Parallel TTr1-Tensor: Randomized Compression-based Scheme for Tensor Train Rank-1 Decomposition

Zeliang Zhang, Junzhe Zhang, Guoping Lin, Zeyuan Yin, Kun He
Huazhong Uniersity of Science and Technology
{zeliangzhang, junzhezhang, u201812716, zeyuanyin, brooklet60}@hust.edu.cn

Abstract

Tensor train rank-1 (TTr1) decomposition, as a variant of tensor train decomposition, is widely used in quantum machine learning to decompose high-order tensors into low-order tree-like tensor networks. However, due to the storage limitation, tensors can not be fully loaded in the main memory simultaneously. A key step in the optimization of TTr1 is the utilization of Singular Value Decomposition (SVD) for the tensor matricization, and memory limitation becomes the bottleneck of existing TTr1 decomposition. In this work, we propose *Parallel TTr1-Tensor*, a randomized compression-based scheme for tensor train rank-1 decomposition by trading computation for storage. In addition, we maximize the parallelism of tensor operations to accelerate the proposed scheme on GPUs. In experiments, we test the relative error of reconstruction and running time of our scheme on various scales. Our method achieves a maximum of $151.2\times$ speedup on GPUs versus CPUs with less than 20% relative errors. To our knowledge, this is the first work to utilize the compression-based technique to handle the TTr1 decomposition bottleneck due to the limited memory.

1 Introduction

In recent years, quantum machine learning algorithms [1–3], as a bridge for classical machine learning and statistical physics, have been appreciated due to its solid theoretical foundation, advantages of quantum computing acceleration, fewer number of parameters and good interpretability in physics. Tensor decomposition [4] is a powerful technique used on quantum machine learning methods [5, 6] to decompose high-order tensors into low order ones while mining potential patterns and features of the tensors. Tensor train rank-1 (TTr1) [10] decomposition is one of the basic tensor decomposition methods applied to tensor networks [8, 9].

Basic TTr1 decomposition method needs to load the full tensor into the main memory. However, in recent years, due to the fact that the scale of tensors becomes increasingly large, the limited memory becomes the bottleneck of TTr1 decomposition for the task of decomposing large scale tensors.

Randomized technology is a powerful computation acceleration technique which has been proposed and studied for decades [11–13]. Randomized compression-based tensor decomposition has drawn increasing attention in the academy. Many algorithms have been proposed for large scale tensor decomposition, such as PARACOMP [14] for CP decomposition [15] and randomized Tucker decomposition [16, 17]. In terms of TTr1 decomposition, there is a fact to confront that it lacks efficient algorithms for large scale tensors and randomized technique has not been applied to TTr1 decomposition. In this work, we explore the effectiveness of compression-based technique and propose a *Parallel TTr1-Tensor* method to achieve a randomized TTr1 decomposition.

First Workshop on Quantum Tensor Networks in Machine Learning, 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

Our main contributions are listed as follows: (1) By exploiting the compression-based technique, we propose two compression-based strategies, Gaussian Compression and High-order Sketch Compression, to achieve randomized tensor train rank-1 decomposition. (2) We make fully use of the parallelism for tensor operations, design and achieve an efficient randomized TT-Tensor scheme on GPUs. (3) We exploit the block strategy to compress large scale tensors and achieve the large scale TTr1 decomposition of which the maximum scale is up to $5000 \times 5000 \times 5000$.

2 Tensor Decomposition and Compression-based Methods

2.1 Notations

we use uppercase calligraphic letters to denote third-order tensors, e.g., $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and uppercase boldface letters to denote matrices, e.g., $A \in \mathbb{R}^{I \times J}$. We use $\odot$ for Khatri-Rao product, $\circ$ for Kruskal product, $*$ for Hadmard product which is also named the element-wise product, $\otimes$ for Kronecker product, $||\cdot||_F$ for frobenius norm, and $\times_n$ for the mode n contraction between two tensors.

2.2 Tensor Train Rank-1 Decomposition

Tensor train rank-1 decomposition factorizes a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, into a series of orthogonal factors that can be written as $\mathcal{X} = \sum_{i=1}^{I} \sum_{j=1}^{J} \sigma_i \sigma_{i,j} \mathbf{a}_i \circ \mathbf{b}_{i,j} \circ \mathbf{c}_{i,j}$, where $\mathbf{a}_i \in \mathbb{R}^I$, $\mathbf{b}_{i,j} \in \mathbb{R}^J$ and $\mathbf{c}_{i,j} \in \mathbb{R}^K$. TTr1 decomposition uses sequential SVD method (*TTr1-SVD*) to obtain the factorization factors. Also, for simplification, we rewritten the above equation as $\mathcal{X} = \sum_{n=1}^{IJ} \sigma_n \mathbf{a}_n \circ \mathbf{b}_n \circ \mathbf{c}_n$.

For TTr1-SVD algorithm, consider a tensor $\mathcal{X} \in \mathbb{R}^{2 \times 3 \times 4}$, the tensor is first be unfolded to mode-1 matricization $\boldsymbol{X}_{(1)}$. Using SVD algorithm, it can be written as a sum of rank-1 terms $\boldsymbol{X}_{(1)} = \sum_{i=1}^{2} \sigma_i \mathbf{u}_i \circ \mathbf{v}_i$ where $\mathbf{u}_i \in \mathbb{R}^{2 \times 1}$ and $\mathbf{v}_i \in \mathbb{R}^{3 \times 1}$. Then each $\mathbf{v}_i$ will be reshaped to V_i and using SVD, it can be rewritten as $V_i = \sum_{j=1}^{3} \sigma_{i,j} \mathbf{u}_{i,j} \circ \mathbf{v}_{i,j}$, $i = 1, 2$. Combining the above steps, we can rewritten the tensor as $\mathcal{X} = \sum_{i=1}^{2} \sum_{j=1}^{3} \sigma_i \sigma_{i,j} \mathbf{u}_i \circ \mathbf{u}_{i,j} \circ \mathbf{v}_{i,j}$. For further simplification, we can rewrite it as $\mathcal{X} = \sum_{n=1}^{6} \sigma_n \mathbf{a}_n \circ \mathbf{b}_n \circ \mathbf{c}_n$.

2.3 High-order Count Sketch Method

High-order Count Sketch (HCS) [18] is a novel dimensionality reduction method that preserves the relevant information in high order data like tensors, through multiple smaller hash functions to sketch the high-order tensors for reducing the size. Furthermore, HCS can be used for approximate computation of various tensor operations like tensor contraction and Khatri-Rao product.

3 The Proposed Parallel TTr1-Tensor Scheme

3.1 Scheme Overview

The *Parallel TTr1-Tensor* scheme mainly consists of three stages, namely compression stage, factorization stage and recovery stage. As illustrated in Figure 1 and Algorithm 1, a tensor $\mathcal{X} \in \mathbb{R}^{100 \times 100 \times 100}$ is first compressed into a set of compression replicas $\mathcal{Y}_p \in \mathbb{R}^{10 \times 10 \times 10}$, $p = 1, 2, 3$. Techniques applied to the compression stage consist of the random Gaussian mode matrix compression, or the random hash-based High-order Sketch method. Then, in the factorization stage it factorizes all the compression replicas independently to obtain the tensor train rank-1 factors $\hat{\sigma}_{p,i} \in \mathbb{R}, \hat{\sigma}_{p,i,j} \in \mathbb{R}, \hat{\mathbf{a}}_{p,i} \in \mathbb{R}^{10}, \hat{\mathbf{b}}_{p,i,j} \in \mathbb{R}^{10}$ and $\hat{\mathbf{c}}_{p,i,j} \in \mathbb{R}^{10}$, $p = 1, 2, 3$, $i = 1, 2, 3$, $j = 1, 2, 3$, in the compression space. Finally, the corresponding recovery method is utilized

Algorithm 1 The proposed Parallel TTr1-Tensor scheme

```
Require: tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$; dimensions of compression replicas $L, M, N$; number of replicas $P$
Ensure: mode matrices $(A, B, C)$
1: $\mathcal{Y}_p \leftarrow$ compress $\mathcal{X}$ according to (1) or (2), for $p = 1, .., P$
2: for $p = 1$ to $P$ do
3:     $(A_p, B_p, C_p) \leftarrow$ the TTr1SVD decomposition of $\mathcal{Y}_p$
4:     $(A_p, B_p, C_p) \leftarrow$ the decompression of $(A_p, B_p, C_p)$ according to (4) or (5)
5: end for
6: $(A, B, C) \leftarrow$ recovery using $(A_p, B_p, C_p)$, $p = 1, 2, ..., P$
7: return $(A, B, C)$
```

to recover the estimation of the tensor train rank-1 factors in the original space. A least square problem will be solved for the random Gaussian mode matrices compression method while the reverse transformation and averaging all the respective estimations are needed for the High-order Sketch method. After the recovery stage, we obtain the estimation of tensor train rank-1 factors $\sigma_i \in \mathbb{R}, \sigma_{i,j} \in \mathbb{R}, \mathbf{a}_i \in \mathbb{R}^{100}, \mathbf{b}_{i,j} \in \mathbb{R}^{100}$ and $\mathbf{c}_{i,j} \in \mathbb{R}^{100}$, $i = 1, 2, 3$, $j = 1, 2, 3$, in the original space.

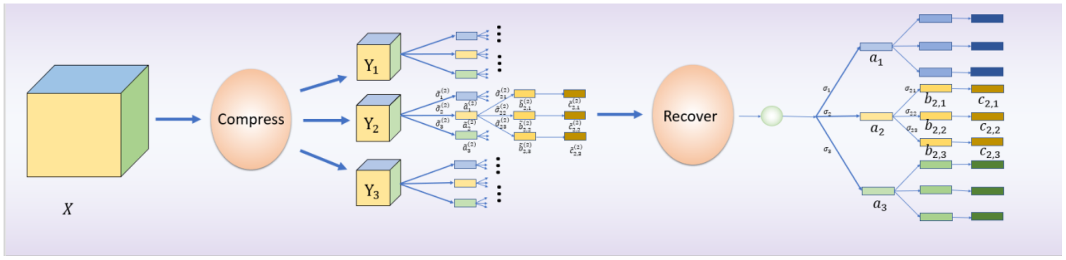

Figure 1: An example for our parallel TTr1-Tensor scheme. A tensor $\mathcal{X} \in \mathbb{R}^{100 \times 100 \times 100}$ is first compressed into 3 replicas, then each replica is factorized to a tree-like structure. The recovery stage is executed to recover the factorization factors.

3.2 Compression Stage

In our *Parallel TTr1-Tensor* scheme, we propose two methods, random Gaussian mode matrix compression and random hash-based High-order sketch, to compress the original tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ into a series of replicas $\mathcal{Y}_p \in \mathbb{R}^{L \times M \times N}$, $p = 1, 2, ..., P$, of which the size along each mode is reduced.

3.2.1 Gaussian Compression

By exploiting the Gaussian compression method (*GCS*), the scheme compresses the tensor $\mathcal{X}$ into a relatively small tensor $\mathcal{Y}$, the (i, j, k)-th element of $\mathcal{Y}$ can be written in the scalar form:

$$\mathcal{Y}_{lmn} = \mathit{GCS}(\mathcal{X})_{lmn} = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} U_{il} V_{jm} W_{kn} \mathcal{X}_{ijk}, \tag{1}$$

where $U \in \mathbb{R}^{I \times L}$, $V \in \mathbb{R}^{J \times M}$, $W \in \mathbb{R}^{K \times N}$ are drawn from the identical independent Gaussian distribution.

3.2.2 High-order Sketch Compression

Besides the Gaussian compression method, we also utilize the High-order Sketch Compression method (*HCS*) in our scheme to compress the tensor $\mathcal{X}$ into a relatively small tensor $\mathcal{Y}$, the (i, j, k)-th element of $\mathcal{Y}$ can be written in a scalar form:

$$\mathcal{Y}_{lmn} = \mathit{HCS}(\mathcal{X})_{lmn} = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} \delta_1(i, l) \delta_2(j, m) \delta_3(k, n) \mathbf{s}_{1,l} \mathbf{s}_{2,m} \mathbf{s}_{3,n} \mathcal{X}_{ijk}, \tag{2}$$

where δ_1 is a function of two variables $\delta_1(i,l) = 1$ if $\mathbf{h}_1(i) = l$ else 0, $\mathbf{s}_1 \in \mathbb{R}^I$ is a random sign function which randomly maps from $[1,2,...,I]$ to $[\pm 1]$, $\mathbf{h}_1 \in \mathbb{R}^I$ is a random hash function which randomly maps from $[1,2,..,I]$ to $[1,2,...,L]$, and likewise for δ_2, δ_3, $\mathbf{s}_2 \in \mathbb{R}^J$, $\mathbf{s}_3 \in \mathbb{R}^K$, $\mathbf{h}_2 \in \mathbb{R}^J$ and $\mathbf{h}_3 \in \mathbb{R}^K$.

The above compression methods will be executed P times to generate a series of independent compression replicas $\mathcal{Y}_p, p = 1,2,..,P$. There exists a trade-off between the compression ratio and the repeated times P. With the size of replicas reduced, P needs to be sufficiently large to guarantee a tolerant variance between the unbiased estimation and the true value.

3.3 Factorization Stage

We modified the original TTr1-SVD algorithm to accelerate the factorization stage. Compared with the vanilla TTr1-SVD that selects all the eigenvectors at both SVD stages to construct the factorization vectors, the modified algorithm just selects the first R_1 left singular vectors for the first SVD stage, and the first R_2 left singular vectors for the second SVD stage, which reduces the number of branches of factorization tree from $L \times min(M,N)$ to R_1R_2, where R_1 and R_2 are error-based self adaptive in the process and (L,M,N) are the size of tensor to be decomposed. $A(:,i)$, columns of B_i and C_i are respectively composed of TTr1 vectors $(\mathbf{a}_j, \mathbf{b}_j, \mathbf{c}_j)$, $j = 1,2,..,R_2$. Formally, given a tolerance of reconstruction error of matrix for SVD stages, we compute $X_{(1)} = A\Sigma D$, where $A \in R^{I\times R_1}$, then reshape $D(i,:) \in \mathbb{R}^{MN}$ to $D_i \in R^{M\times N}$, $i = 1,2,...R_1$, and we have $D_i = B_i\Sigma_i' C_i^T$, where $B_i \in \mathbb{R}^{M\times R_2}$ and $C_i \in \mathbb{R}^{N\times R_2}$. $\hat{\sigma}_i$ and $\hat{\sigma}_{i,j}$ can be obtained from Σ and Σ_i' and we have $\hat{\sigma}_n$=$\hat{\sigma}_{1,i,j} \times \hat{\sigma}_{i,j}$ when $n = i \times R_2 + j$ and $\hat{\sigma}_{1,i,j} = \hat{\sigma}_i$.

After the compression stage, we obtain P compression replicas $\mathcal{Y}_p$, $p = 1,2,3...,P$, with the Gaussian compression or High-order Sketch method. At the factorization stage, each compression replica $\mathcal{Y}_p$ will be factored independently into $(\hat{\sigma}_{p,n}, \hat{\mathbf{a}}_{p,n}, \hat{\mathbf{b}}_{p,n}, \hat{\mathbf{c}}_{p,n})$, $n = 1,2,..,R_1R_2$ by our modified TTr1-svd, where $\hat{\sigma_{p,n}}$ is a scalar and $\hat{\mathbf{a}}_{p,n} \in \mathbb{R}^L$, $\hat{\mathbf{b}}_{p,n} \in \mathbb{R}^M$, $\hat{\mathbf{c}}_{p,n} \in \mathbb{R}^N$ are the tensor train rank-1 decomposition orthogonal factors of the compression replica $\mathcal{Y}_p$ along each mode. We first rearrange the order of the factors $(\sigma_{p,n}, \mathbf{a}_{p,n}, \mathbf{b}_{p,n}, \mathbf{c}_{p,n})$ by the magnitude of $\sigma_{p,n}$, $n = 1,2,..,N$, and multiply $\hat{\sigma}_{p,n}$ with $\hat{a}_{p,n}$, $p = 1,2,...,P$. Then we merge the orthogonal factors along the same mode to $(A_p$,B_p,$C_p)$, where $A_p \in \mathbb{R}^{L\times R_1R_2}$ is the column concatenation and repeat along each branch in the factorization tree of $\hat{a}_{p,n}$, $n = 1,2,..,R_1$, and likewise for $B_p \in \mathbb{R}^{M\times R_1R_2}$ and $C_p \in \mathbb{R}^{N\times R_1R_2}$.

3.4 Recovery Stage

We obtain the tensor train rank-1 orthogonal factors of the replicas after the factorization stage. Here, we sufficiently utilize the property of the Kronecker product [19] and the uniqueness of the tensor train rank-1 decomposition to respectively recover the factors from the compression or sketch space to the original space.

3.4.1 Recovery for Gaussian Compression

Recall the tensor train rank-1 decomposition of $\mathcal{Y}_p$ is the sum of the outer product of the orthogonal factors along each mode, i.e. $\mathcal{Y}_p = \sum_{n=1}^{R_1R_2} \hat{\sigma}_{p,n}(\hat{\mathbf{a}}_{p,n} \odot \hat{\mathbf{b}}_{p,n} \odot \hat{\mathbf{c}}_{p,n})$. The Gaussian compression method actually multiplies the random matrices along each mode to reduce the scale. From the property of the Kronecker product and the uniqueness of tensor train rank-1 decomposition, we have

$$\mathcal{Y}_p = (U_p^T \otimes V_p^T \otimes W_p^T)(\sum_{n=1}^{R_1R_2} \sigma_n(\mathbf{a}_n \odot \mathbf{b}_n \odot \mathbf{c}_n)) = \sum_{n=1}^{R_1R_2} \sigma_n((U_p^T\mathbf{a}_n) \odot (V_p^T\mathbf{b}_n) \odot (W_p^T\mathbf{c}_n)), \quad (3)$$

where (U_p, V_p, W_p) are the random mode matrices exploited in the Gaussian compression method, and $(\sigma_n, \mathbf{a}_n, \mathbf{b}_n, \mathbf{c}_n)$ are the tensor train rank-1 decomposition orthogonal factors. Then, we have $\hat{\mathbf{a}}_{p,n} = U_p^T\mathbf{a}_n$ while $\hat{\mathbf{a}}_{p,n}$ is the factorization mode factors obtained from the compression replicas and likewise for $\hat{\mathbf{b}}_{p,n}$ and $\hat{\mathbf{c}}_{p,n}$, $n = 1,2,...,R_1R_2$. Recall that we have concatenated the orthogonal factors with the same mode to (A_p, B_p, C_p), we can rewrite the above equation as $A_p = U^TA$, and likewise for B_p and C_p, $p = 1,2,...,P$.

To recover for Gaussian compression method, we solve a least square problem

$$\begin{bmatrix} A_1 \\ \vdots \\ A_P \end{bmatrix} = \begin{bmatrix} U_1^T \\ \vdots \\ U_P^T \end{bmatrix} A \tag{4}$$

to obtain A and likewise for B and C.

3.4.2 Recovery for High-order Sketch Compression

We rewrite Eq.(2) in the following tensor form:

$$\mathcal{Y} = HCS(X) = (\mathcal{S} * \mathcal{X}) \times_1 H_1 \times_2 H_2 \times_3 H_3, \tag{5}$$

where $\mathcal{S} = \mathbf{s}_1 \odot \mathbf{s}_2 \odot \mathbf{s}_3$ and $H_m(i,j) = 1$ if and only if $\mathbf{h}_m(i) = j$ else 0, $m = 1,2,3$. The recovery for the orthogonal factor is $a_{p,n} = s_1 * (H_1 \hat{\mathbf{a}}_{p,n})$ and likewise for the recovery of $\mathbf{b}_{p,n}$ and $\mathbf{c}_{p,n}$, $p = 1,2,..,P$, $n = 1,2,..,R_1R_2$. To recover for High-order Sketch compression, we correspondingly average the mode matrices (A_p, B_p, C_p), $p = 1,2,...,P$, and obtain (A,B,C).

4 Optimization for GPU

4.1 Handle Large Scale Tensors

Due to the memory limitation, a large tensor can not be loaded and compressed to small replicas directly. To process large tensors, we adopt a partitioning strategy in which we first split the tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ into multiple blocks $\mathcal{B}_n \in \mathbb{R}^{d_1\times d_2\times d_3}$, $n = 1,2,...,\frac{IJK}{d_1d_2d_3}$. Similar to the block matrix multiplication, the full tensor contraction can also be computed via merging the computational results of all blocks.

For Gaussian compression method, we exploit the related components of (U_p, V_p, W_p) to compress each block $\mathcal{B}_n$ to replicas $\mathcal{Y}_p^n$, $p = 1,2,...P$. Finally, they will be accumulated to the final replicas $\mathcal{Y}_p$ respectively, i.e. $\mathcal{Y}_p {+}= \mathcal{Y}_p^n$, $p = 1,2,..,P$. Likewise for High-order sketch method, related random hash and sign matrices will be computed in time to map the block tensor from the original space to the sketch space of reduced sizes.

4.2 Improve the Utilization of GPU

We exploit the popular open-source library *Pytorch* to accelerate the matrix multiplication operation. To improve the utilization of GPUs, we execute the batch matrix multiplications on the compression stage both for Gaussian compression and High-order sketch method. The batch SVD method is executed at the factorization stage to parallelly compute the nodes on the second level and leaves on the third level of the decomposition tree.

4.3 Reduce the Communication Cost

In our *Parallel TTr1-Tensor*, we adopt the strategy that we first split the tensor into multiple blocks and compress every block into the component of replicas. When the size of block is fixed for the limitation of memory, the total number of tensor blocks increases rapidly with the increase of the tensor scale. The data transfer introduces a considerable time consumption. Thus in our optimization scheme, we directly access raw data including the tensor block data from RAM to GPUs on global memory of GPUs, and ensure that there is no intermediate data transferred back to CPUs during the compression stage to minimize the expensive time consumption introduced by the data transfer between CPUs and GPUs.

5 Performance Evaluation

In this section we provide a series of experimental results to evaluate the performance of the proposed method. All experiments are on a server with two Intel(R) Xeon(R) Gold 5218 CPUs and each CPU has 12 cores @2.30GHz supporting 24 hardware threads. There is a TITAN RTX GPU consisting of 24 GB device memory. There are 251 GB DDR4 memories on the server. For High-order sketch compression method, we mainly make a research at the influence on the relative error by the compression ratio and the number of compression replicas. For Gaussian compression, we mainly make a research at the comparison on the relative error and time consumption between the schemes on CPU and GPU. In each case, we generate the mode matrices $A \in \mathbb{R}^{I\times R}$, $B \in \mathbb{R}^{J\times R}$, $C \in \mathbb{R}^{K\times R}$ from an independent normal distribution to generate the tensor $\mathcal{X}^{I\times J\times K}$ where $I = J = K$ ranging from 100 to 1500. The relative error is defined as $\frac{||\mathcal{X}-\hat{\mathcal{X}}||_F}{||\mathcal{X}||_F}$ where $\hat{\mathcal{X}}$ is the reconstruction of $\mathcal{X}$.

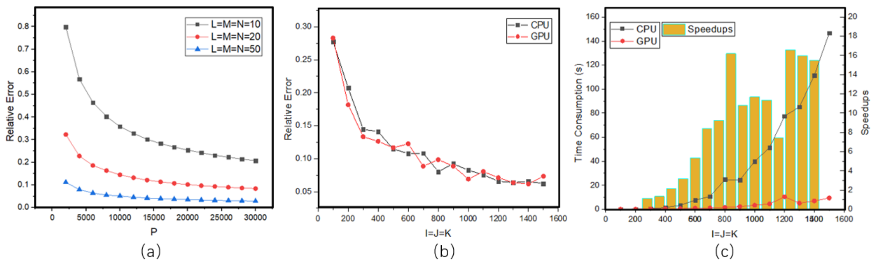

Figure 2: (a) shows the influence on the number of replicas P, and the compression ratio on the relative construction error with High-order Sketch Compression. (b) and (c) respectively show the comparison on the relative construction error and time consumption between the CPU and GPU implementation with Gaussian Compression.

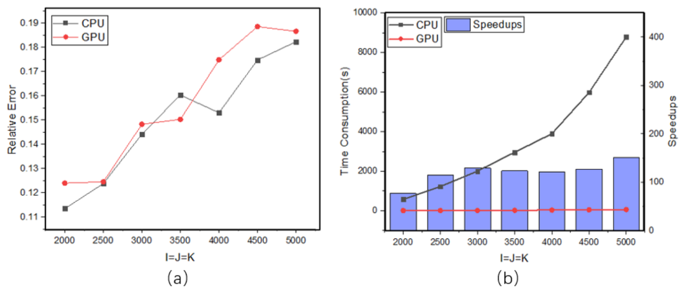

Figure 3: (a) and (b) respectively show the comparison on the relative construction error and time consumption between the CPU and GPU implementation for large scale tensors with Gaussian Compression.

In Figure 2 (a), we can observe that a larger P yields preciser results, but the regularity of the compress ratio is opposite. For example, the relative error drops from 79% to 20% when we keep the compress ratio to 10, where the compress ratio 10 indicates we compress each dimension of the original tensor to 10% in the compressed version, and set $P = 30,000$ instead of 2000. We achieve 3% relative error when the compress ratio is set as 2 after repeating 30,000 times. And the error decays slower when P is large enough. When the compress ratio is 10, the relative error drops from 79% to 28% if we change the repeat time P from 2,000 to 16,000; on contrast, the relative error only drops from 28% to 20% when P rises to 30,000, which also happened for other compress ratios. It can also be implicitly seen that the relative error can be controlled under a tolerant level with a relative small compression rate when P is fixed as a constant.

Figure 2 (b) shows the comparison of the relative error between the CPU version and the GPU version, in which condition we set the compression ratio as a constant value of 10. Obviously the estimation tends to reduce as the scale grows with the scope from $5\% \sim 30\%$. And the relative errors of the CPU version and the CPU version are almost constant. Figure 2(c) shows the comparison of the time consumption between the CPU version and the GPU version. It can be seen that the curves of the time consumption on CPU and GPU both grow along with the scale grow of the tensors. And the CPU version grows exponentially, with a maximum of 146s at the scale of $1500 \times 1500 \times 1500$, while the GPU version grows at a much slower speed, with a maximum of 9.4s at the scale of $1500 \times 1500 \times 1500$, reducing around 138s when compared to the CPU version. The GPU version achieves an average of $8.63\times$ speedup with up to $16.59\times$ speedup.

Figure 3 (a) and (b) show the comparison of the relative error and time consumption between the CPU and GPU implementations when we apply our TTr1-tensor scheme to larger tensors. To trade the computation for storage, we set the size of the replicas as constant $100 \times 100 \times 100$ to reduce the memory cost. All the relative errors increase when the matrix scale increases, but the error is always controlled under 20%. And the GPU implementation achieves a high performance with up to around $151.2\times$ speedup to the CPU implementation. For the reported maximum scale of $5000 \times 5000 \times 5000$, the GPU implementation only needs $58.97s$ while the CPU implementation needs $8788.74s$, achieving $151.2\times$ speedup.

6 Conclusion

In this work, we propose a parallel TTr1-Tensor scheme for tensor train rank-1 decomposition that utilizes the compression-based technique to factorize a tensor into tree-like structures. We provide acceleration techniques for CPU as well as GPU implementations. In experiments, we test various tensor scales and obtain a low reconstruction error. In future work, we will further explore the potential of our scheme to achieve tensor train rank-1 decomposition on matrices in trillion or even billion scale, and we will develop a more efficient parallelism scheme with the help of multi-GPUs.

References

[1] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe Seth Lloyd. Quantum machine learning. *Nature*, Volume 549, pp. 195–202, 2017.

[2] M. Schuld, I. Sinayskiy, F. Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, Volume 56, pp. 172–185, 2015.

[3] Jeremy Adcock, Euan Allen, Matthew Day, Stefan Frick, Janna Hinchliff, Mack Johnson, Sam Morley-Short, Sam Pallister, Alasdair Price, Stasja Stanisic. Advances in quantum machine learning. *Nature*, arXiv:1512.02900.

[4] Tamara G. Kolda, Brett W. Bader. Tensor Decompositions and Applications. *SIAM*, Volume 51, pp. 455–500, 2009.

[5] Yuwang Ji, Qiang Wang. A Survey on Tensor Techniques and Applications in Machine Learning. *IEEE Access*, Volume 7, pp. 162950–162990, 2019.

[6] Thomas Papastergiou, Evangelia I. Zacharaki, Vasileios Megalooikonomou. Tensor Decomposition for Multiple-Instance Classification of High-Order Medical Data. *SIAM*, Volume 2018.

[7] I. V. Oseledets. Tensor-Train Decomposition. *SIAM Journal on Scientific Computing*, Volume 33, pp. 2295–2317, 2011.

[8] Edwin Stoudenmire, David J. Schwab. Supervised Learning with Quantum-Inspired Tensor Networks. *Advances in Neural Information Processing Systems 29*, 2016.

[9] Roman Orus. A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States. *Annals of Physics* Volume 349, pp.117-158, 2014.

[10] Kim Batselier, Haotian Liu, Ngai Wong. A Constructive Algorithm for Decomposing a Tensor into a Finite Sum of Orthonormal Rank-1 Terms. *SIAM Journal on Matrix Analysis and Applications*, Volume 36, pp. 1315–1337, 2015.

[11] Z. Zhang, X.-Y. Liu, P. Zhou. Trillion-Tensor: Trillion-Scale CP Tensor Decomposition. *IJCAI 2020 Workshop on Tensor Network Represenations in Machine Learning*, 2020.

[12] Liu, Xiao-Yang and Zhu, Yanmin and Kong, Linghe and Liu, Cong and Gu, Yu and Vasilakos, Athanasios V and Wu, Min-You. CDC: Compressive data collection for wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, Volume 26, pp. 2188–2197, 2014.

[13] N. Sidiropoulos and A. Kyrillidis. Multi-way compressed sensing for sparse low-rank tensors. *IEEE Signal Processing Letters*, Volume 19, pp. 757–760, 2012.

[14] Nicholas D. Sidiropoulos, Evangelos E. Papalexakis, and Christos Faloutsos. Parallel Randomly Compressed Cubes. *IEEE Signal Processing Magazine* , pp. 57–70, 2014.

[15] J. D. Carroll and J. J. Chang. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika*, Volume 35 , pp. 283–219, 1970.

[16] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, Volume 31, pp. 279–311, 2011.

[17] Salman Ahmadi-Asl, Andrzej Cichocki, Anh Huy Phan, Ivan Oseledets, Stanislav Abukhovich, Toshihisa Tanaka. Randomized Algorithms for Computation of Tucker decomposition and Higher Order SVD (HOSVD). *Numerical Analysis*, arXiv:2001.07124, 2020.

[18] Y. Shi and A. Anandkumar. Higher-order count sketch: Dimensionality reduction that retains efficient tensor operations. *SIAM Journal on Scientific Computing*, arXiv preprint arXiv:1901.11261, 2019.

[19] J. Brewer. Kronecker products and matrix calculus in system theory. *IEEE Trans. Circuits Syst.*, Volume 19, pp. 772–781, 1978.