Accurate and Fast Recovery of Network Monitoring Data with GPU-Accelerated Tensor Completion

Kun Xie[1,2], *Member, IEEE*, Yuxiang Chen[1], Xin Wang[3], *Member, IEEE*,
Gaogang Xie[4,5], *Member, IEEE*, Jiannong Cao[6], *Fellow, IEEE*, Jigang Wen[4],
Guangming Yang[7], Jiaqi Sun[7]

[1] College of Computer Science and Electronics Engineering, Hunan University, Changsha 410082, China
[2] Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen 518000, China
[3] Department of Electrical and Computer Engineering, State of New York University at Stony Brook, NY 11794, USA
[4] Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China
[5] School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100190, China
[6] Department of computing, The Hong Kong Polytechnic University, Hong Kong
[7] Research Institute of Intelligent Network and Terminal of China Telecom, Guangzhou 510630, China

***Abstract*—Monitoring the performance of a large network would involve a high measurement cost. To reduce the overhead, sparse network monitoring techniques may be applied to select paths or time intervals to take the measurements, while the remaining monitoring data can be inferred leveraging the spatial-temporal correlations among data. The quality of missing data recovery, however, highly relies on the specific inference technique adopted. Tensor completion is a promising technique for more accurate missing data inference by exploiting the multi-dimensional data structure. However, data processing for higher dimensional tensors involves a large amount of computation, which prevents conventional tensor completion algorithms from practical application in the presence of large amount of data.**

This work takes the initiative to investigate the potential and methodologies of performing parallel processing for high-speed and high accuracy tensor completion over Graphics Processing Units (GPUs). We propose a GPU-accelerated parallel Tensor Completion scheme (GPU-TC) for accurate and fast recovery of missing data. To improve the data recovery accuracy and speed, we propose three novel techniques to well exploit the tensor factorization structure and the GPU features: *grid-based tensor partition*, independent task assignment based on *Fisher-Yates shuffle*, *sphere facilitated* and *memory-correlated scheduling*. We have conducted extensive experiments using network traffic trace data to compare the proposed GPU-TC with the state of art tensor completion algorithms and matrix-based algorithms. The experimental results demonstrate that GPU-TC can achieve significantly better performance in terms of two relative error ratio metrics and computation time.

I. Introduction

It is very important to efficiently monitor the network to track the network status as well as troubleshoot network incidents and performance issues such as packet losses, latency spikes, and traffic burst [1], [2]. With the emergence of software defined network (SDN) techniques, it becomes more critical to obtain network states at lower time and spatial granularity. This will allow a SDN controller to perform fine-grade network management and traffic engineering for significantly higher network performance.

Several network monitoring systems are proposed to monitor networks in the past few years, where Pingmesh [3] and NetNORAD [4] are two examples. These monitoring systems generally send probe packets between each pair of network nodes, which creates a monitoring cost at $O(n^2)$ for a network of n nodes. It is common for a modern data center to contain hundreds of thousands of servers and the scale is growing rapidly at an exponential rate [5]. The network monitoring becomes a challenge as the size of network constantly increases. Relying on the knowledge of routing paths and network topology, recently deTector [6] proposes to detect and localize network failures with the probing of selected paths to reduce the measurement cost. However, it requires the paths selected well cover all transmission links in the data center, so the measurement cost is still high.

Some recent studies show that network monitoring data such as end-to-end latency and flow traffic have hidden spatio-temporal correlations and thus the matrix of monitoring data has the low rank feature [7], [8]. This inspires the development of novel sparse network monitoring technique [9]. Sample-based network monitoring is applied in this technique, where measurements are only taken between some random node pairs or at some intervals for a given node pair, and the data of other paths are inferred leveraging the spatio-temporal correlations in network monitoring data. As only a few paths need to be probed, the measurement cost can be largely reduced.

As part of the applications of network state tracking and forecasting [10], anomaly detection [11]–[13] and failure recovery are highly sensitive to the missing performance data. This makes the accurate recovery of missing monitoring data from partially observed network states an important procedure in sparse network monitoring. The data inference leverages the spatial-temporal correlations, and its quality highly relies on the inference technique adopted.

Various studies have been made to handle and recover the missing monitoring data. Designed based on purely spatial [14]–[16] or temporal [17], [18] information, the data recovery performance of most known approaches is low. To utilize both spatial and temporal information, recent studies exploit matrix completion [9], [19]–[23] to recover the missing data from a low-rank matrix. Although these approaches present good performance under low data missing ratio, when the data missing ratio is large, the recovery performance suffers as the two-dimensional matrix-based data analysis has the limitation in extracting information.

For better data recovery, it helps to represent monitoring data as a higher dimensional array called *tensor*, a higher-order generalization of vector and matrix. Exploiting the inherent relationship

among higher dimensional data, tensor-based data analysis has shown that tensor models can take full advantage of the multilinear structures to provide better data understanding and information precision. Tensor-based methods have proven to be good analytical tools for dealing with the multi-dimensional data in a variety of fields, including the analysis of web graphs [24], knowledge bases [25], chemometrics [26], signal processing [27], computer vision [28], and transportation management [29]–[31].

Despite that the accuracy of missing data recovery is higher by modeling the monitoring data as a tensor and exploiting the tensor completion technique, compared to vectors and matrices, the processing of higher dimensional tensor data would involve higher computation overhead thus slower speed. When the tensor size is large, it is difficult to perform tensor completion on a traditional CPU platform.

We investigate the potential and methodologies of performing high-speed and high accuracy tensor completion through parallel processing over Graphics Processing Units (GPUs). Although some recent studies [32]–[34] try to investigate parallel and distributed tensor completion by dividing a large tensor into smaller ones, the schemes are not designed to run on the GPU platform. Without well exploiting the special architecture of GPU, these algorithms are inefficient to run in parallel over GPU, which will compromise the speed of tensor completion.

Despite the popularity of GPUs in high-performance and scientific computing and the ever growing computing capabilities of general purpose hardware, the use of GPUs in tensor completion for missing data recovery poses a set of challenges:

- *Efficient parallel programming to well utilize thread architecture.* GPU is a massively parallel processor that supports thousands of active threads. The thread array is divided into multiple threadblocks. The threads within a block cooperate via shared memory while threads in different blocks can not cooperate. To efficiently exploit GPU resources, it requires a programming model to well utilize the thread architecture for higher and efficient parallelism.
- *Efficient job scheduling to take full advantage of on-chip memory for higher performance.* GPU includes on-chip and off chip memory. The off-chip memory is large but slow, while on-chip shared memory has limited space but fast and is a critical resource for good performance. However, random access of data or variables may result in high cache miss rate thus high communication cost to transfer the data from off-chip memory to on-chip memory. To ensure high performance parallel tensor completion, it calls for specific task scheduling scheme to increase the hit rate of cached data in the shared memory.

In light of the above challenges, we propose a GPU accelerated parallel Tensor Completion scheme (GPU-TC) for accurate and fast missing monitoring data recovery. We model the volumes of traffic flows between network nodes as a multi-way tensor (called traffic tensor) and use the network traffic data recovery as an example to present the design of GPU-TC and demonstrate its effectiveness in quick recovery of missing tensor data.

According to the GPU architecture, we propose novel techniques in GPU-TC at two levels, a high level that assigns tasks among threadblocks and a low level that handles the parallel execution using threads inside each threadblock. Following are the main proposed techniques in this paper:

1) ***High level: Gridding a large tensor to enable high level parallel processing among multiple threadblocks.*** GPU-TC exploits the factorization structure of the tensor completion solution to elegantly grid the large traffic tensor into small sub-tensors with each associated with sub-block factor matrices. To train the sub-block factor matrices and also facilitate fusing these matrices to form large factor matrices, based on the distributed application of Stochastic Gradient Descent (SGD), we divide the training process into multiple iteration epochs each consisting of multiple rounds to scan all observed data.
2) ***High level: Enabling simple task scheduling to well exploit the memory architecture of GPU platform.*** To increase the cache hit ratio in the local shared memory of GPU, GPU-TC applies a novel *sphere* structure to well represent the neighbor relationship among sub-tensors, based on which we propose a *memory-correlated scheduling* algorithm to determine the execution sequence of sub-tensors in theadblocks.
3) ***High level: Enabling simple and independent task assignment for parallel task executions in multiple threadblocks across different iteration epochs.*** We propose an algorithm based on Fisher-Yates shuffle that only needs to assign independent tasks to different threadblocks in the first round of an epoch. In the remaining rounds, the subtasks are scheduled to execute in the threadblocks following our well designed memory-correlated scheduling algorithm without need of explicitly assigning sub-tasks. We demonstrate with theoretical analysis that our scheduling technique can provide simple and independent task scheduling to enable parallel executions in every rounds of all epochs. The independent task execution of the whole process makes the factor matrix training process more random, which speeds up the convergence process of tensor completion as well as helps to further increase the data recovery accuracy.
4) ***Low level: Enabling parallel running of algorithms using multiple threads in each threadblock.*** To well exploit GPU's SIMD (Single Instruction Multiple Data) architectures for high computational performance, we divide SGD update rules into vector operations with each vector having the same length, and propose the parallel running of threads for vector operations.

Using real traffic traces Abilene [35] and GÈANT [36], we implement the proposed GPU-TC on the GPU platform. We compare the proposed algorithms with the state of art tensor completion algorithms and the matrix-based algorithms, and our results demonstrate that GPU-TC can achieve significantly better performance in terms of two relative error ratio metrics and computation time.

To the best of our knowledge, this is the first work that enables the accurate inference of missing monitoring data through fast tensor completion over GPUs and we demonstrate its capability in the context of processing network traffic data. We expect that GPU's massively-parallel processing power and tensor's ability of extracting hidden patterns and structures of data in high dimensions open a venue for high performance data recovery and analysis.

The rest of the paper is organized as follows. Section II presents the related work. The preliminaries of tensor and GPU are presented in Section III. We present our system model and problem formulation in Section IV, and our solution overview in Section V. We describe our algorithms for independent task

assignment, memory-correlated scheduling, and GPU running in Section VI, Section VII, and Section VII, respectively. Finally, we implement the proposed GPU-TC on the GPU platform and evaluate the performance using real traffic trace data in Section VIII, and conclude the work in Section IX.

II. Related work

We are not aware of any other work that provide practical techniques to exploit GPU to accelerate the tensor completion, and apply efficient tensor completion to quickly recover the network traffic data. Following we review some literature work that has certain parts relevant to our work.

A. Network monitoring

A number of systems have been proposed to monitor the network performance. Pingmesh [3] and NetNORAD [4] adopt an end-to-end probing approach to measure network latency and packet loss. As they inject probes between each pair of network nodes without selection, it will introduce too much overhead to consume the network bandwidth. To reduce the measurement cost, deTector [6], a topology-aware network monitoring system is proposed. It requires the knowledge of the network topology and routing protocol in a network, and uses source routing to control the probe path to guarantee that all transmission links in the network are well covered by the probes. Although its well-controlled probes significantly reduce the probing overhead as compared to Pingmesh and NetNORAD, the measurement cost is still high. As introduced in Introduction, to further reduce the measurement cost, a novel sparse network monitoring [9] is proposed recently, where only a few paths are selected for measurements. It infers the data of other paths leveraging the spatio-temporal correlations in the network monitoring data. Although promising, its performance relies on the missing data recovery procedure.

Although matrix-completion-based missing data recovery algorithms [9], [19]–[23] provide good recovery performance when the data missing ratio is low, a matrix is still not enough to capture the comprehensive correlations among the traffic data, and the data recovery performance is still low when the data missing ratio is large. For example, although the traffic matrix in [19] can catch the spatial correlation among flows and the small-scale temporal feature, it can not incorporate other features such as the traffic periodicity across days.

To capture more spatial-temporal features in the traffic data, recent work in [37], [38] proposes to apply the tensor completion to traffic recovery. Although these initial efforts demonstrate that the potential of applying tensor-based schemes for better data recovery, tensor computations would involve larger amount of computation, which makes it difficult to apply this technique to large-scale traffic monitoring in a practical Internet environment.

B. Tensor completion

With the rapid progress of sparse representation, following the compressive sensing [39], [40] and matrix completion [41]–[45] to process one-dimensional and two-dimensional data arrays, tensor completion has attracted lots of research interests recently. Several tensor completion algorithms [46]–[49] are proposed to capture the global data structure for recovering the missing data via a high-order CANDECOMP PARAFAC (CP) decomposition [50], [51] and Tucker decomposition [52].

Tensor completion algorithms usually execute iteratively. The computational core of each iteration is a special operation called the matricized tensor times Khatri-Rao product (MTTKRP), which involves high computational cost and requires large memory. Several techniques [53]–[56] are proposed to reduce the computation and space cost of MTTKRP by carefully reordering the operations and exploiting the sparsity of real-world tensors. However, as most of these methods need fully matricized tensor data for each step, they still involve high memory and computation costs and suffer from low convergence speed. Mostly developed for centralized and in-memory computation on a single machine, these algorithms are difficult to be executed in parallel on the GPU platform.

Some recent studies [32], [33] propose to divide the large-scale tensor into smaller ones. Then factor matrices are reconstituted from the estimated sub-factors. However, the parallel machines have to communicate with each other or the server center to obtain the updated factor matrices in a straightforward way, which incurs high communication cost. Although the overall partition methods are sound for parallel algorithms, these methods can not run efficiently over the GPU platform. Recently, [34] proposes to perform parallel tensor completion based on alternating least squares (ALS), stochastic gradient descent (SGD), and coordinate descent (CCD++). However, like most of existing studies [32], [33], [34] is not designed to run over GPU. Without considering special GPU architecture, the proposed approaches can not be directly applied to the GPU platform to achieve good performance.

Besides tensor completion, several parallel or distributed matrix completion algorithms [57]–[61] are proposed recently to speed up the procedure of missing data recovery with the data are modeled with a matix. Among which, only [61] runs over the GPU platform. However, this algorithm does not well exploit GPU's special architecture (especially on-chip shared memory) for higher performance.

Compared with matrix completion, tensor completion incurs much higher computational overhead, and it is more important to speed up tensor completion. Methods for parallel matrix completion often can not be directly applied for efficient parallel tensor completion.

To achieve a high parallelism gain in GPU, its special architecture features (special thread, thread block, and special memory architecture) should be well exploited. Different from current studies, we propose GPU-TC, the first work to design and implement parallel tensor completion algorithm on the GPU platform taking the GPU's architecture features into the consideration. The experiment results show that GPU-TC can achieve highly accurate traffic data recovery with a short computation time.

III. Preliminaries

In this paper, scalars are denoted by lowercase letters ($a, b, \cdots$), vectors are written in boldface lowercase ($\mathbf{a}, \mathbf{b}, \cdots$), and matrices are represented with boldface capitals ($\mathbf{A}, \mathbf{B}, \cdots$). Higher-order tensors are written as calligraphic letters ($\mathcal{A}, \mathcal{B}, \cdots$). The elements of a tensor are denoted by its symbolic name with indexes in subscript. For example, the i-th entry of a vector $\mathbf{a}$ is denoted by a_i, the element (i, j) of a matrix $\mathbf{A}$ is denoted by a_{ij}, and the element (i, j, k) of a third-order tensor $\mathcal{A}$ is denoted by a_{ijk}. The row (column) vectors of a matrix are denoted by the symbolic name of the matrix with indexes in subscript. For example, the i-th row (column) vector of a matrix $\mathbf{A}$ is denoted by $\mathbf{a_i}$ ($\mathbf{a_{(i)}}$).

A. Tensor and Decomposition

A *tensor* is a multi-dimensional array, and is a higher-order generalization of a vector (first-order tensor) and a matrix (second-order tensor). An N-way or Nth-order tensor (denoted as $\mathcal{A} \in$

$\mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$) is an element of the tensor product of N vector spaces, where N is the order of $\mathcal{A}$, also called way or mode. The element of $\mathcal{A}$ is denoted by $a_{i_1,i_2,\cdots,i_N}$, $i_n \in \{1, 2, \cdots, I_n\}$ with $1 \leq n \leq N$. Some preliminaries in this paper can be found in [62]–[64].

Definition 1. *Slices are two-dimensional sub-arrays, defined by fixing all indexes but two.*

In Fig. 1, a 3-way tensor $\mathcal{X}$ has horizontal, lateral and frontal slices, which are denoted by $x_{i::}$, $x_{:j:}$ and $x_{::k}$, respectively.

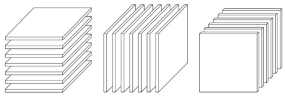


Fig. 1. Tensor slices.

Definition 2. *The outer product of two column vectors $\mathbf{a} \circ \mathbf{b}$ is the matrix defined by: $(\mathbf{a} \circ \mathbf{b})_{ij} = a_i b_j$.*

Definition 3. *The outer product $\mathcal{A} \circ \mathcal{B}$ of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N_1}}$ and a tensor $\mathcal{B} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_{N_2}}$ is the tensor of the order $N_1 + N_2$ defined by*

$$(\mathcal{A} \circ \mathcal{B})_{i_1,i_2,\cdots,i_{N_1},j_1,j_2,\cdots,j_{N_2}} = a_{i_1,i_2,\cdots,i_{N_1}} b_{j_1,j_2,\cdots,j_{N_2}} \quad (1)$$

for all values of the indexes.

Since vectors are first-order tensors, the outer product of three column vectors $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ is a tensor given by:

$$(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})_{ijk} = a_i b_j c_k \quad (2)$$

for all values of the indexes.

Definition 4. *A 3-way tensor $\mathcal{X}$ is a rank one tensor if it can be written as the outer product of three column vectors, i.e. $\mathcal{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$.*

In this paper, we model the traffic data as a 3-way tensor and our focus is thus on 3-way tensor.

Definition 5. *The rank of a 3-way tensor is the minimal number of rank one tensors that can be added to generate the tensor, i.e. the smallest R, such that $\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_{(r)} \circ \mathbf{b}_{(r)} \circ \mathbf{c}_{(r)}$.*

Definition 6. *The idea of CANDECOMP/PARAFAC (CP) decomposition is to express a tensor as the sum of a finite number of rank one tensors. A 3-way tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ can be expressed as*

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_{(r)} \circ \mathbf{b}_{(r)} \circ \mathbf{c}_{(r)}, \quad (3)$$

with an entry calculated as

$$x_{ijk} = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr} \quad (4)$$

where $R > 0$, a_{ir}, b_{jr}, c_{kr} are the i-th, j-th, and k-th entry of vectors $\mathbf{a}_{(r)} \in \mathbb{R}^I$, $\mathbf{b}_{(r)} \in \mathbb{R}^J$, and $\mathbf{c}_{(r)} \in \mathbb{R}^K$, respectively.

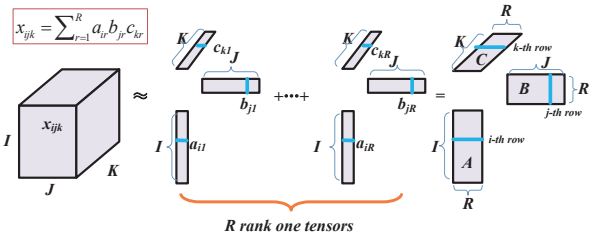


Fig. 2. CP decomposition of a 3-way tensor as sum of R outer products (rank one tensors).

Fig. 2 illustrates the CP decomposition. By collecting the vectors in the rank one components, we have tensor $\mathcal{X}$'s factor matrices $\mathbf{A} = [\mathbf{a}_{(1)}, \cdots, \mathbf{a}_{(R)}] \in \mathbb{R}^{I \times R}$, $\mathbf{B} = [\mathbf{b}_{(1)}, \cdots, \mathbf{b}_{(R)}] \in \mathbb{R}^{J \times R}$, and $\mathbf{C} = [\mathbf{c}_{(1)}, \cdots, \mathbf{c}_{(R)}] \in \mathbb{R}^{K \times R}$. Using the factor matrices, we can rewrite the CP decomposition as follows.

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_{(r)} \circ \mathbf{b}_{(r)} \circ \mathbf{c}_{(r)} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \quad (5)$$

An entry x_{ijk} can be calculated as the sum of the dot product of row vectors $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$. That is, $x_{ijk} = \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k$, where $\mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}$ is the dot product of the three row vectors $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$ with $\mathbf{a}_i \in \mathbb{R}^{1 \times R}$, $\mathbf{b}_j \in \mathbb{R}^{1 \times R}$, and $\mathbf{c}_k \in \mathbb{R}^{1 \times R}$. In this paper, based on CP decomposition, we design a GPU-based parallel tensor completion algorithm for traffic data recovery.

B. GPU architecture

In this paper, we implement our GPU-TC on NVIDIA GPU. Following, we illustrate its basic architecture.

A NVIDIA GPU has three main components: threads, thread-blocks (also called blocks) and memory hierarchy. Compute Unified Device Architecture (CUDA) is released by NVIDIA to simplify the general-purpose programming of GPUs. Threads in a CUDA kernel are organized in two levels, as shown in Fig. 3(a). At the top level, a kernel consists of a 1D/2D grid threadblocks each internally containing multiple threads organized in either 1, 2 or 3 dimensions. One grid corresponds to one kernel call. One or more thread blocks can be executed by a single Streaming Multiprocessor (SM). In Fig. 3(b), two blocks are executed in Multiprocessor 0.

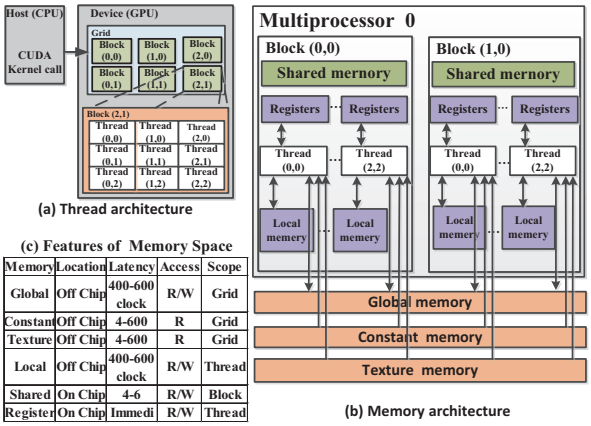


Fig. 3. NVIDIA GPU architecture.

A CUDA program can use different types of memory. As shown in Fig. 3(b), the *Global memory* is normally used for copying the input and output data to and from the main memory of the host. Although the space is large, it is off-chip and slow. The local variables used by each thread are independent and are preferentially stored in registers for fast access. When registers are not enough, data are stored in the *local memory*. Both local memory and registers are visible to only individual threads and cannot be programmed directly by the programmer. *Constant memory* is off-chip with limited space to store a small number of constants (or non-modifiable input data) accessible by all threads. *Texture memory* is a subset of global memory that is accessed via the texture fetch unit. Different from the global memory, the constant and texture memory can only be read but not written by a function. *Shared memory* is a small on-chip software-managed memory on the NVIDIA platform. Each thread block has a shared memory visible to all its threads and with the same lifetime as the block. The latency of accessing a shared memory without bank conflicts is fast and similar to a register access. The effective use of the shared memory can have a large impact on the performance.

To take advantage of the GPU architecture for fast traffic

recovery, we will exploit the shared memory as a user managed cache: first loading the data needed for a tensor completion task from the global memory into the shared memory (coalesced), then processing the data, and finally writing results back to the global memory (coalesced). As the shared memory is limited, to achieve a good performance with a low data retrieving cost, we need a high cache hit ratio. In Section VII, we will propose our memory-correlated scheduling algorithm to well utilize the shared memory.

IV. System model and problem

We will illustrate how our algorithms enable the parallelization of tensor completion using the traffic tensor as an example. We first introduce our traffic tensor model, and then formulate the traffic data recovery problem.

Based on the analyses of real traffic trace, our recent work on tensor completion [8], [38], [65], [66] reveals that the traffic data have the features of temporal stability, spatial correlation, and periodicity. To fully exploit these traffic features for accurate traffic data recovery, following [8], [38], [65], in this paper, we model the traffic data as a 3-way tensor $\mathcal{M} \in \mathbb{R}^{I \times J \times K}$, where K corresponds to the number of origin and destination (OD) pairs in the network, and there are J days to consider with each day having I time intervals. Our recent work [8], [38], [65] has demonstrated that the traffic tensor has low-rank, which satisfies the prerequisite for using the tensor completion to recover the missing traffic data.

In the traffic tensor model, an entry m_{ijk} contains the flow traffic volume data between OD pair k in the time interval i of day j. For the Abilene trace [35], $I = 288$, $K = 144$, and $J = 168$. Our GPU-TC, however, is general and does not depend on how the traffic tensor is modeled.

As partial traffic pairs are often monitored to reduce the measurement load and also there are an unavoidable data losses upon severe communication conditions, $\mathcal{M}$ is generally an incomplete tensor. If there are no traffic data between a pair of nodes in a given time interval, it leaves the corresponding entry in $\mathcal{M}$ empty. The traffic recovery problem is to recover the tensor $\mathcal{M}$ based on its samples through the tensor factorization. With the factorization based on the popular CP decomposition, the problem is defined as

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} L(\mathbf{A},\mathbf{B},\mathbf{C}) = \left\| \left(\mathcal{M} - \sum_{r=1}^{R} \mathbf{a}_{(\mathbf{r})} \circ \mathbf{b}_{(\mathbf{r})} \circ \mathbf{c}_{(\mathbf{r})} \right)_{\Omega} \right\|_F^2 \tag{6}$$

which requires the finding of the factor matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$ to approximate the tensor $\mathcal{M}$ with the minimum loss $L(\mathbf{A},\mathbf{B},\mathbf{C})$. $\mathbf{a}_{(\mathbf{r})}$, $\mathbf{b}_{(\mathbf{r})}$, and $\mathbf{c}_{(\mathbf{r})}$ are the r-th column of the factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, Ω is the set of indices of the sample entries in $\mathcal{M}$. Based on the definition of the Frobenius norm $\|\|_F$, we can rewrite the loss function as a sum of local losses over the sample m_{ijk} as

$$L(\mathbf{A},\mathbf{B},\mathbf{C}) = \sum_{(i,j,k)\in\Omega} L_{ijk}(\mathbf{A},\mathbf{B},\mathbf{C}) \tag{7}$$

where $L_{ijk}(\mathbf{A},\mathbf{B},\mathbf{C}) = (m_{ijk} - \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k)^2$ with $\mathbf{a}_i, \mathbf{b}_j, \mathbf{c}_k$ denoting the i-th, j-th, and k-th row of the factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, respectively. As the local loss function $L_{ijk}(\mathbf{A},\mathbf{B},\mathbf{C})$ only depends on $\mathbf{a}_i, \mathbf{b}_j, \mathbf{c}_k$, we can rewrite this loss function as $L_{ijk}(\mathbf{a}_i, \mathbf{b}_j, \mathbf{c}_k)$.

After getting three factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, the original traffic tensor can be recovered by

$$\hat{\mathcal{M}} = \sum_{r=1}^{R} \mathbf{a}_{(\mathbf{r})} \circ \mathbf{b}_{(\mathbf{r})} \circ \mathbf{c}_{(\mathbf{r})} \tag{8}$$

with the recovered entry

$$\hat{m}_{ijk} = \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k. \tag{9}$$

V. Solution Overview

In this section, we first introduce the basic method we take to solve our problem, based on which we outline our methodologies in inferring the missing data of the traffic tensor through parallel calculation on a GPU platform.

A. SGD-based traffic recovery

Because the loss in Eq. (6) is a non-convex function of the factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$, the traffic recovery problem in (6) is difficult to solve. Among various existing optimization methods for the loss minimization, the gradient descent (GD) algorithm is the simplest. It iteratively takes small steps to move towards the optimal solution in the direction of the negative gradient of loss. With the need of scanning all samples in the set Ω to compute the gradient of $L(\mathbf{A},\mathbf{B},\mathbf{C})$ in each iteration step, its computation overhead is large. Instead, we propose to take the stochastic gradient descent (SGD) method to reduce the load and complete the tensor. We first estimate the local gradient ${L_{ijk}}'(\mathbf{A},\mathbf{B},\mathbf{C})$ of an entry m_{ijk} randomly selected from the sample set Ω, and then estimate the global gradient as $\hat{L}'(\mathbf{A},\mathbf{B},\mathbf{C}) = N{L_{ijk}}'(\mathbf{A},\mathbf{B},\mathbf{C})$, where $N = |\Omega|$.

Based on the local gradient, we design our SGD-based traffic recovery scheme in Algorithm 1. It randomly selects a sample m_{ijk} and applies the update rules (10)-(12) to update the factor matrices iteratively to reduce the loss in the tensor completion process. The iteration will continue until the loss difference between two consecutive iterations is smaller than a given threshold value or the given number of iterations is reached. The parameter ε in (10)-(12) is the learning rate.

Algorithm 1 SGD based traffic recovery

Input: a sample set Ω; initial factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$

1: **while** not converged **do**
2: Select a sample entry $(i,j,k) \in \Omega$ uniformly at random
3:
$$\mathbf{a}_i' \leftarrow \mathbf{a}_i - \varepsilon N \frac{\partial}{\partial \mathbf{a}_i} L_{ijk}(\mathbf{a}_i, \mathbf{b}_j, \mathbf{c}_k) \tag{10}$$
4:
$$\mathbf{b}_j' \leftarrow \mathbf{b}_j - \varepsilon N \frac{\partial}{\partial \mathbf{b}_j} L_{ijk}(\mathbf{a}_i, \mathbf{b}_j, \mathbf{c}_k) \tag{11}$$
5:
$$\mathbf{c}_k' \leftarrow \mathbf{c}_k - \varepsilon N \frac{\partial}{\partial \mathbf{c}_k} L_{ijk}(\mathbf{a}_i, \mathbf{b}_j, \mathbf{c}_k) \tag{12}$$
6: $\mathbf{a}_i = \mathbf{a}_i'$
7: $\mathbf{b}_j = \mathbf{b}_j'$
8: $\mathbf{c}_k = \mathbf{c}_k'$
9: **end while**

B. Parallelization of the tensor completion

Although Algorithm 1 can be applied to the tensor factorization, it is not applicable to handle a large amount of data. The procedures in (10)-(12) are inherently sequential and difficult to be taken in parallel over the GPU platform.

If we check carefully the algorithm, in each iteration only one row of the factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are updated. Specifically, for a randomly selected entry $(i,j,k) \in \Omega$, only the i-th row of the factor matrix $\mathbf{A}$ (i.e., $\mathbf{a}_i$), the j-th row of the factor matrix $\mathbf{B}$ (i.e., $\mathbf{b}_j$) , and the k-th row of the factor matrix $\mathbf{C}$ (i.e. $\mathbf{c}_k$) are updated. In a new iteration, if an entry (i',j',k') with $i' \neq i$, $j' \neq j$, and $k' \neq k$ is selected, their rows of the factor matrices

$\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ (i.e., $\mathbf{a}_{i'}$, $\mathbf{b}_{j'}$, and $\mathbf{c}_{k'}$) can be updated in parallel with the previous iteration.

We can extend this idea to the parallel execution of multiple gradients. For a series of local gradients corresponding to entries from a set $\Omega_1 = \{\{i,j,k\} : i \in I_1, j \in J_1, k \in K_1\}$ and $\Omega_2 = \{\{i,j,k\} : i \in I_2, j \in J_2, k \in J_2\}$, where $I_1 \cap I_2 = \varnothing, J_1 \cap J_2 = \varnothing, K_1 \cap K_2 = \varnothing$, we could run these operations completely in parallel. We call Ω_1 and Ω_2 independent sampling sets. An example in Fig. 4 illustrates this feasibility. This feature provides us the possibility of designing a partition-based parallel tensor completion algorithm.

To facilitate sub-task scheduling in practical systems, assuming the threads in GPU platform are grouped into s threadblocks, we can grid the traffic tensor $\mathcal{M}$ into $s \times s \times s$ sub-tensors, each is denoted as $\mathcal{Z}_{i,j,k}$ $(1 \le i \le s, 1 \le j \le s, 1 \le k \le s)$. Fig. 5 shows an example with $s = 3$. Denote $\mathcal{Z}_{i,j,k}$'s corresponding local factor matrices as $\mathbf{A_i}$, $\mathbf{B_j}$, and $\mathbf{C_k}$. As the sub-tensor $\mathcal{Z}_{i,j,k}$ includes multiple samples, the block factor matrices $\mathbf{A_i}$, $\mathbf{B_j}$ $\mathbf{C_k}$ include multiple rows of matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, which correspond to the samples in the sub-tensor. With the tensor gridding, if $i \neq i', j \neq j', k \neq k'$, two sub-tasks corresponding to two sub-tensors $\mathcal{Z}_{i,j,k}$ and $\mathcal{Z}_{i',j',k'}$ can be run concurrently.

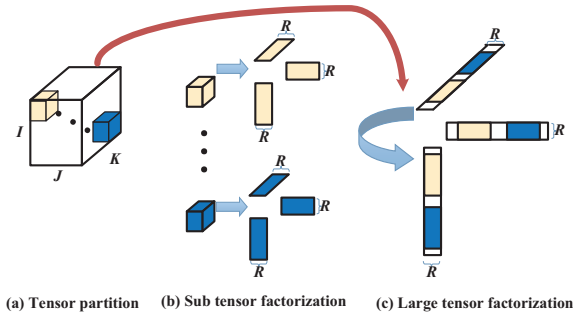


Fig. 4. Partition based tensor completion: (1) The large tensor is partitioned into small sub-tensors; (b) The sub-tensor factorizations are executed in parallel. If samples contained in sub-tensors are independent, the rows of the factor matrices corresponding to them are also independent. We can further combine these factor matrices to form larger factor matrices corresponding to a larger part of the tensor in Fig. 4(c).

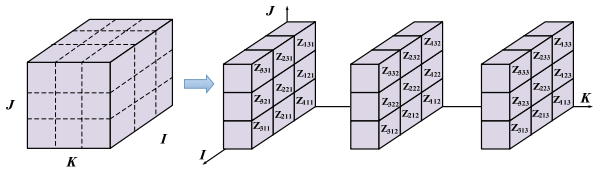


Fig. 5. Grid based tensor data division.

VI. Independent Task Assignment

In SGD-based traffic recovery (Algorithm 1), to train the factor matrices for tensor completion, one sample data entry is randomly selected in an iteration step. We refer the time duration taken for all the sample entries in the tensor to be selected as an *epoch*. To take full advantage of the GPU for parallel processing, we schedule s sub-tasks to execute inside s threadblocks in a time slot, and we call it a *round*. One epoch can consist of multiple rounds. A complete traffic recovery algorithm may include multiple epochs before the algorithm converges and finds the solution. To speed up the algorithm convergence process, the task assignments for SGD updates in different epochs should be independent.

To achieve the independency of task assignment in a round and across epochs, we apply the Fisher-Yates shuffle [67], [68], also known as the Knuth shuffle. Fisher-Yates shuffle has following good properties: 1) low time complexity at $O(n)$ and space complexity at $O(1)$; 2) unbiased with equal likelihood for all outcomes. As Fisher-Yates shuffle outperforms other shuffle algorithms (Bay's shuffle, Perfect Shuffle, Random Shuffle) in time and space [69]–[71], it is well known as the best shuffle algorithm for random permutation. As our main goal is to achieve fast tensor completion with independent task assignment, we exploit the low cost Fisher-Yates shuffle in our task assignment algorithm.

To assign s sub-tasks to s threadblocks in a round, we use the lists I_I, I_J, and I_K to hold the indexes for the sub-tensors, with the number of indexes in each list equal to s. We apply Fisher-Yates shuffling to generate random permutations of the lists I_I, I_J, and I_K, and assign the tasks in Algorithm 2.

Algorithm 2 independent task assignment based on Fisher-Yates shuffling

1: $I_I = \{1, 2, \cdots, s\}$, $I_J = \{1, 2, \cdots, s\}$, and $I_K = \{1, 2, \cdots, s\}$
2: $p = s - 1$
3: **while** $p > 0$ **do**
4: $\quad i = random\,(0, p)$, $j = random\,(0, p)$, $k = random\,(0, p)$
5: $\quad$ swap:$I_I[p] \leftrightarrow I_I[i]$, $I_J[p] \leftrightarrow I_J[j]$, $I_K[p] \leftrightarrow I_K[k]$
6: $\quad p - -$
7: **end while**
8: **for** $p = 1, \cdots, s$ **do**
9: $\quad$ Assign sub-task $\mathcal{Z}_{I_I[p], I_J[p], I_K[p]}$ to threadblock p
10: **end for**

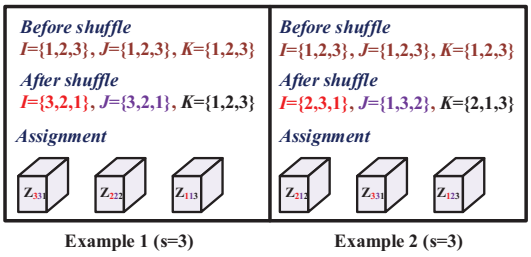


Fig. 6. Example of independent task assignment.

The sub-tasks executed in parallel in an iteration round should correspond to independent sample sets. Denoting the s sub-tensors as $\mathcal{Z}_{i_1 j_1 k_1}, \mathcal{Z}_{i_2 j_2 k_2}, \cdots, \mathcal{Z}_{i_s j_s k_s}$, $\forall p, q \in \{1, \cdots, s\}$, they must meet the condition $i_p \neq i_q, j_p \neq j_q, j_p \neq j_q$. To meet the requirement, the elements in each of the index lists I_I, I_J, and I_K should not be duplicated. On line 1, we initialize the three lists sequentially as $\{1, 2, \cdots, s\}$. Actually, we can initialize these lists with this set of s elements in any order without impacting the results of the random permutation on line 4. After executing the shuffling process on lines 2-7, we can assign s sub-tasks, each corresponding to one sub-tensor, to s threadblocks on lines 8-10. Due to the uniqueness of the elements in each index list, when an index is picked by a sub-tensor, the following sub-tensors will not use the same index. Therefore, the shuffling process also ensures that the gradient and loss updates in each epoch follow a random sequence.

Fig. 6 shows one example of applying the proposed independent task assignment algorithm for two epochs. In epoch 1, after shuffling, the index lists are $I_I = \{3, 2, 1\}$, $I_J = \{3, 2, 1\}$, and

$I_K = \{1, 2, 3\}$, therefore, sub-tensors $\mathcal{Z}_{331}$, $\mathcal{Z}_{222}$, and $\mathcal{Z}_{113}$ are assigned to 3 threadblocks. Obviously, these 3 sub-tensors are independent as the index lists do not have duplicate elements in each list. In epoch 2, after the shuffling, different index lists are obtained, which results in the assignment different from epoch 1. As Algorithm 2 is unbiased, every permutation of I_I, I_J, and I_K in each epoch is equally likely.

The randomness of parameter updates and the independence of task scheduling in different epochs will ensure the factor matrix training process more random, which will help to further increase the tensor recovery accuracy. In Section VIII-C, we will show the performance gain on recovery accuracy with the use of our shuffling technique.

Although an epoch includes multiple rounds, we only apply our Algorithm 2 to assign tasks to threadblocks for the first round. For the following rounds, we apply the memory-correlated scheduling algorithm presented in Section VII to determine the following task execution sequence. This will exploit the correlation of data to increase the memory catching ratio and simplify the task scheduling procedure.

VII. Memory Correlated Scheduling

As introduced earlier, each multiprocessor of GPU has an on-chip shared memory for the threads in a block to access quickly without the need of accessing the global memory frequently. If the data in the on-chip memory can be better utilized, we can reduce the amount of data to transfer from the global memory to the shared memory, thus reducing the communication cost and time. In order to achieve fast processing, our algorithm takes into account the correlation of data in sub-tensors to schedule their task sequences. The questions we need to answer are: 1) How to determine the correlation of data? and 2) What rules to follow so the scheduling can be easily performed while meeting the memory correlation requirements?

Following the tensor partition in Fig. 5, if $\mathcal{Z}_{i,j,k}$ is assigned to a threadblock, the measured data samples in $\mathcal{Z}_{i,j,k}$ as well as the three block factor matrices $\mathbf{A_i}$, $\mathbf{B_j}$, and $\mathbf{C_k}$ need to be transmitted to the shared memory. Among all sub-tensors not yet scheduled to run, if sub-tensor $\mathcal{Z}_{i,j,k'}$ is scheduled to execute in the same threadblock in the next round, only the sample data in $\mathcal{Z}_{i,j,k'}$ and one block factor matrix ($\mathbf{C_{k'}}$) need to transmit to the shared memory. The other two block factor matrices ($\mathbf{A_i}$, $\mathbf{B_j}$) cached in the shared memory can be reused.

This example indicates that the indices of the sub-tensors can determine the correlation of the data needed for processing in tensors. Based on this observation, we consider the three indices of a sub-tensor as a bit string, and use Hamming distance to measure their correlation. The Hamming distance between two bit strings of equal length is the number of positions at which their corresponding bits are different.

A sub-tensor $\mathcal{Z}'$ is called the neighbor of a sub-tensor $\mathcal{Z}$ only if their distance is 1. To increase the chance of reusing the data stored in the on-chip memory thus the cache hit ratio, we consider a neighbor-based scheduling. In order for the data in the shared memory to have a higher correlation level, our scheduling algorithm abides by the following two principles:

Scheduling based on slice. If $\mathcal{Z}_{i,j,k}$ is assigned to a threadblock p $(1 \leq p \leq s)$ in the first round of an epoch, the set of sub-tensors on the slice $\mathcal{Z}_{::k}$ should be assigned to the same threadblock p in the following rounds of the epoch, thus a new task in the threadblock can reuse at least one block factor matrix $\mathbf{C_k}$ from the previous round.

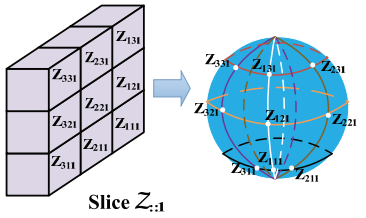


Fig. 7. Sphere based slice representation for slice $\mathcal{Z}_{::1}$ in Fig. 5.

Latitude first and Longitude second (LALO). We use a sphere to track the set of neighboring sub-tensors on a slice, and schedule tasks for the sub-tensors following LALO sequence.

To build a sphere for a slice, a column and a row on the slice correspond to a Longitude circle and Latitude circle of the sphere, respectively. Fig. 7 shows an example to map sub-tensors of the slice $\mathcal{Z}_{::1}$ in Fig. 5 to the sphere, where one-hop nodes on either a Longitude circle or a Latitude circle are neighbors.

In the first round of an epoch, we assign s sub-tensors to s threadblocks. In order for these sub-tensors to be not dependent, they must lie on s different slices. We use a bit to ***track*** whether a sub-tensor has been executed or not in an epoch. After the task assignment for the first round, if a sub-tensor $\mathcal{Z}_{i,j,k}$ is currently executed in a threadblock, to identify the sub-tensor to run in the same threadblock in the next round, we propose the scheduling of sub-tensors on the same slice following LALO:

- **Latitude first**: Check the next sub-tensor in the clockwise direction of the Latitude circle of $\mathcal{Z}_{i,j,k}$, if this sub-tensor is not executed before, schedule this sub-tensor to run with this threadblock in the next round; otherwise, check along the Longitude circle of $\mathcal{Z}_{i,j,k}$.
- **Longitude second**: Check the next sub-tensor in the counter-clockwise direction of the Longitude circle of $\mathcal{Z}_{i,j,k}$, if it is not executed before, schedule it to run with this threadblock in the next round. If this is not true, it indicates that all the sub-tensors on the slice have been scheduled to run once.

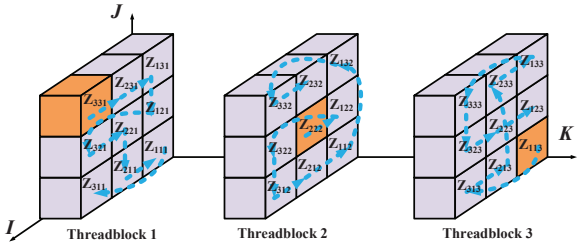


Fig. 8. Executing sequence of each threadblock.

Fig. 8 illustrates that the proposed memory-correlated scheduling algorithm can work with two basic steps following our two design principles. Assume the tangerine sub-tensors $\mathcal{Z}_{3,3,1}$, $\mathcal{Z}_{2,2,2}$, and $\mathcal{Z}_{1,1,3}$ are initially assigned in the first round to run in threadblocks 1, 2, and 3 following the Algorithm 2. First, according to the *Scheduling based on slice* principle, sub-tensors on slices $\mathcal{Z}_{::1}$, $\mathcal{Z}_{::2}$, and $\mathcal{Z}_{::3}$ will be assigned to the threadblocks 1, 2, and 3, in the following rounds of the epoch. Second, if $\mathcal{Z}_{i,j,k}$ is assigned to a threadblock k for the first round, the sequence of sub-tensors on the slice $\mathcal{Z}_{::k}$ will be assigned to run in the threadblock k in each round following the LALO principle: first following the clockwise direction of the Latitude circle of $\mathcal{Z}_{i,j,k}$ until all the sub-tensors on

the Latitude circle are tracked; then going to the Longitude circle to track the one-hop counter-clockwise neighboring sub-tensor; and then going to another Latitude circle, until all the sub-tensors on one sphere are tracked and one epoch ends. Fig. 9 shows the complete task execution sequence of all the three threadblocks for the example in Fig. 8, where the sub-tensors scheduled to execute in each round are independent.

round	0	1	2	3	4	5	6	7	8
block 1	$\mathcal{Z}_{331}$	$\mathcal{Z}_{231}$	$\mathcal{Z}_{131}$	$\mathcal{Z}_{121}$	$\mathcal{Z}_{321}$	$\mathcal{Z}_{221}$	$\mathcal{Z}_{211}$	$\mathcal{Z}_{111}$	$\mathcal{Z}_{311}$
block 2	$\mathcal{Z}_{222}$	$\mathcal{Z}_{122}$	$\mathcal{Z}_{322}$	$\mathcal{Z}_{312}$	$\mathcal{Z}_{212}$	$\mathcal{Z}_{112}$	$\mathcal{Z}_{132}$	$\mathcal{Z}_{332}$	$\mathcal{Z}_{232}$
block 3	$\mathcal{Z}_{113}$	$\mathcal{Z}_{313}$	$\mathcal{Z}_{213}$	$\mathcal{Z}_{233}$	$\mathcal{Z}_{133}$	$\mathcal{Z}_{333}$	$\mathcal{Z}_{323}$	$\mathcal{Z}_{223}$	$\mathcal{Z}_{123}$

Fig. 9. Tensor sequence in each round is independent.

With our design, only sub-tensors in the first round need to be assigned independently using the Algorithm 2. Then the principle LALO is followed to run in all the following rounds of a epoch. This makes our scheduling very simple.

In Theorem 1, we theoretically prove that the LALO principle provides the simple and independent task scheduling to enable parallel executions.

Theorem 1. *Given independent task assignments in the first round of an epoch, the tasks scheduled following the principle **LALO** guarantees that all the tasks executed in the s threadblocks in each following round are independent.*

Proof: Let $\mathcal{Z}_{i_1^t j_1^t k_1^t}, \mathcal{Z}_{i_2^t j_2^t k_2^t}, \cdots, \mathcal{Z}_{i_s^t j_s^t k_s^t}$ be the task list at the t-th round. As the initial assignment $\mathcal{Z}_{i_1^0 j_1^0 k_1^0}, \mathcal{Z}_{i_2^0 j_2^0 k_2^0}, \cdots, \mathcal{Z}_{i_s^0 j_s^0 k_s^0}$ is independent, we have $i_p^0 \neq i_q^0, j_p^0 \neq j_q^0, j_p^0 \neq j_q^0$ for $\forall p, q \in \{1, \cdots, s\}$.

When $t = 1$, the one-hop sub-tensor found on the latitude circle of $\mathcal{Z}_{i_r^0 j_r^0 k_r^0}$ along the clockwise direction is $\mathcal{Z}_{i_r^1 j_r^1 k_r^1}$ (where $r \in \{1, \cdots, s\}$), whose indexes are

$$\begin{cases} i_r^1 \leftarrow \begin{cases} i_r^0 - 1 & if \quad i_r^0 - 1 \neq 0 \\ s & otherwise \end{cases} \\ j_r^1 \leftarrow j_r^0 \\ k_r^1 \leftarrow k_r^0 \end{cases} \tag{13}$$

As $i_p^0 \neq i_q^0, j_p^0 \neq j_q^0, j_p^0 \neq j_q^0$, $\forall p, q \in \{1, \cdots, s\}$, we easily have $i_p^1 \neq i_q^1, j_p^1 \neq j_q^1, j_p^1 \neq j_q^1$. Therefore, the sub-tensors in round 1 are independent.

Suppose $t = m$ and $\mathcal{Z}_{i_1^m j_1^m k_1^m}, \mathcal{Z}_{i_2^m j_2^m k_2^m}, \cdots, \mathcal{Z}_{i_s^m j_s^m k_s^m}$ are independent tasks with $i_p^m \neq i_q^m, j_p^m \neq j_q^m, j_p^m \neq j_q^m$ for $p, q \in \{1, \cdots, s\}$. According to the **LALO** principle, if $(m \bmod s) \neq s - 1$, the next sub-tensor selected to run at the round $m + 1$ is found by moving clockwise one hop on the Latitude circle of the round m task:

$$\begin{cases} i_r^{m+1} \leftarrow \begin{cases} i_r^m - 1 & if \quad i_r^m - 1 \neq 0 \\ s & otherwise \end{cases} \\ j_r^{m+1} \leftarrow j_r^m \\ k_r^{m+1} \leftarrow r_r^m \end{cases} \tag{14}$$

If $(m \bmod s) = s - 1$, the next sub-tensor to run in the round $m + 1$ is selected by moving anticlockwise one hop along Longitude circle of the task of round m:

$$\begin{cases} i_r^{m+1} \leftarrow i_r^m \\ j_r^{m+1} \leftarrow \begin{cases} j_r^m - 1 & if \quad j_r^m - 1 \neq 0 \\ s & otherwise \end{cases} \\ k_r^{m+1} \leftarrow k_r^m \end{cases} \tag{15}$$

In either the case above, $\forall p, q \in \{1, \cdots, s\}$, as $i_p^m \neq i_q^m, j_p^m \neq j_q^m, j_p^m \neq j_q^m$, we have $i_p^{m+1} \neq i_q^{m+1}, j_p^{m+1} \neq j_q^{m+1}, j_p^{m+1} \neq j_q^{m+1}$ according to Eqs. (15) and (14). Therefore, the tasks scheduled in $m + 1$ round $\mathcal{Z}_{i_1^{m+1} j_1^{m+1} k_1^{m+1}}, \mathcal{Z}_{i_2^{m+1} j_2^{m+1} k_2^{m+1}}, \cdots, \mathcal{Z}_{i_s^{m+1} j_s^{m+1} k_s^{m+1}}$ are all independent. The proof completes. ■

In our scheduling, a large traffic tensor is gridded into $s \times s \times s$ sub-tensors. In each execution round, s threadblocks in GPU are assigned to run tasks for s sub-tensors. To reduce the memory access latency and communication cost, in GPU-TC, sub-tensors and their block factor matrices are transmitted from the global memory to the shared memory. After each round of execution, one block factor matrix from each sub-tensor needs to be transmitted back to the global memory and used by other threadblocks in the next round. When s becomes larger, more sub-tensors can be executed in parallel with each sub-tensor having a smaller size. However, a small sub-tensor introduces frequent data exchanges between the global memory and the shared memory, which further increases the communication cost and the execution time. In Section VIII-B, we present the performance results impacted by parameter s and set s according to the results.

According to GPU architecture, the operations in GPU-TC can be classified into two levels (Fig. 10), a high level that assigns tasks among threadblocks and a low level that handles the parallel execution using threads inside each threadblock. The high level has the following operations:

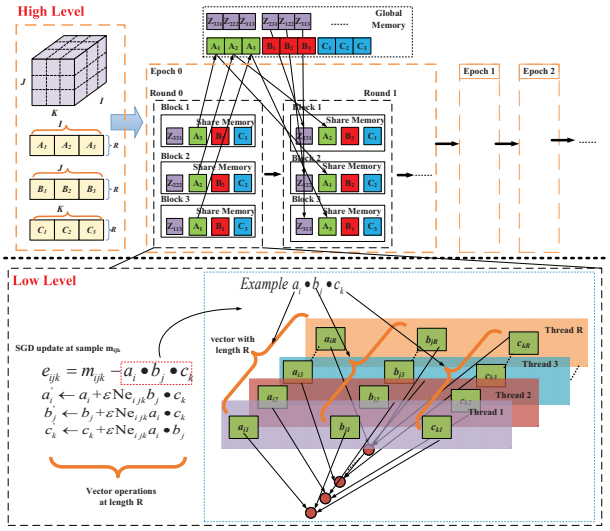


Fig. 10. Complete algorithm.

1) It grids the traffic tensor $\mathcal{M}$ into $s \times s \times s$ sub-tensors, and also divides the factor matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ into s block factor matrices $\mathbf{A_i}$, $\mathbf{B_j}$, and $\mathbf{C_k}$ with $1 \leq i, j, k \leq s$.
2) In the first round of an epoch, task assignment with Fisher-Yates shuffling in Algorithm 2 is applied to assign independent tasks to multiple threadblocks for concurrent execution. In the remaining rounds, the sub-tasks are scheduled to execute in the threadblocks following the memory-correlated scheduling algorithm in Section VII.
3) GPU-TC performs above operations iteratively in multiple epochs until reaching the convergence or the given number of epochs.

At the low level, each sub-task in the threadblock further involves multiple SGD updates. In each SGD update, rules (10)-(12) are applied to update one sample m_{ijk}. Specially, rules (10)-(12) can be executed through the following 4 operations $(I) : e_{ijk} = m_{ijk} - \mathbf{a}_i \bullet \mathbf{b}_j \bullet c_k$, $(II) : \mathbf{a}'_i \leftarrow \mathbf{a}_i + \varepsilon \mathrm{Ne}_{ijk} \mathbf{b}_j \bullet \mathbf{c}_k$, $(III) : \mathbf{b}'_j \leftarrow \mathbf{b}_j + \varepsilon \mathrm{Ne}_{jk} \mathbf{a}_i \bullet \mathbf{c}_k$, $(IV) : \mathbf{c}'_k \leftarrow \mathbf{c}_k + \varepsilon \mathrm{Ne}_{ijk} \mathbf{a}_i \bullet \mathbf{b}_j$,

where ε and N denote the learning rate and the total number of samples, respectively. Obviously, operations (I), (II), (III), and (IV) are vector operations with the length equal to the tensor rank R. As GPU are SIMD (Single Instruction Multiple Data) architectures [72], we can let R threads to work in parallel to perform the vector operations in SGD update. In Fig. 10, using $\mathbf{a}_i \bullet \mathbf{b}_j \bullet c_k$ as an example, we illustrate how R threads work in parallel when they handle a vector operation.

Instead of setting the learning rate ε to a fix value, following the bold driver method often used for gradient decent, we adapt the learning rate starting from an initial value ε_0: (1) Increasing the ε by a small percentage (say, 5%) whenever the loss decreases over an epoch, and (2) Drastically decreasing the ε (say, by 50%) if the loss increases. Within each epoch, the learning rate remains fixed. Specifically, we try learning rates $1, 1/2, 1/4, \cdots, 1/2^{d-1}$; the learning rate that gives the best result is selected as ε_0.

VIII. Performance Evaluations

We use three public traffic traces Abilene [35] and GÈANT [36] and a synthetic trace to evaluate the performance of our proposed GPU-TC. The synthetic trace is a large trace data which is generated through following steps. We firstly generate 3 factor matrices with their size being 2888×400, 1680×400, and 1440×400, respectively. Using these factor matrices, the synthetic tensor $\mathcal{M}$ can be calculated through Eq.(5).

In the experiment, we apply the proposed tensor completion scheme to recover the full traffic data from some samples. We set the default sampling ratio to 40%. Then, using the raw trace data as reference, we calculate the performance metrics by comparing the recovered data with the original data in the trace. We use three metrics to evaluate the recovery performance, which are defined as follows.

$$\textbf{Error(sample)} = \frac{\sqrt{\sum_{(i,j,k)\in\Omega}(m_{i,j,k} - \hat{m}_{i,j,k})^2}}{\sqrt{\sum_{(i,j,k)\in\Omega}(m_{i,j,k})^2}} \tag{16}$$

and

$$\textbf{Error(un-sample)} = \frac{\sqrt{\sum_{(i,j,k)\in\overline{\Omega}}(m_{i,j,k} - \hat{m}_{i,j,k})^2}}{\sqrt{\sum_{(i,j,k)\in\overline{\Omega}}(m_{i,j,k})^2}} \tag{17}$$

where m_{ijk} and $\hat{m}_{ijk}$ denote the raw data and the recovered data at (i,j,k)-th element of $\mathcal{M}$ where $1 \le i \le I$, $1 \le j \le J$ and $1 \le k \le K$.

Training Time: the average number of seconds taken to recover the traffic tensor by training the factor matrices.

Specially, **Error(sample)** is the relative error to evaluate the impact of tensor completion on the data elements with observed values already, and **Error(un-sample)** is error for the element locations with the values inferred from the tensor completion.

The proposed algorithm is implemented on two platforms with today's community hardware. 1) GPU platform: GPU-TC is implemented on a NVIDIA GPU. Specifically, we use the NVIDIA's CUDA SDK 9.2 on NVIDIA GeForce GTX 960. CUDA is started by a host thread of the CPU, which transfers the traffic data to the GPU device memory before invoking the GPU kernel codes. When the kernel execution finishes, the recovered traffic data are transferred back to the CPU main memory. 2) Multi-core CPU platform: We also implement the proposed algorithms in multi-core CPU, Intel®Xeon®CPU E5-2620 (2.00GHz) (totally 12 Cores).

We first investigate the impacts of parameters, based on which, we provide proper parameter setting for performance studies of GPU-TC. Then we compare the performance of our proposed algorithms with that of the state of art tensor completion algorithms and the matrix-based algorithms. Finally, we compare the speed of GPU-TC with that of a tensor completion running on a multi-core CPU.

As all tensor completion and matrix completion algorithms are executed iteratively to train the parameters needed, for a fair comparison, we adopt the same two stop conditions: 1) The difference in the recovery loss between two consecutive iterations is smaller than a given threshold value, set to 10^{-6} in this paper; 2) The maximum number of iterations is reached, and we set the threshold to 100 in this paper. The iteration process will continue until either of the two stop conditions is satisfied.

As GPU-TC and peer algorithms all have random components (e.g., the random initialization of the factor matrices), the results may vary across different runs. For more reliable evaluation, in each parameter setting, we repeat our experiments 40 times and present the average results in the experiments if not specifically stated.

A. CP rank R

From literature studies [73], [74], we know that the CP-rank impacts the recovery accuracy. Meanwhile, from Section VII, we know that the number of threads involved in each threadblock is determined directly by rank R.

To identify the proper CP-rank setting for our traffic tensor, we vary the CP-rank R and show the Error(un-sample) in Fig. 11. As expected, the error decreases with the increase of CP rank, as an under-estimated rank R makes the CP decomposition far from capturing the full structure of the traffic tensor. After R reaches 96 (Abilene), 125 (GÈANT) and 400 (Synthetic), further increasing R will not bring much gain in reducing the recovery error. Moreover, in GPU, the multiprocessor creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps [75]. To utilize warp model in GPU for more efficient execution, R is set to satisfy that $R \bmod 32 = 0$. Therefore, we set $R = 96$ (Abilene), $R = 128$ (GÈANT) and $R = 416$ (Synthetic) in the rest of experiments.

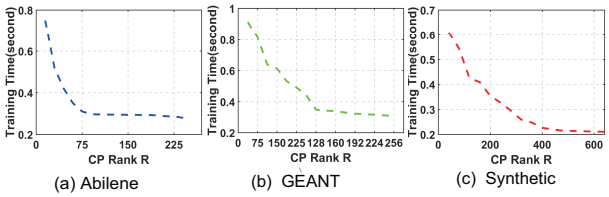


Fig. 11. The impact of CP rank.

B. Gridding parameter s

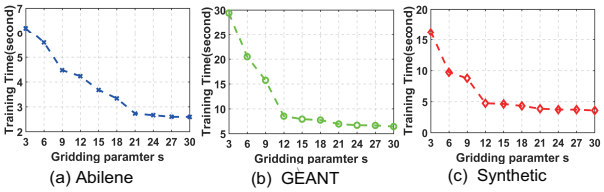


Fig. 12. Parameter s.

At the end of Section VII, we have discussed that there is a tradeoff to set parameter s. Fig. 12 shows the training time with

different s. When $s = 21$ (Abilene), $s = 12$ (GÈANT) and $s = 12$ (Synthetic), GPU-TC achieves the fastest speed to recover traffic tensor. Consequently, we set $s = 21$ (Abilene), $s = 12$ (GÈANT), and $s = 12$ (Synthetic) in the rest of experiments.

C. Effectiveness of independent task assignment

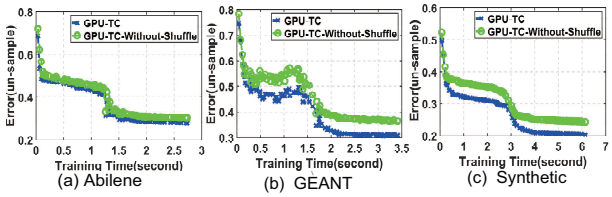


Fig. 13. Effectiveness of shuffle based task assignment.

Our GPU-TC updates the factor matrices iteratively. To make the task assignment independent in different epochs, we exploit Fisher-Yates shuffle in our task assignment algorithm in Section VI. To evaluate the effectiveness of the algorithm, we also implement GPU-TC-Without-Shuffle where the task execution sequence in each epoch remains the same. Fig. 13 shows the recovery performance with the elapsing of the training time. As the independent task execution makes the training of factor matrices more random in different epochs, GPU-TC achieves lower recovery errors than GPU-TC-Without-Shuffle. These results demonstrate that our shuffle-based task assignment algorithm can effectively improve the recovery accuracy.

D. Effectiveness of fast calculating using shared memory

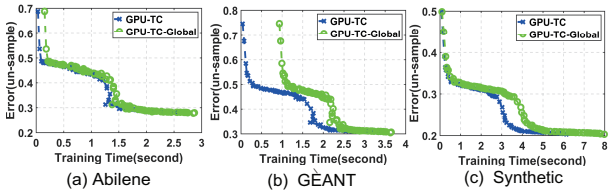


Fig. 14. Effectiveness of fast calculating using shared memory.

In GPU-TC, we exploit the shared memory of GPU architecture for fast tensor recovery. Specially, as the size of shared memory is limited, we propose a memory-correlated scheduling algorithm in Section VII to well utilize the shared memory. To evaluate the effectiveness of our mechanism in utilizing the shared memory, we also implement GPU-TC-Global where all its mechanisms are similar to GPU-TC except that the data are stored and accessed through the global memory instead of the shared memory. In Fig. 14, compared with GPU-TC-Global, our GPU-TC achieves much lower computation time with a larger speed, which demonstrates the effectiveness of our scheduling scheme in taking advantage of the shared memory of GPU architecture to maximize the parallelization gain for fast traffic recovery.

E. Comparison with other tensor completion algorithms

To compare the recovery performance with the literature tensor completion algorithms, we implemented four tensor completion algorithms which apply different methods to fit the CP model to incomplete data sets. The first is our *GPU-TC*. The second CP_{wopt} [47] solves a weighted least squares problem with a first-order optimization, while the third CP_{opt} solves a least-square optimization problem with a gradient-based optimization. The last tensor completion is implemented based on our Algorithm 1, which takes a Stochastic Gradient Descent (SGD) approach and is termed CP_{sgd}. CP_{wopt} and CP_{opt} are implemented using the Tensor Toolbox [76]. GPU-TC is implemented on GPU platform, while the other three algorithms are implemented on the same CPU platform without parallel CPU multi-core execution.

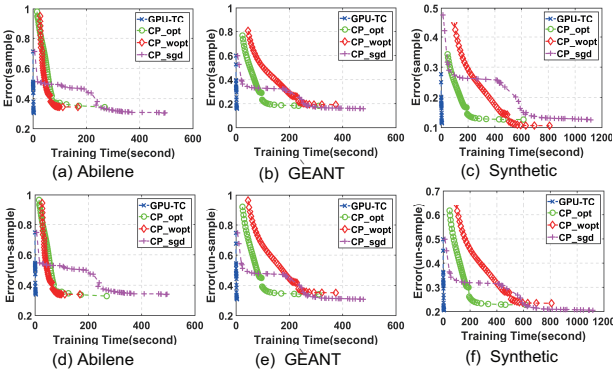


Fig. 15. Speed comparison with other tensor completion algorithms.

Fig. 15 shows the recovery accuracy with the elapsing of the training time. As expected, the Errors of the items with original data samples are smaller than those of un-sample with the item values completely inferred without original data, as the tensor completion process uses the sampled data to train the model. However, as we use random initialization for each run of the experiments, in some running cases, the model randomly falls into a local optimal that happens to be good with the un-sample data. In the figures, we present the average results of multiple runs of the experiments to alleviate the issue. Obviously, GPU-TC achieves significantly faster speed. GPU-TC is 60-180 times faster than other tensor completion implementations, which demonstrates that GPU-TC can well exploit the GPU architecture to achieve significant performance gain for tensor completion. The running speeds of the implemented algorithms are impacted by the size of the data sets, the data correlations in the data sets, as well as the mechanisms adopted in tensor completion algorithms. As a result, GPU-TC performs differently on the 3 datasets.

After 100 epochs, all algorithms converge. We find that our GPU-TC achieves slightly lower Error(un-sample). In GPU-TC, the iterative execution of SGD is based on gridding. The gridding-based execution and the shuffling process adopted in Algorithm 2 can more randomly select the observed data to better train the factor matrices and improve the tensor completion accuracy.

We take two types of sampling strategies: 1) Strategy 1 (randomly sampling): we randomly select samples in the whole tensor; 2) Strategy 2 (continuously sampling): we randomly select sampling locations in the first lateral slice, then use the same locations to take samples in the following lateral slices. The default sampling strategy in the experiments is strategy 1. Fig. 16 and Fig. 17 further show the performance results under different sampling strategies with different sampling ratios. We repeat our experiments 40 times. To show the statistic results, a box plot (a.k.a. box and whisker diagram) can display the distribution of data based on the five number summary:minimum, first quartile, median, third quartile, and maximum. We draw the curves to denote the average results. Fig. 16 and Fig. 17 achieve similar results.

Obviously the error ratio decreases with the increase of samples to obtain more observed data, as the final CP-model can more accurately capture the hidden structure of the traffic data to improve

the recovery accuracy. In these figures, we can see that our GPU-TC can achieve the highest recovery performance with the least error ratio under all different sampling ratios. This demonstrates that our GPU-TC algorithm has the good ability of capturing the global information in the traffic data to recover the missing data with a high accuracy.

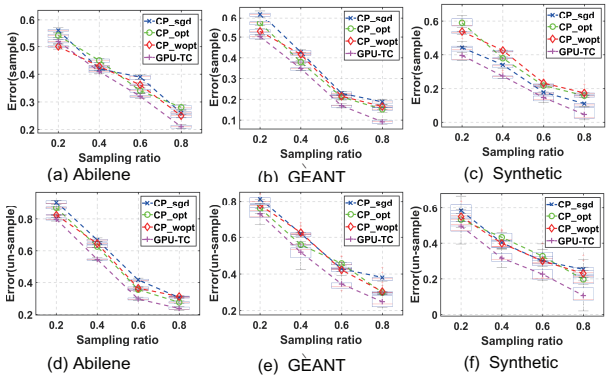


Fig. 16. Accuracy comparison with other tensor completion algorithms under sampling strategy 1.

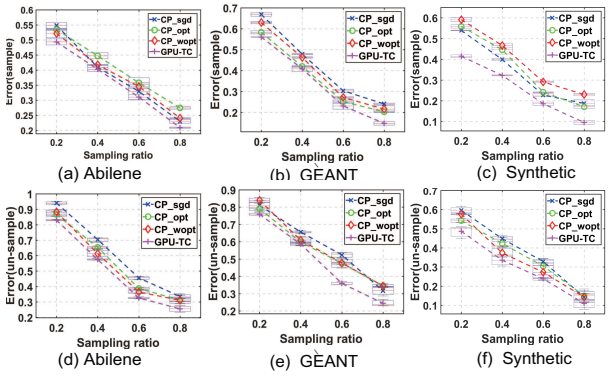


Fig. 17. Accuracy comparison with other tensor completion algorithms under sampling strategy 2.

From the probability theory and statistics, the cumulative distribution function (CDF) describes the probability for a real-valued random variable X to have a value less than or equal to x, and $F(x) = p(X \leq x)$. Therefore, to describe the distribution of individual relative error ratios $\frac{|m_{i,j,k} - \hat{m}_{i,j,k}|}{m_{i,j,k}}$ for all i, j, and k, Fig. 18 shows the CDF results of the individual error ratios. Obviously, our GPU-TC has much better recovery performance and can achieve low error ratio with a high probability.

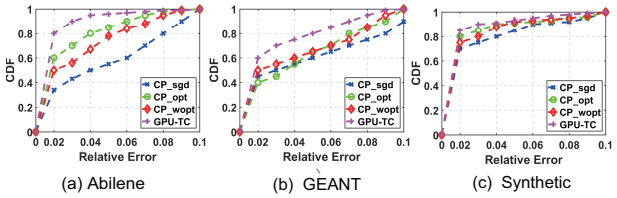


Fig. 18. CDF of individual relative error ratio.

F. Effectiveness of multiple GPUs

Our techniques are easily extended to a platform with multiple GPUs. To well utilize the resources of multiple GPUs, we first partition a large tensor into multiple sub-tensors and assign sub-tensor completion as sub-tasks to different GPUs to run in parallel. For the task assigned to a single GPU, the techniques in GPU-TC are exploited.

Fig. 19 shows the comparison of results with GPU-TC executed on the platforms equipped with single GPU and two GPUs, denoted as GPU-TC-SingleGPU and GPU-TC-MultiGPU, respectively. Rather than 2x acceleration, the speed of GPU-TC-MultiGPU is 1.6-1.8 times that of single GPU, due to the communication overhead.

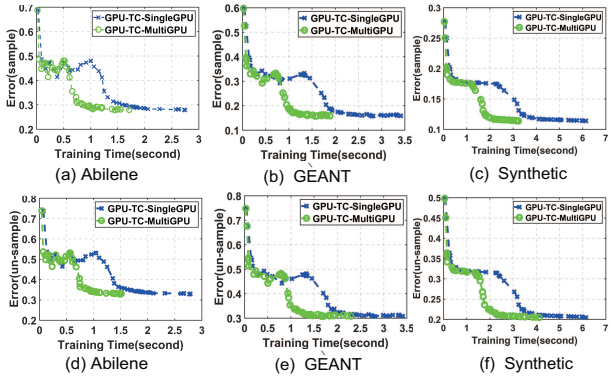


Fig. 19. Effectiveness of multiple GPUs.

G. Performance under different GPU versions: RTX and GTX

RTX and GTX are the two generations of NVIDIA platforms. RTX is an upgraded version of the GTX platform with powerful AI enhanced graphics technology. Fig. 20 presents the performance results under GTX960 and RTX2060, whose core frequencies are 1279MHz and 1354MHz respectively. As expected, RTX2060 is 1.3-1.5 times faster than GTX960 in completing the task.

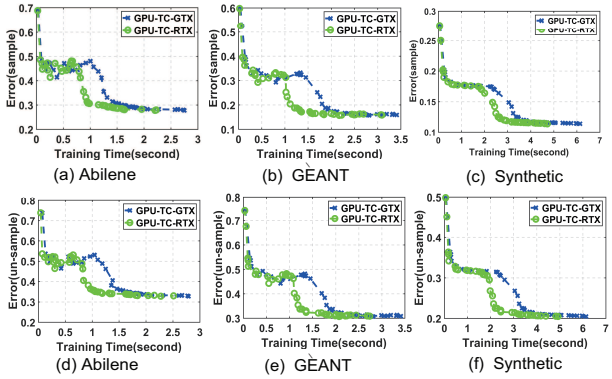


Fig. 20. Performance under different GPU versions: RTX and GTX.

H. Comparison with matrix completion algorithms

As current network monitoring data are mainly modeled as a matrix, the matrix-completion-based algorithm is proven to achieve the best data recovery performance. In this part, we further implement other five matrix completion algorithms (including NMF [41], $SRMF$ [19], $SRSVD$ [19], SVT [42], and $LMaFit$ [43]) for the performance comparison. All the five matrix completion algorithms are applied to the traffic matrix defined in $SRMF$ [19] with the row denoting OD pairs and the column denoting time interval. Fig. 21 utilizes a box plot to better show the statistical performance results. Obviously, in Fig. 21, our GPU-TC achieves

the best recovery performance among all the algorithms studied under all different metrics.

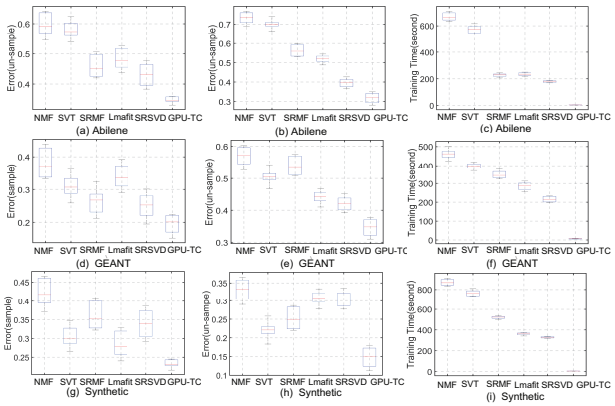


Fig. 21. Comparison with matrix completion algorithms.

The matrix-pattern based traffic recovery only needs to calculate two-dimension information, while GPU-TC needs to mine three dimensional information and thus requires more computation. Although the computation involved for tensor completion is larger than that under matrix-based recovery algorithms, Fig. 21(c), Fig. 21(f), and Fig. 21(i) show that GPU-TC is 50-180 times faster than other matrix completion implementations, as our proposed algorithms can take full advantage of GPU to achieve a large parallelization gain.

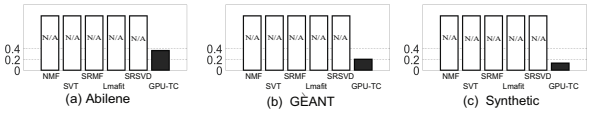


Fig. 22. Error(un-sample) under consecutive data missing.

In Fig. 22, we drop all consecutive measurements over 50 minutes, and then calculate the Error(un-sample) on the 50 minutes data. The consecutive data missing, obviously, results in the consecutive column missing in the traffic matrix. From the literature work, we know that the conventional matrix completion algorithms can only recover data if there is no completely empty row or column. As they do not have effect on these missing entries, in the Fig. 22, we use N/A to denote the none-effectiveness of these algorithms. GPU-TC, however, can recover the consecutively missing data with the Error(un-sample) only 0.36 (Abilene), 0.2 (GÈANT) and 0.13(Synthetic), respectively.

GPU-TC utilizes the information along three dimensions, while the matrix completion only considers the constraints along two particular dimensions. This is the key reason why our GPU-TC outperforms the matrix completion-based algorithms.

I. Comparison with multi-core CPU

To evaluate the performance gain of GPU and the efficiency of using our methods to manage GPU, we further implement CPU-Multi-Core. CPU-Multi-Core follows our proposed parallel tensor completion algorithm to partition the large tensor and run the sub-tasks over the Multi-core CPU platform. To further investigate how the number of CPU cores impacts the recovery performance, we vary the number of cores from 2 to 12. As expected, the performance gain achieved by parallel multi-core execution increases when more CPU cores are utilized in Table.I. Even though the number of CPU-cores reaches 12, our GPU-TC is still near 22 times faster than that under CPU-12Core. These results demonstrate that GPU-TC can exploit the massively-parallel processing power of GPU to bring significantly larger parallelization gain compared to multi-core CPU.

TABLE I
SPEED COMPARISON

	Abilene		GÈANT		Simulate	
	Time (second)		Time (second)		Time (second)	
	means	variance	means	variance	means	variance
GPU-TC	2.73	0.14	3.41	0.18	4.12	0.09
CPU-2Core	410.6	0.13	420.5	0.15	510.2	0.11
CPU-3Core	395.2	0.15	401.2	0.16	470.2	0.12
CPU-4Core	320.5	0.15	350.2	0.19	420.8	0.09
CPU-5Core	256.9	0.16	305.2	0.14	370.2	0.16
CPU-6Core	210.5	0.09	265.4	0.20	320.1	0.08
CPU-7Core	185.3	0.18	205.8	0.16	256.7	0.10
CPU-8Core	152.8	0.12	167.2	0.21	208.1	0.14
CPU-9Core	112.5	0.15	126.5	0.13	167.3	0.16
CPU-10Core	101.6	0.19	108.5	0.16	143.2	0.15
CPU-11Core	90.6	0.16	92.6	0.17	108.2	0.12
CPU-12Core	60.5	0.16	75.2	0.16	90.4	0.13

IX. CONCLUSION

We present GPU-TC, a high performance scheme for accurate and high-speed tensor completion with Graphics Processing Units (GPUs). GPU-TC exploits the factorization structure of the tensor completion solution to elegantly grid the tensor into small sub-tensors, which allows the exploration of the massively-parallel processing power of GPU for fast processing. To speed up the tensor completion process and increase the data recovery accuracy, we propose a task assignment algorithm based on Fisher-Yates shuffle to achieve independent task execution across different iteration epochs. To well exploit the GPU's special architecture to maximize its parallelism gain, GPU-TC applies a novel sphere structure and memory-correlated scheduling algorithm to schedule the execution sequence of sub-tensors in theadblocks to increase the cache hit ratio in the shared memory. We have implemented our algorithms over multiple platforms and compared their performance with those of the current state of tensor completion and matrix completion algorithms. The experiments with real traffic traces demonstrate the effectiveness and efficiency of our well-designed GPU-TC even on today's commodity hardware.

Although we present our scheme (GPU-TC) using the Internet traffic data recovery as an example, GPU-TC is flexible to apply in various applications with the tensor completion process to accurately and quickly infer the missing data.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 61972144, 61572184, 61725206, and 61976087, in part by the Hunan Provincial Natural Science Foundation of China under Grant 2017JJ1010, in part by U.S. NSF under Grant ECCS 78929 and CNS 1526843, in part by the Open Project Funding (CARCH201809) of State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, in part by CERNET Innovation Project under Grant NGII20190118, in part by Open Foundation of State key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) under Grant SKLNST-2018-1-20, and in part by the Peng Cheng Laboratory Project of Guangdong Province under Grant PCL2018KP004.

REFERENCES

[1] P. Tune and M. Roughan, "Spatiotemporal traffic matrix synthesis," in *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 579–592, ACM, 2015.

[2] I. Cunha, R. Teixeira, D. Veitch, and C. Diot, "Predicting and tracking internet path changes," in *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 122–133, ACM, 2011.

[3] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 139–152, ACM, 2015.

[4] A. Adams, P. Lapukhov, and J. H. Zeng, "Netno-rad: Troubleshooting networks via end-to-end probing," *Facebook White Paper, available online at: https://code. facebook. com/posts/1534350660228025/netnorad-troubleshooting-networks-via-end-to-endprobing*, 2017.

[5] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 75–86, ACM, 2008.

[6] Y. Peng, J. Yang, C. Wu, C. Guo, C. Hu, and Z. Li, "detector: a topology-aware monitoring system for data center networks," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pp. 55–68, 2017.

[7] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices," in *In SIGCOMM'09:Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pp. 267–278, 2009.

[8] K. Xie, C. Peng, X. Wang, G. Xie, J. Wen, J. Cao, D. Zhang, and Z. Qin, "Accurate recovery of internet traffic data under variable rate measurements," *IEEE/ACM Transactions on Networking*, vol. 26, no. 3, pp. 1137–1150, 2018.

[9] K. Xie, L. Wang, X. Wang, G. Xie, G. Zhang, D. Xie, and J. Wen, "Sequential and adaptive sampling for matrix completion in network monitoring systems," in *IEEE INFOCOM*, 2015.

[10] C. R. Kalmanek, S. Misra, and Y. R. Yang, *Guide to reliable internet services and applications*. Springer Science & Business Media, 2010.

[11] K. Xie, X. Li, X. Wang, J. Cao, G. Xie, J. Wen, D. Zhang, and Z. Qin, "On-line anomaly detection with high accuracy," *IEEE/ACM Transactions on Networking*, vol. 26, pp. 1222–1235, June 2018.

[12] K. Xie, X. Li, X. Wang, G. Xie, J. Wen, and D. Zhang, "Graph based tensor recovery for accurate internet anomaly detection," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 1502–1510, IEEE, 2018.

[13] K. Xie, X. Li, X. Wang, G. Xie, J. Wen, J. Cao, and D. Zhang, "Fast tensor factorization for accurate internet anomaly detection," *IEEE/ACM transactions on networking*, vol. 25, no. 6, pp. 3794–3807, 2017.

[14] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, "Structural analysis of network traffic flows," in *ACM SIGMETRICS*, 2003.

[15] Y. Zhang, M. Roughan, C. Lund, and D. Donoho, "Estimating point-to-point and point-to-multipoint traffic matrices: An information-theoretic approach," in *IEEE/ACM Trans. Netw*, pp. 947–960, 2005.

[16] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *Acm Sigcomm Computer Communication Review*, vol. 34, no. 4, pp. 219–230, 2004.

[17] Y.Vardi, "Network tomography," *J. Amer. Statist. Assoc.*, vol. vol. 91, no. 433,, p. pp. 365–377, 1996.

[18] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," *ACM IMW*, 2002.

[19] M. Roughan, Y. Zhang, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices (extended version)," *Networking IEEE/ACM Transactions on*, vol. 20, no. 3, pp. 662 – 676, 2012.

[20] M. Mardani and G. Giannakis, "Robust network traffic estimation via sparsity and low rank," in *IEEE ICASSP*, 2013.

[21] R. Du, C. Chen, B. Yang, and X. Guan, "Vanet based traffic estimation: A matrix completion approach," in *IEEE GLOBECOM*, 2013.

[22] G. Gürsun and M. Crovella, "On traffic matrix completion in the internet," in *ACM IMC 2012*.

[23] Y.-C. Chen, L. Qiu, Y. Zhang, G. Xue, and Z. Hu, "Robust network compressive sensing," in *ACM MobiCom*, 2014.

[24] T. Kolda and B. Bader, "The tophits model for higher-order web link analysis," in *Workshop on link analysis, counterterrorism and security*, vol. 7, pp. 26–29, 2006.

[25] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell, "Toward an architecture for never-ending language learning.," in *AAAI*, vol. 5, p. 3, 2010.

[26] C. J. Appellof and E. Davidson, "Strategies for analyzing data from video fluorometric monitoring of liquid chromatographic effluents," *Analytical Chemistry*, vol. 53, no. 13, pp. 2053–2056, 1981.

[27] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *Signal Processing Magazine, IEEE*, vol. 32, no. 2, pp. 145–163, 2015.

[28] S. Aja-Fernández, R. de Luis Garcia, D. Tao, and X. Li, *Tensors in image processing and computer vision*. Springer Science & Business Media, 2009.

[29] H. Tan, G. Feng, J. Feng, W. Wang, Y.-J. Zhang, and F. Li, "A tensor-based method for missing traffic data completion," *Transportation Research Part C: Emerging Technologies*, vol. 28, pp. 15–27, 2013.

[30] H. Tan, J. Feng, G. Feng, W. Wang, and Y.-J. Zhang, "Traffic volume data outlier recovery via tensor model," *Mathematical Problems in Engineering*, vol. 2013, 2013.

[31] H. Tan, Y. Wu, G. Feng, W. Wang, and B. Ran, "A new traffic prediction method based on dynamic tensor completion," *Procedia-Social and Behavioral Sciences*, vol. 96, pp. 2431–2442, 2013.

[32] A. H. Phan and A. Cichocki, "Parafac algorithms for large-scale problems," *Neurocomputing*, vol. 74, no. 11, pp. 1970–1984, 2011.

[33] A. Beutel, P. P. Talukdar, A. Kumar, C. Faloutsos, E. E. Papalexakis, and E. P. Xing, "Flexifact: Scalable flexible factorization of coupled tensors on hadoop.," in *SDM*, pp. 109–117, SIAM, 2014.

[34] S. Smith, J. Park, and G. Karypis, "An exploration of optimization algorithms for high performance tensor completion," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 31, IEEE Press, 2016.

[35] "The abilene observatory data collections. http://abilene. internet2.edu /observatory/data-collections.html.,"

[36] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, 2006.

[37] K. Xie, C. Peng, X. Wang, G. Xie, and J. Wen, "Accurate recovery of internet traffic data under dynamic measurements," in *IEEE INFOCOM*, 2017.

[38] K. Xie, L. Wang, X. Wang, G. Xie, J. Wen, and G. Zhang, "Accurate recovery of internet traffic data: A tensor completion approach," in *IEEE INFOCOM*, 2016.

[39] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory*, vol. 52, no. 2, pp. 489–509, 2006.

[40] J. Haupt, W. U. Bajwa, M. Rabbat, and R. Nowak, "Compressed sensing for networked data," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 92–101, 2008.

[41] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, pp. 556–562, 2001.

[42] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[43] Z. Wen, W. Yin, and Y. Zhang, "Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm," *Mathematical Programming Computation*, vol. 4, no. 4, pp. 333–361, 2012.

[44] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, vol. 9, no. 6, pp. 717–772, 2009.

[45] R. H. Keshavan, A. Montanari, and S. Oh, "Matrix completion from a few entries," *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp. 2980–2998, 2010.

[46] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 208–220, 2013.

[47] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data," *Chemometrics and Intelligent Laboratory Systems*, vol. 106, no. 1, pp. 41–56, 2011.

[48] E. Acar and T. G. Dunlavy, Daniel M.and Kolda, "A scalable optimization approach for fitting canonical tensor decompositions," *Journal of Chemometrics*, vol. 25, no. 2, p. 67–86, 2011.

[49] S. Gandy, B. Recht, and I. Yamada, "Tensor completion and low-n-rank tensor recovery via convex optimization," *Inverse Problems*, vol. 27, no. 2, p. 025010, 2011.

[50] J. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[51] R. A. Harshman, "Foundations of the parafac procedure: Models and conditions for an" explanatory" multi-modal factor analysis," *UCLA Working Papers in Phonetics*, vol. 16, no. 1, p. 84, 1970.

[52] L. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[53] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 316–324, ACM, 2012.

[54] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "Haten2: Billion-scale tensor decompositions," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pp. 1047–1058, IEEE, 2015.

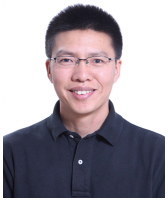
[55] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "Splatt: Efficient and parallel sparse tensor-matrix multiplication," in *Parallel and Distributed*

Processing Symposium (IPDPS), 2015 IEEE International, pp. 61–70, IEEE, 2015.
[56] J. H. Choi and S. Vishwanathan, "Dfacto: Distributed factorization of tensors," in *Advances in Neural Information Processing Systems*, pp. 1296–1304, 2014.
[57] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 69–77, ACM, 2011.
[58] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin, "A fast parallel sgd for matrix factorization in shared memory systems," in *Proceedings of the 7th ACM conference on Recommender systems*, pp. 249–256, ACM, 2013.
[59] H. Yun, H.-F. Yu, C.-J. Hsieh, S. Vishwanathan, and I. Dhillon, "Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion," *Proceedings of the VLDB Endowment*, vol. 7, no. 11, pp. 975–986, 2014.
[60] J. Oh, W.-S. Han, H. Yu, and X. Jiang, "Fast and robust parallel sgd matrix factorization," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 865–874, ACM, 2015.
[61] X. Xie, W. Tan, L. L. Fong, and Y. Liang, "Cumf_sgd: Fast and scalable matrix factorization," *arXiv preprint arXiv:1610.05838*, 2016.
[62] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications.," *Siam Review*, vol. 51, no. 3, pp. 455–500, 2009.
[63] L. De Lathauwer, "Blind separation of exponential polynomials and the decomposition of a tensor in rank-(l_r,l_r,1) terms," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 4, pp. 1451–1474, 2011.
[64] B. W. Bader and T. G. Kolda, "Algorithm 862: Matlab tensor classes for fast algorithm prototyping," *ACM Transactions on Mathematical Software (TOMS)*, vol. 32, no. 4, pp. 635–653, 2006.
[65] K. Xie, L. Wang, X. Wang, G. Xie, J. Wen, G. Zhang, J. Cao, D. Zhang, K. Xie, X. Wang, *et al.*, "Accurate recovery of internet traffic data: A sequential tensor completion approach," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 2, pp. 793–806, 2018.
[66] K. Xie, X. Wang, X. Wang, Y. Chen, G. Xie, Y. Ouyang, J. Wen, J. Cao, and D. Zhang, "Accurate recovery of missing network measurement data with localized tensor completion," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2019.
[67] R. A. Fisher, F. Yates, *et al.*, "Statistical tables for biological, agricultural and medical research.," *Statistical tables for biological, agricultural and medical research.*, no. Ed. 3., 1949.
[68] R. Durstenfeld, "Algorithm 235: random permutation," *Communications of the ACM*, vol. 7, no. 7, p. 420, 1964.
[69] C. Bays and S. Durham, "Improving a poor random number generator," *ACM Transactions on Mathematical Software (TOMS)*, vol. 2, no. 1, pp. 59–64, 1976.
[70] P. Diaconis, R. Graham, and W. M. Kantor, "The mathematics of perfect shuffles," *Advances in applied mathematics*, vol. 4, no. 2, pp. 175–196, 1983.
[71] "Randomness is hard: learning about the fisher-yates shuffle algorithm random number generation," *https://medium.com/@oldwestaction/randomness-is-hard-e085decbcbb2*.
[72] D. Song and S. Chen, "Exploiting simd for complex numerical predicates," in *Data Engineering Workshops (ICDEW), 2016 IEEE 32nd International Conference on*, pp. 143–149, IEEE, 2016.
[73] Q. Zhao, L. Zhang, and A. Cichocki, "Bayesian cp factorization of incomplete tensors with automatic rank determination," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 37, no. 9, pp. 1751–1763, 2015.
[74] E. S. Allman, P. D. Jarvis, J. A. Rhodes, and J. G. Sumner, "Tensor rank, invariants, inequalities, and applications," *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 3, pp. 1014–1045, 2013.
[75] C. Nvidia, "Nvidia cuda c programming guide," *Nvidia Corporation*, vol. 120, no. 18, p. 8, 2011.
[76] B. W.Bader, T. G.Kolda, *et al.*, "Matlab tensor toolbox version 2.5," January 2012.

Kun Xie received the Ph.D. degree in computer application from Hunan University, Changsha, China, in 2007. She is currently a Professor with Hunan University, and Peng Cheng Laboratory. She has published over 60 articles in major journals and conference proceedings, including IEEE/ACM TON, IEEE TMC, IEEE TC, IEEE TWC, IEEE TSC, SIGMOD, INFOCOM, ICDCS, SECON, DSN, and IWQoS. Her research interests include network measurement, network security, big data, and AI.

Yuxiang Chen is now a PhD candidate in Hunan University. His research interests include parallel computing and matrix/tensor completion.

Xin Wang (M'1 / ACM'4) received the Ph.D. degree in electrical and computer engineering from Columbia University, USA. She is currently an Associate Professor with the Department of Electrical and Computer Engineering, The State University of New York at Stony Brook, USA. Her research interests include algorithm and protocol design in wireless networks and communications, mobile and distributed computing, and networked sensing and detection. She was a member of ACM in 2004. She received the NSF Career Award in 2005 and the ONR Challenge Award in 2010.



Gaogang Xie received the B.S. degree in physics, the M.S. and Ph.D. degrees in computer science from Hunan University in 1996, 1999, and 2002, respectively. He is currently a Professor with the Computer Network Information Center (CNIC), Chinese Academy of Sciences (CAS), and the University of Chinese Academy of Sciences and the Vice President of CNIC. His research interests include Internet architecture, packet processing and forwarding, and Internet measurement.

Jiannong Cao (M'93-SM'05-FM'14) received the Ph.D degree in computer science from Washington State University, Pullman, WA, USA, in 1990. Dr. is currently the Otto Poon Charitable Foundation Professor in Data Science, Chair Professor of Department of Computing at The Hong Kong Polytechnic University, Hong Kong. He is also the director of the Internet and Mobile Computing Lab (IMCL) in the department and the director of University's Research Facility in Big Data Analytics (UBDA). His research interests include parallel and distributed computing, wireless sensing and networks, pervasive and mobile computing, and big data and cloud computing.

Jigang Wen received the Ph.D. degrees in computer application from Hunan University, China, in 2011. He was a Research Assistant with the Department of Computing, The Hong Kong Polytechnic University, from 2008 to 2010. He is currently with the Computer Network Information Center (CNIC), Chinese Academy of Sciences. His research interests include wireless networks and mobile computing, and high-speed network measurement and management.

Guangming Yang is a senior expert from Research Institute of Intelligent Network and Terminal of China Telecom. His research interests include IP RAN/STN (smart transport network), network monitoring and SLA commitment.

SunJiaqi Sun is a network engineer from Research Institute of Intelligent Network and Terminal of China Telecom. Her research interests include IP RAN/STN (smart transport network), network monitoring and SLA commitment.