \sum_{\substack{b=1\\a\neq b\neq c}}^n W_{jb}D_bF_{ab}. \tag{A.7d}$$

Consider (A.7a). As $D_k^2 = 1$ and $\mathbf{F}$ is symmetric, the expectation is just affected by $F_{ac}^2$. We then have

$$\begin{aligned}
\mathbb{E}\sum_{\substack{a,c=1\\a\neq c}}^n W_{ia}W_{ic}D_aD_cW_{jc}W_{ja}D_cD_aF_{ac}F_{ca} &= \sum_{\substack{a,c=1\\a\neq c}}^n W_{ia}W_{ic}W_{jc}W_{ja}\mathbb{E}[F_{ac}^2]\\
&\leq \frac{\ell^2}{s_k^* n^2}\sum_{\substack{a,c=1\\a\neq c}}^n W_{ia}W_{ic}W_{jc}W_{ja} \leq \frac{\ell^2}{s_k^* n^2}\sum_{a,c=1}^n W_{ia}W_{ic}W_{jc}W_{ja}\\
&= \frac{\ell^2}{s_k^* n^2}\left(\sum_{a=1}^n W_{ia}W_{ja}\right)^2 = \frac{\ell^2}{s_k^* n^2}\left[\mathbf{W}\mathbf{W}^\top\right]_{ij}^2 = \delta_{ij}\frac{\ell^2}{s_k^* n^2}.
\end{aligned}$$

In all three of the remaining parts, (A.7b), (A.7c), and (A.7d), distributing the expectation to the independent random components gives us the expectation of the product of independent Rademacher entries. This means all three of these parts are equal to 0. With all these pieces, we now have $\mathbb{E}[N_{ij}^2] \leq \frac{\ell^2}{s_k^* n^2} + \delta_{ij}\frac{\ell^2}{s_k^* n^2}$. With the bound on $\mathbb{E}[N_{ij}^2]$, we can now bound the expectation of the norm of $\mathbf{N}$: $\mathbb{E}\|\mathbf{N}\|_2^2 \leq \mathbb{E}\|\mathbf{N}\|_F^2 = \mathbb{E}\sum_{i,j=1}^r N_{ij}^2 \leq \frac{\ell^2}{s_k^* n^2}\sum_{i,j=1}^r(1+\delta_{ij}) = (r^2+r)\frac{\ell^2}{s_k^* n^2}$. Then, using Markov's inequality, $\|\mathbf{N}\|_2 \leq \frac{\beta\ell}{n}\sqrt{\frac{(r^2+r)}{s_k^*}}$ with probability at least $1-\frac{1}{\beta}$.

Now consider the term $\|\mathbf{G}^\dagger\|_2$. Recalling that $\mathbf{G} = \mathbf{M}+\mathbf{N}$, we can express this instead as $\mathbf{G} = (\mathbf{I}+\mathbf{N}\mathbf{M}^{-1})\mathbf{M}$. Then we can write $\mathbf{G}^\dagger = \mathbf{M}^{-1}(\mathbf{I}+\mathbf{N}\mathbf{M}^{-1})^\dagger$. Taking norms, we have $\|\mathbf{G}^\dagger\|_2 \leq \|\mathbf{M}^{-1}\|_2\|(\mathbf{I}+\mathbf{N}\mathbf{M}^{-1})^\dagger\|_2 \leq \frac{n}{\ell}\sum_{k=0}^\infty \|\mathbf{N}\mathbf{M}^{-1}\|_2^k$, where we use the Taylor expansion $(\mathbf{I}+\mathbf{N}\mathbf{M}^{-1})^\dagger = \sum_{k=0}^\infty(-\mathbf{N}\mathbf{M}^{-1})^k$ (see [18, Corollary 5.6.16] for more details). We can then write $\|\mathbf{G}^\dagger\|_2 \leq \frac{n}{\ell}\sum_{k=0}^\infty\left(\|\mathbf{N}\|_2\|\mathbf{M}^{-1}\|_2\right)^k$. We now consider $\|\mathbf{N}\|_2\|\mathbf{M}^{-1}\|_2$ before the entire expression. As $\|\mathbf{M}^{-1}\|_2 = n/\ell$, $\|\mathbf{N}\|_2\|\mathbf{M}^{-1}\|_2 \leq \sqrt{\frac{\beta^2(r^2+r)}{s_k^*}} \leq 1-\frac{1}{\alpha}$ with probability at least $1-\frac{1}{\beta}$, where the last inequality comes from (5.5). Then, $\|\mathbf{G}^\dagger\|_2 \leq \frac{n}{\ell}\sum_{k=0}^\infty\left(1-\frac{1}{\alpha}\right)^k = \frac{n\alpha}{\ell}$. Our smallest singular value is then $\frac{1}{\sigma_{\min}^2(\mathbf{W}\mathbf{\Omega})} = \|\mathbf{G}^\dagger\|_2 \leq \frac{\alpha n}{\ell}$, with probability at least $1-\frac{1}{\beta}$. Taking the square root, we obtain the desired result. ☐

## REFERENCES

[1] S. Ahmadi-Asl, S. Abukhovich, M. G. Asante-Mensah, A. Cichocki, A. H. Phan, T. Tanaka, and I. Oseledets, *Randomized algorithms for computation of Tucker decomposition and higher order SVD (HOSVD)*, IEEE Access, 9 (2021), pp. 28684–28706, https://doi.org/10.1109/ACCESS.2021.3058103.

[2] H. Al Daas, G. Ballard, L. Grigori, S. Kumar, and K. Rouse, *Communication lower bounds and optimal algorithms for multiple tensor-times-matrix computation*, SIAM J Matrix Anal. Appl., 45 (2024), pp. 450–477, https://doi.org/10.1137/22M1510443.

[3] G. Ballard, A. Klinvex, and T. G. Kolda, *TuckerMPI: A parallel C++/MPI software package for large-scale data compression via the Tucker tensor decomposition*, ACM Trans. Math. Software, 46 (2020), 13, https://doi.org/10.1145/3378445.

[4] K. Batselier, W. Yu, L. Daniel, and N. Wong, *Computing low-rank approximations of large-scale matrices with the tensor network randomized SVD*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 1221–1244, https://doi.org/10.1137/17M1140480.

[5] C. Battaglino, G. Ballard, and T. G. Kolda, *A practical randomized CP tensor decomposition*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 876–901, https://doi.org/10.1137/17M1112303.

[6] W. H. Cabot and A. W. Cook, *Reynolds number effects on Rayleigh–Taylor instability with possible implications for type Ia supernovae*, Nature Phys., 2 (2006), pp. 562–568, https://doi.org/10.1038/nphys361.

[7] V. T. Chakaravarthy, J. W. Choi, D. J. Joseph, X. Liu, P. Murali, Y. Sabharwal, and D. Sreedhar, *On optimizing distributed Tucker decomposition for dense tensors*, in 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, Piscataway, NJ, 2017, pp. 1038–1047, https://doi.org/10.1109/IPDPS.2017.86.

[8] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, *Collective communication: Theory, practice, and experience*, Concurr. Comput. Pract. Exp., 19 (2007), pp. 1749–1783, https://doi.org/10.1002/cpe.1206.

[9] M. Che and Y. Wei, *Randomized algorithms for the approximations of Tucker and the tensor train decompositions*, Adv. Comput. Math., 45 (2019), pp. 395–428, https://doi.org/10.1007/s10444-018-9622-8.

[10] M. Che, Y. Wei, and H. Yan, *The computation of low multilinear rank approximations of tensors via power scheme and random projection*, SIAM J. Matrix Anal. Appl., 41 (2020), pp. 605–636, https://doi.org/10.1137/19M1237016.

[11] M. Che, Y. Wei, and H. Yan, *An efficient randomized algorithm for computing the approximate Tucker decomposition*, J. Sci. Comput., 88 (2021), pp. 1–29, https://doi.org/10.1007/s10915-021-01545-5.

[12] J. Choi, X. Liu, and V. Chakaravarthy, *High-performance dense Tucker decomposition on GPU clusters*, in SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2018, pp. 543–553, https://doi.org/10.1109/SC.2018.00045.

[13] H. A. Daas, G. Ballard, P. Cazeaux, E. Hallman, A. Międlar, M. Pasha, T. W. Reid, and A. K. Saibaba, *Randomized algorithms for rounding in the tensor-train format*, SIAM J Sci. Comput., 45 (2023), pp. A74–A95.

[14] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278, https://doi.org/10.1137/S0895479896305696.

[15] L. De Lathauwer, B. De Moor, and J. Vandewalle, *On the best rank-*1 *and rank-(R*1*,R*2*,...,Rn) approximation of higher-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342, https://doi.org/10.1137/S0895479898346995.

[16] N. B. Erichson, K. Manohar, S. L. Brunton, and J. N. Kutz, *Randomized CP tensor decomposition*, Mach. Learn. Sci. Technol., 1 (2020), 025012, https://doi.org/10.1088/2632-2153/ab8240.

[17] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288, https://doi.org/10.1137/090771806.

[18] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, 2012.

[19] R. Jin, T. G. Kolda, and R. Ward, *Faster Johnson–Lindenstrauss transforms via Kronecker products*, Inf. Inference, 10 (2021), pp. 1533–1562, https://doi.org/10.1093/imaiai/iaaa028.

[20] O. Kaya and Y. Robert, *Computing dense tensor decompositions with optimal dimension trees*, Algorithmica, 81 (2019), pp. 2092–2121, https://doi.org/10.1007/s00453-018-0525-3.

[21] O. Kaya and B. Uçar, *High performance parallel algorithms for the Tucker decomposition of sparse tensors*, in 45th International Conference on Parallel Processing (ICPP '16), IEEE Computer Society, Los Alamitos, CA, 2016, pp. 103–112, https://doi.org/10.1109/ICPP.2016.19.

[22] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500, https://doi.org/10.1137/07070111X.

[23] H. Kolla, X.-Y. Zhao, J. H. Chen, and N. Swaminathan, *Velocity and reactive scalar dissipation spectra in turbulent premixed flames*, Combust. Sci. Technol., 188 (2016), pp. 1424–1439, https://doi.org/10.1080/00102202.2016.1197211.

[24] Z. Li, Q. Fang, and G. Ballard, *Parallel Tucker decomposition with numerically accurate SVD*, in 50th International Conference on Parallel Processing, ICPP '21, ACM, New York, 2021, 49, https://doi.org/10.1145/3472456.3472472.

[25] R. Minster, A. K. Saibaba, and M. E. Kilmer, *Randomized algorithms for low-rank tensor decompositions in the Tucker format*, SIAM J. Math. Data Sci., 2 (2020), pp. 189–215, https://doi.org/10.1137/19M1261043.

[26] A.-H. Phan, P. Tichavsky, and A. Cichocki, *Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations*, IEEE Trans. Signal Process., 61 (2013), pp. 4834–4846, https://doi.org/10.1109/TSP.2013.2269903.

[27] Y. Sun, Y. Guo, C. Luo, J. Tropp, and M. Udell, *Low-rank Tucker approximation of a tensor from streaming data*, SIAM J. Math. Data Sci., 2 (2020), pp. 1123–1150, https://doi.org/10.1137/19M1257718.

[28] J. A. Tropp, *Improved analysis of the subsampled randomized Hadamard transform*, Adv. Adapt. Data Anal., 3 (2011), pp. 115–126, https://doi.org/10.1142/S1793536911000787.

[29] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, *A new truncation strategy for the higher-order singular value decomposition*, SIAM J. Sci. Comput., 34 (2012), pp. A1027–A1052, https://doi.org/10.1137/110836067.

[30] A. S. J. W. Wolf, *Low Rank Tensor Decompositions for High Dimensional Data Approximation, Recovery and Prediction*, Ph.D. thesis, Technical University of Berlin, Berlin, 2019, https://doi.org/10.14279/depositonce-8109.

[31] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert, *A fast randomized algorithm for the approximation of matrices*, Appl. Comput. Harmon. Anal., 25 (2008), pp. 335–366.

[32] K. Zhao, S. Di, X. Lian, S. Li, D. Tao, J. Bessac, Z. Chen, and F. Cappello, *SDRBench: Scientific data reduction benchmark for lossy compressors*, in 2020 IEEE International Conference on Big Data (Big Data), IEEE, Piscataway, NJ, 2020, pp. 2716–2724, https://doi.org/10.1109/BigData50022.2020.9378449.

[33] G. Zhou, A. Cichocki, and S. Xie, *Decomposition of Big Tensors with Low Multilinear Rank*, preprint, arXiv:1412.1885, 2014.