2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application

Towards Optimized Streaming Tensor Completion on multiple GPUs

Jiwei Hao[1], Hailong Yang[1], Qingxiao Sun[1], Huaitao Zhang[2], Zhongzhi Luan[1], Depei Qian[1]
[1]*School of Computer Science and Engineering*, Beihang University, Beijing, China, 100191
[2]*School of Computer Science and Technology*, Xi'an Jiaotong University, Shaanxi, China, 710049
{jiweihao, hailong.yang, qingxiaosun, 07680, depeiq}@buaa.edu.cn, 2211110532@stu.xjtu.edu.cn

***Abstract*—Tensor completion is a prevailing method for predicting the unobserved or missing data in incomplete tensors. In many real-world scenarios, incomplete tensors can grow in multiple modes with streaming fashion. Such tensors are generally transformed into a dynamically changing tensor sequence known as Multi-Aspect Streaming Tensor (MAST) sequence. However, existing optimization techniques for a single static tensor completion cannot scale well to large streaming tensor sequences with dynamically growing properties. In this paper, we develop an efficient streaming tensor completion library *cuSTC* on multiple graphics processing units (GPUs). *cuSTC* adopts a novel TB-COO storage format with tiling and bit mapping data structure. In addition, *cuSTC* implements a two-level tensor partition scheme to accelerate the prevalent dynamic alternating least squares (ALS) algorithm for MAST sequence. Finally, *cuSTC* designs a warp-level reduction scheme to reduce atomic operations in tensor decomposition. We compare *cuSTC* with SPLATT tensor completion library and achieve 48.69×, 24.22×, 4.54×, 3.58×, 12.26× performance speedup on five real-world datasets.**

***Index Terms*—Streaming Tensor Completion, Multiple GPUs, Performance Optimization**

I. Introduction

Tensors can represent multi-way real-world data and are widely applied in domains including telecommunications, computer vision, and recommendation systems. Among those applications, numerous primitive tensors are being partially observed, and it is essential to utilize observed entries to impute the missing ones, such as image in-painting and preference prediction. The growing demands on the performance of the above applications drive the research on optimization techniques for tensor completion.

Beyond the traditional tensor completion for a single static tensor, due to the prosperity of online information systems, real-world high-dimensional data often changes dynamically, denoted as the tensor sequence. Tensor sequences can be divided into two categories depending on whether there are dependencies among the tensors in sequence. This paper focuses on streaming tensor completion for Multi-Aspect Streaming Tensor (MAST) sequence [1], where the tensors in sequence grow in multiple modes with dependency. Furthermore, since the tensor slices in MAST sequence share common data, a new dynamic algorithm can leverage the former completion results to improve the performance without deteriorating accuracy [2].

Due to the increasing demand for computational performance from various fields, research works have been devoted to accelerating decomposition-based tensor completion for both static and dynamic tensors over the past few years. A mode-agnostic sparse tensor bit encoding format for multicore CPUs has been designed by Helal *et al.* [3]. Smith *et al.* [4] extend static tensor decomposition algorithm to many-core processors, such as Intel Knights Landing. Due to the massive parallelism capability, Graphical Processing Units (GPUs) have gained popularity in optimizing numerical algorithms and have become a profitable candidate for improving the performance of MAST completion algorithm, which the existing studies have not explored. However, several challenges need to be addressed to achieve efficient streaming tensor completion for MAST sequence on multiple GPUs. Firstly, unpredictable sparsity patterns in the tensor sequence when updating factor matrices exacerbate cache thrashing and memory bandwidth contention. Secondly, the expensive computational Matricized-Tensor Times Khatri-Rao Product (MTTKRP) for sparse tensors becomes a bottleneck for streaming tensor completion. Finally, a sophisticated partition strategy is needed to mitigate load imbalance when scaling to multiple GPUs.

To address the above challenges, we propose a library *cuSTC* to exploit the performance potential of GPUs. And To our knowledge, this is the first work to establish an efficient streaming tensor completion library on multiple GPUs. Specifically, This paper makes the following contributions:

- We improve the performance of widely applied algorithms in tensor completion for multi-aspect streaming tensor sequence, leveraging the advantage of TB-COO storage format and unique characteristics of GPU.
- We design a partition mechanism for large tensor slices in the tensor sequence on multiple GPUs and implement a warp-level reduction scheme to reduce atomic operations in tensor decomposition, which can effectively improve the performance of streaming tensor completion.
- We develop the first streaming tensor completion library *cuSTC* on multiple GPUs and evaluate it on real-world datasets. The experiment results demonstrate *cuSTC* achieves performance speedup compared to the SOTA libraries and good scalability on multiple GPUs.

II. Background

A. Notations and Preliminaries

First, we will introduce the preliminaries for the key symbols to formulate the tensor completion, which are listed in

979-8-3503-1993-4/22/$31.00 ©2022 IEEE
DOI 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00177

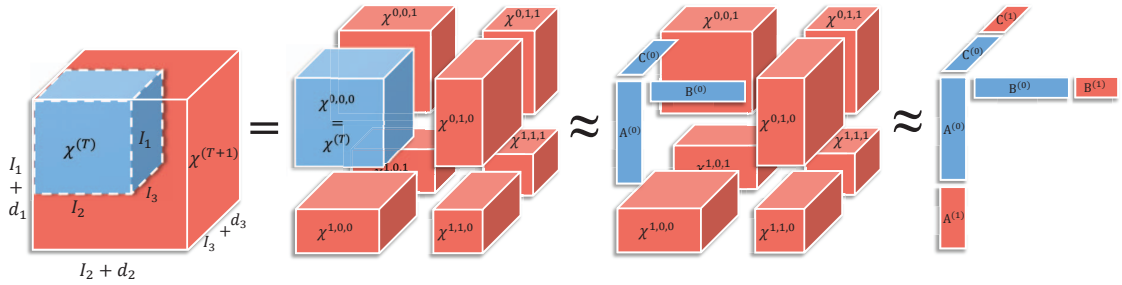


Fig. 1: The illustration of dynamic tensor decomposition in MAST.

Table I. To preserve the clarity and brevity of this article, we will concentrate on the three-dimensional tensor and mode-1 operations. After that, the scheme for higher dimensions and other modes can be easily derived.

TABLE I: Key symbols and operations.

Notation	Definition
$\mathcal{X}$	A multi-dimensional tensor
$\mathcal{X}_{i,j,k}$	An element at (i,j,k) of $\mathcal{X}$
$\{\mathcal{X}^{(T)}\}$	A sequence of multi-dimensional tensors $\mathcal{X}^{(1)},...,\mathcal{X}^{(T)},...$
$\mathbf{A}$	A two-dimensional matrix
$\mathbf{A}_{i,j}$	An element at (i,j) of $\mathbf{A}$
$\{\mathbf{A},\mathbf{B}\}$	Concatenate matrice $\mathbf{A}$ and $\mathbf{B}$ along the first dimension
$\odot$	The symbol for Kronecker product
$\circledast$	The symbol for Hadamard product
$[\![\cdot]\!]$	The symbol for Kruskal operator, e.g, $\mathcal{X} \approx [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$
$\|\mathbf{A}\|_*$	The symbol for nuclear norm of A

B. Multi-aspect Streaming Tensor Completion

Decomposition-based tensor completion utilizes Canonical polyadic decomposition(CPD), a generalization of Singular Value Decomposition and decomposes a tensor $\mathcal{X}$ with rank F into the summation of F rank-one matrixes. Based on the partitioning property of CPD algorithm, the completion of the tensor sequence can utilize the factor matrices generated previously. As illustrated in Figure 1, in the MAST sequence $\{\mathcal{X}^{(T)}\}$, the tensor $\mathcal{X}^{(T)}$ at time T is the subset of the tensor $\mathcal{X}^{(T+1)}$ at time $T+1$. The completion of $\mathcal{X}^{(T+1)}$ can utilize the factor matrices that approximate the preceding tensor $\mathcal{X}^{(T)}$ to reduce the computation. And then, the completion problem on $\mathcal{X}^{(T+1)}$ can be converted to impute the missing entries of the relative complement of $\mathcal{X}^{(T)}$ within $\mathcal{X}^{(T+1)}$.

Dynamic Low-Rank Tensor Completion (LRTC): The MAST completion can be regarded as a dynamic LRTC problem and transferred to a rank-minimization problem [2]. To solve this NP-hard rank minimization problem, tensor rank can be relaxed by summing the nuclear norms of the factor matrices, as described in Equation 1, where $\mathcal{X}$ indicates the complete tensor, Ω is a binary tensor to mark which entries of $\mathcal{X}$ are observed, and $\mathcal{T}$ indicates the observed entries of $\mathcal{X}$ [2]. The Gaussian loss function $\left\|\mathcal{X}-\hat{\mathcal{X}}\right\|$ is denoted as $\mathcal{L}$. Then, the MAST completion problem can be modeled as Equation 2, where $\hat{\mathcal{X}}^{(0)}$ indicates the completed tensor previous step and the factor matrix $\mathbf{A}=\{\mathbf{A}^{(0)},\mathbf{A}^{(1)}\}$.

$$\min_{\mathcal{X},\mathbf{A},\mathbf{B},\mathbf{C}} \left\|\mathcal{X}-\hat{\mathcal{X}}\right\| + \alpha_1\|\mathbf{A}\|_* + \alpha_2\|\mathbf{B}\|_* + \alpha_3\|\mathbf{C}\|_*, \quad s.t.\ \mathcal{X}\circledast\Omega=\mathcal{T},\ \hat{\mathcal{X}}=[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]. \tag{1}$$

$$\min_{\mathcal{X},\mathbf{A},\mathbf{B},\mathbf{C}} \sum_{(i_1,i_2,i_3)\in\Theta} \left\|\mathcal{X}-[\![A^{(i_1)},B^{(i_2)},C^{(i_3)}]\!]\right\| + \alpha_1\|\mathbf{A}\|_* + \alpha_2\|\mathbf{B}\|_* + \alpha_3\|\mathbf{C}\|_* + \lambda(\hat{\mathcal{X}}^{(0)}-[\![\mathbf{A}^{(0)},\mathbf{B}^{(0)},\mathbf{C}^{(0)}]\!]) \quad s.t.\ \Theta=\{0,1\}^3\setminus\{0,0,0\}. \tag{2}$$

Dynamic Alternating Least Squares (ALS): As illustrated in Figure 1, the tensor $\mathcal{X}^{(T+1)} \in \mathbb{R}^{(I_1+d_1)\times(I_2+d_2)\times(I_3+d_3)}$ is partitioned into eight sub-tensors, and $\mathcal{X}^{0,0,0}=\mathcal{X}^{(T)} \in \mathbb{R}^{I_1\times I_2\times I_3}$. With the partition property of CPD algorithm, the sub-matrices of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ can be leveraged to approximate the sub-tensors. Therefore, the loss $\left\|\mathcal{X}-\hat{\mathcal{X}}\right\|$ can be rewritten as the summation of loss of sub-tensors as Equation 2. Based on new loss function, the factor matrices in ALS algorithm are updated as Equation 3 and Equation 4, where $\tilde{\mathbf{A}}$, $\tilde{\mathbf{B}}$ and $\tilde{\mathbf{C}}$ are the results of $\mathcal{X}^{(T)}$ and μ is the forgetting factor. By replacing $\mathcal{X}^{(0,0,0)}$ with the factor matrix updated in previous step, the term $\mathcal{X}^{(0,0,0)}(\mathbf{C}_0\odot\mathbf{B}_0)$ is substituted by $\mu\tilde{\mathbf{A}}(\tilde{\mathbf{C}}^{\mathrm{T}}\mathbf{C}\circledast\tilde{\mathbf{B}}^{\mathrm{T}}\mathbf{B})$ which leads to the reduction of time complexity from $O(RI_1I_2I_3)$ to $O(R^2(I_1+I_2+I_3))$. After updating the factor matrix of all tensor modes, the missing value of observed tensor $\mathcal{T}$ can be completed as Equation 5.

$$\mathbf{A}^{(0)} \leftarrow \frac{\mu\tilde{\mathbf{A}}(\tilde{\mathbf{C}}^{\mathrm{T}}\mathbf{C}_0\circledast\tilde{\mathbf{B}}^{\mathrm{T}}\mathbf{B}_0)+\sum_{(j,k)\neq(0,0)}\mathcal{X}_{(1)}^{(0,j,k)}(\mathbf{C}_k\odot\mathbf{B}_j)}{(\sum_{k=0}^{1}\mathbf{C}_k^{\mathrm{T}}\mathbf{C}_k)\circledast(\sum_{k=0}^{1}\mathbf{B}_k^{\mathrm{T}}\mathbf{B}_k)} \tag{3}$$

$$\mathbf{A}^{(1)} \leftarrow \frac{\sum_{\forall(j,k)}\mathcal{X}_{(1)}^{(1,j,k)}(\mathbf{C}_k\odot\mathbf{B}_j)}{(\sum_{k=0}^{1}\mathbf{C}_k^{\mathrm{T}}\mathbf{C}_k)\circledast(\sum_{k=0}^{1}\mathbf{B}_k^{\mathrm{T}}\mathbf{B}_k)} \tag{4}$$

$$\mathcal{X} \leftarrow \mathcal{T}+\Omega^C\circledast[\![A,\ B,\ C]\!] \tag{5}$$

C. Sparse Tensor Storage Format

The sparse tensor storage formats can exploit sparse attribute of tensors and play a role in optimizing tensor algorithms by reducing memory consumption and random memory access.

The sparse formats can be divided into two categories: COO-based and CSF-based. As the recursive CPD algorithm with the tree-like CSF format [5] does not fit in GPU architecture [6], we only explain the COO-based formats. COO is the most intuitive format where the indices and values are stored straightly. However, when performing the ALS algorithm to update $\mathbf{A}$, the factor matrices of other modes are involved in the calculation. To leverage the phenomenon described above, F-COO [6] compressed the mode-1 indices of non-zero elements to two flag arrays: the start flag (*sf*) and bit flag (*bf*). However, retrieving the mode-1 indices from data in F-COO format when updating the factor matrix of other modes leads to performance degradation. In this paper, we utilize a novel tensor storage format TB-COO (Section III-A) for tensor completion on multiple GPUs.

III. Methodology

In this section, we begin with describing the data structure and relevant operation of TB-COO sparse storage format. Then, by exploiting TB-COO format, we develop the optimization methodology with implementation details for dynamic tensor completion algorithms of the MAST sequence.

A. TB-COO Sparse Storage Format

Figure 2 demonstrates the data structure of COO sparse format and TB-COO sparse format. To optimize the memory consumption and memory access on GPU architecture, TB-COO leverages a bit encoding scheme and a tiling scheme. There are two arrays in TB-COO, named *directory* and *entry*. In TB-COO format, the non-zero values in COO format are sorted by mode-1 indices and stored linearly in array *entry*. The elements in *entry* are partitioned at length *T*. Meanwhile, to retrieve the mode-1 indices efficiently, there are three elements at the head of each entry, where the first element *sp* and the second element *lp* record the index of data in *directory* acting as pointers. The third element is the bitmap which indicates the variance of mode-1 indices. If mode-1 index of i^{th} non-zero value is different from $(i-1)^{th}$ non-zero value, the i^{th} bit in *bitmap* is 0, and vice versa. With the bitmap, the pointer to mode-1 indices for non-zero elements can be easily computed through bit operation, thereby reducing global memory access compared to the traditional COO format. Considering a sparse tensor with *m* different mode-1 index and *nnz* non-zero values, the memory footprint in COO format requires $4nnz$ number of memory. In contrast, TB-COO format requires $3(nnz + \frac{nnz}{T}) + m$ number of memory. Since $m \ll nnz$, TB-COO can decrease memory storage significantly. Compared to the tree-based CSF format, TB-COO can better fit GPU architecture because TB-COO does not rely on recursive algorithms for the MTTKRP routine, and its tiles are more friendly for GPU threads.

B. Optimizing Multi-Aspect Streaming Tensor Completion

We optimize the dynamic ALS algorithm for multi-aspect streaming tensor completion. There are several differences in the dynamic ALS algorithm between MAST sequences and the static pattern for independent tensor sequences. First, only the incremental part relative to the tensor in the previous time step needs to be processed in the MTTKRP routine. After that, an efficient filter is necessary to filter the incremental elements and convert COO storage format to TB-COO format. The processing logic of computing dynamic ALS algorithm for the MAST sequence is illustrated in Algorithm 1.

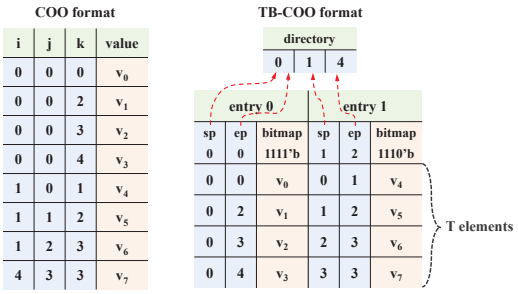


Fig. 2: A sparse tensor stored in COO and TB-COO formats.

Algorithm 1: Dynamic ALS algorithm

Input: Previous factor matrix $\tilde{\mathbf{A}}\ \tilde{\mathbf{B}}\ \tilde{\mathbf{C}}$, Previous tensor mode *pmode*, Tensor $\mathcal{X}$, Tile size T, Rank R, Warp size W
Output: Updated factor matrix $\mathbf{A}\ \mathbf{B}\ \mathbf{C}$

$\mathcal{X}_{inc} = filte\_and\_convert(\mathcal{X},\ pmode)$;
repeat
  for $i = 1$ **to** n **do**
    /* Randomly select a mode from modes which not updated(example mode-1) */
    $H = (\mathbf{C}^T\mathbf{C}) \circledast (\mathbf{B}^T\mathbf{B})$;
    $M = MTTKRP(\mathcal{X}_{inc}, \mathbf{B}, \mathbf{C}, T, R, W)$;
    $T = \tilde{\mathbf{A}}[\![(\tilde{\mathbf{C}}^T\mathbf{C}_0) \circledast (\tilde{\mathbf{B}}^T\mathbf{B}_0)]\!]$;
    $Solve\ \hat{A} = \{T, M\}\ H^{\dagger}$
    update $\mathbf{A}_0$ via Equation 3;
    update $\mathbf{A}_1$ via Equation 4;
    $\mathbf{A} = \{\mathbf{A_0},\ \mathbf{A_1}\}$;
until *convergence or reaching maximum number of iterations*;
Function $MTTKRP(\mathcal{X}, B, C, T, R, W)$**:**
  for *each thread* **do**
    $laneid = threadid\ mod\ W$;
    $warpid = threadid\ /\ W$;
    $mask = get\_warpmask(warpid)$;
    $sp,\ lp,\ bitmap = get\_tile\_attr(\mathcal{X}, warpid, laneid)$;
    $i = get\_i\_index(\mathcal{X}.directory, sp, lp, bitmap, laneid)$
    $j,\ k,\ v = get\_data(\mathcal{X}.entry, T, warpid, laneid)$;
    for $m = 0$ **to** R **do**
      $buffer[\![m]\!] = C[\![c * R + m]\!] * C[\![b * R + m]\!]$;
    for $n = 0$ **to** R **do**
      $tmp = v * buffer[\![m]\!]$;
      $warp\_reduce(M, i, laneid, tmp, bitmap, mask)$;
  $return\ M$;

Before performing multiple-GPU dynamic ALS to decompose the latest tensor in tensor sequence, the tensor modes in the previous step are exploited to extract the elements that do

not appear in the original tensor (line 1). After the incremental part $\mathcal{X}_{inc}$ relative to the previous tensor is filtered, it is split and the storage format is converted. The sparse part in COO format is divided evenly into segments, and the elements in each segment are stored in TB-COO format. Then, each GPU updates the factor matrices with its assigned segment according to the dynamic ALS algorithm. The host device records a range of tensor segments on each GPU and controls the reduction of factor matrices via the $AllReduce$ primitive provided by NVIDIA Collective Communications Library (NCCL).

When processing the separate segments on the GPUs, we perform the second level of the splitting scheme with TB-COO format. One *entry* in TB-COO storage is a tile assigned to the warp and the *entry* length *T* is no more than the warp size. If the number of non-zero elements in the separate segment is indivisible by *T*, some idle threads in the last warp are ignored by the *warpmask* stored in shared memory. The tiling scheme increases data reuse and reduces global memory accesses in the Hadamard routine (line 4) and MTTKRP routine (line 5). In MAST, dynamic ALS can utilize the hardmard and matrix multiple results of the factor matrix to substitute the computation of MTTKRP(line 6) for mode-N sparse tensor, which reduces the time complexity from $O(NR(nnz(\mathcal{X})))$ to $O(R^2(I_1 + ... + I_N))$ and $(I_1 + ... + I_N) \ll nnz(\mathcal{X})$. Further, we leverage the warp shuffle mechanism to optimize the reduction operation in MTTKRP and Hadamard routines. Each thread first generates the mask for active threads in warp (line 16) and then obtains indices and values from sparse tensor stored in TB-COO (line 17∼19). The intermediate results of MTTKRP are calculated and stored in the shared memory(line 20∼21). After the active threads complete the calculation of intermediate results, they conduct the warp-level reduction via $warp\_reduce$ (line 24) and write result to global memory, which reduces the atomic operations.

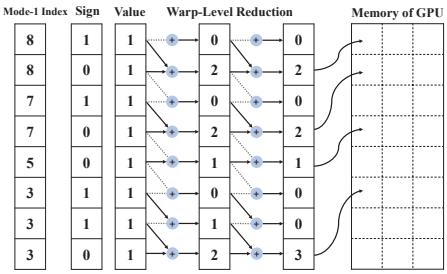


Fig. 3: The illustration of warp-level reduction for optimizing result reduction in MTTKRP and Hadamard routines on GPU.

The warp-level reduction scheme is illustrated in Figure 3. We generate signal variable $sign$ via the bit operation on $bitmap$ to take control of two-stage warp-level reduction. In each stage, the thread with $laneid\ i$ receives the intermediate data from the thread with $laneid\ i + 1$. The original data in $laneid\ i$ is discarded if the $sign$ in $laneid\ i$ is 1. The received data in $laneid\ i$ is accumulated if the $sign$ in $laneid\ i+1$ is 1. After two stages, the summation in the thread with $sign = 1$ is accumulated to the final result in the global memory.

An iteration of the dynamic ALS algorithm generally updates the factor matrices corresponding to each mode in sequence order (1, 2, 3, ...), where the optimization process is settled after the factor matrices are initialized. To avoid this situation, we adopt the randomization scheme proposed in [7] accelerate convergence. Therefore, at each step of the iterative loop, we randomly select an unmodified factor matrix and conduct the update operation.

IV. Evaluation

A. Experiment Setup

1) Datasets: For a comprehensive evaluation, we work with publicly available real-world datasets, including YELP [8], MovieLens [9], DARPA [10], Nell-1 and Nell-2 [11]. Table III shows the dataset details in terms of total dimensions, initial dimensions, increasing step, and sparsity. Initial dimensions indicate the size of the first tensor in the tensor sequence. Increasing step is set to simulate the scenario where real-world tensors grow and generate tensor sequences of five tensors for each dataset. Sparsity is calculated as $\frac{nnz(\mathcal{X})}{Total\ Dimensions}$.

2) Streaming Tensor Completion Libraries: To our knowledge, no public streaming tensor completion libraries are implemented on GPU. Moreover, we only found one public framework named *SIITA* [1] implemented in MATLAB. For a comprehensive comparison, we extend the state-of-the-art CSF-based library *SPLATT* [7] by separating the streaming process into several static tensor completion epochs. We also implement a COO-based streaming tensor completion library *COO-GPU* with the optimization that adopts the tiling strategy with shared memory. The execution time is taken from the average runtime of each iteration loop in seconds, and The convergence of tensor decomposition is measured by Root Mean Square Error (RMSE).

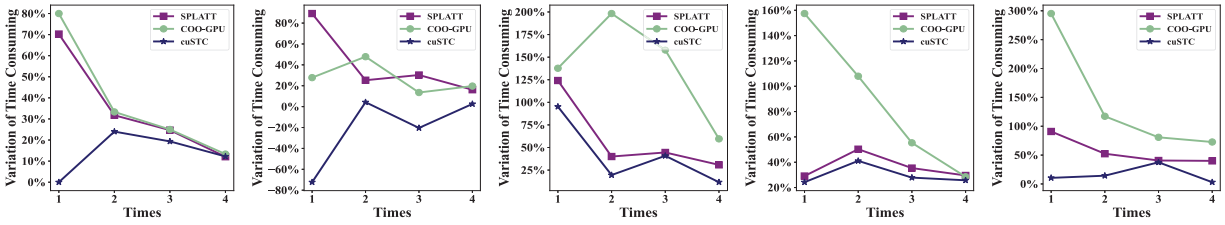
3) Hardware and Software Platforms: We conduct the experiments on a Linux server with two sockets, Intel Xeon Gold 6230R CPUs and two Nvidia Tesla V100 GPUs. The hardware and software specifications are presented in Table IV. The Intel MKL and OpenMP are utilized to parallelize all tensor completion libraries on CPUs, whereas the Nvidia cuSOLVER executes linear algebra routines on GPUs. To further evaluate the scalability of *cuSTC*, we conduct more experiments on another server with Intel Gold 6240R CPUs and eight Nvidia V100 GPUs connected via NVLink.

B. Performance Analysis

The performance comparison of *cuSTC*, *SIITA* and *COO-GPU* is shown in Table II, where *SPLATT* is selected as the baseline. The symbol $\emptyset$ indicates failing to converge within 10 hours, symbol $-$ indicates the invalid speedup due to failing to converge, and symbol ∞ indicates failing to converge with RMSE metric. For the tensor corresponding to each epoch in the tensor sequence, *cuSTC* achieves performance acceleration for whole datasets on both single GPU and multiple GPUs. The

TABLE II: The performance comparison of *cuSTC*, *SPLATT*, *SIITA* and *COO-GPU*.

1-Dataset	2-epoch	SPLATT			SIITA			COO-GPU			cuSTC Single GPU			cuSTC Multiple GPUs		
		3-Time	4-Speedup	5-RMSE	6-Time	7-Speedup	8-RMSE	9-Time	10-Speedup	11-RMSE	12-Time	13-Speedup	14-RMSE	15-Time	16-Speedup	17-RMSE
YELP	1	0.074	1.00	3.95	192.68	7.27E-4	6.00E+04	0.0050	14.80	4.00	0.0025	29.60	4.00	0.0018	**41.11**	4.00
	2	0.13	1.00	3.93	1132.60	1.94E-4	∞	0.0090	14.00	4.00	0.0025	50.40	4.00	0.0025	**50.40**	4.00
	3	0.17	1.00	3.96	2799.69	9.64E-4	∞	0.012	13.83	4.01	0.0031	53.55	4.01	0.0026	**63.85**	4.01
	4	0.21	1.00	3.95	∅	–	∞	0.015	13.80	4.03	0.0037	55.95	4.03	0.0036	**57.50**	4.03
	5	0.23	1.00	3.95	∅	–	∞	0.017	13.65	4.03	0.0043	53.95	4.03	0.0037	**62.70**	4.03
Movie Lens	1	0.046	1.00	3.74	263.44	1.10E-3	1.00E+05	0.09	0.51	3.68	0.017	2.71	3.68	0.0086	**5.35**	3.72
	2	0.087	1.00	3.72	2123.38	1.93E-4	∞	0.12	0.76	3.65	0.0047	18.51	3.71	0.0033	**26.36**	3.66
	3	0.11	1.00	3.74	∅	–	∞	0.017	0.64	3.67	0.0049	22.24	3.67	0.0036	**30.28**	3.67
	4	0.14	1.00	3.72	∅	–	∞	0.19	0.74	3.68	0.0039	36.41	3.68	0.0039	**36.41**	3.68
	5	0.16	1.00	3.70	∅	–	∞	0.23	0.71	3.69	0.004	41.25	3.7	0.0038	**43.42**	3.72
DARPA	1	0.58	1.00	1.20	∅	–	∞	0.46	1.25	1.20	0.21	**5.27**	1.2	0.13	4.46	1.20
	2	1.30	1.00	1.16	∅	–	∞	1.10	1.18	1.16	0.41	3.71	1.16	0.32	**4.06**	1.16
	3	1.82	1.00	1.13	∅	–	∞	3.29	0.66	1.13	0.49	5.71	1.13	0.43	**5.05**	1.13
	4	2.63	1.00	1.13	∅	–	∞	8.48	0.31	1.12	0.69	3.81	1.13	0.55	**4.78**	1.13
	5	3.44	1.00	1.12	∅	–	∞	13.53	0.24	1.12	0.77	4.21	1.12	0.85	**3.81**	1.12
Nell-1	1	1.34	1.00	22.27	∅	–	∞	1.49	0.90	35.30	0.43	4.47	35.30	0.29	**7.05**	35.30
	2	1.73	1.00	15.27	∅	–	∞	3.83	0.45	10.40	0.56	3.09	10.40	0.37	**4.68**	10.40
	3	2.60	1.00	14.68	∅	–	∞	7.97	0.33	14.60	0.79	3.29	14.50	0.53	**4.91**	14.60
	4	3.52	1.00	13.06	∅	–	∞	12.38	0.28	15.10	1.01	3.49	15.10	0.68	**5.18**	15.10
	5	4.49	1.00	15.17	∅	–	∞	15.42	0.29	14.90	1.27	3.54	14.90	0.79	**5.68**	14.90
Nell-2	1	0.11	1.00	29.45	132.38	8.48E-4	∞	0.15	0.74	87.70	0.019	5.79	87.40	0.011	**10.00**	87.60
	2	0.21	1.00	73.02	2346.33	8.77E-5	∞	0.59	0.36	73.40	0.021	10.00	73.40	0.014	**15.00**	73.40
	3	0.32	1.00	68.18	∅	–	∞	1.27	0.25	49.40	0.024	13.33	49.40	0.018	**17.78**	49.60
	4	0.45	1.00	58.53	∅	–	∞	2.30	0.20	83.29	0.033	13.64	83.20	0.021	**21.43**	83.20
	5	0.63	1.00	76.34	∅	–	∞	3.97	0.16	65.00	0.034	18.53	65.00	0.023	**27.39**	65.10



(a) YELP (b) MovieLens (c) DARPA (d) Nell-1 (e) Nell-2

Fig. 4: The variations rate of time consumption for *cuSTC*, *SPLATT* and *COO-GPU*. This figure represents the variation of the time consumption in the current epoch tensor relative to the time consumption of completing the previous epoch tensor.

TABLE III: Tensor datasets used for evaluation.

Datasets	Total Dimensions	Initial Dimensions	Increasing Step	Sparity
YELP	71K×16K×108	11K×4K×104	15K, 3K, 4	2.72E-6
MovieLens	72K×11K×157	32K×3K×154	10K, 2K, 3	8.04E-5
DARPA	22K×22K×24M	2K×22K×4M	5K, 13, 5M	2.41E-9
Nell-1	3M×2M×25M	3M×400K×5M	1, 400K, 4M	9.6E-13
Nell-2	12K×9K×29K	2K×9K×9K	2K, 1, 5K	2.46E-5

TABLE IV: Hardware and software specifications.

	Intel Xeon Gold 6230R	Nvidia Tesla V100
Microarchitecture	Broadwell	Volta
Frequency	2.1GHz	1.23GHz
Memory Size	384 GB	32 GB
FP64 Performance	873.6 GFLOPS	7.066 TFLOPS
Compiler	Intel Compiler 21.1.1	CUDA 10.2

highest speedup evaluated in each tensor dataset is bolded in Table II. Overall, *cuSTC* achieves the best performance among all tensor completion libraries while achieving similar or even better RMSE over the baseline.

Compared to *SPLATT*, *cuSTC* achieves the speedups of 48.69×, 24.22×, 4.54×, 3.58×, 12.26× on YELP, MovieLens, DARPA, Nell-1, Nell-2 datasets, respectively. Column 13 shows the detailed speedups of each epoch. Meanwhile, *COO-GPU* with a single GPU achieves 14.01×, 0.67×, 0.73×, 0.45×, 0.34× speedups over *SPLATT*. *SPLATT* with CSF format gains notable performance on CPU platform, even outperforming *COO-GPU* with COO format on most tensors. *cuSTC* outperforms *COO-GPU* and *SPLATT* benefiting from the two-level tensor partition scheme and GPU warp reduction optimization. The tensor partition scheme cooperates with the TB-COO format, generating bitmaps for warp reduction optimization. When handling large-size tensors, the performance of *SIITA* lags behind other libraries by several orders of magnitude. In addition, *SIITA* fails to converge after the same order of iterations. Therefore, we only compare *cuSTC* with *COO-GPU* and *SPLATT* in the following.

The time consumption variation comparison for tensor completion libraries is shown in Figure 4. It can be observed that the variation rate of *cuSTC* is less than *SPLATT* and *COO-GPU* during the almost whole completion process for all datasets. The reason is that *cuSTC* utilizes the factor matrices generated in the previous epoch to substitute the MTTKRP calculation on the tensor subset in processing. In addition, there are negative values of variation rate at the first time of *cuSTC*. This is because *cuSTC* conducts MTTKRP calculation

for the whole tensor at the first time and conducts dynamic ALS algorithm in the subsequent time steps. The variation of *SPLATT* and *COO-GPU* decreases due to the higher base time consumption as the tensor dimension increases. However, *SPLATT* and *COO-GPU* still obtain higher time consumption than *cuSTC* in the current epoch (deduced by Table II).

C. Scalability Analysis

We evaluate *cuSTC* on a Linux server with eight Nvidia V100 to demonstrate its strong scalability. Execution times are recorded at the last epoch to measure speed changes with increasing GPUs. We select the Nell-1 dataset and a large dataset Yahoo [12] that contains 257 million non-zero values. Meanwhile, we have experimented with other large sparse tensor datasets, such as Amazon-T and Patents-T, but their runtime data storage is unsuitable for eight GPUs. The initial size setting for Yahoo dataset is 45K×113×61 and the increasing step is 40K×5×10. As seen in Figure 5, The speedup trend of Nell-1 slows down slightly as the number of GPUs increases. This is due to the higher data transfer cost and synchronization among GPUs. Despite this, *cuSTC* achieves good scalability on both Nell-1 and Yahoo datasets.

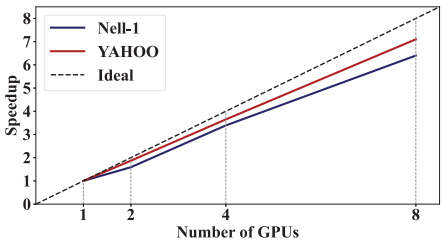


Fig. 5: Strong scalability on Nell-1 and Yahoo dataset.

V. Related Work

To accelerate Canonical Polyadic tensor decomposition (CPD), various optimization techniques has been proposed. Helal *et al.* [3] proposed a mode-agnostic compressed format called ALTO for storing tensors. This approach offers better load balancing and locality while performing a single MTTKRP on CPU. These studies proposed for tensor decomposition have affected and promoted the advance of tensor completion. Kurt *et al* [13] proposed a data movement-aware MTTKRP algorithm for load-balanced parallel computation on multiprocessors CPU. Generalized from matrix cases, various research have been proposed for optimizing tensor completion. Smith *et al.* [7] implemented several optimization algorithms for tensor completion on shared- and distributed-memory architectures. However, these three algorithms are only optimized for multicore CPUs. Song et al. [2] proposed the first MAST completion framework, which can deal with the all-mode-change tensor dynamics while imputing the missing entries. Nimishakavi *et al.* [1] proposed an inductive framework for incorporating side information along multiple modes for tensor completion in multi-aspect streaming settings.

VI. Conclusion

In this paper, we develop the first streaming tensor completion library *cuSTC* on multiple GPUs with dynamic alternating least squares optimization algorithm. *cuSTC* utilizes a novel sparse tensor format TB-COO with tiling and bitmap data structure. Besides, *cuSTC* designs a streaming tensor partition scheme for improving load balance when scaling to multiple GPUs. Based on TB-COO format, *cuSTC* implements a warp-level reduction scheme to reduce atomic operations. The experiment results demonstrate that *cuSTC* achieves up to 48.69× speedup compared to the state-of-the-art tensor completion. *cuSTC* also achieves good scalability when scaling to eight Nvidia V100 GPUs on real-world datasets.

Acknowledgment

This work is supported by National Key Research and Development Program of China (Grant No. 2020YFB1506703), National Natural Science Foundation of China (Grant No. 62072018), and Fundamental Research Funds for the Central Universities. Hailong Yang is the corresponding author.

References

[1] M. Nimishakavi, B. Mishra, M. Gupta, and P. Talukdar, "Inductive framework for multi-aspect streaming tensor completion with side information," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 307–316.

[2] Q. Song, X. Huang, H. Ge, J. Caverlee, and X. Hu, "Multi-aspect streaming tensor completion," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 435–443.

[3] A. E. Helal, J. Laukemann, F. Checconi, J. J. Tithi, T. Ranadive, F. Petrini, and J. Choi, "Alto: Adaptive linearized storage of sparse tensors," in *Proceedings of the ACM International Conference on Supercomputing*, 2021, p. 404–416.

[4] S. Smith, J. Park, and G. Karypis, "Sparse tensor factorization on many-core processors with high-bandwidth memory," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 1058–1067.

[5] S. Smith and G. Karypis, "Tensor-matrix products with a compressed sparse tensor," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 1–7.

[6] B. Liu, C. Wen, A. D. Sarwate, and M. M. Dehnavi, "A unified optimization approach for sparse tensor operations on gpus," in *2017 IEEE international conference on cluster computing (CLUSTER)*. Hawaii, USA: IEEE, 2017, pp. 47–57.

[7] S. Smith, J. Park, and G. Karypis, "Hpc formulations of optimization algorithms for tensor completion," *Parallel Computing*, vol. 74, pp. 99–117, 2018.

[8] YELP, "Yelp," 2015. [Online]. Available: http://www.yelp.com/dataset_challenge/

[9] G. Research, "Movielens," 2019. [Online]. Available: http://grouplens.org/datasets/movielens/

[10] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "Haten2: Billion-scale tensor decompositions," in *IEEE International Conference on Data Engineering (ICDE)*. Seoul, South Korea: IEEE, 2015, pp. 1047–1058.

[11] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis. (2017) FROSTT: The formidable repository of open sparse tensors and tools. [Online]. Available: http://frostt.io/

[12] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer, "The yahoo! music dataset and kdd-cup'11," in *Proceedings of the 2011 International Conference on KDD Cup 2011-Volume 18*. San Diego, USA: JMLR.org, 2011, pp. 3–18.

[13] S. E. Kurt, S. Raje, A. Sukumaran-Rajam, and P. Sadayappan, "Sparsity-aware tensor decomposition," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 952–962.