High Performance Third-order Tensor-Train and Tensor-Ring Decompositions on GPUs

Hao Hong [1], Weiqin Tong [1], Tao Zhang [1], Xiaoyang Liu [2]

[1] *Shanghai University, No.99 Shangda Road BaoShan District, Shanghai, 200444, China*
[2] *Columbia University, 116th St & Broadway, New York, 10027, USA*

Abstract
Tensor decomposition is an essential tool for analyzing data in many fields, such as sociology, financial encryption and signal processing. According to the “Curse of Dimensionality,” the time and the space cost of the tensor decomposition increase quickly with the tensor size. The high-performance GPU-based tensor-train (TT) and tensor-ring (TR) decompositions implementations are proposed in this paper. Firstly, we utilize the high-parallel Jacobi-based singular value decomposition (SVD) for replacing the traditional SVD to match the GPU structure. Secondly, we design a high-performance matrix multiplication on GPU. Thirdly, by observing data storage, we propose optimized memory access to reduce the memory footprint. Moreover, we conducted experiments to verify the performance of our algorithm on a V100 GPU. Our optimized GPU-based TT and TR decomposition implementations get maximum of 6.67× and 6.36× speedups over the basic implementations.

Keywords
TT decomposition, TR decomposition, GPU, Jacobi SVD

1. Introduction

Tensor decomposition is an extension of matrix decomposition in higher dimensions. It is an essential tool in social relation prediction [1], financial encryption [2], and image processing [3]. Where tensor-train (TT) and tensor-ring (TR) decomposition have been widely used in signal processing [4], computer vision [5], and data mining [6]. According to the “Curse of Dimensionality,” tensor decomposition’s time and space cost increase quickly with the size and dimension of the tensor. It is a critical mission to develop high-performance tensor decompositions. Currently, CPU-based TT and TR decompositions [9, 10] do not take full advantage of the algorithms’ parallelism, making it difficult to process large amounts of data.

This paper utilizes GPUs to achieve high-performance TT and TR decomposition algorithms. Moreover, we conducted experiments comparing existing CPU algorithms with our optimized GPU algorithms, showing that our optimized TT and TR decomposition implemen-tations on GPUs are efficient.

There are three major contributions to this paper:

- This paper proposes high-performance GPU-based third-order TT and TR decom-positions with the same accuracy as CPUs.
- This paper proposes efficient memory access of tensors in GPUs, an efficient diago-nal matrix and matrix multiplication, and utilizes the high-parallel Jacobi-based SVD for replacing the traditional SVD. The optimized algorithms reduce memory footprint and tensor matricization.
- This paper conducts experiments to verify the performance of TT and TR decompositions on one Tesla V100 GPU. The optimized TT and TR decomposition implementations get maximum of 6.67× and 6.36× speedups over the GPU basic implementations.

ISCIPT2022@7th International Conference on Computer and Information Processing Technology, August 5-7, 2022, Shenyang, China
EMAIL: honghao@shu.edu.cn (Hao Hong); xl2427@columbia.edu (Weiqin Tong); baxlumen@shu.edu.cn (Tao Zhang); taozhang@shu.edu.cn (Xiao-Yang Liu)

2. Tensor-Train and Tensor-Ring Decompositions

This section describes notations, TT decomposition, and TR decomposition.

2.1 Operations and Notations

This paper utilizes boldface lowercase letters $\mathbf{a} \in \mathbb{R}^{n}$, boldface uppercase letters $\boldsymbol{A} \in \mathbb{R}^{n_1 \times n_2}$, and uppercase calligraphic letters $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ to denote vectors, matrices, and tensors, respectively. Tensor contractions is represented with the $\circ$ symbol.

2.2 Tensor-Train Decomposition

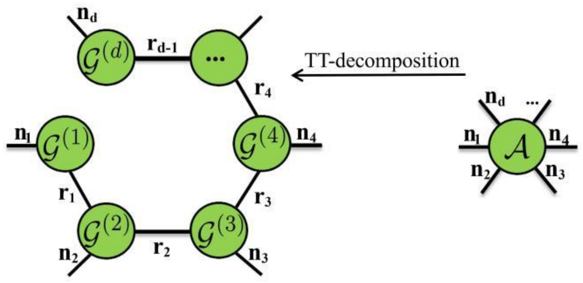

Figure 1: The display diagram of the *d*-th tensor tensor-train decomposition.

Figure 1 shows that the TT decomposition [9] uses three third-order core tensors to express a third-order tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ by tensor contractions:

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

where $\mathcal{G}^{(k)} \in \mathrm{R}^{r_{k-1} \times n_k \times r_k}$ expresses the k-th core tensor. $[r_0, r_1, r_2, r_3]$ expresses the TT-ranks where $r_0 = r_3 = 1$. Therefore, $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(3)}$ are second-order tensors (matrices).

The tensor-train structure is one of the tensor networks and is represented in Figure 1 by the graphical modeling [7]. The connections between two tensors indicate tensor contractions. The original tensor $\mathcal{A}$ is obtained by the tensor contractions of all tensors on the left. The steps of third-order TT decomposition [9] are described in Algorithm 1. In the third and tenth lines of Algorithm 1, we convert a tensor $\mathcal{C}^{(k-1)}$ into a matrix $\mathbf{C}$ with $r_{k-1} n_k$ rows and $\prod_{i=k+1}^{3} n_i$ columns and a matrix $\boldsymbol{U}$ into a tensor $\mathcal{G}^{(k)}$ with n_k columns, r_k in the third direction, and r_{k-1} rows and by reshaping operations, respectively.

2.3 Tensor-Ring Decomposition

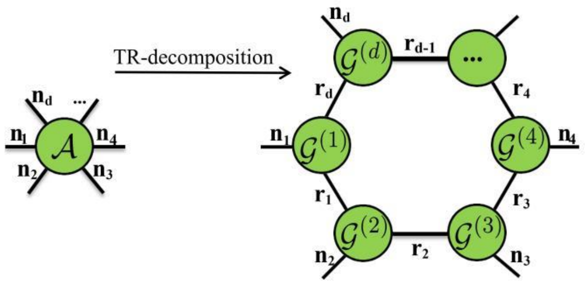

Figure 2: The display diagram of the *d*-th tensor tensor-ring decomposition.

Figure 2 shows that the TR decomposition [10] uses three third-order core tensors $\mathcal{G}^{(k)} \in \mathbb{R}^{r_k \times n_k \times r_{k+1}}, k = 1,2,3$ to express a third-order tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ by tensor contractions:

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

where tensor contractions are calculated between tensors and also between $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(3)}$. $[\mathrm{r}_0, \mathrm{r}_1, \mathrm{r}_2, \mathrm{r}_3]$ expresses the TR-ranks. Because of the ring structure of TR tensors, $\mathrm{r}_0 = \mathrm{r}_3$ do not need to be forced to equal 1, which is used to distinguish between TR and TT structures. TR structure is another special case of tensor networks and steps of TR decomposition [10] are described in Algorithm 2.

3. High-performance Third-order Tensor-Train and Tensor-Ring Decompositions on GPUs

3.1 Parallelization Schemes

In this section, we propose three parallel optimizations for TT decomposition in Algorithm 1 and TR decomposition in Algorithm 2.

Jacobi SVD in Parallel

In TT and TR decompositions, the matrix SVD operations take up the most time, reaching 67%. The traditional SVD operation is not matched to GPU structure because of its low parallelism characteristics. As a substitute, Jacobi SVD [8] is adopted to match the GPU's high-parallelism feature. Jacobi SVD needs an iteration number and an accuracy to determine when the algorithm terminates. Under the single precision of data and calculation, this paper set the maximum iteration to 100 and the accuracy to 10e-8 for getting the minimum error in experiments.

Algorithm 1 Third-order Tensor-Train Decomposition[9]

```
Input: Tensor $\mathcal{A} \in R^{n_1 \times n_2 \times n_3}$, pre-specified accuracy $\varepsilon$.
Output: Cores $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \mathcal{G}^{(3)}$,  $r_0, r_1, r_2, r_3 \in N^+$.
 1: $\mathcal{C}^{(0)} = \mathcal{A}$, $r_0 = r_1 = r_2 = r_3 = 1$,
 2: for $k = 1$ to $2$ do
 3:     $C = \mathrm{reshape}(\mathcal{C}^{(k-1)}, [r_{k-1} n_k, \prod_{i=k+1}^{3} n_i])$,
 4:     Compute SVD: $C = USV^T$, and $s = \mathrm{diag}(S)$,
 5:     $\delta = \frac{\varepsilon}{\sqrt{3-1}} \|s\|_2,\ \gamma = 0,\ r_k = \sharp(s)$,
 6:     while $\gamma \le \delta$ do
 7:         $\gamma = \gamma + s_{r_k}^2,\ r_k = r_k - 1$,
 8:     end while
 9:     $r_k = r_k + 1,\ U = U(:, 1 : r_k)$,
        $S = S(1 : r_k, 1 : r_k),\ V^T = V^T(1 : r_k, :)$,
10:     $\mathcal{G}^{(k)} = \mathrm{reshape}(U, [r_{k-1}, n_k, r_k])$,
11:     $\mathcal{C}^{(k)} = \mathrm{reshape}(SV^T, [r_k, \prod_{i=k+1}^{3} n_i, r_{k+1}])$,
12: end for
13: $\mathcal{G}^{(3)} = \mathcal{C}^{(2)}$. =0
```

The algorithm stops when the number of iterations reaches the maximum number of iterations or the error between the repaired matrix and the original matrix reaches a preset threshold.

Diagonal Matrix and Matrix Multiplication in Parallel

Diagonal matrix and matrix multiplication is the operation with the second longest time occupation in the eleventh line of Algorithm 1 and the eleventh and fifth lines of Algorithm 2. The time cost of these operations increases rapidly with the dimension size of data. We find that the traditional processes introduce redundant calcula-tions because values exist only on the diagonal. Therefore, we accelerate the computation using the following parallel computation method:

$$\boldsymbol{S}\boldsymbol{V}^{\mathrm{T}} = \mathrm{parallel}\left(\mathrm{s}_{\mathrm{k}} \cdot \boldsymbol{V}_{\mathrm{k}}^{\mathrm{T}}\right), \tag{3}$$

where s_{k} and $\boldsymbol{V}_{\mathrm{k}}^{\mathrm{T}}$ represent the *k*-th value and row of matrix $\boldsymbol{S}$ on the diagonal and matrix $\boldsymbol{V}^{\mathrm{T}}$. This parallel computation method takes advantage of parallelism and reduces redundant computations.

Element-wise Product in Parallel

The sixth line to the ninth line of Algorithm 1 and the fifth line to the seventh line of Algorithm 2 are the element-wise products which can be calculated in parallel. We utilize the following parallel method to perform the element-wise product $s \cdot s$, with $m = \#(s)$:

$$\text{parallel}\left(s^{(0)}_{m-k+1} = s_k \cdot s_k\right), 1 \leq k \leq m. \tag{4}$$

Algorithm 2 Third-order Tensor-Ring Decomposition[10]

```
Input: Tensor $\mathcal{A} \in R^{n_1 \times n_2 \times n_3}$, pre-specified accuracy $\varepsilon$.
Output: Cores $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \mathcal{G}^{(3)}$, $r_0, r_1, r_2, r_3 \in N^+$.
 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,
    $C = \text{reshape}(\mathcal{C}^{(0)}, [r_0 n_1, n_2 n_3])$,
 3: $C = USV^T$, $s = \text{diag}(S)$,
 4: $\delta = \frac{\varepsilon}{\sqrt{3-1}}\|s\|_2$, $\gamma = 0$, $m = \sharp(s)$,
 5: while $\gamma \leq \delta$ do
 6:     $\gamma = \gamma + s_m^2, m = m - 1$,
 7: end while
 8: $m = m + 1$, $U = U(:, 1:m)$,
    $S = S(1:m, 1:m)$, $V^T = V^T(1:m, :)$,
 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: $\mathcal{G}^{(1)} = \text{permute}(\text{reshape}(U, [n_1, r_0, r_1]), [2, 1, 3])$,
11: $\mathcal{C} = \text{permute}(\text{reshape}(SV^T, [r_0, r_1, n_2 n_3]), [2, 3, 1])$,
12: $C = \text{reshape}(\mathcal{C}, [r_1 n_2, n_3 r_0])$,
13: Repeat Line 3 to 8, and set $r_2 = m, r_3 = r_0$
14: $\mathcal{G}^{(2)} = \text{reshape}(U, [r_1, n_2, r_2])$,
15: $\mathcal{G}^{(3)} = \text{reshape}(SV^T, [r_2, n_3, r_3])$. =0
```

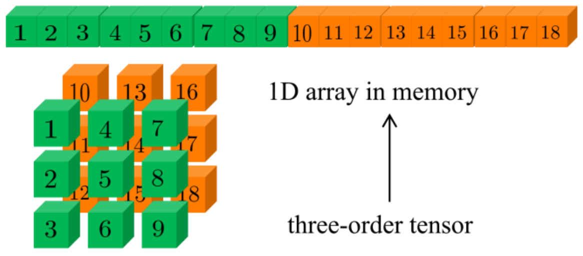

Figure 3: A schematic diagram of the tensor's layout in memory.

3.2 Optimized Memory Access

Figure 3 exhibits a third-order tensor' column-major layout in memory. The tensor data is stored as a front slice of the column master. We adopt the column-major layout in memory to meet the data reading requirements of two libraries: cuSOLVER and cuBLAS. In addition, this kind of memory access method can directly get the mode-1 unfolding of the tensor without the tensor matricization reducing the overhead.

To reduce the memory footprint and the overhead of truncation operations in Algorithm 1 and Algorithm 2, the front truncation sub-sections of matrix $\boldsymbol{V}^{\mathrm{T}}$ and vector s are calculated directly in the eleventh line of Algorithm 1 and the eleventh and fifth lines of Algorithm 2. We utilize the direct conversion in memory to reduce the overhead of tensor permuting operations in the tenth and eleventh lines of Algorithm 2. Moreover, through this optimized memory access method, the reshape operations are eliminated in Algorithms. These algorithms generate a lot of intermediate variables, which introduces much memory footprint and time overhead. Therefore, we reuse the allocated memory and dynamically delete and allocate intermediate variables in GPU memory to reduce memory consumption.

3.3 Efficient Data Transfer

The input and output data volumes of TT and TR decompositions increase quickly with tensor sizes, which results in high time consumption of data transfer between CPUs and GPUs. To reduce the overhead of access space in transmission, the cores $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ are combineded into an array c. Meanwhile, $[n_1, n_2, n_3]$ are used to store dimensions of the input tensor and $[\mathrm{r}_0, \mathrm{r}_1, \mathrm{r}_2, \mathrm{r}_3]$ are used to store TR-ranks or TT-ranks. Therefore, k-th core is acquired through $\mathcal{G}_k = \mathrm{reshape}\left(\mathbf{c}\left(\prod_{j=1}^{k-1} \mathrm{r_j n_j r_{j+1}}, \prod_{j=1}^{k} \mathrm{r_j n_j r_{j+1}}\right), [\mathrm{r_{k-1}}, \mathrm{n_k}, \mathrm{r_k}]\right)$, $\mathbf{c}\left(\prod_{j=1}^{k-1} \mathrm{r_j n_j r_{j+1}}, \prod_{j=1}^{k} \mathrm{r_j n_j r_{j+1}}\right)$ denotes the elements from $\prod_{j=1}^{k-1} \mathrm{r_j n_j r_{j+1}}$ to $\prod_{j=1}^{k} \mathrm{r_j n_j r_{j+1}}$.

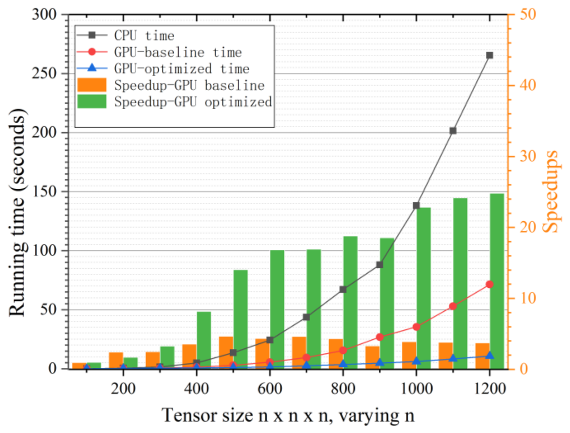

Figure 4: Speedups and running time of third-order TT decomposition on two Intel CPUs and a Tesla V100 GPU.

4. Performance Evaluation

Our experiments run on a server with 80 GB host memory. The server is equipped with two Intel Xeon E5-2640 V4 CPUs. Each CPU has ten cores supporting twenty hardware threads. Moreover, the server is equipped with one Tesla V100 GPU with 32GB device memory and 5,120 CUDA cores @1.53 GHz. We focus on the speedups of our experiment result: speedup = (CPU running time)/(GPU running time). The relative square error (RSE) is utilized to measure the error of data before and after decomposition: RSE $= ||\mathcal{A} - \mathcal{G}^{(1)} \circ \mathcal{G}^{(2)} \circ \mathcal{G}^{(3)}||_{\mathrm{F}} / ||\mathcal{A}||_{\mathrm{F}}$. The experiment tensor data are obtained by tensor contractions of three small tensors. For the Jacobi SVD, under single precision, the accuracy ε is set to10e-8, and the max iteration time is set to 100.

The speedups and running time of our optimized third-order TT decomposition are exhibited in Figure 4. The tensor sizes vary from $100 \times 100 \times 100$ to $1{,}200 \times 1{,}200 \times 1{,}200$. The CPU implemen-tation is referred from MATLAB code [9]. Because of the GPU memory size, the maximum tensor size that can be processed is 1,200 × 1,200 × 1,200 on Tesla V100 GPU. Compared with the CPU implementations, the optimized GPU implement-tation obtains 14.25× on average and up to 24.80× speedups, which are higher than the GPU baseline implementation. The RSE of CPU and GPU are on the 10e-4 level. In our experiment, the speedups of the optimized implementations have a general upward trend.

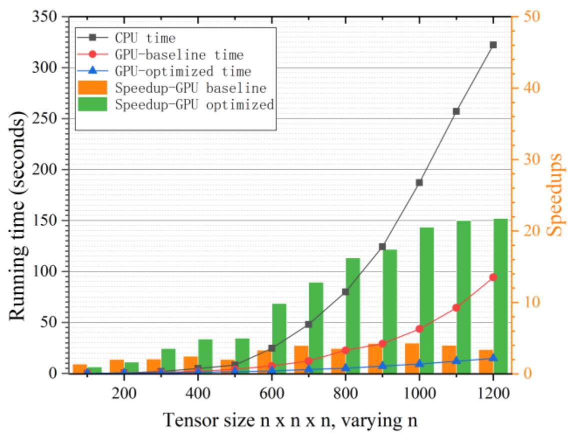

Figure 5: Speedups and running time of third-order TR decomposition on two Intel CPUs and a Tesla V100 GPU.

The speedups and running time of our optimized third-order TR decomposition are exhibited in Figure 5. The CPU implementation is referred from MATLAB code [11]. Compared with the CPU implementations, our optimized GPU implementation achieves 11.35× on average and up to 21.77× speedups, which are higher than GPU baseline implementations. The RSE of CPU and GPU are also on the 10e-4 level. Because of the overhead of iteration and data transfer, the speedup is less than one when the size of tensor is $100 \times 100 \times 100$. The speedups of the optimized TR decomposition keep increasing with the size of tensor.

5. Conclusions

High-performance third-order tensor-train and tensor-ring decomposition implementations on GPUs are proposed in this paper. To improve the efficiency of the algorithms, three optimization strategies are proposed. First, efficient memory access is proposed to reduce the memory footprint. Second, parallelization strategies are widely adopted in algorithms to match GPUs. Third, we use the high-parallel Jacobi SVD to reduce time for critical calculations.

Moreover, we experimentally verify the advantages of our optimized decomposition algorithms. The third-order TT and TR decompositions get maximum of 6.67× and 6.36× speedups. Implementing multi-GPU implementations of high-order TT and TR decompositions is our future work. Meanwhile, the optimized third-order TT and TR decomposition algorithms will be combined into the cuTensor library [6].

6. References

[1] Rettinger A, and et al. Context-aware tensor decomposition for relation prediction in social networks[J]. Social Network Analysis and Mining, 2012, 2(4): 373-385.
[2] Charlier J, Falk E, and et al. User-device authentication in mobile banking using APHEN for PARATUCK2 tensor decomposition[C]//2018 IEEE International Conference on Data Mining Workshops (ICDMW). IEEE, 2018: 886-894.
[3] Huang L, Wu X, and et al. Color demosaicking via nonlocal tensor representation[C]//2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2017: 1812-1816.
[4] Han X, Wu B, and et al. Tensor FISTA-Net for real-time snapshot compressive imaging[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2020, 34(07): 10933-10940.
[5] Ma J, Liu X Y, and et al. Deep tensor admm-net for snapshot compressive imaging[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019: 10223-10232.

[6] Zhang T, Liu X Y, and et al. cuTensor-Tubal: Efficient primitives for tubal-rank tensor learning operations on GPUs[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(3): 595-610.

[7] Roberts C, Milsted A, and et al. Tensornetwork: A library for physics and machine learning[J]. arXiv preprint arXiv:1905.01330, 2019.

[8] Drmač Z, Veselić K. New fast and accurate Jacobi SVD algorithm. I[J]. SIAM Journal on matrix analysis and applications, 2008, 29(4): 1322-1342.

[9] Oseledets I V. Tensor-train decomposition[J]. SIAM Journal on Scientific Computing, 2011, 33(5): 2295-2317.

[10] Zhao Q, Zhou G, and et al. Tensor ring decomposition[J]. arXiv preprint arXiv:1606.05535, 2016.

[11] Mickelin O, Karaman S. On algorithms for and computing with the tensor ring decomposition[J]. Numerical Linear Algebra with Applications, 2020, 27(3): e2289.