cuTensor-TT/TR: High Performance Third-order Tensor-Train and Tensor-Ring Decompositions on GPUs

Hao Hong[1] , **Tao Zhang**[1,3,*] , **Xiao-Yang Liu**[2]
[1]School of Computer Engineering and Science, Shanghai University, Shanghai, China
[2]Department of Electrical Engineering, Columbia University, USA
[3]Shanghai Institute for Advanced Communication and Data Science, Shanghai, China
honghao@shu.edu.cn, taozhang@shu.edu.cn, xl2427@columbia.edu

Abstract

Tensor decompositions are widely used for processing multi-dimensional data in machine learning. However, the time and space cost of tensor decompositions increases rapidly with the size and dimension of tensors. In this paper, we propose high performance GPU implementations for the third-order tensor-train (TT) decomposition and tensor-ring (TR) decomposition. First, we propose to utilize highly-parallel Jacobi-based singular value decomposition (SVD) on GPU. Second, we parallelize diagonal matrix and matrix multiplication on GPU. Thirdly, we optimize the transfer and memory access of intermediate variables to further improve performance. On a Tesla V100 GPU, we tested tensors up to 1,200 $\times$ 1,200 $\times$ 1,200. Compared with the basic GPU implementations, the proposed GPU implementations of TT and TR decompositions achieve up to 6.67$\times$ and 6.36$\times$ speedups, respectively.

Keywords GPU, tensor-train decomposition, tensor-ring decomposition, Jacobi SVD

1 Introduction

Tensor decompositions are extensions of matrix factorization to high dimensions. They are powerful in analyzing big data, such as social networks, finance and quantum physics [2] [4]. Tensor decompositions have become basic tools in data mining [1], computer vision [3] [5] and deep learning [10] [9]. With the ever-growing demands of efficient big data analytics, developing efficient tensor decompositions becomes a critical task. Exiting works proposed CPU-based tensor-train (TT) and tensor-ring (TR) decompositions [7] [11], which did not fully exploit the parallelism of tensor operations, making them impractical for processing big data sets in machine learning.

In this paper, we propose high performance GPU implementations for third-order TT and TR decompositions. We parallelize low-rank dense tensor TT and TR decompositions on many-core GPU. Evaluation results show that the GPU-based TT and TR decompositions achieve good performance.

Our major contributions are summarized as follows.

- We implement third-order TT and TR decompositions on GPU, which achieve high performance and the same accuracy as CPU. To the best of our knowledge, this is the first work for GPU-based high-performance TT and TR decompositions.
- We propose optimization strategies, including a faster and more efficient tensor access in GPU memory and an optimized parallel diagonal matrix and matrix multiplication on GPU. We reduce GPU memory consumption and avoid the operations of tensor matrixing and vector diagonalization to improve performance.
- We evaluate the performance of TT and TR decompositions with experiments on a Tesla V100 GPU. The speedup of the TT decomposition is up to 6.67$\times$ over a GPU baseline. The speedup of the TR decomposition is up to 6.36$\times$ over a GPU baseline.

*Corresponding author: Tao Zhang. Tao Zhang is supported by Science and Technology Committee of Shanghai Municipality under grant No. 19511121002 and No. 19DZ2252600.

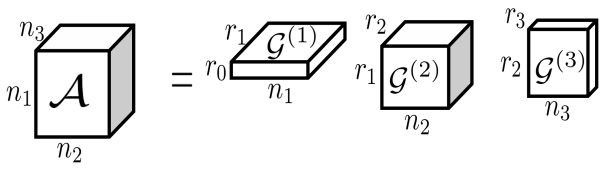

Figure 1: A third-order tensor-train decomposition.

2 Third-order TT and TR Decompositions

We briefly describe third-order tensor-train (TT) and tensor-ring (TR) decomposition algorithms.

2.1 Notions and Operations

We use boldface lowercase letters $\boldsymbol{a} \in \mathbb{R}^n$ to denote vectors, boldface uppercase letters $\boldsymbol{A} \in \mathbb{R}^{n_1 \times n_2}$ for matrices and uppercase calligraphic letters $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ for third-order tensors. We use $\circ$ to denote the tensor contraction operation.

2.2 Third-order TT Decomposition

A TT decomposition [7], as shown in Fig. 1, expresses a third-order tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ as contractions of three core tensors:

$$\mathcal{A} = \mathcal{G}^{(1)} \circ \mathcal{G}^{(2)} \circ \mathcal{G}^{(3)},$$

where $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ is the k-th core tensor. The auxiliary indices $[r_0, r_1, r_2, r_3]$ are the tensor-train ranks (TT-ranks), and $r_0 = r_3 = 1$ for third-order tensors. Thus, $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(3)}$ are matrices and $\mathcal{G}^{(2)}$ is a third-order tensor.

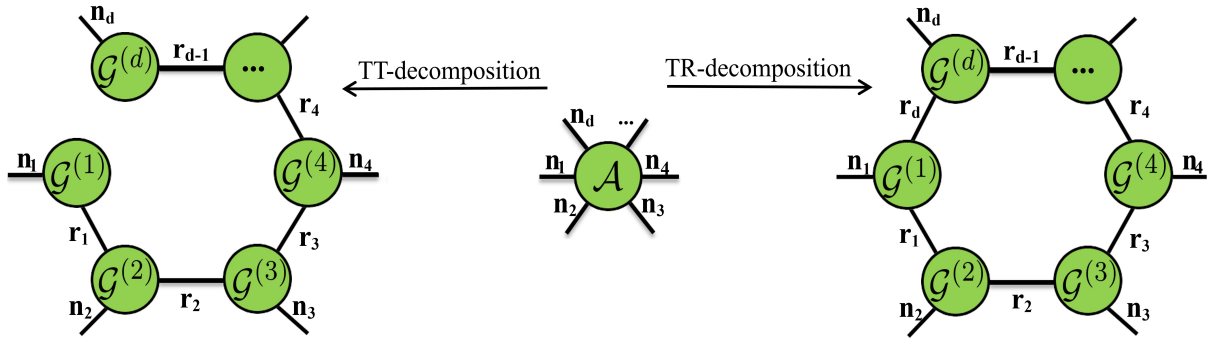

Figure 2: An illustration of TT and TR decompositions for a d-th order tensor.

Algorithm 1 Third-order Tensor-Train Decomposition [7]

Input: Tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, pre-specified accuracy ε.
Output: Cores $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \mathcal{G}^{(3)}, r_0, r_1, r_2, r_3 \in \mathbb{N}^+$.

$\mathcal{C}^{(0)} = \mathcal{A}, r_0 = r_1 = r_2 = r_3 = 1,$
for $k = 1$ to 2 **do**
  $\boldsymbol{C} = \text{reshape}(\mathcal{C}^{(k-1)}, [r_{k-1}n_k, \prod_{i=k+1}^{3} n_i]),$
  Compute SVD: $\boldsymbol{C} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T,$ and $\boldsymbol{s} = \text{diag}(\boldsymbol{S}),$
  $\delta = \frac{\varepsilon}{\sqrt{3-1}}\|\boldsymbol{s}\|_2,\ \gamma = 0,\ r_k = \sharp(\boldsymbol{s}),$
  while $\gamma \leq \delta$ **do**
    $\gamma = \gamma + s_{r_k}^2,\ r_k = r_k - 1,$
  end while
  $r_k = r_k + 1,\ \boldsymbol{U} = \boldsymbol{U}(:, 1:r_k),$
  $\boldsymbol{S} = \boldsymbol{S}(1:r_k, 1:r_k),\ \boldsymbol{V}^T = \boldsymbol{V}^T(1:r_k, :),$
  $\mathcal{G}^{(k)} = \text{reshape}(\boldsymbol{U}, [r_{k-1}, n_k, r_k]),$
  $\mathcal{C}^{(k)} = \text{reshape}(\boldsymbol{S}\boldsymbol{V}^T, [r_k, \prod_{i=k+1}^{3} n_i, r_{k+1}]),$
end for
$\mathcal{G}^{(3)} = \mathcal{C}^{(2)}.$

The TT decomposition is a special case of tensor network and can be represented by graphical model [8], as shown in Fig. 2. The connection "leg" between two circles represents the contraction operation of two tensors. Through the contraction of every small tensors, we can get the original tensor $\mathcal{A}$. Alg. 1 describes the procedures of the third-order TT decomposition [7]. The $\boldsymbol{C} = \text{reshape}(\mathcal{C}^{(k-1)}, [r_{k-1}n_k, \prod_{i=k+1}^{3} n_i])$ operation changes the tensor $\mathcal{C}^{(k-1)}$ to a matrix $\boldsymbol{C}$ with $r_{k-1}n_k$ rows and $\prod_{i=k+1}^{3} n_i$ columns. The $\mathcal{G}^{(k)} = \text{reshape}(\boldsymbol{U}, [r_{k-1}, n_k, r_k])$ operation changes the matrix $\boldsymbol{U}$ to a tensor $\mathcal{G}^{(k)}$ with r_{k-1} rows, n_k columns, and r_k in the third direction.

2.3 Third-order TR Decomposition

A TR decomposition [11], as shown in Fig. 2, represents a third-order tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ with three third-order latent tensors $\mathcal{G}^{(k)} \in \mathbb{R}^{r_k \times n_k \times r_{k+1}}, k = 1, 2, 3$:

$$\mathcal{A} = \mathcal{G}^{(1)} \circ \mathcal{G}^{(2)} \circ \mathcal{G}^{(3)},$$

where we also calculate the contraction between $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(3)}$. Auxiliary indices $[r_1, r_2, r_3]$ are the tensor-ring ranks (TR-ranks). Because of the trace characteristic of TR-format tensor, $r_4 = r_1$, which leads to the main difference between the TT decomposition and the TR decomposition. The TR decomposition is also a special case of tensor networks.

Algorithm 2 Third-order Tensor-Ring Decomposition [11]

Input: Tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, pre-specified accuracy ε.
Output: Cores $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \mathcal{G}^{(3)}, r_0, r_1, r_2, r_3 \in \mathbb{N}^+$.

1: $\mathcal{C}^{(0)} = \mathcal{A}, m = r_0 = r_1 = r_2 = r_3 = 1,$
2: Choose the first mode as the start point,
   $\boldsymbol{C} = \text{reshape}(\mathcal{C}^{(0)}, [r_0 n_1, n_2 n_3]),$
3: $\boldsymbol{C} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T,\ \boldsymbol{s} = \text{diag}(\boldsymbol{S}),$
4: $\delta = \frac{\varepsilon}{\sqrt{3-1}}\|\boldsymbol{s}\|_2,\ \gamma = 0,\ m = \sharp(\boldsymbol{s}),$
5: **while** $\gamma \leq \delta$ **do**
6: $\gamma = \gamma + s_m^2, m = m - 1,$
7: **end while**
8: $m = m + 1,\ \boldsymbol{U} = \boldsymbol{U}(:, 1:m),$
   $\boldsymbol{S} = \boldsymbol{S}(1:m, 1:m),\ \boldsymbol{V}^T = \boldsymbol{V}^T(1:m, :),$
9: Split ranks r_0, r_1 by
   $\min_{r_0, r_1} |r_0 - r_1|,$ s.t. $r_0 r_1 = m,$
10: $\mathcal{G}^{(1)} = \text{permute}(\text{reshape}(\boldsymbol{U}, [n_1, r_0, r_1]), [2, 1, 3]),$
11: $\mathcal{C} = \text{permute}(\text{reshape}(\boldsymbol{S}\boldsymbol{V}^T, [r_0, r_1, n_2 n_3]), [2, 3, 1]),$
12: $\boldsymbol{C} = \text{reshape}(\mathcal{C}, [r_1 n_2, n_3 r_0]),$
13: Repeat Line 3 to 8, and set $r_2 = m, r_3 = r_0$
14: $\mathcal{G}^{(2)} = \text{reshape}(\boldsymbol{U}, [r_1, n_2, r_2]),$
15: $\mathcal{G}^{(3)} = \text{reshape}(\boldsymbol{S}\boldsymbol{V}^T, [r_2, n_3, r_3]).$

Alg. 2 describes the procedures of the third-order TR decomposition. Because of the ring-shaped feature, the third-order TR decomposition needs to choose a start point. The $\mathcal{C} = \text{permute}(\text{reshape}(\boldsymbol{S}\boldsymbol{V}^T, [r_1, r_2, n_2 n_3]), [2, 3, 1])$ operation transposes the tensor dimensions from $[r_1, r_2, n_2 n_3]$ to $[r_2, n_2 n_3, r_1]$.

3 Efficient Third-order TT and TR Decompositions on GPU

3.1 Parallelization Schemes

For third-order TT and TR decompositions in Alg. 1 and Alg. 2, we use the following parallel optimizations.

Parallel Jacobi SVD

The most time consuming step is matrix SVD, which occupies about 68% running time. We find that conventional SVD possesses low parallelism, thus is not suitable for GPU. Instead, we use the Jacobi SVD method that has much higher parallelism and matches the computing characteristics of GPU. In the Jacobi SVD method, we need to specify an accuracy and an iteration number. In our experiment, when the calculation and storage precision are single precision, we set the accuracy 10e-8 and the maximum iteration 100 to get the smallest error. If the error between restored matrix and original matrix meets the preset threshold or the iteration reaches the maximum iteration number, the algorithm terminates.

Parallel Diagonal Matrix Times Matrix

In the Line 11 of Alg. 1 and the Lines 11 and 15 of Alg. 2, there are diagonal matrix and matrix multiplications and computing costs increases quickly with the increase of the size of matrices. Because of the diagonal matrix only having values on the diagonal, there is useless calculation in the process of diagonal matrix and matrix multiplication. To accelerate, we replace with the multiplication of corresponding lines of two matrices:

$$\boldsymbol{S}\boldsymbol{V}^T = \text{parallel}(s_k \cdot \boldsymbol{V}_k^T),$$

where s_k represents the k-th value in $\boldsymbol{S}$ on the diagonal and $\boldsymbol{V}_k^T$ represents the k-th row of $\boldsymbol{V}^T$. Through this optimization, we also eliminate the diagonalization of matrix on GPU.

Parallel Element-wise Product

In the Lines 6 to 9 of Alg. 1 and the Lines 5 to 7 of Alg. 2, we extract the parallelizable part. We calculate the element wise product of vector $\boldsymbol{s} \cdot \boldsymbol{s}$ in parallel:

$$\text{parallel}(s^{(0)}_{m-k+1} = s_k \cdot s_k),\ 1 \le k \le m,$$

where $m = \sharp(\boldsymbol{s})$. Then we use $s^{(0)}$ to calculate the r_k.

3.2 Optimizing Memory Access

Fig. 3 shows the layout of a tensor. The slices of a tensor are stored contiguously and each slice is stored in column-major layout. There are two reasons we use this kind of storage. First, we design to meet requirement of bottom layer of two CUDA libraries, cuBLAS and cuSOLVER. Secondly, we can easily obtain the truncation part of mode-1 unfolding of tensor and remove the overhead of tensor matrixing.

In the Line 11 of Alg. 1 and the Lines 11&15 of Alg. 2, we directly calculate the front truncation part of the matrix $\boldsymbol{V}^T$ and vector $\boldsymbol{s}$ to reduce the computation and to eliminate the truncation operations in the Line 9 of Alg. 1 and the Line 8 of Alg. 2. In the Lines 10&11 of Alg. 2, we propose the direct transformation in one-way array to replace the permuting of tensor. With this memory access strategy, we can eliminate reshape operations in these two Algorithms. Meanwhile, in these two algorithms, there are many intermediate variables that take up many space to affect subsequent calculations. Thus, we allocate and deallocate intermediate data structures dynamically and reuse allocated data structures in GPU memory, which effectively reduces memory consumption.

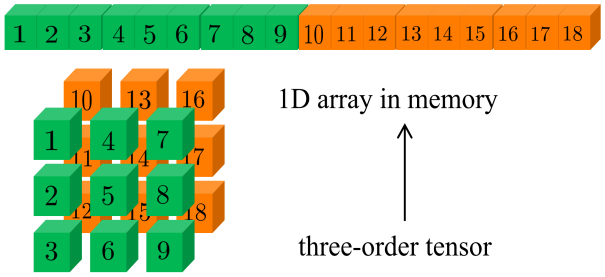

Figure 3: Tensors are stored as an 1D array in memory.

3.3 Efficient Data Transfer

TT and TR decomposition algorithms 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 GPU and CPU. We combine the cores $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ into a new one-dimensional array $\boldsymbol{c}$. Meanwhile, we use $[n_1, n_2, n_3]$ to store dimensions of tensor and $[r_0, r_1, r_2, r_3]$ to store TT-ranks and TR-ranks. The k-th core can be obtained by $\mathcal{G}_k = \text{reshape}(\boldsymbol{c}(\prod_{j=1}^{k-1} r_j n_j r_{j+1}, \prod_{j=1}^{k} r_j n_j r_{j+1}), [r_{k-1}, n_k, r_k])$, where $\boldsymbol{c}(\prod_{j=1}^{k-1} r_j n_j r_{j+1}, \prod_{j=1}^{k} r_j n_j r_{j+1})$ gets the parameters from $\prod_{j=1}^{k-1} r_j n_j r_{j+1}$ to $\prod_{j=1}^{k} r_j n_j r_{j+1}$ of $\boldsymbol{c}$.

4 Performance Evaluation

We evaluate the performance on a server that has two dual Intel Xeon E5-2640 V4 CPUs, each having 10 cores @2.4 GHz supporting 20 harware threads with hyperthreading. There is one Tesla V100 GPU with $5,120$ CUDA cores @1.53 GHz and 32GB device memory. There are 80 GB DDR4 memories @2.133 GHz on the server. We are interest in the speedups of the Tesla V100 GPU over two dual 10-core CPUs, defined as *(CPU running time)/(GPU running time)*. For error rate, we measure the relative square error (RSE), which is defined as $RSE = \|\mathcal{G}^{(1)} \circ \mathcal{G}^{(2)} \circ \mathcal{G}^{(3)} - \mathcal{A}\|_F / \|\mathcal{A}\|_F$. We construct a low rank tensor of rank 50 by multiplication of two matrices and then tensor product with with a matrix. We set the accuracy 10e-8 and the max iteration times 100 of the Jacobi SVD and prespecified $\varepsilon =$ 10e-6 under single precision.

Fig. 4 shows the running time and speedups of the third-order TT decomposition on GPU and CPU for different tensor sizes. The tensor size varies from $100 \times 100 \times 100$ to $1,200 \times 1,200 \times 1,200$. The CPU implementation used MATLAB code from [7]. The maximum tensor size can be supported on the GPU is $1,200 \times 1,200 \times 1,200$ due to the GPU memory size. Compared to the CPU implementation, the GPU baseline achieves an average of $3.50\times$ and up to $6.10\times$ speedups. When the size of tensor comes over $500 \times 500 \times 500$, speedups of the GPU baseline get smaller because of the increased cost of low parallelism SVD. Compared to the CPU implementation, the optimized GPU implementation achieves an average $14.25\times$ and up to $24.80\times$ speedups. The RSE reaches the 10e-4 level on GPU and CPU. When the size of the tensor is bigger than $500 \times 500 \times 500$, the speedups has an overall upward trend.

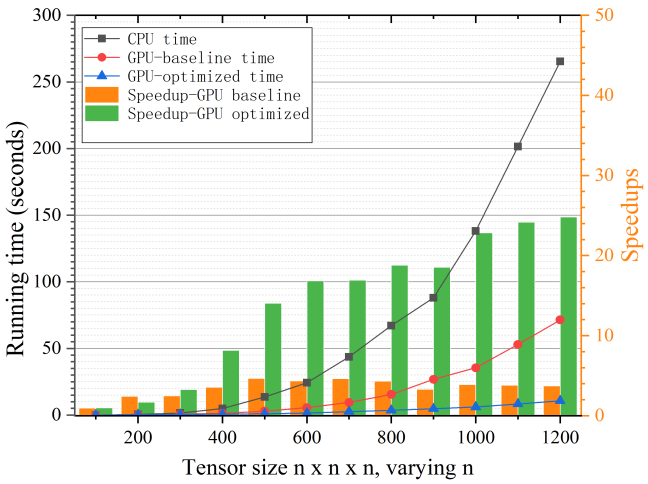

Figure 4: Running time and speedups of third-order TT decomposition on Tesla V100 GPU and two 10-core CPUs.

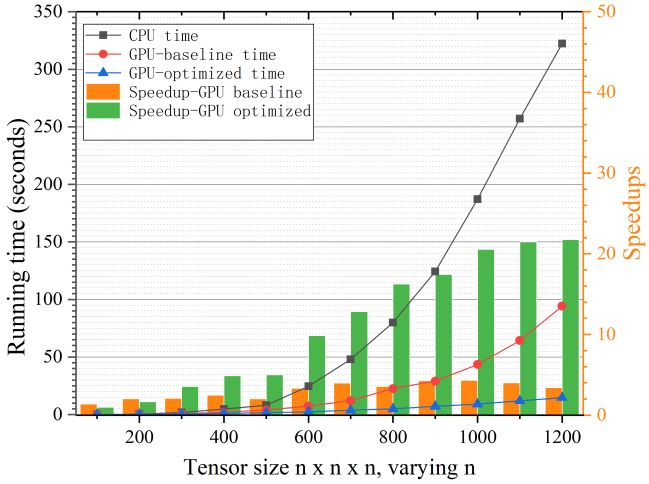

Figure 5: Running time and speedups of third-order TR decomposition on Tesla V100 GPU and two 10-core CPUs.

Fig. 5 shows the running time and speedups of the third-order TR decomposition on GPU and CPU with different tensor sizes and $r_1 = 2$. The CPU implementation used MATLAB code from [6]. Compared to the CPU implementation, the GPU baseline achieves an average of 3.07× and up to 4.31× speedups and the optimized GPU implementation achieves an average 11.35× and up to 21.77× speedups. The RSE reaches the 10e-4 level on GPU and CPU. When the tensor size is $100 \times 100 \times 100$, the speedups is less than one because of the iterations cost and data transfer. The speedups keep increasing with the growing of tensor size.

5 Conclusions

In this paper, we have proposed high performance GPU implementations for third-order TT and TR decompositions. We propose an efficient memory access in GPU memory and an optimized parallel diagonal matrix and matrix multiplication on GPU. We utilize the highly-parallel Jacobi-based SVD to match the SIMT GPU architectures. Through the experiments, we obtain up to 6.67× and 6.36× speedups for third-order TT and TR decompositions, respectively. Our future work will focus on higher-order TT and TR decompositions on multiple GPUs and we will incorporate TT and TR decompositions into the cuTensor library [9].

References

[1] Yuanzhe Cai, Miao Zhang, Dijun Luo, Chris Ding, and Sharma Chakravarthy. Low-order tensor decompositions for social tagging recommendation. In *Proceedings of the fourth ACM International Conference on Web Search and Data Mining*, pages 695–704, 2011.

[2] 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*, 32(2):145–163, 2015.

[3] Xiaochen Han, Bo Wu, Zheng Shou, Xiao-Yang Liu, Yimeng Zhang, and Linghe Kong. Tensor FISTA-Net for real-time snapshot compressive imaging. In *AAAI*, pages 10933–10940, 2020.

[4] Lili Huang, Xuan Wu, Wenze Shao, Hongyi Liu, Zhihui Wei, and Liang Xiao. Color demosaicking via nonlocal tensor representation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1812–1816. IEEE, 2017.

[5] Jiawei Ma, Xiao-Yang Liu, Zheng Shou, and Xin Yuan. Deep tensor ADMM-Net for snapshot compressive imaging. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10223–10232, 2019.

[6] Oscar Mickelin and Sertac Karaman. On algorithms for and computing with the tensor ring decomposition, 2018, arXiv:1807.02513.

[7] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[8] Chase Roberts, Ashley Milsted, Martin Ganahl, Adam Zalcman, Bruce Fontaine, Yijian Zou, Jack Hidary, Guifre Vidal, and Stefan Leichenauer. Tensornetwork: A library for physics and machine learning. *arXiv preprint arXiv:1905.01330*, 2019.

[9] Tao Zhang, Xiao-Yang Liu, Xiaodong Wang, and Anwar Walid. cuTensor-Tubal: Efficient primitives for tubal-rank tensor learning operations on gpus. *IEEE Trans. Parallel Distrib. Syst.*, 31(3):595–610, 2020.

[10] Yimeng Zhang, Xiao-Yang Liu, Bo Wu, and Anwar Walid. Video synthesis via transform-based tensor neural networks. In *ACM Multimedia*, 2020.

[11] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535*, 2016.