CUTENSOR-TUBAL: OPTIMIZED GPU LIBRARY FOR LOW-TUBAL-RANK TENSORS

Tao Zhang[†, ‡] *, Xiao-Yang Liu*[*] *(co-primary author)*

[†]Shanghai University, [*]Columbia University,
[‡]Shanghai Institute for Advanced Communication and Data Science

ABSTRACT

In this paper, we optimize the computations of third-order low-tubal-rank tensor operations on many-core GPUs. Tensor operations are compute-intensive and existing studies optimize such operations in a case-by-case manner, which can be inefficient and error-prone. We develop and optimize a BLAS-like library for the low-tubal-rank tensor model called *cuTensor-tubal*, which includes efficient GPU primitives for tensor operations and key processes. We compute tensor operations in the frequency domain and fully exploit tube-wise and slice-wise parallelisms. We design, implement, and optimize four key tensor operations namely t-FFT, inverse t-FFT, t-product, and t-SVD. For t-product and t-SVD, cuTensor-tubal demonstrates significant speedups: maximum $29.16\times$, $6.72\times$ speedups over the non-optimized GPU counterparts, and maximum $16.91\times$ and $27.03\times$ speedups over the CPU implementations running on dual 10-core Xeon CPUs.

***Index Terms*—** Low-tubal-rank tensor model, GPU, cuTensor-tubal library, tensor product, tensor SVD

1. INTRODUCTION

Real-world data are often modeled as sparse or low-rank tensors [1, 2, 3, 4]. The low-tubal-rank tensor model [5, 6] generalizes classical matrix algorithms to a tensor space, including SVD (Singular Value Decomposition), QR decomposition, multiplication, normalization and power method. This model has been used for data completion [7, 8], MRI imaging [9], tensor sparse coding [10], indoor localization for smartphones [11], seismic data processing [12], and wireless tomographic imaging [13].

However, tensor algorithmic workloads are compute-intensive because their complexity grows exponentially with the order of tensors. Existing works exploit GPUs to accelerate tensor operations, applying different approaches to optimize specific tensor operations such as tensor contraction [14, 15], factorization [16, 17], transpose [18, 19] and tensor-matrix multiplication [20, 21]. In contrast, we provide an optimized library of common tensor operations and key processes for the low-tubal-rank tensor model on GPUs, in order to efficiently support a wide range of applications.

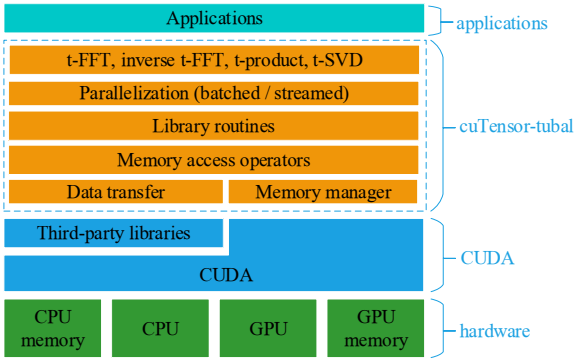

Fig. 1. System architecture of the cuTensor-tubal library.

The primary challenges associated with the optimization of tensor operations on GPUs include the following: 1) obtaining suitable parallelization schemes for tensor operations exhibiting data-dependent and data-independent computation patterns on GPUs; 2) designing efficient data transfer and memory access to facilitate computations; and 3) fully utilizing hardware resources, particularly on-board and in-chip memories. We propose several optimization methods for low-tubal-rank tensor operations to address these challenges. We encapsulate these high-performance tensor operations and key processes into an open-source library called *cuTensor-tubal*, whose system architecture is shown in Fig. 1. Evaluation results show that the library achieves good performance. For t-product and t-SVD, cuTensor-tubal demonstrates significant speedups: maximum $29.16\times$, $6.72\times$ speedups over the non-optimized GPU counterparts, and maximum $16.91\times$ and $27.03\times$ speedups over the CPU implementations running on dual 10-core Xeon CPUs. The cuTensor-tubal library is available at: http://www.tensorlet.com.

2. THIRD-ORDER LOW-TUBAL-RANK TENSOR MODEL

The low-tubal-rank model was initially proposed in [5, 6]. We briefly summarize key concepts and operations, while detailed information can be found in [8].

This research is partially supported by Natural Science Foundation of Shanghai under grant No. 17ZR1409800 and the Science and Technology Committee of Shanghai Municipality under Grant No. 16010500400.

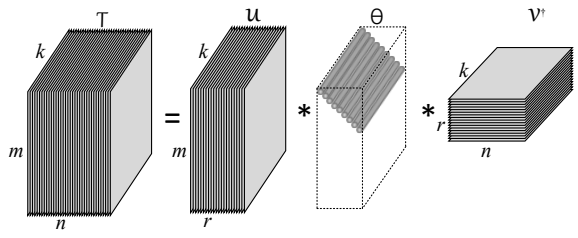

Fig. 2. t-SVD of an $m \times n \times k$ tensor of tubal-rank r.

2.1. Notations

We use uppercase calligraphic letters to denote third-order tensors, e.g., $\mathcal{X} \in \mathbb{R}^{m \times n \times k}$. For tensor $\mathcal{T} \in \mathbb{R}^{m \times n \times k}$, the (i, j, ℓ)-th entry is $\mathcal{T}(i, j, \ell)$, or concisely represented as $\mathcal{T}_{ij\ell}$. Let $[n]$ denote the set $\{1, ..., n\}$.

2.2. Tensor Operations

Tubes/fibers, and slices: A tube/fiber is a 1-D section by fixing all indices except one, while a slice is a 2-D section by fixing all except two indices. We use $\mathcal{T}(:, j, \ell)$, $\mathcal{T}(i, :, \ell)$, $\mathcal{T}(i, j, :)$ to denote the mode-1, mode-2, mode-3 tubes that are vectors, and $\mathcal{T}(:, :, \ell)$, $\mathcal{T}(:, j, :)$, $\mathcal{T}(i, :, :)$ to denote the frontal, lateral, horizontal slices that are matrices. For easy representation, we denote $\mathcal{T}^{(\ell)} = \mathcal{T}(:, :, \ell)$.

t-transpose: $\mathcal{T}^\dagger$ is obtained by transposing each frontal slice and then reversing the order of transposed frontal slices 2 through k. That is, $\mathcal{T}^\dagger(:, :, 1) = (\mathcal{T}(:, :, 1))^H$ and for $2 \le \ell \le k$, $\mathcal{T}^\dagger(:, :, \ell) = \mathcal{T}^{\dagger(\ell)} = (\mathcal{T}(:, :, k + 2 - \ell))^H$ (the transpose of matrix $\mathcal{T}(:, :, k + 2 - \ell)$).

t-FFT: we define $\widetilde{\mathcal{T}}$ as the frequency domain representation of $\mathcal{T}$ by taking the Fourier transform along the third dimension of $\mathcal{T}$, i.e., $\widetilde{\mathcal{T}}(i, j, :) = \text{fft}(\mathcal{T}(i, j, :))$. In MATLAB notations, $\widetilde{\mathcal{T}} = \text{fft}(\mathcal{T}, [\,], 3)$, and one can also compute $\mathcal{T}$ from $\widetilde{\mathcal{T}}$ via $\mathcal{T} = \text{ifft}(\widetilde{\mathcal{T}}, [\,], 3)$.

t-product: For two tubes/vectors of the same size, say $\mathbf{a}, \mathbf{b} \in \mathbb{R}^k$, let $\mathbf{a} * \mathbf{b}$ denote the *circular convolution* that preserves the dimension. Then, the t-product $\mathcal{C} = \mathcal{A} * \mathcal{B}$ of $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times k}$ and $\mathcal{B} \in \mathbb{R}^{n_2 \times n_3 \times k}$ is a tensor of size $n_1 \times n_3 \times k$, where $\mathcal{C}(i, j, :) = \sum_{s=1}^{n_2} \mathcal{A}(i, s, :) * \mathcal{B}(s, j, :)$, for $i \in [n_1]$ and $j \in [n_3]$.

t-SVD: For $\mathcal{T} \in \mathbb{R}^{m \times n \times k}$, the t-SVD of $\mathcal{T}$ is given by $\mathcal{T} = \mathcal{U} * \Theta * \mathcal{V}^\dagger$ as shown in Fig. 2, where $\mathcal{U}$ and $\mathcal{V}$ are orthogonal tensors of sizes $m \times m \times k$ and $n \times n \times k$, respectively; and Θ is an f-diagonal tensor of size $m \times n \times k$ and the tubes are called the eigentubes of $\mathcal{T}$. The **tensor tubal-rank** of a third-order tensor $\mathcal{T}$ is the number of non-zero tubes in Θ, denoted as r. Suppose $\mathcal{T}$ has tubal-rank r, then the reduced t-SVD of $\mathcal{T}$ is $\mathcal{T} = \mathcal{U} * \Theta * \mathcal{V}^\dagger$, where $\mathcal{U} \in \mathbb{R}^{m \times r \times k}$ and $\mathcal{V} \in \mathbb{R}^{n \times r \times k}$ satisfying $\mathcal{U}^\dagger * \mathcal{U} = \mathcal{I}$, $\mathcal{V}^\dagger * \mathcal{V} = \mathcal{I}$, and Θ is a f-diagonal tensor of size $r \times r \times k$. This reduced version of t-SVD will be used throughout the paper.

2.3. Operations in Frequency Domain

Recall the Convolution Theorem that *circular convolution in the time domain corresponds to point-wise multiplication in the frequency domain*, thus tensor operations of the low-tubal-rank tensor model can be efficiently computed in the frequency domain as in [8, 22]. As shown in Alg. 1, to compute $\mathcal{C} = \mathcal{A} * \mathcal{B}$, we could compute $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{B}}$, then perform front-slice-wise matrix multiplications (denoted as $\cdot\S$) of these two tensors in the frequency domain to derive $\widetilde{\mathcal{C}}$, and finally obtain $\mathcal{C}$ by applying the inverse FFT on $\widetilde{\mathcal{C}}$.

ALGORITHM 1: Computing t-product in the frequency domain

1: **Input**: Tensors $\mathcal{A} \in \mathbb{R}^{m \times p \times k}$, $\mathcal{B} \in \mathbb{R}^{p \times n \times k}$.
2: **Output**: $\mathcal{C} \in \mathbb{R}^{m \times n \times k}$
3: $\widetilde{\mathcal{A}} = \text{fft}(\mathcal{A}, [\,], 3)$;
4: $\widetilde{\mathcal{B}} = \text{fft}(\mathcal{B}, [\,], 3)$;
5: $\widetilde{\mathcal{C}} = \widetilde{\mathcal{A}} \cdot \S \widetilde{\mathcal{B}}$; % perform front-slice-wise matrix multiplication
6: $\mathcal{C} = \text{ifft}(\widetilde{\mathcal{C}}, [\,], 3)$;

3. EFFICIENT PARALLEL TENSOR OPERATIONS

We present a novel unified computation flow for the third-order low-tubal-rank tensor model:

- 1) obtain the frequency domain representation of the input tensor by performing Fourier transform along the third-dimension (tube-wise DFT), called the t-FFT;
- 2) in the frequency domain, the tensor operations are separated into multiple independent (complex) matrix operations that possess strong parallelism. We provide batch mode for these (complex) matrix operations on many-core GPU architectures;
- 3) converting the frequency domain results back to the time domain, called the inverse t-FFT.

3.1. Overview of the cuTensor-tubal Library

The system architecture of cuTensor-tubal is presented in Fig. 1. cuTensor-tubal runs on top of the hardware, CUDA and third-party libraries such as MAGMA. To improve performance, cuTensor-tubal utilizes highly optimized single and batched matrix routines from the cuBLAS and MAGMA libraries. The cuTensor-tubal library is divided into five layers. The lowest layer consists of two modules: the data transfer module handles the data transfer between the CPU and the GPU; the memory manager is in charge of the allocation and release of the GPU memory. The second layer comprises the memory access operators for efficient fetching and storing of tubes or slices. The third layer contains library routines implemented in cuTensor-tubal. The fourth layer consists of two

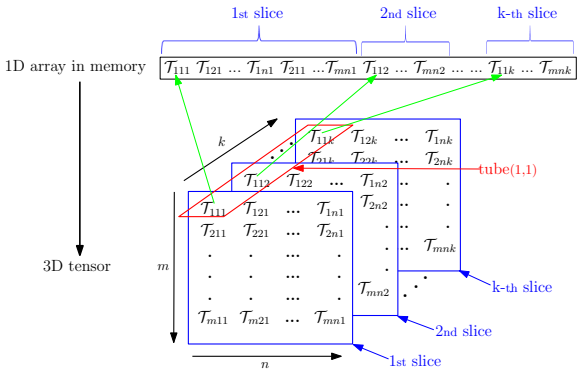

Fig. 3. Tensors are stored as an 1D array in memory.

parallelization schemes. The highest layer contains four tensor operations implemented using the routines and modules below them.

3.2. Efficient Data Transfer

Tensor operations are data-intensive and their input and output data increase rapidly with the size of tensors, leading to a significant time overhead for transferring data between the CPU and GPU. From Sec. 2, we observe that t-product takes two input tensors while t-SVD outputs three tensors. We exploit this feature to overlap computation with data transfer. For t-product, we overlap the Fourier transform of $\mathcal{A}$ with the data transfer of $\mathcal{B}$. For t-SVD, we overlap the inverse Fourier transform of $\widetilde{\Theta}$ with the data transfer of $\mathcal{U}$, and overlap the inverse Fourier transform of $\widetilde{\mathcal{V}}$ with the data transfer of Θ.

3.3. Uniform Memory Access

Fig. 3 illustrates the memory layout of a tensor. We observe that slice elements are stored contiguously while tube elements are stored separately. In the computation flow of tensor operations, the t-FFT and inverse t-FFT operations fetch tubes while the matrix operations fetch slices. We design four memory access operators to provide uniform and easy memory accesses for all tensor operations: tube-strided-fetch, tube-strided-store, slice-fetch and slice-store. The tube-strided-fetch and the tube-strided-store operator are used for the t-FFT and inverse t-FFT operations to fetch and store tubes from the 1D data in memory. The slice-fetch and the slice-store operator can be used by the matrix operators to fetch and store slices in the memory.

3.4. Parallelization Schemes

We design two parallelization schemes for efficient t-FFT, inverse t-FFT, and complex matrix operations on the GPU.

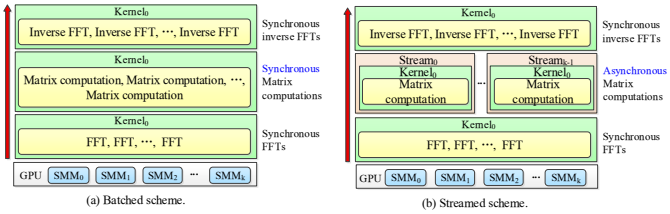

Fig. 4. Batched scheme and streamed scheme.

3.4.1. *Batched Scheme*

This scheme advocates the synchronous paradigm, which is designed for tensor operations with data-independent (i.e. regular) computation patterns such as t-product. In this scheme, only one CUDA kernel is launched in each step to compute multiple Fourier transforms, inverse Fourier transforms or matrix computations. This scheme requires a batched routine in each step of a tensor operation to perform multiple computations *synchronously*, as shown in Fig. 4(a).

3.4.2. *Streamed Scheme*

This scheme advocates the asynchronous paradigm, which is designed for tensor operations with data-dependent (i.e. irregular) computation patterns such as t-SVD. This scheme differs from the batched scheme only in the matrix computation step, in which it launches multiple CUDA streams to perform the matrix computations. Each stream contains a CUDA kernel to compute a single matrix computation, as shown in Fig. 4(b). Therefore, the matrix computations in different streams are executed *asynchronously*.

3.5. Exploit Conjugate Symmetry to Save Computation

The t-FFT has the conjugate symmetry property along the third dimension. That is, for $\mathcal{A} \in \mathbb{R}^{m\times n\times k}$, we know that $\widetilde{\mathcal{A}}$ satisfies:

$$\widetilde{\mathcal{A}}^{(\ell)} = \text{conjugate}(\widetilde{\mathcal{A}}^{(k-\ell+2)}), \quad \ell = \left\lceil \frac{k+1}{2} \right\rceil + 1, ..., k,$$

where $\widetilde{\mathcal{A}}^{(\ell)} = \widetilde{\mathcal{A}}(:,:,\ell)$. We exploit this property in our implementation that reduces the computational time. For example, to compute a t-product $\mathcal{C} = \mathcal{A} * \mathcal{B}$, we only compute for half the slices of $\mathcal{A}$ and $\mathcal{B}$ and for the rest slices, we take conjugates.

4. EXPERIMENTAL RESULTS

We use a server as the experiment platform. The server has two Intel Xeon E5-2640 V4 CPUs, each having 10 cores @2.4GHz supporting 20 hardware threads with hyper-threading. There is a Tesla V100 GPU which consists of 5120 CUDA cores @1.53 GHz and 32 GB device memory. There

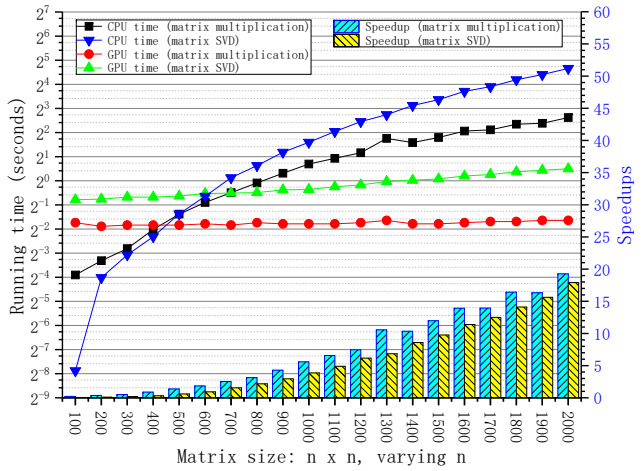

Fig. 5. Running time and speedups of matrix multiplication and SVD on Tesla V100 GPU and two 10-core CPUs, respectively.

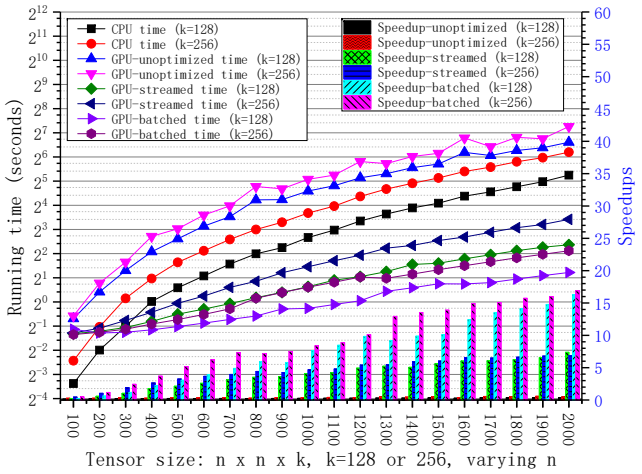

Fig. 6. Running time and speedups of t-product.

are 80 GB DDR4 memories @2.133 GHz on the server. We show the speedups of tensor operations on the Tesla V100 GPU versus dual 10-core CPUs. The speedup is defined as: (CPU running time)/(GPU running time).

Fig. 5 shows the running time and speedups of matrix multiplication and SVD on the GPU and two CPUs, respectively. Compared to the CPU implementations, the GPU matrix multiplication achieves an average of $7.37\times$ and up to $19.27\times$ speedups, and matrix SVD achieves an average of $6.02\times$ and up to $17.91\times$ speedups, respectively. These speedups are the baseline reference to understand the relative computation power of the Tesla V100 GPU and two CPUs.

Fig. 6 shows the running time and speedups of t-product on different hardware and tensor sizes. Compared to t-product running on two CPUs, the GPU batched and streamed t-product achieves an average of $8.54\times$ and $4.28\times$, and up to $16.91\times$ and $7.38\times$ speedups, respectively. The parallelism and GPU utilization grow with tensor sizes, which improves GPU performance. The unoptimized GPU implementation achieves average $0.34\times$ and up to $0.58\times$ speedups, which is even slower than the CPU implementations. The unoptimized GPU t-product employed none of the optimization techniques presented in Section 3, especially the parallelization schemes. As a result, it conducts single FFT, matrix multiplication or inverse FFT instead of batched computations on the GPU, leading to very low GPU utilizations and performance.

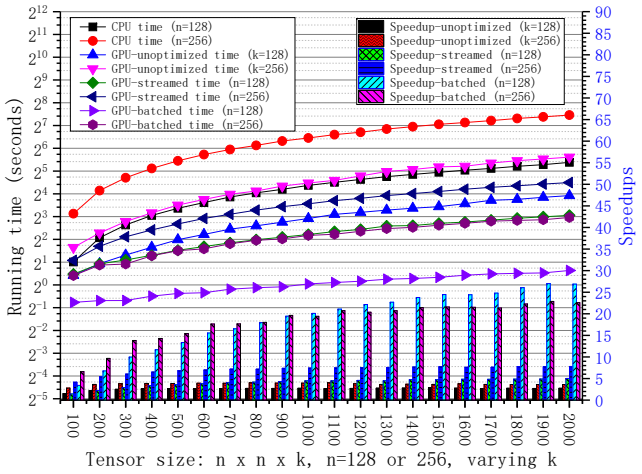

Fig. 7. Running time and speedups of t-SVD.

Fig. 7 shows the running time and speedups of t-SVD on different hardware and tensor sizes. Compared to t-SVD running on two CPUs, the GPU batched and streamed t-SVD achieves average $18.63\times$ and $5.58\times$, and up to $27.03\times$ and $7.80\times$ speedups, respectively. The unoptimized GPU implementation achieves average $3.17\times$ and up to $4.02\times$ speedups, which is slower than the streamed or batched implementations. This demonstrates the effectiveness of the optimization techniques in Sec. 3.

5. CONCLUSIONS

In this paper, we have proposed a cuTensor-tubal library of common tensor operations for the low-tubal-rank model. The cuTensor-tubal library exploits the separability in the frequency domain and maps the parallelism between tube-wise Fourier transforms and slice-wise matrix operations onto SIMT GPU architectures. Our library incorporates two parallelization schemes and several optimizations. We compared the running time and speedups of different GPU implementations and showed the effectiveness of our code optimizations. In summary, tensor operations in cuTensor-tubal achieve a maximum $27.03\times$ speedup versus two 10-core CPUs. The cuTensor-tubal library can be used to accelerate various applications.

6. REFERENCES

[1] Andrzej Cichocki, Danilo Mandic, Lieven De Lathauwer, Guoxu Zhou, Qibin Zhao, Cesar Caiafa, and Huy Anh Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[2] Lili Huang, Xuan Wu, Wenze Shao, Hongyi Liu, Zhihui Wei, and Liang Xiao, "Color demosaicking via nonlocal tensor representation," in *IEEE ICASSP*, 2017, pp. 1812–1816.

[3] Feifan Guan, Gangyi Jiang, Yang Song, Mei Yu, Zongju Peng, and Fen Chen, "No-reference hdr image quality assessment method based on tensor space," in *IEEE ICASSP*, 2018, pp. 1218–1222.

[4] Kazuyoshi Yoshii, "Correlated tensor factorization for audio source separation," in *IEEE ICASSP*, 2018, pp. 731–735.

[5] Misha E Kilmer and Carla D Martin, "Factorization strategies for third-order tensors," *Linear Algebra and its Applications*, vol. 435, no. 3, pp. 641–658, 2011.

[6] Misha E Kilmer, Karen Braman, Ning Hao, and Randy C Hoover, "Third-order tensors as operators on matrices: A theoretical and computational framework with applications in imaging," *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 1, pp. 148–172, 2013.

[7] Zemin Zhang and Shuchin Aeron, "Exact tensor completion using t-svd," *IEEE Transactions on Signal Processing*, vol. 65, no. 6, pp. 1511–1526, 2017.

[8] Xiao-Yang Liu, Shuchin Aeron, Vaneet Aggarwal, and Xiaodong Wang, "Low-tubal-rank tensor completion using alternating minimization," in *(Major revision) IEEE Transactions on Information Theory*, 2018.

[9] Oguz Semerci, Ning Hao, Misha E Kilmer, and Eric L Miller, "Tensor-based formulation and nuclear norm regularization for multienergy computed tomography," *IEEE Transactions on Image Processing*, vol. 23, no. 4, pp. 1678–1693, 2014.

[10] Fei Jiang, Xiao-Yang Liu, Hongtao Lu, and Ruimin Shen, "Efficient multi-dimensional tensor sparse coding using t-linear combinations," in *AAAI Conference on Artificial Intelligence*, 2018.

[11] Xiao-Yang Liu, Shuchin Aeron, Vaneet Aggarwal, Xiaodong Wang, and Min-You Wu, "Adaptive sampling of rf fingerprints for fine-grained indoor localization," *IEEE Transactions on Mobile Computing*, vol. 15, no. 10, pp. 2411–2423, 2016.

[12] Songjie Liao, Xiao-Yang Liu, Feng Qian, Miao Yin, and Guang-Min Hu, "Tensor super-resolution for seimic data," in *IEEE ICASSP*.

[13] Tao Deng, Xiao-Yang Liu, Feng Qian, Manyuan Zhang, and Anwar Walid, "Tensor sensing for rf tomographic imaging," in *IEEE International Conference on Multimedia and Expo (ICME)*, 2018.

[14] Thomas Nelson, Axel Rivera, Prasanna Balaprakash, Mary Hall, Paul D Hovland, Elizabeth Jessup, and Boyana Norris, "Generating efficient tensor contractions for gpus," in *International Conference on Parallel Processing (ICPP)*, 2015, pp. 969–978.

[15] Yang Shi, U N Niranjan, Animashree Anandkumar, and Cris Cecka, "Tensor contractions with extended blas kernels on cpu and gpu," in *IEEE International Conference on High Performance Computing Data and Analytics*, 2016, pp. 193–202.

[16] Jukka Antikainen, Jiri Havel, Radovan Josth, Adam Herout, Pavel Zemcik, and Markku Hautakasari, "Nonnegative tensor factorization accelerated using gpgpu," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, pp. 1135–1141, 2011.

[17] Benyou Zou, Cuiping Li, Liwen Tan, and Hong Chen, "Gputensor: efficient tensor factorization for context-aware recommendations," *Information Sciences*, vol. 299, pp. 159–177, 2015.

[18] Dmitry I Lyakh, "An efficient tensor transpose algorithm for multicore cpu, intel xeon phi, and nvidia tesla gpu," *Computer Physics Communications*, vol. 189, pp. 84–91, 2015.

[19] Antti-Pekka Hynninen and Dmitry I Lyakh, "cutt: A high-performance tensor transpose library for cuda compatible gpus," *arXiv preprint arXiv:1705.01598*, 2017.

[20] Jiajia Li, Yuchen Ma, Chenggang Yan, and Richard Vuduc, "Optimizing sparse tensor times matrix on multi-core and many-core architectures," in *IEEE Workshop on Irregular Applications: Architecture and Algorithms (IA3)*. IEEE, 2016, pp. 26–33.

[21] David M Rogers, "Efficient primitives for standard tensor linear algebra," in *XSEDE16 Conference on Diversity, Big Data, and Science at Scale*. ACM, 2016, p. 14.

[22] Xiao-Yang Liu and Xiaodong Wang, "Fourth-order tensors with multidimensional discrete transforms," *arXiv preprint arXiv:1705.01576*, 2017.