Supplementary Material for Toward Scalable Tucker Decomposition: Skew-Aware Multi-Level Partitioning with GPU–Storage Co-Processing

Seung Hyeon Song*[§], Jihye Lee[†][§], Chanki Kim[‡][¶], Kang-Wook Chon*[¶]
*KOREATECH, [†]ETRI, [‡]Jeonbuk National University
Republic of Korea
{adsllove1, kw.chon}@koreatech.ac.kr, jihye.lee@etri.re.kr, carisis@jbnu.ac.kr

I. Extended experimental study

In the supplementary material, we provide extended experiments conducted on a system equipped with a larger main memory (256 GB) than that used in the main paper (64 GB), in order to examine the impact of main memory capacity on execution behavior and performance. These extended experiments focus on comparisons with SOTA methods.

A. Experimental setup

1) Datasets: We evaluate GSP-Tucker using 14 real-world sparse tensor datasets summarized in Table I. The tensors range from three to five modes and contain up to 4.6 billion nonzeros.

2) Environment: All single-machine experiments, including those for GSP-Tucker, are conducted on a server equipped with dual Intel Xeon E5-2630 v4 CPUs (2.20 GHz, 10 cores per socket), two NVIDIA GTX 1080 GPUs (8 GB of GPU device memory each), 256 GB of main memory, and a 1 TB SSD. For the distributed baselines, we use an Amazon EC2 cluster consisting of one master and 20 m5.xlarge worker nodes, each providing 4 vCPUs, 16 GB of memory, and 50 GB of storage with 10 Gbps network bandwidth. This configuration offers a total of 80 vCPUs, 336 GB of memory, and 1.05 TB of aggregate storage.

3) Comparison Methods: We compare GSP-Tucker with representative Tucker decomposition methods described in Table II. All baselines use official source codes released by the authors and use default configurations for each baseline except for GTA. For ensuring a fair comparison, we re-implement GTA, which is developed with OpenCL, using CUDA. For the experiments of the proposed method (i.e., GSP-Tucker), we use the partition parameters (i.e., $\{P_n\}_{n=1}^{N}$) listed in Table III for each dataset. These partition parameters are automatically determined by considering the capacity of GPU device memory so that the number of I/O transfers between the GPU and the host is minimized.

We set the regularization parameter λ=0.001 and measure the average elapsed time per iteration. For fair comparison, loading and finalization times are excluded for all baselines except GSP-Tucker, where the first tensor split loading (with one SSD) and result return are included. GPU-based methods include data transfer between GPU device memory and main memory.

Hereafter, O.O.M. denotes an out-of-memory error, S.E. indicates a storage error caused by excessive intermediate data, T.O. refers to a timeout where execution exceeds the predefined limit (10,800 seconds), R.E. represents a runtime error, and G.O.O.M. denotes a GPU out-of-memory failure.

B. Comparisons with the state-of-the-art methods

In the supplementary material, Table III reports the time per iteration of representative Tucker decomposition frameworks across real-world tensor datasets with Tucker ranks $R = 10$ and $R = 20$, evaluated on a system equipped with 256 GB of main memory. All GPU-based methods are executed using two GPU devices, while CPU-based parallel methods use 12 CPU threads. The distributed methods (HaTen2 and MuLOT) are evaluated on the same Amazon EC2 cluster, consisting of one master node and 20 worker nodes, each running two workers (i.e., Mappers/Reducers or Spark Executors).

Compared to the experimental results obtained under the 64 GB main-memory configuration presented in the manuscript, the larger memory resource substantially reduces the number of O.O.M. failures across many baseline methods. In particular, several CPU-based methods that previously terminated due to memory exhaustion are now able to complete execution on a broader set of datasets, which could indicate that main memory capacity was a dominant limiting factor under tighter configurations.

However, despite increased memory availability, execution failures are not eliminated. A significant fraction of methods still suffer from S.E., T.O., R.E., G.O.O.M., and O.O.M. errors, especially on large-scale and highly skewed tensors. This observation suggests that simply provisioning more main memory alleviates memory pressure but does not fundamentally resolve scalability issues arising from intermediate data materialization and skew-induced imbalance.

In contrast, GSP-Tucker completes all experiments without O.O.M. or G.O.O.M. failures under the same 256 GB setting, while it maintains stable iteration times across both small and large datasets. These results indicate that the robustness of

[§] Equal contribution.
[¶] Corresponding authors.

TABLE I: Specifications of the real-world tensor datasets used in the experiments. Each dataset is denoted by its full name (Name) and abbreviation (Abb.). Order represents the tensor dimensionality, and Dimension size lists the size of each mode. $nnz(\boldsymbol{\mathcal{X}})$ denotes the number of nonzero entries in tensor $\boldsymbol{\mathcal{X}}$, and Density is defined as $nnz(\boldsymbol{\mathcal{X}})/\prod_{n=1}^{N} I_n$. The last five columns report data sizes (GB) at different processing stages: raw input (text), converted input (COO), intermediate data, factor matrices, and total memory cost when the Tucker rank $R = 20$. These datasets range from millions to billions of nonzeros, covering 3D to 5D tensors used for evaluating scalability and robustness of GSP-Tucker.

Name	**Abb.**	**Order**	**Dimension size**	$nnz(\boldsymbol{\mathcal{X}})$	**Density**	**Input in text format (GB)**	**Input in COO format (GB)**	**Intermediate data (GB)**	**Factors (GB)**	**Total memory cost (GB)**
Delicious-3d	D3	3	533K × 17M × 2.4M	140M	6.14109E-12	3.02	2.61	20.88	3.02	34.81
NELL-1	N1	3	2.9M × 2.1M × 25.4M	143.6M	9.05415E-13	3.7	2.67	21.40	3.56	38.12
NELL-2	N2	3	12K × 9K × 29K	77M	2.40224E-05	1.4	1.43	11.46	0.00	12.90
Amazon Reviews	AR	3	4M × 1.7M × 1.8M	1.7B	1.12798E-10	42.8	32.44	259.55	1.26	294.75
Flickr-3d	F3	3	320K × 28M × 1.6M	113M	7.80441E-12	2.57	2.1	16.82	4.48	35.21
Reddit-2015	RE	3	8.2M × 177K × 8.1M	4.6B	3.97443E-10	105.54	87.31	698.49	2.46	789.96
Vast-2015-3d	VA3	3	165K × 11K × 2	26M	6.91494E-03	0.42	0.48	3.88	0.02	4.39
Uber pickups	UP	4	183 × 24 × 1K × 2K	3M	3.84967E-04	0.05	0.07	0.49	0.00	0.57
NIPS publications	NP	4	2K × 3K × 14K × 17	3M	1.82988E-06	0.05	0.07	0.46	0.00	0.53
Chicago-crime-comm	CCC	4	6K × 24 × 77 × 32	5M	1.45720E-02	0.07	0.12	0.79	0.00	0.91
Enron Emails	EE	4	6K × 6K × 244K × 1K	54M	5.4581E-09	1.12	1.21	8.08	0.04	9.33
Flickr-4d	F4	4	320K × 28M × 1.6M × 731	113M	1.06764E-14	2.99	2.52	16.82	4.48	37.33
Chicago-crime-geo	CCG	5	6K × 24 × 380 × 395 × 32	6M	8.87395E-06	0.12	0.16	0.94	0.00	1.11
LBNL-Network	LN	5	1.6K × 4K × 1.6K × 4K × 868K	1.7M	4.23071E-14	0.05	0.04	0.25	0.14	0.43

TABLE II: Comparison of GSP-Tucker and the existing methods. The performance bottlenecks are highlighted in red. While GSP-Tucker achieves superior computational efficiency, the other approaches exhibit one or more performance bottlenecks.

	CPU-based					**Distributed**		**GPU-based**			
	P-Tucker [1]	**FTCom [2]**	**SHOT [3]**	**VeST [4]**	**SGD-Tucker [5]**	**HaTen2 [6]**	**MuLOT [7]**	**ParTI! [8]**	**GTA [9]**	**GPUTucker [10]**	**GSP-Tucker**
Speed	Low	Low	Low	Low	Low	Low	Low	High	High	High	High
Limit of scalability	Main memory	Storage	Storage	Main memory	Main memory	Distributed storage	Distributed storage	GPU device memory	GPU device memory	Main memory	Storage
Network overhead	X	X	X	X	X	O	O	X	X	X	X
Skew-aware	X	X	X	X	X	X	X	X	X	X	O

TABLE III: Times per iteration for varying datasets with Tucker ranks R set to 10 and 20 across datasets (sec.). GSP-Tucker shows no O.O.M. or G.O.O.M. failures across all experiments, achieving superior performance over CPU-based and distributed methods, comparable speed to the existing GPU-based methods on small datasets, and stable scalability on large tensors where other GPU-based methods fail due to memory errors.

Datasets (Partition param.)	**CPU-based**										**Distributed**				**GPU-based**							
	P-Tucker [1]		**S-HOT [3]**		**SGDTucker [5]**		**FTCom [2]**		**VeST [4]**		**HaTen2 [6]**		**MuLOT [7]**		**ParTI! [8]**		**GTA [9]**		**GPUTucker [10]**		**GSP-Tucker**	
	$R = 10$	$R = 20$	$R = 10$	$R = 20$	$R = 10$	$R = 20$	$R = 10$	$R = 20$	$R = 10$	$R = 20$	$R = 10$	$R = 20$	$R = 10$	$R = 20$	$R = 10$	$R = 20$	$R = 10$	$R = 20$	$R = 10$	$R = 20$	$R = 10$	$R = 20$
D3 (1-7-2)	256	1905	S.E.	S.E.	T.O.	T.O.	R.E.	R.E.	T.O.	T.O.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	G.O.O.M.	G.O.O.M.	G.O.O.M.	G.O.O.M.	77	524	49	377
N1 (2-1-9)	270	1976	S.E.	S.E.	T.O.	T.O.	R.E.	R.E.	T.O.	T.O.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	G.O.O.M.	G.O.O.M.	G.O.O.M.	G.O.O.M.	236	1768	214	1681
N2 (1-1-1)	139	1082	S.E.	S.E.	T.O.	T.O.	783	1150	T.O.	T.O.	O.O.M.	O.O.M.	2236	7651	30	44	44	233	20	106	15	88
AR (11-4-3)	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	214	3086
F3 (1-8-1)	213	1551	S.E.	S.E.	T.O.	T.O.	R.E.	R.E.	T.O.	T.O.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	G.O.O.M.	G.O.O.M.	G.O.O.M.	G.O.O.M.	64	241	36	194
RE (16-1-15)	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	1812	8192
VA3 (1-1-1)	157	1224	S.E.	S.E.	T.O.	T.O.	R.E.	R.E.	T.O.	T.O.	T.O	T.O.	R.E.	R.E.	R.E.	R.E.	13	65	15	48	9	34
UP (1-1-1-1)	91	1457	S.E.	S.E.	T.O.	T.O.	48	652	T.O.	T.O.	T.O.	T.O.	1131	2584	R.E.	R.E.	13	136	15	202	8	119
NP (1-1-1-1)	94	1491	S.E.	S.E.	T.O.	T.O.	82	R.E.	T.O.	T.O.	T.O.	T.O.	515	R.E.	11	128	6	128	6	93	6	87
CCC (1-1-1-1)	192	3053	S.E.	S.E.	T.O.	T.O.	142	2038	T.O.	T.O.	T.O.	T.O.	1558	R.E.	R.E.	R.E.	20	237	17	202	17	195
EE (1-1-1-1)	1652	T.O.	S.E.	S.E.	T.O.	T.O.	1357	T.O.	T.O.	T.O.	O.O.M.	O.O.M.	T.O.	T.O.	G.O.O.M.	G.O.O.M.	176	2639	130	2022	122	1973
F4 (1-8-1-1)	3008	T.O.	S.E.	S.E.	T.O.	T.O.	R.E.	R.E.	T.O.	T.O.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	G.O.O.M.	G.O.O.M.	G.O.O.M.	G.O.O.M.	370	8241	329	8018
CCG (1-1-1-1-1)	3453	T.O.	S.E.	S.E.	T.O.	T.O.	R.E.	R.E.	R.E.	T.O.	T.O.	T.O.	T.O.	T.O.	G.O.O.M.	G.O.O.M.	287	8542	259	7411	250	7407
LN (1-1-1-1-1)	2061	T.O.	S.E.	S.E.	T.O.	T.O.	R.E.	R.E.	T.O.	T.O.	T.O.	T.O.	T.O.	T.O.	G.O.O.M.	G.O.O.M.	178	8542	166	7411	165	7491

GSP-Tucker is primarily attributed to its execution strategy, which explicitly bounds intermediate data growth and controls skew at the partition level, rather than relying on increased memory capacity alone.

References

[1] S. Oh, N. Park, S. Lee, and U. Kang, "Scalable tucker factorization for sparse tensors-algorithms and discoveries," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 1120–1131.

[2] D. Lee, J. Lee, and H. Yu, "Fast tucker factorization for large-scale tensor completion," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 1098–1103.

[3] J. Oh, K. Shin, E. E. Papalexakis, C. Faloutsos, and H. Yu, "S-hot: Scalable high-order tucker decomposition," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2017, pp. 761–770.

[4] M. Park, J.-G. Jang, and L. Sael, "Vest: Very sparse tucker factorization of large-scale tensors," in *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2021, pp. 172–179.

[5] H. Li, Z. Li, K. Li, J. S. Rellermeyer, L. Chen, and K. Li, "Sgd_tucker: A novel stochastic optimization strategy for parallel sparse tucker decomposition," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1828–1841, 2020.

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

[7] A. Gillet, É. Leclercq, and N. Cullot, "Multi-level optimization of the canonical polyadic tensor decomposition at large-scale: Application to the stratification of social networks through deflation," *Information Systems*, vol. 112, p. 102142, 2023.

[8] Y. Ma, J. Li, X. Wu, C. Yan, J. Sun, and R. Vuduc, "Optimizing sparse tensor times matrix on gpus," *Journal of Parallel and Distributed Computing*, vol. 129, pp. 99–109, 2019.

[9] S. Oh, N. Park, J.-G. Jang, L. Sael, and U. Kang, "High-performance tucker factorization on heterogeneous platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2237–2248, 2019.

[10] J. Lee, D. Han, O.-K. Kwon, K.-W. Chon, and M.-S. Kim, "Gputucker: Large-scale gpu-based tucker decomposition using tensor partitioning," *Expert Systems with Applications*, vol. 237, p. 121445, 2024.