Supplementary Material for GTA

Sejoon Oh*, Namyong Park†, Jun-Gi Jang#, Lee Sael‡, U Kang§

*#§‡ *Seoul National University, Korea*

† *Carnegie Mellon University, USA*

* ohhenrie@snu.ac.kr † namyongp@cs.cmu.edu # elnino4@snu.ac.kr ‡ saellee@gmail.com § ukang@snu.ac.kr

Abstract

In this supplementary material, we first suggest how to measure the reconstruction error of GTA using GPUs. Moreover, we provide full proofs of the row-wise update rules of GTA-PART and GTA-FULL as well as theoretical complexities of GTA-PART and GTA-FULL. Finally, we offer additional experimental results of GTA-PART and GTA-FULL in terms of speed and accuracy for synthetic and real-world tensors, where all contents were introduced in the main paper.

1 Measuring reconstruction error using GPUs

1.1 Partially Observable Tucker Factorization (POTF)

Given a tensor $\mathcal{X}$, factor matrices $\mathbf{A}^{(n)}$, and a core tensor $\mathcal{G}$, reconstruction error for POTF is defined by the following.

$$\sum_{\forall (i_1,\dots,i_N)\in\Omega}\left(\mathcal{X}_{(i_1,\dots,i_N)} - \sum_{\forall (j_1,\dots,j_N)\in\mathcal{G}} \mathcal{G}_{(j_1,\dots,j_N)} \prod_{n=1}^{N} a^{(n)}_{i_n j_n}\right)^2 \tag{1}$$

Equation (1) is computed only for observable entries Ω, and an observable entry is independent of each other in terms of calculating (1). Hence, we can utilize nonzero-based parallelization on GPUs for computing (1). Specifically, all observable entries of $\mathcal{X}$ are uniformly distributed to T_2 GPU cores, and each GPU core calculates partial reconstruction error for its allocated observable entries. Finally, CPU collects all partial reconstruction error from GPU cores and computes summation of them. The time complexity of computing reconstruction error for POTF is $O(N|\Omega|J^N/T_2)$ since the total number of computation for computing (1) is $O(N|\Omega|J^N)$, and it is parallelized by T_2 GPU cores.

1.2 Fully Observable Tucker Factorization (FOTF)

Given a tensor $\mathcal{X}$, factor matrices $\mathbf{A}^{(n)}$, and a core tensor $\mathcal{G}$, reconstruction error for FOTF is defined by the following.

$$\sum_{\forall (i_1,\dots,i_N)\in\Omega}\left(\mathcal{X}_{(i_1,\dots,i_N)} - \sum_{\forall (j_1,\dots,j_N)\in\mathcal{G}} \mathcal{G}_{(j_1,\dots,j_N)} \prod_{n=1}^{N} a^{(n)}_{i_n j_n}\right)^2 + \sum_{\forall (i_1,\dots,i_N)\notin\Omega}\left(\sum_{\forall (j_1,\dots,j_N)\in\mathcal{G}} \mathcal{G}_{(j_1,\dots,j_N)} \prod_{n=1}^{N} a^{(n)}_{i_n j_n}\right)^2 \tag{2}$$

As the second term in Equation (2), it is not straightforward to parallelize (2) using GPUs, we transform (2) into by the following.

$$\begin{aligned}\leftrightarrow & \sum_{\forall\alpha\in\Omega}\left(\mathcal{X}_\alpha - \sum_{\forall\beta\in\mathcal{G}}\mathcal{G}_\beta\prod_{n=1}^{N}a^{(n)}_{i_n j_n}\right)^2 + \sum_{\forall\alpha\in\mathcal{X}}\left(\sum_{\forall\beta\in\mathcal{G}}\mathcal{G}_\beta\prod_{n=1}^{N}a^{(n)}_{i_n j_n}\right)^2 - \sum_{\forall\alpha\in\Omega}\left(\sum_{\forall\beta\in\mathcal{G}}\mathcal{G}_\beta\prod_{n=1}^{N}a^{(n)}_{i_n j_n}\right)^2 \\ \leftrightarrow & \sum_{\forall\alpha\in\Omega}\left(\mathcal{X}_\alpha^2 - 2\mathcal{X}_\alpha\sum_{\forall\beta\in\mathcal{G}}\mathcal{G}_\beta\prod_{n=1}^{N}a^{(n)}_{i_n j_n}\right) + \sum_{\forall\alpha\in\mathcal{X}}\left(\sum_{\forall\beta\in\mathcal{G}}\mathcal{G}_\beta\prod_{n=1}^{N}a^{(n)}_{i_n j_n}\right)^2\end{aligned} \tag{3}$$

Note that α and β indicate an entry of tensor $\mathcal{X}$ and $\mathcal{G}$, respectively. The first term of the second line in Equation (3) is easily parallelized on GPUs similarly to the POTF case (Section 1.1). Moreover, we can efficiently compute the second term of the second line in Equation (3) by the factorization and caching technique used for GTA-FULL (refer to the main paper). The time complexity of computing reconstruction error for FOTF is $O(NIJ^{2N}/T_1 + N|\Omega|J^N/T_2)$ since the first term's time complexity is $O(N|\Omega|J^N/T_2)$ and the second term's time complexity is $O(NIJ^{2N}/T_1)$.

2 Full Proofs of the Row-wise Update Rules

Definition 1 (Partially Observable Tucker Factorization). Given a tensor $\mathcal{X}$ $(\in \mathbb{R}^{I_1\times\dots\times I_N})$ with observable entries Ω, the goal of partially observable Tucker factorization of $\mathcal{X}$ is to find factor matrices $\mathbf{A}^{(n)}$ $(\in \mathbb{R}^{I_n\times J_n})$ and a core tensor $\mathcal{G}$ $(\in \mathbb{R}^{J_1\times\dots\times J_N})$, which minimize Equation (4).

$$L(\mathcal{G},\mathbf{A}^{(1)},\dots,\mathbf{A}^{(N)}) = \sum_{\forall (i_1,\dots,i_N)\in\Omega}\left(\mathcal{X}_{(i_1,\dots,i_N)} - \sum_{\forall (j_1,\dots,j_N)\in\mathcal{G}} \mathcal{G}_{(j_1,\dots,j_N)} \prod_{n=1}^{N} a^{(n)}_{i_n j_n}\right)^2 + \lambda\sum_{n=1}^{N}\|\mathbf{A}^{(n)}\|^2 \tag{4}$$

Theorem 1 (Row-wise Update Rule for Factor Matrices of* GTA-PART*).

$$\underset{[a^{(n)}_{i_n 1},...,a^{(n)}_{i_n J_n}]}{\arg\min} \quad L(\mathcal{G}, \mathbf{A}^{(1)}, ..., \mathbf{A}^{(N)}) = \mathbf{c}^{(n)}_{i_n:} \times [\mathbf{B}^{(n)}_{i_n} + \lambda \mathbf{I}_{J_n}]^{-1} \tag{5}$$

$$\text{where the } (j_1, j_2)\text{th entry of } \mathbf{B}^{(n)}_{i_n} (\in \mathbb{R}^{J_n \times J_n}) : \sum_{\forall (i_1,...,i_N) \in \Omega^{(n)}_{i_n}} \delta^{(n)}_{(i_1,...,i_N)}(j_1) \delta^{(n)}_{(i_1,...,i_N)}(j_2), \tag{6}$$

$$\text{the } j\text{th entry of } \mathbf{c}^{(n)}_{i_n:} (\in \mathbb{R}^{J_n}) : \sum_{\forall (i_1,...,i_N) \in \Omega^{(n)}_{i_n}} \mathcal{X}_{(i_1,...,i_N)} \delta^{(n)}_{(i_1,...,i_N)}(j), \tag{7}$$

$$\text{the } j\text{th entry of } \delta^{(n)}_{(i_1,...,i_N)} (\in \mathbb{R}^{J_n}) : \sum_{\forall (j_1 \cdots j_{n-1}, j, j_{n+1} \cdots j_N) \in \mathcal{G}} \mathcal{G}_{(j_1 \cdots j_{n-1}, j, j_{n+1} \cdots j_N)} \prod_{k \neq n} a^{(k)}_{i_k j_k}. \tag{8}$$

Proof 1.

$$\frac{\partial L}{\partial a^{(n)}_{i_n j_n}} = 0, \forall j_n, 1 \le j_n \le J_n$$

$$\Leftrightarrow \sum_{\forall (i_1,...,i_N) \in \Omega^{(n)}_{i_n}} \left(\left(\mathcal{X}_{(i_1,...,i_N)} - \sum_{\forall (j_1,...,j_N) \in \mathcal{G}} \mathcal{G}_{(j_1,...,j_N)} \prod_{n=1}^{N} a^{(n)}_{i_n j_n} \right) \times \left(- \delta^{(n)}_{(i_1,...,i_N)}(j_n) \right) \right) + \lambda a^{(n)}_{i_n j_n} = 0$$

$$\Leftrightarrow \sum_{\forall (i_1,...,i_N) \in \Omega^{(n)}_{i_n}} \left(\left(\sum_{t=1}^{J_n} \delta^{(n)}_{(i_1,...,i_N)}(t) a^{(n)}_{i_n t} \right) \times \left(\delta^{(n)}_{(i_1,...,i_N)}(j_n) \right) \right) + \lambda a^{(n)}_{i_n j_n} = \sum_{\forall (i_1,...,i_N) \in \Omega^{(n)}_{i_n}} \left(\mathcal{X}_{(i_1,...,i_N)} \delta^{(n)}_{(i_1,...,i_N)}(j_n) \right) \tag{9}$$

$\sum_{\forall (i_1,...,i_N) \in \Omega^{(n)}_{i_n}} \left(\left(\sum_{t=1}^{J_n} \delta^{(n)}_{(i_1,...,i_N)}(t) a^{(n)}_{i_n t} \right) \times \left(\delta^{(n)}_{(i_1,...,i_N)}(j_n) \right) \right)$ is expressed as an inner product of the following vectors.

$$\text{Row vector } (1 \times J_n) : [a^{(n)}_{i_n 1}, ..., a^{(n)}_{i_n J_n}]$$

$$\text{Column vector } (J_n \times 1) : \begin{bmatrix} \sum_{\forall (i_1,...,i_N) \in \Omega^{(n)}_{i_n}} \delta^{(n)}_{(i_1,...,i_N)}(1) \delta^{(n)}_{(i_1,...,i_N)}(j_n) \\ \vdots \\ \sum_{\forall (i_1,...,i_N) \in \Omega^{(n)}_{i_n}} \delta^{(n)}_{(i_1,...,i_N)}(J_n) \delta^{(n)}_{(i_1,...,i_N)}(j_n) \end{bmatrix}$$

If we vary j_n from 1 to J_n, the row vector is fixed as $a^{(n)}_{i_n:}$ and the column vector differs.
Thus, we can integrate each column vector into a matrix $\mathbf{B}^{(n)}_{i_n} (\in \mathbb{R}^{J_n \times J_n})$

$$\text{where the } (j_1, j_2)\text{th entry of } \mathbf{B}^{(n)}_{i_n} (\in \mathbb{R}^{J_n \times J_n}) : \sum_{\forall (i_1,...,i_N) \in \Omega^{(n)}_{i_n}} \delta^{(n)}_{(i_1,...,i_N)}(j_1) \delta^{(n)}_{(i_1,...,i_N)}(j_2).$$

λ term is simply transformed into $\lambda \mathbf{I}_{J_n}$, where $\mathbf{I}_{J_n}$ is an identity matrix $(\in \mathbb{R}^{J_n \times J_n})$.
In the same way, the right part of Equation (9) is integrated as $\mathbf{c}^{(n)}_{i_n:} (\in \mathbb{R}^{J_n})$

$$\text{where the } j\text{th entry of } \mathbf{c}^{(n)}_{i_n:} (\in \mathbb{R}^{J_n}) : \sum_{\forall (i_1,...,i_N) \in \Omega^{(n)}_{i_n}} \mathcal{X}_{(i_1,...,i_N)} \delta^{(n)}_{(i_1,...,i_N)}(j).$$

Therefore, Equation (9) is equivalent to

$$[a^{(n)}_{i_n 1}, ..., a^{(n)}_{i_n J_n}] \times [\mathbf{B}^{(n)}_{i_n} + \lambda \mathbf{I}_{J_n}] = \mathbf{c}^{(n)}_{i_n:}$$

Since $\mathbf{B}_{i_n}^{(n)}$ is represented as the sum of rank-1 matrices and $\lambda > 0$, matrix $[\mathbf{B}_{i_n}^{(n)} + \lambda \mathbf{I}_{J_n}]$ is positive-definite and invertible. Hence,

$$\Leftrightarrow [a_{i_n 1}^{(n)}, ..., a_{i_n J_n}^{(n)}] = \mathbf{c}_{i_n:}^{(n)} \times [\mathbf{B}_{i_n}^{(n)} + \lambda \mathbf{I}_{J_n}]^{-1}$$

■

Definition 2 (Fully Observable Tucker Factorization). Given a tensor $\mathcal{X}$ $(\in \mathbb{R}^{I_1 \times \dots \times I_N})$ with nonzero-value entries Ω, the goal of fully observable Tucker factorization of $\mathcal{X}$ is to find factor matrices $\mathbf{A}^{(n)}$ $(\in \mathbb{R}^{I_n \times J_n})$ and a core tensor $\mathcal{G}$ $(\in \mathbb{R}^{J_1 \times \dots \times J_N})$, which minimize Equation (10).

$$\begin{aligned} L(\mathcal{G}, \mathbf{A}^{(1)}, ..., \mathbf{A}^{(N)}) = &\sum_{\forall (i_1,...,i_N) \in \Omega} \left(\mathcal{X}_{(i_1,...,i_N)} - \sum_{\forall (j_1,...,j_N) \in \mathcal{G}} \mathcal{G}_{(j_1,...,j_N)} \prod_{n=1}^{N} a_{i_n j_n}^{(n)} \right)^2 + \\ &\sum_{\forall (i_1,...,i_N) \notin \Omega} \left(\sum_{\forall (j_1,...,j_N) \in \mathcal{G}} \mathcal{G}_{(j_1,...,j_N)} \prod_{n=1}^{N} a_{i_n j_n}^{(n)} \right)^2 + \lambda \sum_{n=1}^{N} \|\mathbf{A}^{(n)}\|^2 \end{aligned} \tag{10}$$

Theorem 2 (Row-wise Update Rule for Factor Matrices of* GTA-FULL*).

$$\underset{[a_{i_n 1}^{(n)}, ..., a_{i_n J_n}^{(n)}]}{\arg\min} \quad L(\mathcal{G}, \mathbf{A}^{(1)}, ..., \mathbf{A}^{(N)}) = \mathbf{c}_{i_n:}^{(n)} \times [\mathbf{B}^{(n)} + \lambda \mathbf{I}_{J_n}]^{-1} \tag{11}$$

$$\text{where the } (j_1, j_2)\text{th entry of } \mathbf{B}^{(n)} (\in \mathbb{R}^{J_n \times J_n}) : \sum_{\forall (i_1,...,i_N) \in \Omega} \delta_{(i_1,...,i_N)}^{(n)}(j_1) \delta_{(i_1,...,i_N)}^{(n)}(j_2), \tag{12}$$

$$\text{the } j\text{th entry of } \mathbf{c}_{i_n:}^{(n)} (\in \mathbb{R}^{J_n}) : \sum_{\forall (i_1,...,i_N) \in \Omega_{i_n}^{(n)}} \mathcal{X}_{(i_1,...,i_N)} \delta_{(i_1,...,i_N)}^{(n)}(j), \tag{13}$$

$$\text{the } j\text{th entry of } \delta_{(i_1,...,i_N)}^{(n)} (\in \mathbb{R}^{J_n}) : \sum_{\forall (j_1 \dots j_{n-1}, j, j_{n+1} \dots j_N) \in \mathcal{G}} \mathcal{G}_{(j_1 \dots j_{n-1}, j, j_{n+1} \dots j_N)} \prod_{k \neq n} a_{i_k j_k}^{(k)}. \tag{14}$$

Proof 2.

$$\frac{\partial L}{\partial a_{i_n j_n}^{(n)}} = 0, \forall j_n, 1 \le j_n \le J_n$$

$$\begin{aligned} \Leftrightarrow &\sum_{\forall (i_1,...,i_N) \in \Omega_{i_n}^{(n)}} \left(\left(\mathcal{X}_{(i_1,...,i_N)} - \sum_{\forall (j_1,...,j_N) \in \mathcal{G}} \mathcal{G}_{(j_1,...,j_N)} \prod_{n=1}^{N} a_{i_n j_n}^{(n)} \right) \times \left(- \delta_{(i_1,...,i_N)}^{(n)}(j_n) \right) \right) \\ &+ \sum_{\forall (i_1,...,i_N) \notin \Omega_{i_n}^{(n)}} \left(\left(\sum_{\forall (j_1,...,j_N) \in \mathcal{G}} \mathcal{G}_{(j_1,...,j_N)} \prod_{n=1}^{N} a_{i_n j_n}^{(n)} \right) \times \left(\delta_{(i_1,...,i_N)}^{(n)}(j_n) \right) \right) + \lambda a_{i_n j_n}^{(n)} = 0 \end{aligned}$$

$$\Leftrightarrow \sum_{\forall (i_1,...,i_N) \in \mathcal{X}} \left(\left(\sum_{t=1}^{J_n} \delta_{(i_1,...,i_N)}^{(n)}(t) a_{i_n t}^{(n)} \right) \times \left(\delta_{(i_1,...,i_N)}^{(n)}(j_n) \right) \right) + \lambda a_{i_n j_n}^{(n)} = \sum_{\forall (i_1,...,i_N) \in \Omega_{i_n}^{(n)}} \left(\mathcal{X}_{(i_1,...,i_N)} \delta_{(i_1,...,i_N)}^{(n)}(j_n) \right) \tag{15}$$

$\sum_{\forall (i_1,...,i_N) \in \mathcal{X}} \left(\left(\sum_{t=1}^{J_n} \delta_{(i_1,...,i_N)}^{(n)}(t) a_{i_n t}^{(n)} \right) \times \left(\delta_{(i_1,...,i_N)}^{(n)}(j_n) \right) \right)$ is expressed as an inner product of the following vectors.

$$\text{Row vector } (1 \times J_n) : [a_{i_n 1}^{(n)}, ..., a_{i_n J_n}^{(n)}]$$

$$\text{Column vector } (J_n \times 1) : \begin{bmatrix} \sum_{\forall(i_1,\ldots,i_N)\in\mathcal{X}} \delta^{(n)}_{(i_1,\ldots,i_N)}(1)\delta^{(n)}_{(i_1,\ldots,i_N)}(j_n) \\ \vdots \\ \sum_{\forall(i_1,\ldots,i_N)\in\mathcal{X}} \delta^{(n)}_{(i_1,\ldots,i_N)}(J_n)\delta^{(n)}_{(i_1,\ldots,i_N)}(j_n) \end{bmatrix}$$

If we vary j_n from 1 to J_n, the row vector is fixed as $a^{(n)}_{i_n:}$ and the column vector differs.
Thus, we can integrate each column vector into a matrix $\mathbf{B}^{(n)}(\in \mathbb{R}^{J_n\times J_n})$

$$\text{where the } (j_1, j_2)\text{th entry of } \mathbf{B}^{(n)}(\in \mathbb{R}^{J_n\times J_n}) : \sum_{\forall(i_1,\ldots,i_N)\in\mathcal{X}} \delta^{(n)}_{(i_1,\ldots,i_N)}(j_1)\delta^{(n)}_{(i_1,\ldots,i_N)}(j_2).$$

λ term is simply transformed into $\lambda\mathbf{I}_{J_n}$, where $\mathbf{I}_{J_n}$ is an identity matrix $(\in \mathbb{R}^{J_n\times J_n})$.
In the same way, the right part of Equation (9) is integrated as $\mathbf{c}^{(n)}_{i_n:}(\in \mathbb{R}^{J_n})$

$$\text{where the } j\text{th entry of } \mathbf{c}^{(n)}_{i_n:}(\in \mathbb{R}^{J_n}) : \sum_{\forall(i_1,\ldots,i_N)\in\Omega^{(n)}_{i_n}} \mathcal{X}_{(i_1,\ldots,i_N)}\delta^{(n)}_{(i_1,\ldots,i_N)}(j).$$

Therefore, Equation (9) is equivalent to

$$[a^{(n)}_{i_n1}, \ldots, a^{(n)}_{i_nJ_n}] \times [\mathbf{B}^{(n)} + \lambda\mathbf{I}_{J_n}] = \mathbf{c}^{(n)}_{i_n:}$$

Since $\mathbf{B}^{(n)}$ is represented as the sum of rank-1 matrices and $\lambda > 0$, matrix $[\mathbf{B}^{(n)} + \lambda\mathbf{I}_{J_n}]$ is positive-definite and invertible. Hence,

$$\Leftrightarrow [a^{(n)}_{i_n1}, \ldots, a^{(n)}_{i_nJ_n}] = \mathbf{c}^{(n)}_{i_n:} \times [\mathbf{B}^{(n)} + \lambda\mathbf{I}_{J_n}]^{-1}$$

■

3 Theoretical Complexities of GTA-PART and GTA-FULL

Theorem 3 (Time complexity of* GTA-PART*). The time complexity of GTA-PART is $O(NIJ^3/T_1 + N^2|\Omega|J^N/T_2)$.

Proof 3. First, in Algorithm 1, it takes $O(N)$ for computing a single entry of $\delta^{(n)}$ (line 11). Thus, if we repeat the computation for all entries of $\delta^{(n)}$, it takes $O(N|\Omega|J^N)$ (lines 9-11). Computing $\delta^{(n)}$ for all factor matrices takes $O(N^2|\Omega|J^N)$, and it is computed in parallel by T_2 GPU cores. Hence, the total time complexity of computing δ is $O(N^2|\Omega|J^N/T_2)$ **(1)**. Given the i_nth row of $\mathbf{A}^{(n)}$ (line 12), updating $\mathbf{B}^{(n)}_{i_n}$ and $\mathbf{c}^{(n)}_{i_n:}$ (line 14-16) takes $O(|\Omega^{(n)}_{i_n}|J^2)$ since $\delta^{(n)}_{\alpha}$ is already calculated. Inverting $[\mathbf{B}^{(n)}_{i_n} + \lambda\mathbf{I}_{J_n}]$ (line 17) takes $O(J^3)$, and updating a row (line 18) takes $O(J^2)$. Thus, the time complexity of updating the i_nth row of $\mathbf{A}^{(n)}$ (lines 13-18) is $O(J^3 + |\Omega^{(n)}_{i_n}|J^2)$. Iterating it for all rows of $\mathbf{A}^{(n)}$ takes $O(IJ^3 + |\Omega|J^2)$. Finally, updating all $\mathbf{A}^{(n)}$ takes $O(NIJ^3 + N|\Omega|J^2)$. Since the updates of factor matrices are executed in parallel with CPUs, the time complexity for updating all $\mathbf{A}^{(n)}$ is $O((NIJ^3 + N|\Omega|J^2)/T_1)$ **(2)**. Reconstruction on GPUs (line 19) takes $O((N|\Omega|J^N)/T_2)$ **(3)**. Summing all complexities (1), (2), and (3) gives us the total time complexity of GTA-PART, which is $O(NIJ^3/T_1 + N^2|\Omega|J^N/T_2)$. ■

Theorem 4 (Time complexity of* GTA-FULL*). The time complexity of GTA-FULL is $O(N^2IJ^{2N}/T_1 + N^2|\Omega|J^N/T_2)$.

Proof 4. The only difference between GTA-FULL and GTA-PART is that there is an additional cost for calculating a cache table $\mathcal{M}$. Hence, the time complexity of GTA-FULL is summation of the time complexities of GTA-PART and computing $\mathcal{M}$. In order to calculate $\mathcal{M}$, we first need to decide two core tensor entries β and γ, which takes $O(J^{2N})$. After that, we apply the factorization technique to compute $\mathcal{M}[\beta][\gamma]$, which takes $O(NI)$. As all entries of $\mathcal{M}$ are computed in parallel with T_1 CPU cores, the time complexity for computing $\mathcal{M}$ is $O(NIJ^{2N}/T_1)$. As we compute $\mathcal{M}$ for all factor matrices $\mathbf{A}^{(n)}$, the total time complexity for computing $\mathcal{M}$ is $O(N^2IJ^{2N}/T_1)$. If we sum the time complexities of GTA-PART and computing $\mathcal{M}$, that is equivalent to the time complexity of GTA-FULL, which takes $O(N^2IJ^{2N}/T_1 + N^2|\Omega|J^N/T_2)$. ■

Theorem 5 (Memory complexity of* GTA-FULL*). The memory complexity of GTA-FULL is $O(J|\Omega| + J^{2N})$.

Proof 5. Compared to GTA-PART, there is an additional intermediate data (a cache table $\mathcal{M}$) for GTA-FULL, and the memory complexity of $\mathcal{M}$ is $O(J^{2N})$. Therefore, the total memory complexity of GTA-FULL is summation of the memory complexities of GTA-PART and $\mathcal{M}$, which is $O(J|\Omega| + J^{2N})$. ■

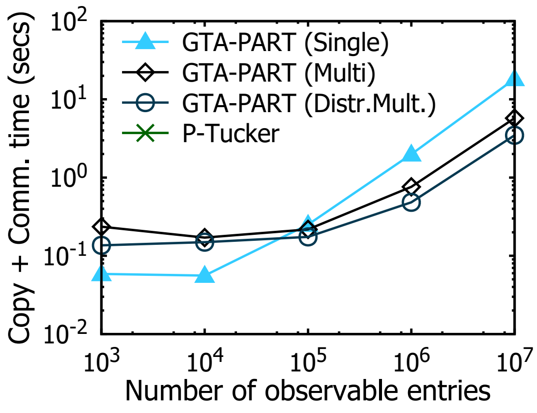
Algorithm 1: GTA Algorithm

```
Input  : Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$,
         factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ $(n = 1, ..., N)$, and
         core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$.
Output: Updated factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ $(n = 1, ..., N)$ and core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$.
1  initialize $\mathbf{A}^{(n)}$ and $\mathcal{G}$ with random values between 0 and 1
2  repeat
3      for n = 1...N do
4          if GTA-FULL then
                                                    ▷ Compute an intermediate table $\mathcal{M}^{(n)}$
5              for $\beta = \forall(j_1, ..., j_N) \in \mathcal{G}$ do                 ▷ parallel, CPU
6                  for $\gamma = \forall(j_1, ..., j_N) \in \mathcal{G}$ do
7                      $\mathcal{M}^{(n)}[\beta][\gamma] \leftarrow \mathcal{G}_\beta \mathcal{G}_\gamma \left( \sum_{\forall \alpha \in \mathcal{X}^{(n)}_{i_n}} \prod_{k \neq n} (a^{(k)}_{\alpha_k \beta_k} a^{(k)}_{\alpha_k \gamma_k}) \right)$
8              calculate $\mathbf{B}^{(n)}$ using $\mathbf{B}^{(n)}[j_1][j_2] = \sum_{\forall \alpha \in \mathcal{X}^{(n)}_{i_n}} \delta^{(n)}_\alpha(j_1) \delta^{(n)}_\alpha(j_2)$
                                                    ▷ Precompute $\delta^{(n)}$ for GTA-PART & GTA-FULL
9          for $\alpha = \forall(i_1, ..., i_N) \in \Omega$ do                          ▷ parallel, GPU
10             for $\beta = \forall(j_1, ..., j_N) \in \mathcal{G}$ do
11                 $\delta^{(n)}_\alpha(j_n) \leftarrow \delta^{(n)}_\alpha(j_n) + \mathcal{G}_\beta \prod_{k \neq n} a^{(k)}_{i_k j_k}$
12         for $i_n$ = 1...$I_n$ do                                             ▷ parallel, CPU
13             for $\alpha = \forall(i_1, ..., i_N) \in \Omega^{(n)}_{i_n}$ do
14                 if GTA-PART then
15                     calculate $\mathbf{B}^{(n)}_{i_n}$ using $\mathbf{B}^{(n)}_{i_n}[j_1][j_2] = \sum_{\forall \alpha \in \Omega^{(n)}_{i_n}} \delta^{(n)}_\alpha(j_1) \delta^{(n)}_\alpha(j_2)$
16                 calculate $\mathbf{c}^{(n)}_{i_n:}$ using $\mathbf{c}^{(n)}_{i_n:}[j] = \sum_{\forall \alpha \in \Omega^{(n)}_{i_n}} \mathcal{X}_\alpha \delta^{(n)}_\alpha(j)$
17             find an inverse of $[\mathbf{B}^{(n)}_{i_n} + \lambda \mathbf{I}_{J_n}]$ or $[\mathbf{B}^{(n)} + \lambda \mathbf{I}_{J_n}]$
18             update $[a^{(n)}_{i_n 1}, \cdots, a^{(n)}_{i_n J_n}] = \mathbf{c}^{(n)}_{i_n:} \times [\mathbf{B}^{(n)}_{i_n} + \lambda \mathbf{I}_{J_n}]^{-1}$ (GTA-PART) or $\mathbf{c}^{(n)}_{i_n:} \times [\mathbf{B}^{(n)} + \lambda \mathbf{I}_{J_n}]^{-1}$ (GTA-FULL)
19     calculate reconstruction error on GPUs
20 until the maximum iteration or $\|\mathcal{X} - \mathcal{X}'\|$ converges;
21 for n = 1...N do
22     $\mathbf{A}^{(n)} \rightarrow \mathbf{Q}^{(n)} \mathbf{R}^{(n)}$                                  ▷ QR decomposition
23     $\mathbf{A}^{(n)} \leftarrow \mathbf{Q}^{(n)}$                                          ▷ Orthogonalize $\mathbf{A}^{(n)}$
24     $\mathcal{G} \leftarrow \mathcal{G} \times_n \mathbf{R}^{(n)}$                                     ▷ Update core tensor $\mathcal{G}$
```

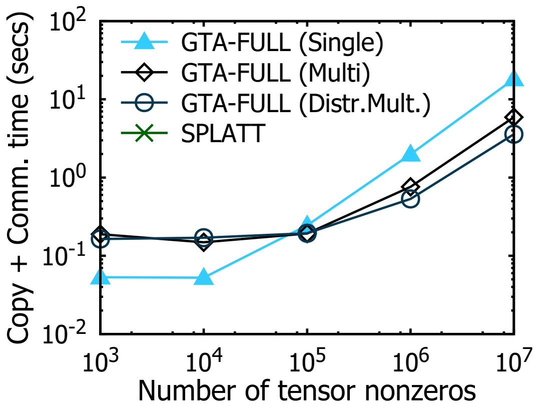
4 Additional experimental results of GTA

4.1 (Data copy + Communication) time analysis of GTA-PART and GTA-FULL

Since SnuCL makes an illusion which converts distributed environment to single-node one with multi-GPUs, we were able to compute (data transfer + communication) time rather than individual time. Figure 1 shows (data transfer + communication) time of GTA and other methods with respect to $|\Omega|$ for synthetic tensors with $N = 4$, $I_n = 10^4$, and $J_n = 10$. Note that the results are elapsed time for (data transfer + communication) per iteration. When $|\Omega|$ is small, communication cost dominates data transfer time, which makes the distributed-GPU version of GTA slower than single-node one. When $|\Omega|$ increases, data transfer time becomes larger than communication cost. Hence, at this time, single-GPU version of GTA has higher (copy + comm.) time value than those of other versions (e.g., copying 100 nonzeros to 1 GPU vs 25 nonzeros to 4 GPUs in parallel). Moreover, P-TUCKER and SPLATT have zero (copy + comm.) time values for all $|\Omega|$ since they are single-machine CPU-based algorithms, which is the main reason why P-TUCKER shows faster speed than GTA-PART when $|\Omega|$ is small.



a (data transfer + communication) time of GTA-PART and P-TUCKER with respect to $|\Omega|$.



b (data transfer + communication) time of GTA-FULL and SPLATT with respect to $|\Omega|$.

Fig. 1: (data transfer + communication) time of GTA and other methods with respect to $|\Omega|$ for synthetic tensors with $N = 4$, $I_n = 10^4$, and $J_n = 10$. When $|\Omega|$ is small, communication cost dominates copy time, and the distributed-GPU version of GTA has higher (copy + comm.) time value than that of single-GPU version. On the other hand, when $|\Omega|$ is large, copy time dominates communication cost, and single-GPU version of GTA has higher (copy + comm.) time value than those of multi- or distributed-GPU versions. Note that P-TUCKER and SPLATT have zero (copy + comm.) time values for all $|\Omega|$ and experimental results are measured per iteration.

4.2 Detailed Running Time Analysis of GTA and other methods

Table 1 shows full experimental results of GTA-PART and P-TUCKER with respect to the number of observable entries of a tensor. We vary $|\Omega|$ from 10^3 to 10^7 while fixing $N = 4$, $I_n = 10^4$, and $J_n = 10$. As shown in Table 1, GTA-PART runs faster than P-TUCKER except for when $|\Omega| = 10^3$; in particular, it is $36.1\times$ faster than P-TUCKER when $|\Omega| = 10^7$. The main computational bottleneck of P-TUCKER is computing intermediate data δ using CPUs. When the number of observable entries is small, distributed-GPU version of GTA-PART runs slower than single-node versions due to communication costs.

Table 2 shows full benchmark results of GTA-FULL and SPLATT with respect to the number of nonzero-value entries of a tensor. We vary $|\Omega|$ from 10^3 to 10^7 while fixing $N = 4$, $I_n = 10^4$, and $J_n = 10$. As shown in Table 2, GTA-FULL runs faster than SPLATT across all $|\Omega|$; in particular, it is $26.4\times$ faster than SPLATT when $|\Omega| = 10^7$. The main computational bottleneck of SPLATT is performing singular value decomposition (SVD) to update factor matrices. When the number of nonzero-value entries is small, distributed-GPU version of GTA-FULL runs slower than single-node versions due to communication costs.

TABLE 1: Performance of GTA-PART and its competitor P-TUCKER for a partially observable synthetic tensor ($N = 4$, $I_n = 10^4$, and $J_n = 10$). **Results in bold** indicate the fastest running time per iteration among all methods. GTA-PART runs up to $36.1\times$ faster than P-TUCKER when $|\Omega| = 10^7$. When the number of observable entries is small, P-TUCKER runs faster than GTA-PART due to the copy-time bottleneck of GTA-PART between CPU and GPU. Notice that all results are measured in seconds.

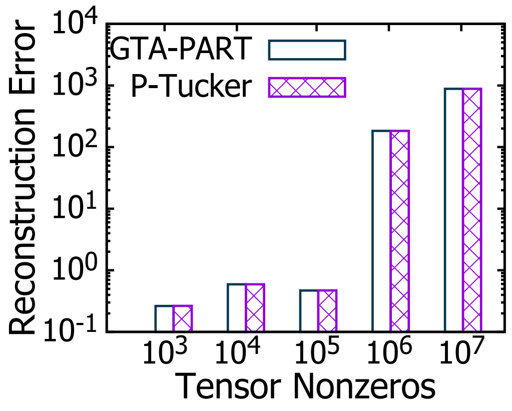
Number of observable entries	**P-TUCKER** (16 CPUs; 0 GPU)	**GTA-PART (Single)** (16 CPUs + 1 GPU)	**GTA-PART (Multi)** (16 CPUs + 4 GPUs)	**GTA-PART (Distributed)** (16 CPUs + 16 GPUs)
10^3	**0.065285**	0.137151	0.365505	0.2985043
10^4	0.364194	**0.145284**	0.270871	0.314541
10^5	3.340793	0.859945	0.437463	**0.330154**
10^6	31.804728	7.313576	2.316254	**1.093719**
10^7	319.91961	68.993037	20.297236	**8.867695**

TABLE 2: Performance of GTA-FULL and its competitor SPLATT for a fully observable synthetic tensor ($N=4$, $I_n=10^4$, and $J_n=10$). **Results in bold** indicate the fastest running time per iteration among all methods. GTA-FULL runs up to $26.4\times$ faster than SPLATT when $|\Omega|=10^7$. When the number of nonzero-value entries is small, distributed-GPU version of GTA-FULL runs slower than single-node versions due to communication costs. Notice that all results are measured in seconds.

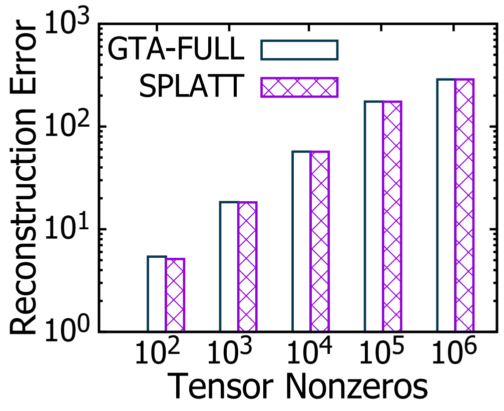
Number of nonzero-value entries	**SPLATT** (16 CPUs; 0 GPU)	**GTA-FULL (Single)** (16 CPUs + 1 GPU)	**GTA-FULL (Multi)** (16 CPUs + 4 GPUs)	**GTA-FULL (Distributed)** (16 CPUs + 16 GPUs)
10^3	601.7066667	**13.848402**	13.854225	15.538099
10^4	619.3263333	13.883727	**13.808904**	15.448877
10^5	606.713	**14.671736**	14.851909	14.805537
10^6	619.6155	20.650712	16.414856	**15.78476**
10^7	622.9545	82.836525	33.773691	**23.621079**

4.3 Correctness Verification of GTA

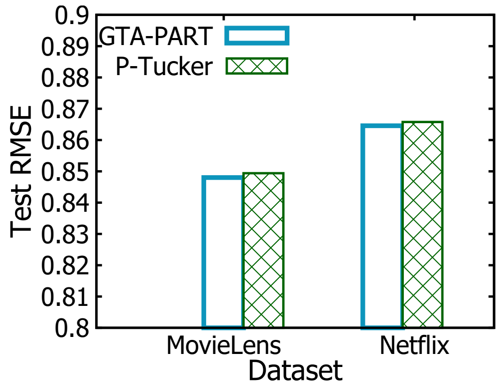
As introduced in the main paper, we verify whether our method GTA is correct or not using two accuracy metrics: reconstruction error and test root mean square error (RMSE). As shown in Figure 2, GTA and other state-of-the-art methods have almost the same reconstruction error and test RMSE, which implies the correctness of GTA. Note that we split the data into training and test set at a ratio of 9:1 and use the synthetic tensor with $N=4$, $I_n=10^4$, and $J_n=10$.



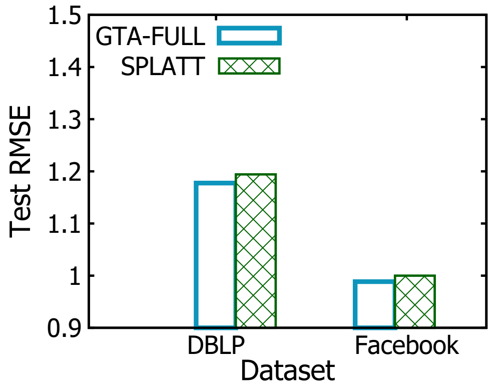
a Reconstruction error of GTA-PART and P-TUCKER with respect to the number of tensor nonzeros.



b Reconstruction error of GTA-FULL and SPLATT with respect to the number of tensor nonzeros.



c Test RMSE of GTA-PART and P-TUCKER for partially observable real-world tensors.



d Test RMSE of GTA-FULL and SPLATT for fully observable real-world tensors.

Fig. 2: Correctness verification of GTA with respect to two accuracy metrics: reconstruction error and test RMSE for synthetic and real-world tensors. GTA and other state-of-the-art methods have almost the same reconstruction error and test RMSE, which implies the correctness of GTA. Note that we split the real-world data into training and test set at a ratio of 9:1 for calculating the test RMSE and use the synthetic tensor with $N=4$, $I_n=10^4$, and $J_n=10$ for calculating the reconstruction error.

Furthermore, as shown in the Table 3, our method GTA shows comparable reconstruction error as the state-of-the-art methods, P-TUCKER and SPLATT. Moreover, GTA presents a stable reconstruction error regardless of random initialization (small standard deviation), similar to HOOI-based method (SPLATT).

Recon. Error	**Init. 1**	**Init. 2**	**Init. 3**	**Init. 4**	**Init. 5**	**Average**	**Standard Deviation**	**Std. Dev. / Avg.**
GTA-PART	202.9837	203.003	203.058	203.02	203.06	203.0243	0.03295	**0.000162**
P-TUCKER	203.01	203.015	203.008	203.036	202.99	203.012	0.0165	**0.000081**
GTA-FULL	321.287	321.256	321.301	321.356	321.372	321.314	0.0486	**0.000151**
SPLATT	320.442	320.393	320.398	320.32	320.393	320.389	0.04397	**0.000137**

TABLE 3: Reconstruction error averaged on five random runs of GTA and competitors for a $100 \times 100 \times 100$ tensor with density = 0.5. GTA shows almost identical reconstruction error to other state-of-the-art methods. Moreover, GTA has a stable reconstruction error regardless of random initialization, similar to the HOOI-based method (SPLATT).