Performance Implication of Tensor Irregularity and Optimization for Distributed Tensor Decomposition

ZHENG MIAO, Hangzhou Dianzi University, China
JON C. CALHOUN and RONG GE, Clemson University, USA
JIAJIA LI, North Carolina State University, USA

Tensors are used by a wide variety of applications to represent multi-dimensional data; tensor decompositions are a class of methods for latent data analytics, data compression, and so on. Many of these applications generate large tensors with irregular dimension sizes and nonzero distribution. CANDECOMP/PARAFAC decomposition (CPD) is a popular low-rank tensor decomposition for discovering latent features. The increasing overhead on memory and execution time of CPD for large tensors requires distributed memory implementations as the only feasible solution. The sparsity and irregularity of tensors hinder the improvement of performance and scalability of distributed memory implementations. While previous works have been proved successful in CPD for tensors with relatively regular dimension sizes and nonzero distribution, they either deliver unsatisfactory performance and scalability for irregular tensors or require significant time overhead in preprocessing. In this work, we focus on medium-grained tensor distribution to address their limitation for irregular tensors. We first thoroughly investigate through theoretical and experimental analysis. We disclose that the main cause of poor CPD performance and scalability is the imbalance of multiple types of computations and communications and their tradeoffs; and sparsity and irregularity make it challenging to achieve their balances and tradeoffs. Irregularity of a sparse tensor is categorized based on two aspects: very different dimension sizes and a non-uniform nonzero distribution. Typically, focusing on optimizing one type of load imbalance causes other ones more severe for irregular tensors. To address such challenges, we propose irregularity-aware distributed CPD that leverages the sparsity and irregularity information to identify the best tradeoff between different imbalances with low time overhead. We materialize the idea with two optimization methods: the prediction-based grid configuration and matrix-oriented distribution policy, where the former forms the global balance among computations and communications, and the latter further adjusts the balances among computations. The experimental results show that our proposed irregularity-aware distributed CPD is more scalable and outperforms the medium- and fine-grained distributed implementations by up to 4.4× and 11.4× on 1,536 processors, respectively. Our optimizations support different sparse tensor formats, such as compressed sparse fiber (CSF), coordinate (COO), and Hierarchical Coordinate (HiCOO), and gain good scalability for all of them.

CCS Concepts: • **Theory of computation → Design and analysis of algorithms;**

Additional Key Words and Phrases: Sparse tensor, tensor decomposition, CPD, irregularity

This research is partially supported by U.S. National Science Foundation Principles and Practice of Scalable Systems (PPoSS) program and by U.S. Department of Energy and Pacific Northwest National Laboratory under Contract No. 532181 and the PNNL Cluster. This research was supported by the U.S. National Science Foundation under Grants SHF-1910197, SHF-1943114, CCF-155151, and OAC-2204011.

Authors' addresses: Z. Miao, Hangzhou Dianzi University, China; email: miaozheng@hdu.edu.cn; J. C. Calhoun and R. Ge, Clemson University; emails: jonccal@clemson.edu.cn, rge@clemson.edu.cn; J. Li, North Carolina State University; email: jiajia.li@ncsu.edu.

2329-4949/2023/06-ART10 $15.00
https://doi.org/10.1145/3580315

ACM Reference format:
Zheng Miao, Jon C. Calhoun, Rong Ge, and Jiajia Li. 2023. Performance Implication of Tensor Irregularity and Optimization for Distributed Tensor Decomposition. *ACM Trans. Parallel Comput.* 10, 2, Article 10 (June 2023), 27 pages.
https://doi.org/10.1145/3580315

1 INTRODUCTION

Tensors are multi-dimensional arrays and often sparse that are utilized by applications spanning a wide range of domain areas, such as quantum chemistry, (healthcare, social network, brain signal, electrical grid) data analytics, signal processing, machine learning, and recommendation systems [1, 4, 14, 16, 18, 22, 28–30]. Tensor decompositions are a class of tensor methods for data analytics, low-rank approximation, data compression, and so on. In this work, we study the **CANDECOMP/PARAFAC decomposition (CPD)**, one of the most popular tensor decompositions.

Large data generated from these applications requires distributed memory implementations due to the large amount of memory requirements and the need for fast execution time. For example, the *amazon* tensor comprises reviews and contains more than 1 billion nonzeros; the state-of-the-art CPD implementation based on CSF format could not analyze it on fewer than eight CPU nodes. Some studies show impressive performance for sparse distributed CPD algorithms [13, 20, 35]. The previous works present medium-grained decomposition that performs an N-dimensional decomposition of the tensor, where N is the number of modes and one-dimensional decompositions of the factor matrices [6, 32, 35]. They have achieved good performance and scalability in CPD for tensors with relatively regular dimension sizes and nonzero distribution, because both computation and communication are balanced well. However, the sparsity and irregularity features and their influence on stages of the CPD algorithm have not been well investigated, which hinders further performance improvement and machine scalability. Other recent works use a fine-grained decomposition of tensors to co-optimize computation and communication [20, 21]. But they require significant time overhead in hypergraph partitioning. In this work, we focus on analysis and optimization of medium-grained tensor distribution to address their limitation that does not scale well for tensors with high sparsity and irregularity.

We categorize the *irregularity* of a sparse tensor based on two aspects: very different dimension sizes and a non-uniform nonzero distribution. Analyzing sparse tensors from various data sources, we observe a tensor could have *dimension(s) much longer relative to the others*. For example, the tensor *fb-m*, a sampled knowledge base dataset, has the first two dimensions around 23 million, while the last dimension is only 166. This phenomenon is common because of different information contained in diverse dimensions: Short dimensions could come from a small range of timestamps, types of relations, and so on, while long dimensions could be users, pages, keywords, papers, and so on. Sparse tensors from real applications tend to have a *non-uniform nonzero distribution*; while different dimension sizes make it worse. The nonzeros could be extremely dense in a couple of regions, but much sparser in other regions in an irregular tensor. In tensor *fb-m*, the nonzero distribution is extremely dense near the diagonal and the corner of the tensor but is very sparse in other regions.

We take medium-grained, bulk-synchronous distributed CPD algorithm SPLATT [35] as an example to illustrate the problems in performance and scalability for irregular tensors. Figure 1 shows the normalized time of the three major stages of CPD using the SPLATT library [35] on 192 and 384 processors. We separate the distributed CPD execution time into three components: communication ('COMM') and two computation components, consisting of the **matricized**

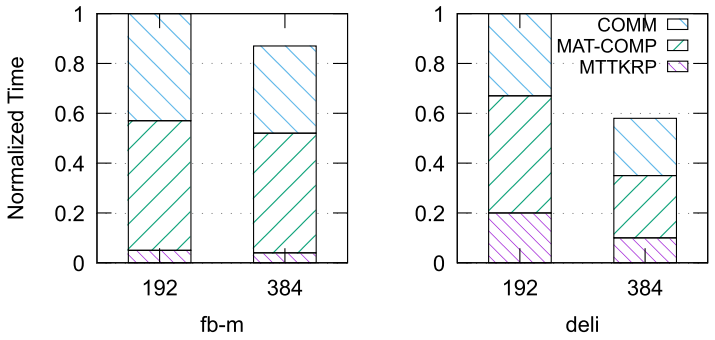


Fig. 1. Execution time breakdown of SPLATT CPD [35] on 192 and 384 processors. Time is normalized to the 192-processor run for each tensor.

tensor times Khatri-Rao product (MTTKRP) and **matrix computations (MAT-COMP)** (see Section 3). For the very irregular tensor *fb-m* (sized $23M \times 23M \times 166$) with both dimension sizes and non-uniform nonzero distribution irregularities, its time reduction from 192 to 384 processors is only 13%, wherein the time for communication, MTTKRP, and matrix computations are slightly decreased by 19%, 19%, and 8%, respectively. SPLATT does not scale well in *fb-m* for two reasons: First, it assumes MTTKRP is the dominant kernel and focuses on optimizing it. Second, it optimizes each kernel in separate stages, and focusing on reducing MTTKRP usually causes more overhead on other kernels. Conversely, the time reduction for the regular tensor *deli* (sized $0.5M \times 17M \times 2.5M$) is 41% (communication 30%, MTTKRP 49%, and matrix computations 47%). SPLATT scales better for *deli* overall but the communication cost could be further improved. More irregular tensors tend to yield lower CPD performance and worse scalability on large distributed systems. This phenomenon has also been observed in distributed dense matrix multiplication when matrix dimensions vary significantly [11, 37].

There are three types of load imbalance that play critical roles in what bottlenecks performance on sparse tensors: tensor nonzero, communication volume, and matrix computation imbalance. To measure these imbalances, we introduce three ratios as metrics (see Section 3.1.2). The state-of-the-art works such as medium- [6, 32, 35] and fine-grained [20, 21] distributed CPDs have made efforts to optimize these three types of imbalance. Medium-grained distributed CPD chooses to optimize them separately. However, when it focuses on balancing tensor nonzero, the other imbalances increase significantly for irregular tensors. Fine-grained distributed CPD [20, 21] utilizes hypergraph partitioners to co-optimize these imbalances, but it requires significantly more time overhead in partitioning than actual CPD computation for large tensors. To address these limitations, we present irregularity-aware CPD that co-optimizes different types of imbalance with a low overhead during preprocessing. Our solution provides two insights: First, by evaluating SPLATT theoretically and experimentally, we reveal that these two irregularities lead to unacceptable load imbalance when distributing a sparse tensor among multiple computing nodes. Furthermore, we outline four findings that influence the performance of existing methods. These findings demonstrate that two stages in the preprocessing *grid configuration* and *distribution policy* are critical for the overall CPD performance and scalability. Second, we leverage the sparsity and irregularity information that reflects in the large imbalance of matrix computation. Either MTTKRP or matrix computation could be dominant for different types of tensors. The matrix computation is usually the bottleneck of performance and scalability for most irregular tensors. Therefore, we identify the dominant imbalance ratio as matrix computation imbalance for irregular tensors and optimize it with higher priority. However, focusing only on balancing

matrix computation makes other imbalances worse. It is important to achieve the best tradeoffs between different imbalances in grid configuration and distribution policy.

Grid configuration. The process grid determines the shape of the decomposition in medium-grained CPD [35]. It is critical to the computation and communication costs and requires an intelligent approach over the costly brute-force method. Configuring processes to obtain the optimal performance for an irregular sparse tensor is challenging and is an open question in many fields. SPLATT proposes an easy-to-use method that assigns more processes to longer tensor modes without consideration of nonzero distribution. However, for irregular tensors, the load balance and communication volume are only known at runtime and hard to measure with simple parameters. *We propose a prediction-based grid configuration method in virtual data distribution to determine the optimal process grid at runtime by considering the two irregularities.* We observe that those grids with better performance are more likely to share the same mode-balanced base obtained by SPLATT's method. Therefore, our method solves the limitations by predicting a process grid with the smallest nonzero imbalance among all candidates from a mode-balanced base. We optimize both communication and computation imbalance with a negligible overhead. We build an intermediate grid by optimizing communication imbalance and further find the optimal process grid by an intelligent prediction.

Distribution policy. A distribution policy determines the partitioning of the nonzero and factor matrices. It is also important for the overall performance of CPD because it has an impact on the imbalance of computation and communication and their tradeoffs. SPLATT optimizes nonzero imbalance in distribution policy and expects their communication imbalance has been optimized in the grid configuration stage. It does not work well for irregular tensors, because their communication imbalance increases significantly after distribution policy. From our profiling, we observe that matrix computational kernels and communication could also dominate the CPD execution time, especially for irregular tensors. The CPD execution time of sparse tensors *fb-m* and *deli* in Figure 1 is dominated by matrix computation kernels, rather than MTTKRP. Communication becomes more dominant when scaling to more processors. The CPD time of tensor *deli* is dominated by communication (57%) on 1,536 processors from our experiments, versus 40% occupied on 768 processors. *We design matrix-oriented distribution policies that begins with the matrix-balancing strategy then adjusts according to our two nonzero-balancing strategies.* Our method is based on the matrix-balancing strategy, because we identify the dominant imbalance ratio as matrix computation imbalance for irregular tensors. We then achieve the best tradeoff between different imbalances by balancing nonzero of each partition independently or based on the differences between them.

Our main contributions are summarized as follows:

- Our work investigates the common algorithm structure of state-of-the-art distributed implementations from theoretical and experimental analysis and observes four findings to guide performance optimization (Section 3).
- We demonstrate that the imbalance of computation and communication, and their tradeoffs, are critical to the overall CPD performance and scalability. We identify the dominant imbalance ratio as matrix computation imbalance for irregular tensors. We propose irregularity-aware CPD that co-optimizes these imbalances with high priority in matrix computation imbalance in grid configuration and distribution policy with a low time overhead (Section 4).
- We demonstrate that our method scales well for both regular and irregular tensors when using up to 1,536 processors and obtains up to 4.4× and 11.4× performance improvement over the distributed medium- and fine-grained CPD libraries [20, 35], respectively (Section 5).
- Our optimizations support different sparse tensor formats such as **compressed sparse fiber (CSF)** and **coordinate (COO)**, and more new formats like **Hierarchical Coordinate (HiCOO)**. Our optimizations gain good scalability for all of them (Section 5.6).

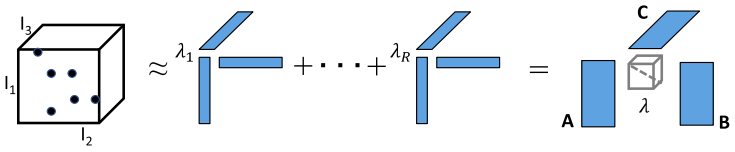


Fig. 2. CPD for a third-order sparse tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$.

Table 1. Symbols and Notations

Symbols	Description
$\mathcal{X}$	A sparse tensor
$\mathbf{X}_{(n)}$	Matricized tensor $\mathcal{X}$ in dimension-n
$\mathbf{A}, \mathbf{B}, \mathbf{C}, \tilde{\mathbf{A}}$	Dense matrices
$\mathbf{a_r}, \mathbf{b_r}, \mathbf{c_r}$	Dense vectors
$\boldsymbol{\lambda}$	Weight vector
N	Tensor order
I_n	Tensor dimension sizes
M	#Nonzeros of the input tensor $\mathcal{X}$
R	Approximate tensor rank (usually a small value)
I_l	Layer size
I_p	#Local matrix rows
P	#MPI processes
$r_{nnz}, r_{vol}, r_{I_p}$	Imbalance ratios for M, communication volume, and I_p

2 BACKGROUND

Tensors, representing multi-dimensional arrays, are one fundamental data representation in real-world HPC applications. We use different fonts for tensors ($\mathcal{X} \in \mathbb{R}^{I \times J \times K}$), matrices ($\mathbf{A} \in \mathbb{R}^{I \times J}$), and vectors ($\mathbf{x} \in \mathbb{R}^{I}$) in this article, following Reference [22]. A nonzero (i, j, k)-element of tensor $\mathcal{X}$ is x_{ijk}. Figure 2 shows a sparse third-order tensor with dots representing nonzero entries. We assume an Nth-order sparse tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with M nonzeros in the subsequent context; sometimes we use a third-order tensor for simplicity. If a tensor $\mathcal{X}$ has one or more dimension(s) that are very small relative to the other dimensions or the nonzero values are not uniformly distributed in one or more dimensions, then we call it an *irregular tensor*. A *slice* is a two-dimensional cross-section of a tensor, obtained by fixing all indices but two, e.g., $\mathbf{S}_{::k} = \mathcal{X}(:, :, k)$. We summarize the symbols and notations in Table 1.

2.1 Distributed CPD

CANDECOMP/PARAFAC decomposition (CPD) factorizes a tensor into a sum of component rank-one tensors [22]. Figure 2 illustrates a third-order CPD. In general, CPD approximates an Nth-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ as

$$\mathcal{X} \approx \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)} \equiv [\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!], \tag{1}$$

where R is the canonical rank of tensor $\mathcal{X}$, the number of component rank-one tensors [22]. In a low-rank approximation, R is usually chosen to be a small number less than 100. The outer product of the vectors $\mathbf{a}_r^{(1)}, \ldots, \mathbf{a}_r^{(N)}$ produces R rank-one tensors. $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}, n = 1, \ldots, N$ are the *factor matrices*, each formed by taking the corresponding vectors as its columns. The vector lambda can

be represented as a superdiagonal lambda tensor. For Tucker decomposition, this core tensor is usually not superdiagonal [39]. We normalize these vectors to unit magnitude and store the factor weights in the vector $\boldsymbol{\lambda} = \{\lambda_1, \ldots, \lambda_r\}$. Typically, the factor matrices $\mathbf{A}^{(n)}$ are given initial values and solved iteratively.

Data decomposition and distribution.

For large tensors, the number of nonzeros M and the resulting factor matrices $\mathbf{A}^{(n)}$ are large and easily exceed the memory capacity of a single node. To meet the needs of large-scale data processing, distributed CPD algorithms, such as coarse-grained [13], medium-grained [2, 35], and fine-grained [20, 21] strategies, have been developed. Medium-grain is one of the most successful from References [2, 6, 32, 35] and is the baseline for this work (described in Section 2.3). To efficiently store large tensors, we consider one state-of-the-art tensor format, **Compressed Sparse Fiber (CSF)** for general unstructured sparse tensors. CSF [35] is a hierarchical and fiber-centric format that effectively generalizes the **Compressed Sparse Row (CSR)** sparse matrix format to tensors.

Distributed algorithm. We focus on the most popular medium-grained, bulk-synchronous distributed CPD algorithms [2, 6, 32, 35], adopted in multiple libraries, including SPLATT, the Surprisingly ParalleL spArse Tensor Toolkit [36], and ENSIGN [23]. It has shown outstanding performance and scalability as well as efficient memory usage compared to the counterparts [13, 20], evaluated in References [2, 6, 32, 35]. Medium-grained tensor distribution, an N-dimensional partitioning (N as tensor order) on a tensor, corresponds to a 2D stationary algorithm in traditional dense matrix multiplication [33], which has been proven to be performance-efficient in the SUMMA algorithm [40] included in ScaLAPACK [12] and PLAPACK [3] libraries.

2.2 Related Work

Distributed CP decompositions. Three major bulk-synchronous distributed CPD algorithms have been proposed: coarse-grained [13], medium-grained [2, 6, 35], and fine-grained [20]. References [32, 35] showed that medium-grained CPD generally obtains the optimal state-of-the-art performance. SPLATT [36] is a popular sparse tensor library that includes medium- and fine-grained distributed CPD implementations. ENSIGN [23] uses special sparse tensor data structures **mode-specific sparse (MSS)** and **mode-generic sparse (MGS)** with an optimization that improves data reuse and reduces redundant computations in tensor decompositions [6]. But ENSIGN requires significantly higher memory usage due to its special data structures. ALTO [15] is proposed as a novel sparse tensor format for high performance of tensor operations. ALTO outperforms the state-of-the-art CPD implementation based on CSF format by using a mode-agnostic tensor representation that improves data locality. Other efforts employed MapReduce/Hadoop or Spark programming models on cloud platforms, such as GigaTensor [18], HaTen2 [17], and CSTF [7]. Our work developed upon medium-grained distributed CPD and through optimizing grid configuration and distribution policy to improve performance.

Grid configuration. Some distributed work also studied the approach to find the optimal process grid configuration. However, they only consider tensor dimension sizes without taking tensor irregularity and sparsity into account [21, 35]. Our work designs irregularity-aware grid configuration based on prediction to generate the most suitable process grid and balances performance impacting factors.

Partitioning methods. Various partitioning methods are proposed to balance computation and communication. Lite is proposed for Tucker decomposition as a lightweight distribution scheme [9]. But due to the difference between Tucker decomposition and CPD, Lite focuses more on balancing computation without explicitly optimizing communication volume [9]. Ballard et al. [5] have discussed communication lower bounds for MTTKRP, but they did not consider computation

and communication in matrix-related kernels. Cartesian partitioning used in medium-grained algorithm focuses on reducing maximum communication volume [35]. Hypergraph partitioning works well in balancing MTTKRP and reducing average communication overhead, but has worse communication balance [2, 20, 35]. CartHP [2] is a novel hypergraph-partitioning model that utilizes sparsity for minimizing the total communication volume. It is possible that hypergraph partitioners can be used to produce similar partition with our method by certain parameters. But it is difficult to find these parameters in hypergraph partitioners. And hypergraph partitioners require significantly more time overhead in partitioning than actual CPD computation for large tensors. Another hypergraph-based partitioning method [19] is proposed for **Non-negative Matrix Factorization (NMF)**, while it focuses more on optimizing communication volume with the sacrifice of other load imbalances. Our partition optimizations are more matrix-oriented and consider the tradeoff in balancing MTTKRP, matrix computation, and communication to obtain the optimal CPD performance.

2.3 Medium-grained, Bulk-synchronous Distributed CPD Algorithm

We extract the general medium-grained, bulk-synchronous distributed CPD algorithm as a template in Algorithm 1, named as MGBS-CPD, extracted from the state-of-the-art works cited in References [2, 6, 35].

Medium-grained data distribution. The medium-grained decomposition uses a nonzero-oriented data decomposition strategy. After loading a tensor file into each process' memory in a distributed way (Line 1), two performance-critical steps follow: process grid configuration (Line 2) and distribution policy determination (Line 3). (Refer to Section 3.2 for details of these two steps.) Based on these two steps, a tensor $\mathcal{X}$ is N-dimensional partitioned into subtensors in a non-overlapping fashion and distributed to processes; each factor matrix $\mathbf{A}^{(n)}$ is distributed to the processes according to the distribution policy on each dimension-n.

Take a $P = 2 \times 3 \times 2$ process grid[1] in Figure 3 as an example. The tensor $\mathcal{X}$ is partitioned to $2 \times 3 \times 2$ subtensors, each associated with a process and saved in its memory. Meanwhile, each $\mathbf{A}^{(n)}$ is partitioned to P submatrices along its dimension with two levels: the *layer-level* corresponds to the tensor computation (dashed red lines) and splits each matrix to sub-matrices $\mathbf{A}_l^{(n)}$ affiliated to its row dimension (blank boxes on $\mathbf{A}$), and the *process-level* further evenly splits $\mathbf{A}_l^{(n)}$ to $\mathbf{A}_p^{(n)}$ for each process p in the corresponding subgrid (dashed lines on $\mathbf{A}$). Dashed orange lines show submatrices for the first process in Figure 3. Note that $\mathbf{A}_p^{(n)}$ is the actual local matrix storage per process, while $\mathbf{A}_l^{(n)}$ is only stored during tensor-matrix computation (MTTKRP, described below).

Bulk-synchronous parallel algorithm. Computation is accordingly partitioned with the above data decomposition—i.e., each process only does local tensor/matrix computation and updates its own matrix partition $\mathbf{A}_p^{(n)}$. Thus, the grid configuration and distribution policy, which determine the data decomposition, play critical roles in the performance of CPD algorithm.

Algorithm 1 shows the bulk-synchronous parallel algorithm for an Nth-order tensor using a traditional alternating least square algorithm [22]. The bulk-synchronous parallel algorithm is generalized from almost all existing distributed CPD-ALS implementations [2, 6, 7, 13, 20, 21, 32, 35]. This is an iterative implementation. In each iteration, matrices are updated one-by-one; each time, all but one matrix are fixed to update the matrix $\tilde{\mathbf{A}}^{(n)}$. The algorithm comprises four main computation kernels. MTTKRP is the only kernel that computes on the sparse tensor and has been studied most for optimization in previous work [20, 21, 35]. The other three compute on dense matrices only. Note that all the four steps except MAT SOLVE have mixed computation and communication.

[1]Due to our hybrid MPI+OpenMP implementation, the MPI processes count is referred in grid configuration.

ALGORITHM 1: Medium-grained, bulk-synchronous distributed CPD-ALS algorithm (MGBS-CPD).

Require: An Nth-order sparse tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with M nonzeros, P MPI processes;
Ensure: Vector $\boldsymbol{\lambda}$ and dense matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}, n = 1, \ldots, N$;

// Variables
Initialize matrices $\mathbf{A}^{(n)}, n = 1, \ldots, N$;
$\mathbf{A}_l^{(n)}$ is the layer-distributed matrix, needed by MTTKRP computation on p.
$\mathbf{U}_n \in \mathbb{R}^{R \times R}, n = 1, \ldots, N$ is local temporary data.
// Preprocessing
1: Distributedly load $\mathcal{X}$ to P MPI processes' local memory
2: **Grid configuration** $\mathcal{G}$: Get rank dimensions $P_n, n = 1, \ldots, N$ decomposed from P and initialize MPI communicator
3: Determine a **distribution policy** $\mathcal{D}$
▷ Tensor partitioning, $\mathcal{X}_p$ locally owned by process p.
4: Redistribute $\mathcal{X}$ according to $\mathcal{D}$
▷ Matrix partitioning, $\mathbf{A}_p^{(n)}$ locally owned by process p.
5: Distribute all $\mathbf{A}^{(n)}$ to $\mathbf{A}_l^{(n)}$ and $\mathbf{A}_p^{(n)}, n = 1, \ldots, N$ according to $\mathcal{D}$
6: Get $\mathcal{X}_p$ after removing empty slices and get index mapping from $\mathcal{X}_p$ to $\mathcal{X}$
7: Get the indices in $\mathbf{A}_p^{(n)}$ that need to communicate in AlltoAll($\mathbf{A}_p^{(n)}$)
8: Randomly initialize $\mathbf{A}_l^{(n)}$
// Computation
9: $\mathbf{A}_l^{(n)} = \text{AlltoAll}(\mathbf{A}_p^{(n)})$; $\mathbf{U}_n = \text{AllReduce}\ (\mathbf{A}_p^{(n)T} \mathbf{A}_p^{(n)})$
10: **do**
11: **for** $n = 1, \ldots, N$ **do**
12: $\tilde{\mathbf{A}}_l^{(n)} = \text{MTTKRP}(\mathcal{X}_p, \mathbf{A}_l^{(1)}, \ldots, \mathbf{A}_l^{(n-1)}, \mathbf{A}_l^{(n+1)}, \ldots, \mathbf{A}_l^{(N)})$ ▷ MTTKRP
13: $\tilde{\mathbf{A}}_p^{(n)} = \text{AlltoAll}(\tilde{\mathbf{A}}_l^{(n)})$
14: $\tilde{\mathbf{A}}_p^{(n)} = \tilde{\mathbf{A}}_p^{(n)}\ (\mathbf{U}_1 * \cdots * \mathbf{U}_N)^{\dagger}$ ▷ MAT SOLVE
15: $\tilde{\boldsymbol{\lambda}} = \text{Normalize}\ (\tilde{\mathbf{A}}_p^{(n)})$ ▷ MAT NORM
16: $\tilde{\mathbf{U}}_n = \text{AllReduce}(\tilde{\mathbf{A}}_p^{(n)T} \tilde{\mathbf{A}}_p^{(n)})$ ▷ MAT A^TA
17: $\tilde{\mathbf{A}}_l^{(n)} = \text{AlltoAll}(\tilde{\mathbf{A}}_p^{(n)})$
18: **end for**
19: **while** fit not change or maximum iterations exhausted

- MTTKRP (Line 12): each process computes the Khatri-Rao product of its subtensor with all but one layer-partitioned $\mathbf{A}_l^{(1)}, \ldots, \mathbf{A}_l^{(n-1)}, \mathbf{A}_l^{(n+1)}, \ldots, \mathbf{A}_l^{(N)}$, which are obtained from remote memory by communicating with other processes.
- MAT SOLVE (Line 14): each process updates $\tilde{\mathbf{A}}_p^{(n)}$ using the Cholesky method[2] based on the temporary results from MTTKRP.
- MAT NORM (Line 15): each process normalizes $\tilde{\mathbf{A}}_p^{(n)}$ locally and then performs a parallel reduction to obtain $\boldsymbol{\lambda}$.

[2]The default matrix solver is Cholesky, because in most cases matrices are small (matrix rank R < 100) and SPD. Both SPLATT and our implementation use SVD solver in case the matrix is not SPD.

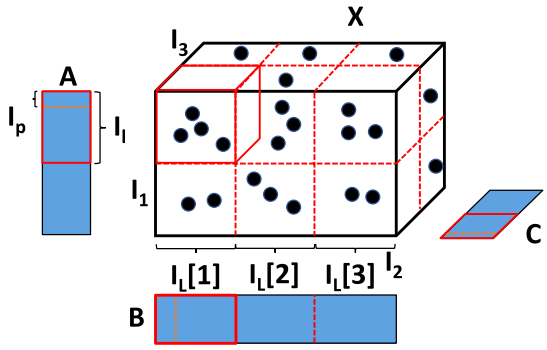


Fig. 3. Tensor and matrix distribution over a $12 = 2 \times 3 \times 2$ process grid. Dotted lines on matrices indicate local matrix storage in one process. The tensor is partitioned into $2 \times 3 \times 2$ subtensors, each mapped to a process. Each factor matrix is first partitioned by the layers (dashed red lines) affiliated with tensor partition and then evenly split among the corresponding process subgrid. Dashed orange lines show submatrices for the first process.

- MAT A^TA (Line 16): each process uses symmetric matrix multiplication locally and then performs a reduction to form the new $\tilde{\mathbf{U}}_n$ for the next iteration.
- Other COMM (Lines 13,17): $\tilde{\mathbf{A}}_p^{(n)}$ is updated by communicating $\tilde{\mathbf{A}}_l^{(n)}$ after local MTTKRP computation. Consequently, communications are involved to update $\tilde{\mathbf{A}}_l^{(n)}$ from $\tilde{\mathbf{A}}_p^{(n)}$ to prepare the layer-partitioned $\tilde{\mathbf{A}}_l^{(n)}$ for the next MTTKRP.

The complexity lies in both communication and local computations influenced by the grid configuration and distribution policy from the preprocessing steps. All communication within CPD computation (Steps 9 to 19 in Algorithm 1) is for dense matrices, while sparse communication only exists in preprocessing (Step 4) for sparse tensors. Due to sparsity of the tensor, the communication volume for dense matrices could be very imbalanced. The computational complexity does not always reflect time overhead. MAT NORM can be more expensive than MAT A^TA and MAT SOLVE, because the latter two operations are implemented with BLAS functions.

3 LEARNING THE PERFORMANCE OF DISTRIBUTED TENSOR DECOMPOSITIONS

This section illustrates the general medium-grained, bulk-synchronous distributed CPD algorithm and its performance problem abstraction and analysis along with our four findings.

3.1 Problem Statement and Analysis

We first present general models to capture the execution time of medium-grained distributed CPD in Algorithm 1. Our target is to find the optimal data distribution by designing a grid configuration and distribution policy, to obtain the best CPD performance, expressed in Equation (2). The optimal grid configuration $\mathcal{G}_{opt}$ and distribution policy $\mathcal{D}_{opt}$ have the minimum overall execution time. The execution time of CPD is dominated by the iterations (Lines 7–16) in Algorithm 1. We use the time of one iteration to represent the CPD execution time, noted by T_{cpd}, which aligns with our experiments.

$$\mathcal{G}_{opt}, \mathcal{D}_{opt} = argmin_{\mathcal{G},\mathcal{D}} T_{cpd} \tag{2}$$

3.1.1 Execution Time Analysis. T_{cpd} consists of the aforementioned five steps: MTTKRP, MAT SOLVE, MAT NORM, MAT A^TA, and other COMM. Due to the bulk synchronous feature of

Table 2. Time Complexity of the Five Steps in MGBS-CPD

Key Steps	Comp	Comm Vol
MTTKRP (T_{mttkrp})	$O(c_N \times R \times M_p)$	/
MAT SOLVE (T_{solve})	$\Theta(R^2 \times I_p)$	/
MAT NORM (T_{norm})	$\Theta(R \times I_p)$	R
MAT A^TA (T_{ata})	$\Theta(R^2 \times I_p)$	R^2
Other COMM (T_{ocomm})	/	$R(I_l - I_p) + RI_p$

MGBS-CPD, T_{cpd} is expressed in Equation (3).

$$\begin{aligned} T_{cpd} &= T_{mttkrp}(c_N, R, M_p) + T_{ocomm}(P, I_l, I_p) \\ &\quad + (T_{solve}(R, I_p) + T_{norm}(P, R, I_p) + T_{ata}(P, R, I_p)) \end{aligned} \tag{3}$$

The time complexity and communication volume per process of each step are listed in Table 2. Two collective communications are employed to synchronize and update local data, *MPI_Alltoall* in Other COMM and *MPI_Allreduce* in MAT NORM and MAT A^TA. Both SPLATT and our algorithm implement *Alltoall* for the communication. We use MPI Communicators for layers and, in each layer, we use *Alltoall* across processors in one layer. The communication time is modeled as $\alpha + \beta n$, where α and β are the memory latency and bandwidth, respectively, and n is the number of bytes to be transferred [38]. We assume the tensor rank R (usually a small value < 100) and $c_N < N$ are constants.[3] T_{cpd} is mainly determined by the number of nonzeros of a local sparse tensor M_p, layer size I_l, and local matrix size I_p, though computation and communication are different functions of these variables. M_p dominates T_{mttkrp}; I_p affects the time complexity of all matrix steps, $T_{solve}, T_{norm}, T_{ata}$; I_l and I_p both influence the other communications T_{ocomm}.

Comparing these steps, we see that, in general, M_p is several orders of magnitude larger than I_l and I_p for relatively small or mildly sparse tensors, where T_{mttkrp} might take a larger percentage in T_{cpd}. However, M_p could be in the similar order-of-magnitude as I_l and I_p for relatively sparse tensors or tensors with irregular shapes, where matrix computations and communication might have non-negligible costs. Besides, we also observe that some configurations of $\mathcal{G}, \mathcal{D}$ could decrease the execution time of one step but increase that of other step(s). (Experiments in Section 3.2 verify this analysis.) Thus, it is non-trivial to infer the optimal settings for $\mathcal{G}, \mathcal{D}$ to gain the highest distributed performance only relying on theoretical analysis even with c_N, P, R all fixed, plus the analysis is closely related to the features of input sparse tensors.

3.1.2 Load Imbalance Ratios. Thus far, we consider M_p, I_l, and I_p as the average values on each process, which is the ideally balanced data distribution. However, in reality, especially for irregular sparse tensors, the data distribution could be very skewed. We present three imbalance ratios as metrics to measure this effect.

We use a more accurate *imbalance ratio* r, adapted from the one used in Reference [35],[4] to represent the imbalance of sparse tensor computation, matrix computation, and communication. From Table 2, sparse tensor computation, MTTKRP, is influenced by M_p. Nonzero imbalance ratio $r_{nnz} = (max\{M_p\} - min\{M_p\})/max\{M_p\}$ represents the gap between the maximal and minimal number of nonzeros assigned to a process among P processes. Our imbalance ratio r, always less than 1.0, better evaluates long and short jobs per process. A ratio close to 0.0 means an ideal, even nonzero distribution; while a ratio close to 1.0 means extreme imbalance indicating that the

[3] c_N is a constant for a given tensor in an MTTKRP algorithm [25, 35].

[4] The nonzero imbalance in Reference [35] represents the gap between the maximal and average number of nonzeros assigned to a process, which cannot measure the imbalance from the short tasks well.

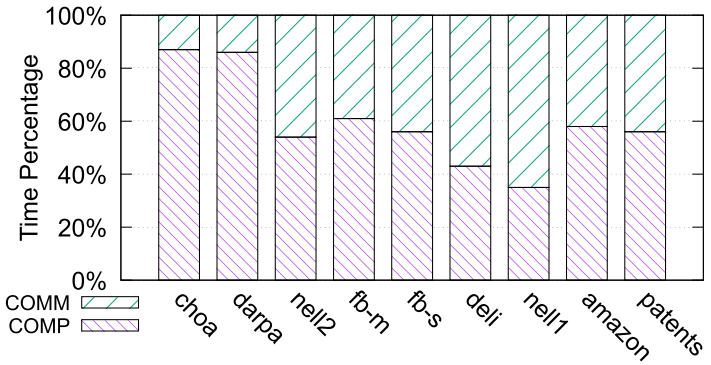


Fig. 4. Computation and communication percentage of CPD.

gap between the longest and shortest MTTKRP execution time is huge. Analogously, r_{Ip} represents the imbalance ratio of I_p, thus for matrix computation, r_{vol} is the imbalance ratio of communication volume. We use the imbalance ratio for communication volume rather than I_l, because the communication volume is influenced by both I_l and I_p; therefore, r_{vol} better represents the communication. The three imbalance metrics help determine the $\mathcal{G}_{opt}, \mathcal{D}_{opt}$ by reflecting features of real sparse tensors from three distribution-related perspectives.

3.2 Findings

Based on our theoretical analysis and the proposed imbalance ratios, we discuss performance findings on MGBS-CPD. The tests are run on the open-source SPLATT MPI library [36], representing a fast state-of-the-art MGBS implementation from References [2, 6, 32, 35].

Finding 1: *Both computation and communication have non-negligible costs, and the dominance varies with tensors.* We compare actual computation and communication time in results and use T_{mttkrp} and T_{ocomm} to give a rough theoretical analysis. Comparing the dominant parameters: M_p and I_l, either one could be larger for different sparse tensors. For example, tensor *choa* has a maximum $M_p = 400K$, $I_l = 15K$, while tensor *deli* has $M_p = 2M$, $I_l = 4M$ on 768 processors. Thus, either computation or communication could be dominant among different tensors. We further study the overall performance of the SPLATT CPD implementation running on 768 processors. Figure 4 depicts the percentage of the execution time taken by computation and all types of communication operations in Algorithm 1, respectively, on nine sparse tensors from real applications (refer to Section 5 for tensor descriptions). Computation takes 35%–81%, while communication takes 19%–65% of the total execution time. Computation largely dominates the CPD execution on two tensors: *choa* and *darpa*; communication largely dominates on tensors *nell1* and *deli*. This matches the M_p and I_l examples given above. The computation is mainly impacted by M_p and the communication is mainly impacted by I_l. Therefore, computation is dominant for tensors whose $M_p > I_l$ (such as *choa* and *darpa*), while communication is dominant for tensors whose $M_p < I_l$ (such as *nell*1 and *deli*). On the rest of five tensors, computation and communication take a similar amount of time with a percentage difference less than 10%. The shifting of dominance between computation and communication among tensors raises the difficulty of performance optimization. Taking tensor dimension sizes into consideration, *fb-m*, *fb-s*, *choa*, and *patents* are more irregular tensors in Table 3 and tend to be computation-dominated, while the other tensors are more communication-dominated or without significant dominance.

Finding 2: *Computation cost is not always dominated by sparse tensor computation, but also dense matrix computations.*

Table 3. Description of Sparse Tensors

Tensors	Dimensions	#Nonzeros	Density
choa	$712K \times 10K \times 767$	27M	5.0×10^{-6}
darpa	$22K \times 22K \times 24M$	28M	2.4×10^{-9}
nell2	$12K \times 9K \times 29K$	77M	2.4×10^{-5}
random	$100K \times 100K \times 100$	100M	1.0×10^{-4}
fb-m	$23M \times 23M \times 166$	100M	1.1×10^{-9}
fb-s	$39M \times 39M \times 532$	140M	1.7×10^{-10}
deli	$533K \times 17M \times 2.5M$	140M	6.1×10^{-12}
nell1	$2.9M \times 2.1M \times 25M$	144M	9.1×10^{-13}
amazon	$4.8M \times 1.8M \times 1.8M$	1,742M	1.1×10^{-10}
patents	$46 \times 239K \times 239K$	3,597M	1.4×10^{-3}

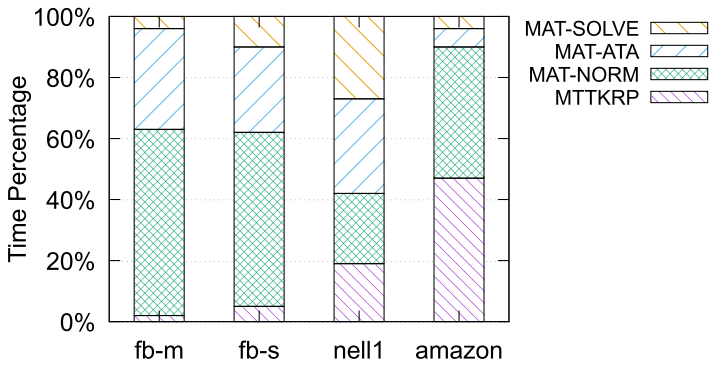


Fig. 5. Time percentage of computational kernels of Cpd.

Compare the computation complexity of matrix operations, MAT SOLVE, NORM, A^TA, versus the Mttkrp complexity in Table 2 shows $I_p < c_N \times M_p$ is generally true if there are not many empty slices in dimension-n. However, $R \times I_p < c_N \times M_p$ is not necessarily true and depends on the values of R, the constant c_N ($R > c_N$ usually), the distribution policy that determines the sparsity pattern of the local tensor $\mathcal{X}_p$ and influence value I_p in the next process-distribution for matrices. This is especially prudent for irregular tensors with $I = \Theta(M)$ on one dimension. If $R \times I_p > c_N \times M_p$, then the complexity of MAT A^TA and SOLVE steps could take more time than Mttkrp. While these matrix operations are all dense and generally perform more efficiently than the sparse Mttkrp, dense matrix computation can influence computational performance. We conclude Finding 2 that Mttkrp is not always the dominant computational kernel in Cpd, the matrix computation kernels are also expensive as tensor rank grows and for tensors with preferable sparse patterns (e.g., irregular tensors). Therefore, the state-of-the-art work [2, 13, 35] that focuses on minimizing the computational cost of Mttkrp may not gain much performance improvement for all types of tensors.

Figure 5 shows the time percentage of the four computational steps on four representative tensors: *fb-m*, *fb-s*, *nell1*, and *amazon*, verifying our theoretical analysis above. For the four tensors, Mttkrp, MAT NORM, MAT A^TA, and MAT SOLVE take 2%–47%, 23%–61%, 6%–33%, and 4%–27% of the Cpd computation time, respectively. The other three computations easily take more execution time than Mttkrp, which needs to be optimized as well for better performance. These insights about dominating costs of Findings 1 and 2 could guide our following optimization for distribution policy.

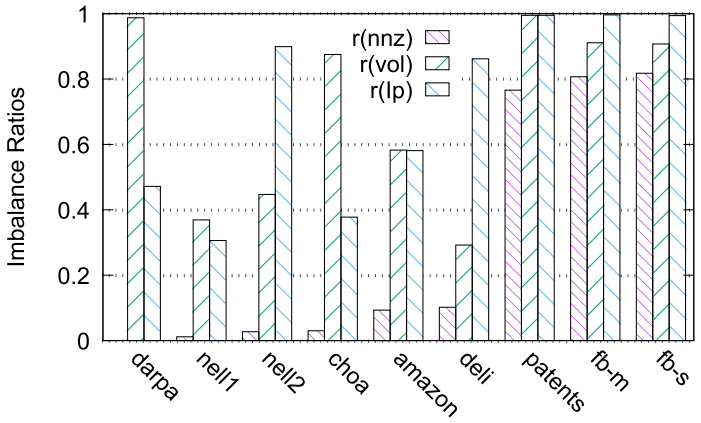


Fig. 6. Load imbalance ratios (r_{nnz}, r_{vol}, and r_{Ip}) for sparse tensors.

Finding 3: *Different load imbalance factors influence computation and communication overhead.*

Figure 6 shows these three ratios r_{nnz}, r_{vol}, and r_{Ip} for sparse tensors as the increasing order of r_{nnz}, where r_{nnz} and r_{Ip} reflect computation imbalance and r_{vol} reflects communication imbalance. The nonzero imbalance is less than 0.2 for the left six tensors, while tensors *patents* ($46 \times 239K \times 239K$), *fb-m* ($23M \times 23M \times 166$), and *fb-s* ($39M \times 39M \times 532$) have a much higher nonzero imbalance, all of which are very irregular in dimension sizes. All the tensors have much higher volume and I_p imbalance ratios than nonzero imbalance ratios. Different from the dominance perspectives in Findings 1 and 2, the imbalance ratios expose the load imbalance issues that influence the overhead of all the five steps in Table 2 correspondingly. Almost all tensors have at least one imbalance ratio with the value higher than 0.8, which indicates the difficulty to do a good tradeoff among the three imbalance ratios. The state-of-the-art work puts efforts on optimizing the nonzero imbalance [10, 35], which only influences sparse tensor computation. Therefore, they only target minimizing the tensor computation imbalance, not communication or the other matrix computation imbalances.

Finding 4: *The grid calculated from only tensor dimensions is usually not the optimal. And different grid configurations could lead to very different distributed* CPD *performance.*

For a given tensor, the process grid on which the tensor is mapped determines the computation and communication costs from the first sight, even before the distribution policy takes effect. Figure 3 shows the tensor and matrix decomposition on 12 processes as a $2 \times 3 \times 2$ grid. Given 12 processes, there are 18 unique configurations on which the tensor can be mapped to the processes. Configurations $12 \times 1 \times 1$, $1 \times 12 \times 1$, and $1 \times 1 \times 12$ are considered as different ones due to partitioning the first, second, and third dimensions correspondingly. A cluster with hundreds of nodes will have thousands of configurations or more. Figure 7 shows all grid configurations for 16 MPI processes, with the execution time varying up to 3.5×. The traditional method for grid configuration computes $4 \times 2 \times 2$ based on *amazon*'s tensor dimension sizes ($4.8M \times 1.8M \times 1.8M$). But the optimal grid is $16 \times 1 \times 1$ due to the tensor's sparsity. Thus, finding the optimal process grid is critical to choosing the distribution policy and overall performance, which also requires an intelligent approach over the costly brute-force method.

4 IRREGULARITY-AWARE CPD

The four findings above motivate our optimizations in considering different tensor irregularity and finding the optimal grid configuration $\mathcal{G}$ and distribution policies $\mathcal{D}$ to improve runtime performance. This section presents our proposed irregularity-aware CPD. We propose new methods

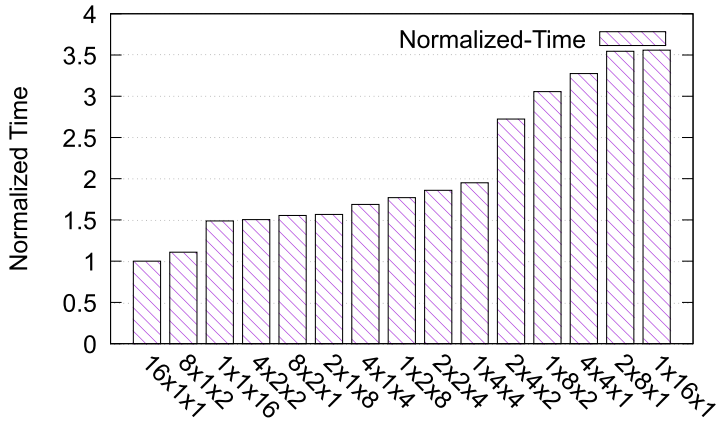


Fig. 7. Normalized time of all possible grid configurations compared to the slowest one for tensor amazon on 16 MPI processes.

for grid configuration and distribution policy, and the implementations of them are detailed in Algorithms 2 and 3.

ALGORITHM 2: Prediction-based grid configuration with $n_{pr} = 2$.

Require: Number of processes P, tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$;
Ensure: Grid configuration $\mathcal{G}_{opt} = \{P_1, P_2, P_3\}$, $P_1 \times P_2 \times P_3 = P$;
1: Initialize intermediate grid $\mathcal{G}_{int} = \{1, 1, 1\}$
// Step 1: intermediate grid generation
2: prs_o = getPrimes(P); ▷ Ordered from large to small
3: $I_{avg} = (I_1 + I_2 + I_3)/3$
4: **for** pr in $prs_o[1:-1]$ **do**
5: $\mathcal{G}_{int}[n]* = pr, s.t. I_n = max\{I_1, I_2, I_3\}$
6: $I_n - = I_{avg}$
7: **end for**
// Step 2: sparsity-aware grid trimming
8: Initialize six grid candidates $\mathcal{G}_1, \ldots, \mathcal{G}_6 = \mathcal{G}_{int}$
9: $\mathcal{G}_i* = (prs_o[-2] * prs_o[-1]), i = \{1, 2, 3, 4, 5, 6\}$ ▷ Assign two smallest primes to six candidates
10: Compute r_{layer_nnz} to predict r_{nnz} of $\mathcal{G}_1, \ldots, \mathcal{G}_6$ with virtual data distribution
11: $\mathcal{G}_{opt} = \mathcal{G}_i, s.t. min_{r_{nnz}}\{\mathcal{G}_1, \ldots, \mathcal{G}_6\}$
12: **Return** $\mathcal{G}_{opt}$;

4.1 Prediction-based Grid Configuration

It is important to find the optimal process grid, because the performance varies a lot between different grid configurations based on our Finding 4 in Section 3. Figure 8 compares two example grid configurations: $2 \times 3 \times 2$ and $2 \times 2 \times 3$. In Conf. 1, tensor $\mathcal{X}$ is split to two pieces in mode-I and three pieces in mode-J; Conf. 2 is the opposite. Distribution on mode-K is the same. Assume $J > K$, ostensibly, Conf. 1 should be more reasonable than Conf. 2 by splitting the larger dimension. For a dense tensor $\mathcal{X}$, this is true. The different matrix distribution on $\mathbf{A}$ and $\mathbf{B}$ could lead to uneven matrix communications, thus influencing overall CPD performance. We prove this using a dense, cubical third-order tensor $\mathcal{X} \in \mathbb{R}^{I \times I \times I}$ along with three matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I \times R}$, $n = 1, 2, 3$, distributed on $P = P_1 \times P_2 \times P_3$. From Algorithm 1, the data to be communicated is dominated by $\tilde{\mathbf{A}}_l^{(n)} - \tilde{\mathbf{A}}_p^{(n)}$

ALGORITHM 3: Matrix-oriented distribution policy generation in tensor dimension n.

Require: Sparse tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, number of processes P_n in dimension-n;
Ensure: Distribution policy $\mathcal{D}$ (a.k.a. layer configuration $\{I_L\}$);

```
// Matrix-balancing strategy: set
for i in P_n do
    I_{L_i} = I_n / P_n;                                   ▷ Initial layer size
end for
if Ordered adjustment then
    // Ordered adjustment strategy: ordered-c
    for i in P_n do
        m_i = #nonzeros in layer L_i                       ▷ c is a user-given parameter
        I_{L_i} -= (m_i − M/P_n)/(c . S_{L_i})
    end for
else if Max-to-min adjustment then
    // Max-min adjustment strategy: max-min
    I'_L: Sorted {I_{L_i}, i = 1, ..., P_n} by #nonzeros in a descending order
    for i in P_n/2 do
        I'_L[i] −= (I'_L[i] − I'_L[P_n − i])/S_n
        I'_L[P_n − i] += (I'_L[i] − I'_L[P_n − i])/S_n
    end for
    I'_L = I_L
end if
Return D = {I_L};
```

and $\tilde{\mathbf{A}}_p^{(n)}$ to communicate in its own layer. For each inside loop, its communication volume in the first dimension is $P_2P_3(\frac{I}{P} + (\frac{I}{P_1} - \frac{I}{P})) = P\frac{I}{P_1^2}$. Thus, the total volume of Cpd in all dimensions is

$$VOL_{comm} = I \times P \times \left(\frac{1}{P_1^2} + \frac{1}{P_2^2} + \frac{1}{P_3^2}\right). \tag{4}$$

According to Cauchy-Schwarz inequality, the minimum of the total volume is obtained when $P_1 = P_2 = P_3$. For a cubical dense tensor, equally splitting the dimension sizes obtains the minimum communication cost. For a tensor with irregular shape, we proportionally assign more processes to a longer dimension to maintain the minimum communication. The state-of-the-art work [21, 35] developed an easy-to-use prediction algorithm based on the above idea. It assigns the number of processes based on the tensor dimension sizes. However, for irregular sparse tensors with a non-uniform nonzero distribution, their method leads to severe imbalance for computation and communication.

To solve their problem, we propose a new online prediction algorithm that simultaneously considers communication volume and nonzero balance when deciding the process grid. Our key idea is to find a process grid with the smallest nonzero imbalance from a mode-balanced foundation. We have two steps to achieve the above goal. First, we build an intermediate process grid that leads to balanced communication and matrix computations based on the existing work [21, 35]. This intermediate grid uses most but not all of the processes. Second, we construct the grid candidates by adjusting the intermediate grid with the remaining process(es) and predict the optimal grid among them. Prediction is leveraged to make a balance among the imbalance ratios in Section 3.1.2.

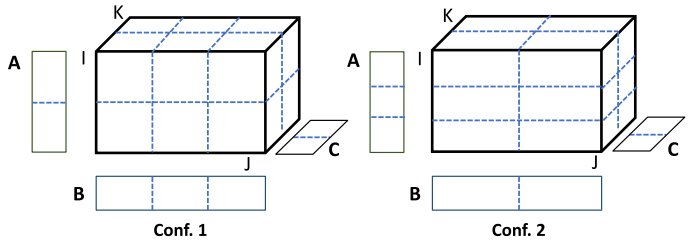


Fig. 8. Two example grid configurations for 12 processes.

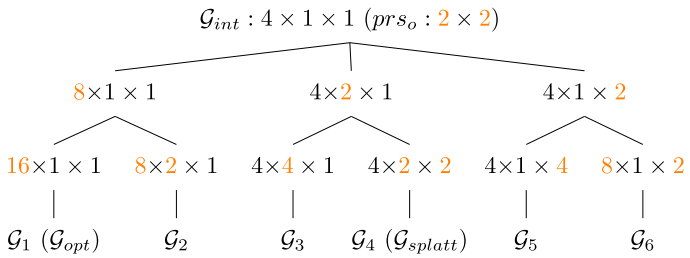


Fig. 9. Six grid candidates on 16 (=$2 \times 2 \times 2 \times 2$) processes for tensor *amazon* with $n_{pr} = 2$. We assign two smallest primes (2×2) to $\mathcal{G}_{int}$ and obtain six grid candidates.

Algorithm 2 illustrates our method. Our goal in the first step is to form an intermediate grid as a base of all candidates. The brute-force results indicate those grids with better performance are more likely to share the same base. For example, 4 of top 5 grids have the base of $4 \times 1 \times 1$ in Figure 7. Therefore, we need to build this balanced base first. To form this intermediate grid, we first find all the prime factors of the total process count and sort them in descending order in prs_o. Using all but the last n_{pr} factors, we form an intermediate grid $\mathcal{G}_{int}$ (Line 8). For example, $n_{pr} = 1$ indicates the smallest prime factor is unused in the intermediate grid. $n_{pr} = 2$ means that we will generate 6 candidates, as Figure 9 shows. We choose $n_{pr} = 2$, because we observe that for most cases the best grid is in 6 candidates and the overhead of computing imbalance ratios for 6 candidates is negligible. Specifically, it repeatedly assigns the largest prime factor to the current longest tensor dimension, which dynamically changes after each loop iteration. After the loop ends, the intermediate grid $\mathcal{G}_{int}$ has a best effort in balancing communication and matrix computations. We assign the remaining n_{pr} primes to form a complete process grid in the following step:

The key idea in the second step is to build all possible candidates and identify the optimal grid among them by predicting their nonzero imbalance. We form six grid candidates from $\mathcal{G}_1$ to $\mathcal{G}_6$ with $\mathcal{G}_{int}$ by assigning two smallest primes to each dimension. Figure 9 displays how we form all candidates from $\mathcal{G}_{int}$ for tensor *amazon* with 16 MPI processes (Lines 8–11 in Algorithm 2). The prime factors are {2,2,2,2}. The first step builds $\mathcal{G}_{int}$ as $4\times1\times1$ by the first two prime factors based on *amazon*'s dimension size as $4.8M \times 1.8M \times 1.8M$. We build six candidates after assigning the remaining n_{pr} primes 2×2. These six candidates are considered to have an equal chance to obtain the optimal performance from the first step with tensor dimension size and implied communication information. To identify the optimal grid among them, we need to predict the nonzero imbalance ratio r_{nnz} for each candidate. We do not actually distribute data to processes in different nodes for each candidate grid. If we want to compute the actual r_{nnz} with the nonzeros of each process (M_p) as stated in Section 3.1.2, then we need to take the tensor slice information to determine the index range of each process. However, the above computation of r_{nnz} has a complexity of $O(c_N \times P \times M_p)$

for each candidate. This is expensive for tensors with large amounts of nonzeros. We present a new metric r_{layer_nnz} as the imbalance ratio of nonzeros among different layers to predict r_{nnz}. Figure 3 displays the layers affiliated with the tensor partition. In mode I_2 there are three layers each with 4 subtensors. Particularly, we take the tensor slice information to compute the nonzeros of each layer Ln_p in each mode. In mode I_n, $r_{layer\_nnz}(n) = (max\{Ln_p\} - min\{Ln_p\})/max\{Ln_p\}$. We then compute r_{layer_nnz} as the average $r_{layer\_nnz}(n)$ for all modes. The total complexity is $O(c_N \times I_n)$. $r_{layer\_nnz} = r_{nnz} = 0$ in a dense tensor or a sparse tensor with an even nonzero distribution. In a sparse tensor with an imbalanced nonzero distribution, r_{layer_nnz} is able to predict r_{nnz} by considering several subtensors as a group. Therefore, compared to r_{nnz}, r_{layer_nnz} can capture the imbalance of nonzero distribution by a low-cost estimation. Finally, we select the grid candidates with the best nonzero balance as the optimal grid $\mathcal{G}_{opt}$. Figure 9 shows that Algorithm 2 predicts the optimal grid as $16 \times 1 \times 1$ as with smallest r_{layer_nnz} for tensor *amazon* on 16 MPI processes. And Figure 7 indicates that our $\mathcal{G}_{opt}$ has a better performance than $\mathcal{G}_{splatt}$ built from SPLATT's grid configuration. The selected grid configuration is used for the following distribution policy and CPD computation.

4.2 Matrix-oriented Distribution Policy

Once we decide on a process grid, the next challenge is to choose a distribution policy that leads to an optimal partitioning of the tensor and matrices and balanced computation and communication and their tradeoffs among the processes. Thus, three parameters M_p, I_p, and I_l in Table 2 are influenced by a distribution policy $\mathcal{D}$. The optimal strategies effectively eliminate performance bottlenecks, resulting in balanced computation and communication and their tradeoffs.

The state-of-the-art work [35] takes a strategy that balances nonzero computation by evenly partitioning tensor nonzeros among the processes, shown in Figure 10(a). It only considers M_p and targets to minimize r_{nnz}. Thus, it is advantageous for CPD dominated by the sparse tensor computation kernel MTTKRP. In general, such tensors have moderate sparsity and uniform nonzero distributions along the dimensions. Nevertheless, this strategy may not be beneficial for irregular tensors. For example, tensor *fb-m* has one dimension size multiple orders-of-magnitude smaller than the others, and its nonzeros mainly reside along a diagonal with increasing density, while most nonzeros concentrate at a bottom corner. Applying the nonzero balancing strategy to such tensors results in severe imbalances in all aspects, including nonzero computation, matrix computations, and communication (see Figure 13). Furthermore, CPD on some tensors under study do not benefit from balanced nonzero computation, as the execution is dominated by communication or matrix computations in Figures 5 and 4. We leverage the sparsity and irregularity information that reflects in matrix computation imbalance. We identify the dominant imbalance ratio as matrix computation imbalance for irregular tensors. Therefore, all our strategies are based on balancing matrix computations and then achieve the best tradeoffs between different imbalances.

To balance matrix computations, we first propose an easy-to-use *set* strategy that balances I_p by evenly partitioning matrices among the processes in every dimension, shown as Figure 10(b). This results in minimal r_{I_p} and balanced matrix computation but could exacerbate the imbalance for nonzero computation. *Set* strategy is advantageous for CPD dominated by matrix computations, typically very sparse tensors with a uniformed distribution of nonzeros, and could tolerate irregular tensor dimension sizes. However, applying the matrix balancing strategy improves the balance for matrix computations and communication but exacerbates the imbalance for nonzero computation. Yet, neither of these two strategies works well for irregular sparse tensors like *fb-m*, because they target to minimize only one imbalance ratio, either r_{nnz} or r_{I_p}, without considering the tradeoffs among the three ratios counting r_{vol} for communication.

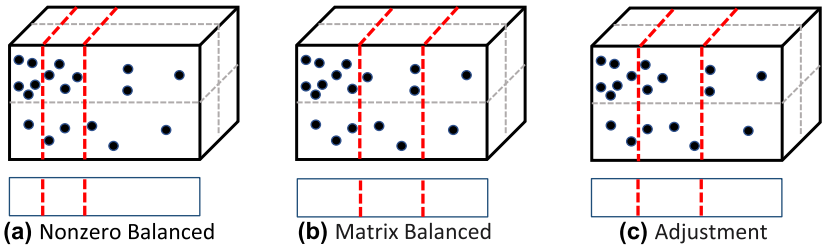


Fig. 10. Distribution policy on 12(=2 × 3 × 2) processes. Layer boundaries in red are adjusted in the 2nd dimension; boundaries in gray are fixed and in the 1st and 3rd dimensions.

The challenge for irregular sparse tensors is extremely high imbalance in both computation and communication, as our Finding 3 in Section 3 shows. Focusing only on optimizing one of imbalance ratios might lead to higher imbalance ratios for the other two. To support irregular sparse tensors, we propose new distribution policies to achieve better tradeoffs between these imbalance ratios. *Our proposed distribution policies begin with the matrix-balancing strategy but adjust according to the nonzero-balancing strategy*, illustrated in Figure 10(c) where red lines are shifted based on (b) but not as skewed as (a). Algorithm 3 shows the three strategies of distribution policies: *set*, *ordered-c*, and *max-min*. From Figure 10, a distribution policy is a layer configuration and represented by $\{I_L\}$, an array of dimension sizes distributed to each process that sum up to the dimension size in dimension-n. Assume the processor grid is $P = P_1 \times P_2 \times \cdots \times P_N$. We first employ *set* strategy by partitioning I_n/P_n consecutive slices of $\mathcal{X}$ to each process in dimension n, yielding balanced matrix computations but potentially skewed nonzeros among processes. We then adjust layer boundaries to mitigate nonzero imbalance using either *ordered-c* or *max-min* strategies.

The key idea of the *ordered-c* strategy is to reduce nonzero imbalance of each partition independently. It adjusts layer boundaries along with the index in each tensor dimension. To achieve this, we first calculate M/P_n as the target nonzero size of each partition in dimension-n and add/remove slices if the nonzeros in a partition are greater/less than the target size. Second, we need to move layer boundaries to make nonzeros in each partition closer to the target nonzero size M/P_n. Assume the current number of nonzeros in the ith partition is m_i and the average number of nonzeros for one slice in this partition is S_{L_i}, then the number of slices to be adjusted is given by $(m_i - M/P_n)/(c \cdot S_{L_i})$, where c is a user-given integer. The larger the c value, the finer the adjustment granularity. Partitioning with $c = 1$ is the same as Splatt for dense tensors or sparse tensors with uniformed nonzero distribution. With larger c, our partitioning keeps more balanced I_p rather than nonzeros for irregular sparse tensor. When c is extremely large, the *ordered-c* strategy has little difference with the *set* strategy, as it has little adjustment. Therefore, we set c as 1 or 2 to achieve better tradeoffs between nonzero and I_p imbalance and distinguish with the *set* strategy.

Instead of adjusting each partition independently, the key idea of the *Max-min* method is to balance nonzeros in partitions based on the differences between them. There is no target nonzero size in this strategy. It moves slices from partitions with the maximal nonzeros to the ones with the minimal nonzeros. We first sort the layer configuration I_L in a descending order and save it as I'_L. By looping the first half of I'_L, the max-min pair is $I'_L[i]$, $I'_L[P_n - i]$, respectively. Second, we adjust layer boundaries of each max-min pair. Let S_n be the average number of nonzeros per slice for all partitions of dimension-n. The number of slices to be adjusted is $(I'_L[i] - I'_L[P_n - i])/S_n$. *Max-min* adjusts only the maximal and minimal nonzero partitions, but might be less accurate in partitioning nonzeros by considering the global slice information with S_n among partitions rather than the local S_{L_i} within a partition. As each partition must contain continuous slices, this method

might involve adjusting boundaries of all partitions. Therefore, we expect lower performance than the first method, but it still outperforms the nonzero-balancing strategy for irregular tensors.

Our proposed prediction-based grid configuration and matrix-oriented distribution policy are directly applied to medium-grained, bulk-synchronous distributed CPD (Algorithm 1) as Lines 2 and 3 separately to gain performance improvement and better scalability.

5 EXPERIMENTAL RESULTS

Platform. We perform experiments on the Constance cluster at the Pacific Northwest National Laboratory; each node has 2 × 12-core Intel Xeon CPU E5-2670 v3 CPUs. The Constance system has 520 2 × 12-core nodes (totaling 12,480 cores), 64 GB DDR4 memory per node on a 56 Gb/s FDR Infiniband interconnect. We use up to a total number of 1,536 cores, with 128 nodes and 12 cores/node, gcc 7.3.0 and OpenMPI 4.0.1 as compilers. Our experiments consume 25% of the whole system. The default BLAS and LAPACK libraries v3.2.1 on Linux are used for the dense matrix routines.

Dataset. We evaluate sparse tensors from real-world applications and a randomly permuted tensor in Table 3, ordered by increasing number of nonzeros. Most of these tensors are from the **Formidable Repository of Open Sparse Tensors and Tools (FROSTT)** [34]. The *darpa* (source IP-destination IP-time triples), *fb-m*, and *fb-s* (entity-entity-relation triples) are from HaTen2 [17], and *choa* (patient-visit-time triples) is built from **electronic health records (EHRs)** [31]. The *random* is a randomly permuted tensor.

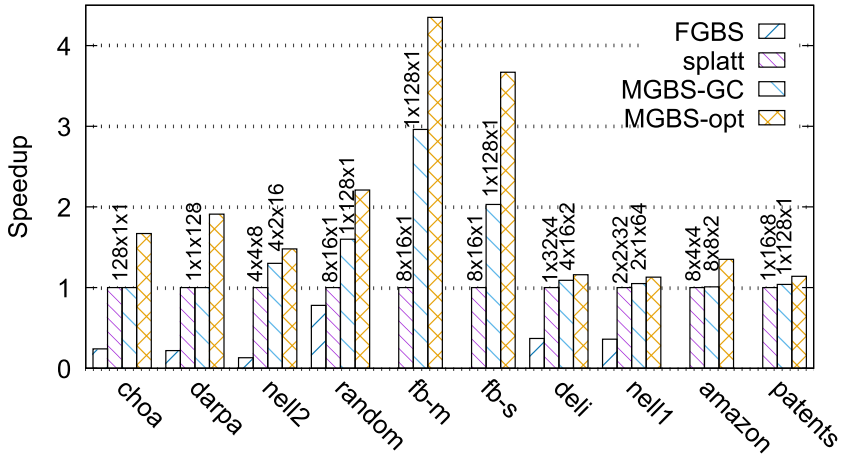
Baseline. We use SPLATT as our baseline, representing a medium-grained, bulk-synchronous distributed CPD [35],[5] which is generally considered faster than MapReduce implementations [17, 18]. We also compare to the fine-grained distributed CPD algorithm (represented as FGBS) from HyperTensor [20].[6] Both medium- and fine-grained CPD are hybrid MPI+OpenMP parallelized. We use 12 threads (referred to as processors uniformly) for each CPU for all experiments and set $R = 32$, as using a different R has no impact on our evaluation. All experiments use single-precision floating point values, and the average execution time of five iterations is reported. Due to the CPD execution time variance on different tensors, we normalize the time of other implementations to medium-grained SPLATT.

5.1 Overall Performance

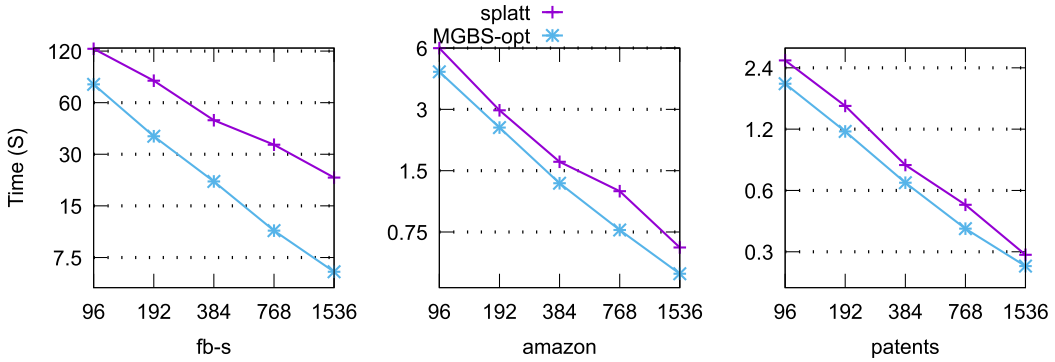
Figure 11(a) shows the speedup of our distributed CPD (MGBS-opt) compared to medium-grained (SPLATT) and fine-grained (FGBS) CPD when using 1,536 processors. The speedup over SPLATT ranges from 1.2× to 4.4× for all nine tensors. The two irregular tensors, *fb-m* and *fb-s*, benefit the most from our methods, because they suffer severe r_{nnz}, r_{vol}, and r_{Ip} imbalance in prior implementations (see Figure 13). Relatively small sparse tensors like *choa*, *darpa*, and *nell2* have a speedup from 1.5× to 1.7×. Other tensors such as *deli* and *amazon* gain a speedup from 1.2× to 1.4× from our methods, even though they have decent balances with SPLATT. The randomly permuted tensor *random* has a relatively regular tensor nonzero distribution, because it is difficult to generate an irregular tensor like those tensors from real-world applications. The tensor *random* has a speedup as 2.2×.

Compared to fine-grained distributed CPD (FGBS) with hypergraph partitioning generated by Zoltan [8], MGBS-opt always performs better by 3.1–11.4×. The missing bars on large and/or irregular tensors, *amazon*, *patents*, *fb-m*, and *fb-s* are due to failures of generating hypergraph partitions by Zoltan on 1,536 processors. We observe that SPLATT achieves higher performance than FGBS on all cases, aligned with Reference [35].

[5]ENSIGN [23] is a closed-sourced, commercial library, and CarHP [2] is not open-sourced.
[6]Implemented in SPLATT as its open-source version.



(a) Overall performance speedup for CPD on 1536 processors.



(b) Strong scalability from 96 to 1536 processors.

Fig. 11. Overall performance comparison and scalability.

Figure 11(a) also presents the performance effect of our prediction-based grid configuration (Algorithm 2) as MGBS-GC. By comparing SPLATT, MGBS-GC, and MGBS-opt, we see the incremental performance from our optimizations. The prediction-based grid configuration and matrix-oriented distribution policy increase the performance by 0%–296% and 7%–91% separately. The labels on top of SPLATT and MGBS-GC bars show their chosen process grids. MGBS-GC and SPLATT obtain the same grid and thus lead to the same performance on *choa* and *darpa*. Our prediction-based grid configuration accelerates performance for 8 out of 10 tensors. It is expensive to find the optimal partition in a large system size. For 128 nodes, it has 36 different grids with 5 partitioning strategies, totally 120 different partitions. Our idea is to find the best overall balanced partition with negligible time overhead. Tensor *fb-m* gets the highest gain at 2.96× with a better grid configuration. These results verify that irregularity-aware grid configuration is critical to CPD performance.

Figure 11(b) demonstrates that MGBS-opt obtains better strong scalability than SPLATT on three large tensors from 96 to 1,536 processors. MGBS-opt shows significantly better scalability than SPLATT on irregular yet sparse tensor *fb-s*. This is because r_{Ip} that impacts matrix computation and communication time reduces significantly in MGBS-opt. Detailed profiling shows that both communication and computation time are closed to be halved as the number of processors

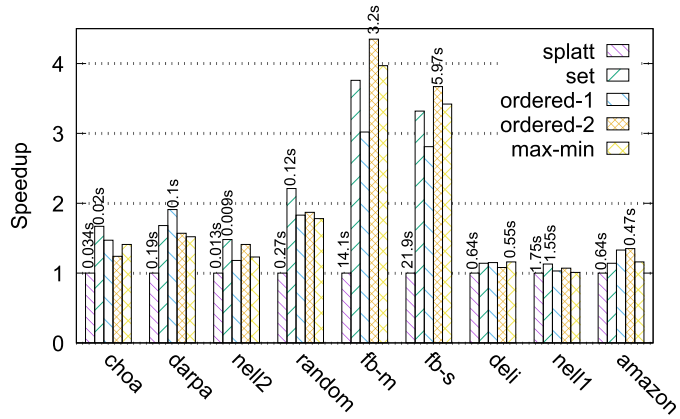


Fig. 12. The effect of different distribution policies: matrix-balancing (*set*), two ordered adjustments (*ordered-1* and *ordered-2*), and max-to-min adjustment (*max-min*). The CPD running time in seconds for SPLATT and our best strategy are shown.

doubles in MGBS-opt. MGBS-opt scales slightly better for matrix computation and communication on tensors *amazon* and *patents*, where MTTKRP occupies a larger time percentage. For other tensors: *fb-m* shows similar scalability to *fb-s*; *deli* and *nell1* are similar to *patents*; both SPLATT and MGBS-opt show good scalability on small tensors *choa*, *darpa*, and *nell2*.

5.2 Balanced Distribution Policy Analysis

Figure 12 shows the speedup of CPD from our four matrix-oriented distribution policies against SPLATT on 1,536 processors. *Set*, *ordered-1*, *ordered-2*, and *max-min* represent the strategies of matrix-balancing, two types of ordered adjustment, and max-min adjustment separately. *Ordered-1* and *ordered-2* incline the adjustment to nonzero and I_p balance, respectively. Overall, our strategies obtain speedup on all tensors. The *set* strategy performs the best on four, *ordered-1* on one, *ordered-2* on three, and *max-min* on one tensor, respectively. All the four strategies achieve significant speedups on the two most-irregular tensors *fb-m* and *fb-s*, with *ordered-2* the most advantageous. An interesting observation is that simple strategies (*set* and *max-min*) could perform the best. These results verify our findings that balancing only nonzeros results in suboptimal performance, and tradeoffs are required among nonzero, matrix computation, and communication volume.

To further understand why some tensors benefit more from our strategies than others, we look into how their imbalance ratios change. We explore two representative tensors in Figure 13 to show our optimization for load imbalance on irregular and regular tensors. The overall performance is comprehensively impacted by r_{nnz}, r_{vol}, and r_{I_P}. Two general observations are obtained: First, no strategy simultaneously obtains the lowest imbalance ratios from all the three aspects: nonzero, matrix computation, and communication. Second, all strategies trade higher r_{nnz} for lower r_{vol} and r_{I_P} to gain performance improvement. The irregular tensor *fb-m* suffers very high imbalance ratios for all strategies in all three aspects. SPLATT has the smallest r_{nnz} balance, *set* has nearly perfect r_{I_p} balance (around 0, invisible in bars), while *ordered-1* gets the best r_{vol} balance. However, *ordered-2* obtains the best performance in Figure 12, since none of SPLATT, *set*, and *ordered-1* obtains a good tradeoff among the three ratios. For $fb-m$, r_{I_p} plays a more important role in overall performance, so a small reduction in r_{I_p} has crucial impact on performance speedup. Different from irregular tensors, regular tensors like *nell1* have much lower imbalance ratios in each category. For *nell*1 r_{nnz} plays a more important role, so our optimization for r_{vol} and r_{I_p} does not translate

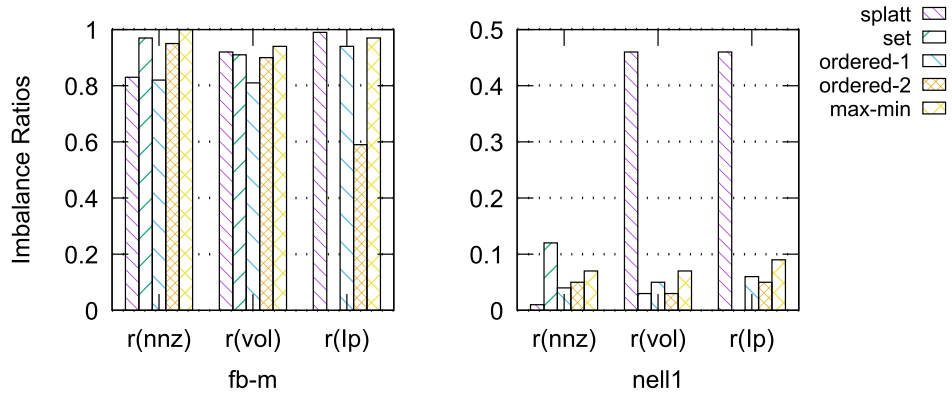


Fig. 13. Load imbalance ratios (r_{nnz}, r_{vol}, and r_{Ip}).

to significant performance improvement. Its r_{nnz} imbalance ratio is actually under control at 1% with Splatt. *Set*, which gets the highest performance gain, has the worst r_{nnz} imbalance but the best r_{I_p} and r_{vol} balance. Regular tensors tend to be easier to get balanced in all categories and the differences among them are small. These results demonstrate that the tradeoff among different load balances is complex and the optimal solution is determined by tensor properties, i.e., sparsity, shape, and distribution of nonzeros among the modes. We identify the dominant imbalance ratio as r_{I_p} for irregular tensors because of its impact on matrix computation. However, the best performance of Cpd is usually not achieved by the optimal r_{I_p}, because other imbalance ratios are also important. It is still very difficult or impossible to obtain the optimal balance simultaneously among all categories, thus a careful tradeoff is required for the best performance.

Guideline for choosing strategies. We provide general guidelines for users to easily pick from the strategies for their own tensors. Our strategies try to find the best tradeoff among three imbalance ratios, though it is difficult to match each strategy for one certain type of tensors. If r_{I_p} is the dominant imbalance factor in Cpd and we need to control it as small as possible, the ascending order of r_{I_p} in our strategies is *set* < *ordered-2* < *ordered-1*. Generally, users could safely choose *set* if lacking statistical information on a sparse tensor, because it always performs better than Splatt on a large cluster, as Figure 12 shows. Our recommendations are as follows: (1) Use *set* for relatively small or regular tensors, as it obtains the smallest r_{I_p} while the other two imbalance ratios have little increase in those tensors such as *choa* and *nell2*; (2) Use *ordered-2* for relatively large and irregular tensors, as it optimizes both r_{vol} and r_{I_P} well on tensors such as *fb-m* and *fb-s*.

5.3 Bottleneck Shifting

We show how Mgbs-opt influences the performance bottleneck of major computation and communication kernels of Cpd for tensors *choa* and *fb-m* in Figure 14. For *choa*, Mgbs-opt shifts the performance bottleneck from communication in Splatt to *MAT-SOLVE* as a result of communication time reduction, while also decreasing the time of *MAT NORM*. For *fb-m*, the Mgbs-opt performance is still dominated by *COMM* as in Splatt, but largely reduced. Since Splatt focuses on optimizing the nonzero imbalance for Mttkrp, which only accounts for a negligible portion (invisible in Figure 14), Mgbs-opt correctly identifies bottlenecks and significantly improves their execution.

5.4 Partitioning Strategies Comparison

Several previous works have compared Mgbs with coarse-grained Cpd [13]. It has been proved that Splatt is 41× to 76× faster than DFacTo on 1,024 cores [35]. Therefore, we no longer compare

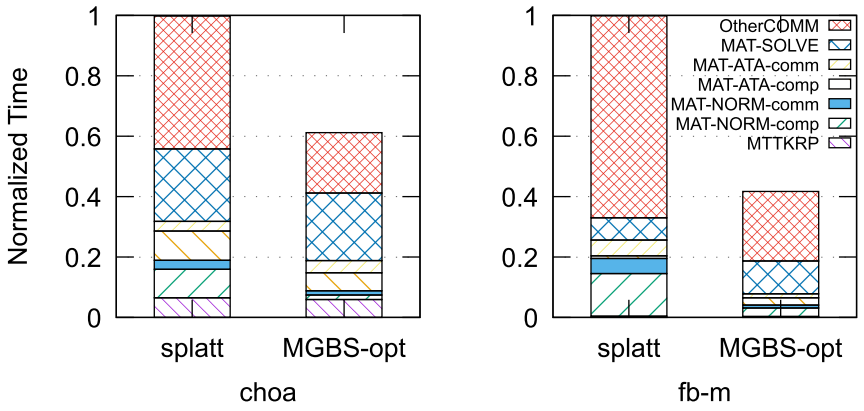


Fig. 14. Time percentage of main kernels.

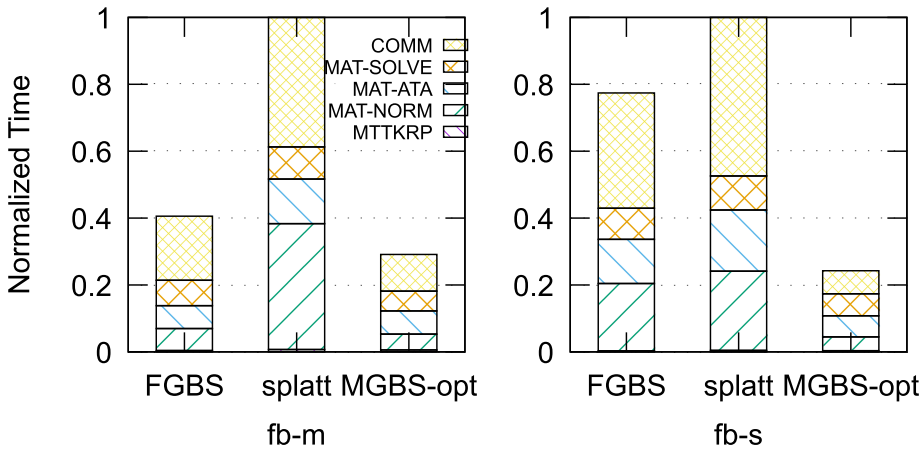


Fig. 15. Time percentage of main kernels for *fb-m* and *fb-s* on 768 processors.

Mgbs-opt with coarse-grained Cpd in this work. We examine the fine-grained distribution with hypergraph partitioning of each tensor generated by Zoltan [8]. Large tensors such as *amazon* and *patents* are unable to compute a hypergraph partitioning due to their memory requirements. Figure 11(a) already shows FGBS achieves lower performance than both Splatt and Mgbs-opt for 5 tensors on 1,536 processors. The hypergraph partitions of *fb-m* and *fb-s* can be generated on 768 processors. Splatt achieves higher performance than fine-grained distribution in 5 out of 7 tensors on 768 processors except for *fb-m* and *fb-s*. Figure 15 displays the normalized time of major computation and communication kernels in FGBS, Splatt, and Mgbs-opt on 768 processors. We first disclose that FGBS performs faster than Splatt on tensors *fb-m* and *fb-s* by 3.2× and 1.3×, but only achieves 70% and 30% of the performance of Mgbs-opt, which further strengthens our motivation of study on irregular tensors. Compared to Splatt, both FGBS and Mgbs-opt significantly improve the performance of matrix-related computations on *fb-m* and achieve similar speedups; while on *fb-s*, FGBS only gains a small improvement over Splatt. This demonstrates the performance improvement of Mgbs-opt is more stable than FGBS on different irregular tensors.

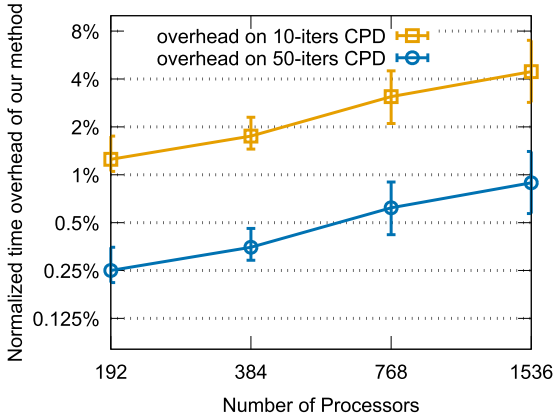


Fig. 16. Time overhead of our method (Algorithms 2 and 3). The time of our method is normalized to CPD time.

5.5 Time Overhead of Irregularity-aware Method

We evaluate time overhead of our irregularity-aware method and compare it with CPD time. Our proposed prediction-based grid configuration incurs trivial time cost in the virtual distribution, as it needs to compute the nonzero imbalance ratio r_{layer_nnz} for each candidate. The cost of our matrix-oriented distribution policy is negligible, because its complexity is $O(P)$. The time cost of irregularity-aware method is mainly determined by the total dimension sizes of the tensor c_N in the complexity of $O(c_N \times I_n)$ in computation of r_{layer_nnz}. Our method does not incur expensive data redistribution, because we only do data distribution once as SPLATT. The number of CPD iterations is determined comprehensively by the size, nonzero distribution, and sparsity of a tensor. We set 10 and 50 as the minimum and maximum iterations, because we observe tensors in our dataset converge for CPD in this range of iterations. Figure 16 displays the average, maximum and minimum time overhead of irregularity-aware method normalized to 10 and 50 CPD iterations for all tensors in our dataset. As the system size increases, the normalized time overhead increases for both cases. This is because our proposed irregularity-aware method is sequential with relatively stable time on different system sizes. The average overhead is 4.5% to 10 CPD iterations and 0.9% to 50 iterations on 1,536 processors. Overall, the time cost of irregularity-aware method is low and acceptable compared to CPD time. And its time overhead is negligible compared to hypergraph partitioning in both fine-grained [20, 21] and medium-grained CPD [2].

5.6 Application to Other Formats

We extend MGBS-opt to support other sparse tensor formats such as the **coordinate (COO)** and **Hierarchical Coordinate (HICOO)** [25] by extending the ParTI library [24]. COO, the simplest yet arguably most popular format by far, stores each nonzero value along with all of its position indices. HICOO [25] format improves upon COO by compressing the indices in units of sparse tensor blocks. Figure 17 plots strong scalability of MGBS-opt applied to COO and HiCOO formats for three tensors on 48 to 1,536 processors. MGBS-opt obtains near-linear scalability for HiCOO on these tensors. With COO format *darpa* on 96 to 192 and *deli* on 48 to 96, processors show super-linear speedup. Detailed profiling shows that computation time for matrix-related kernels reduces by more than half in both cases because of much better matrix-balance. MGBS-opt is flexible to support other variant formats in CSF or COO families [26, 27].

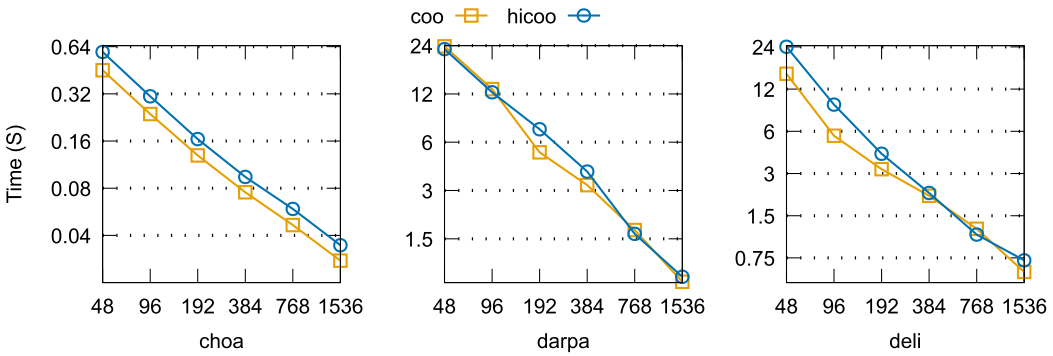


Fig. 17. Scalability of Mgbs-opt applied on ParTI for COO and HiCOO formats.

6 CONCLUSION

Although distributed CANDECOMP/PARAFAC decomposition is well-studied due to the increasing needs of processing large-scale data, the performance implication of tensor irregularity is not well understood. This work presents an irregularity-aware tensor decomposition on a distributed memory system. We thoroughly investigate the performance behavior of an abstract of the state-of-the-art distributed Cpd implementations through theoretical analysis and experimental profiling. From the study, we propose three imbalance ratio metrics and conclude four findings to guide our optimizations: prediction-based grid configuration and matrix-oriented distribution policy. Our optimization-enhanced distributed Cpd achieves up to 4.4× and 11.4× on 1,536 processors against the state-of-the-art medium- and fine-grained distributed implementations. Our optimizations support different sparse tensor formats such as CSF, COO, and HiCOO and gain good scalability for all of them. For future work, we intend to apply our optimizations to other tensor decompositions and adopt shared-memory optimizations like dimension-tree [21] to further improve performance.

REFERENCES

[1] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015), arXiv preprint arXiv:1603.04467.

[2] Seher Acer, Tugba Torun, and Cevdet Aykanat. 2018. Improving medium-grain partitioning for scalable sparse tensor decomposition. *IEEE Trans. Parallel Distrib. Syst.* 29, 12 (2018), 2814–2825.

[3] Phillip Alpatov, Greg Baker, H. Carter Edwards, John Gunnels, Greg Morrow, James Overfelt, and Robert van de Geijn. 1997. PLAPACK Parallel linear algebra package design overview. In *Proceedings of the ACM/IEEE Conference on Supercomputing*. IEEE, 29–29.

[4] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M. Kakade, and Matus Telgarsky. 2014. Tensor decompositions for learning latent variable models. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 2773–2832.

[5] Grey Ballard and Kathryn Rouse. 2020. General memory-independent lower bound for MTTKRP. In *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, 1–11.

[6] Muthu Baskaran, Thomas Henretty, and James Ezick. 2019. Fast and scalable distributed tensor decompositions. In *Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7.

[7] Zachary Blanco, Bangtian Liu, and Maryam Mehri Dehnavi. 2018. CSTF: Large-scale sparse tensor factorizations on distributed platforms. In *Proceedings of the 47th International Conference on Parallel Processing (ICPP'18)*. ACM, New York, NY. DOI: https://doi.org/10.1145/3225058.3225133

[8] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine. 2012. The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring. *Scient. Program.* 20, 2 (2012), 129–150.

[9] Venkatesan T. Chakaravarthy, Jee W. Choi, Douglas J. Joseph, Prakash Murali, Shivmaran S. Pandian, Yogish Sabharwal, and Dheeraj Sreedhar. 2018. On optimizing distributed Tucker decomposition for sparse tensors. In *Proceedings of the 32nd ACM International Conference on Supercomputing (ICS'18)*. 374–384.

[10] M. Ozan Karsavuran, M. Ozan, Seher Acer, and Cevdet Aykanat. 2020. Partitioning models for general medium-grain parallel sparse tensor decomposition. *IEEE Transactions on Parallel and Distributed Systems* 32, 1 (2020), 147–159.

[11] Jieyang Chen, Nan Xiong, Xin Liang, Dingwen Tao, Sihuan Li, Kaiming Ouyang, Kai Zhao, Nathan DeBardeleben, Qiang Guan, and Zizhong Chen. 2019. TSM2: Optimizing tall-and-skinny matrix-matrix multiplication on GPUs. In *Proceedings of the ACM International Conference on Supercomputing*. 106–116.

[12] Jaeyoung Choi, James Demmel, Inderjiit Dhillon, Jack Dongarra, Susan Ostrouchov, Antoine Petitet, Ken Stanley, David Walker, and R. Clinton Whaley. 1996. ScaLAPACK: A portable linear algebra library for distributed memory computers–Design issues and performance. *Comput. Phys. Commun.* 97, 1-2 (1996), 1–15.

[13] Joon Hee Choi and S. Vishwanathan. 2014. DFacTo: Distributed factorization of tensors. In *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 1296–1304.

[14] Andrzej Cichocki. 2014. Era of big data processing: A new approach via tensor networks and tensor decompositions. *CoRR* abs/1403.2048 (2014).

[15] Ahmed E. Helal, Jan Laukemann, Fabio Checconi, Jesmin Jahan Tithi, Teresa Ranadive, Fabrizio Petrini, and Jeewhan Choi. 2021. ALTO: Adaptive linearized storage of sparse tensors. In *Proceedings of the ACM International Conference on Supercomputing*. 404–416.

[16] Joyce C. Ho, Joydeep Ghosh, and Jimeng Sun. 2014. Marble: High-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14)*. ACM, New York, NY, 115–124. DOI: https://doi.org/10.1145/2623330.2623658.

[17] Inah Jeon, Evangelos E. Papalexakis, U. Kang, and Christos Faloutsos. 2015. HaTen2: Billion-scale tensor decompositions. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*.

[18] U. Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. 2012. 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 (KDD'12)*. ACM, New York, NY, 316–324. DOI: https://doi.org/10.1145/2339530.2339583.

[19] Oguz Kaya, Ramakrishnan Kannan, and Grey Ballard. 2018. Partitioning and communication strategies for sparse non-negative matrix factorization. In *Proceedings of the 47th International Conference on Parallel Processing*. 1–10.

[20] Oguz Kaya and Bora Uçar. 2015. Scalable sparse tensor decompositions in distributed memory systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'15)*. ACM, New York, NY. DOI: https://doi.org/10.1145/2807591.2807624

[21] O. Kaya and B. Uçar. 2018. Parallel Candecomp/Parafac decomposition of sparse tensors using dimension trees. *SIAM J. Scient. Comput.* 40, 1 (2018), C99–C130. DOI: https://doi.org/10.1137/16M1102744

[22] T. Kolda and B. Bader. 2009. Tensor decompositions and applications. *SIAM Rev.* 51, 3 (2009), 455–500. DOI: https://doi.org/10.1137/07070111X

[23] Reservoir Labs. 2016. ENSIGN: Multi-Domain Analytics. (2016). Retrieved from https://reservoir-ensign.github.io/usecases/ENSIGN.html.

[24] Jiajia Li, Yuchen Ma, and Richard Vuduc. 2018. ParTI!: A Parallel Tensor Infrastructure for multicore CPUs and GPUs (Version 1.0.0). (Oct. Retrieved from: https://github.com/hpcgarage/ParTI.

[25] Jiajia Li, Jimeng Sun, and Richard Vuduc. 2018. HiCOO: Hierarchical storage of sparse tensors. In *Proceedings of the ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*.

[26] B. Liu, C. Wen, A. D. Sarwate, and M. M. Dehnavi. 2017. A unified optimization approach for sparse tensor operations on GPUs. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*. 47–57. DOI: https://doi.org/10.1109/CLUSTER.2017.75

[27] Israt Nisa, Jiajia Li, Aravind Sukumaran-Rajam, Prasant Singh Rawat, Sriram Krishnamoorthy, and Ponnuswamy Sadayappan. 2019. An efficient mixed-mode representation of sparse tensors. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–25.

[28] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. 2015. Tensorizing neural networks. *CoRR* abs/1509.06569 (2015).

[29] Evangelos E. Papalexakis, Christos Faloutsos, and Nicholas D. Sidiropoulos. 2012. ParCube: Sparse parallelizable tensor decompositions. In *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I (ECML PKDD'12)*. Springer-Verlag, Berlin, 521–536. DOI: https://doi.org/10.1007/978-3-642-33460-3_39

[30] Ioakeim Perros, Robert Chen, Richard Vuduc, and Jimeng Sun. 2015. Sparse hierarchical Tucker factorization and its application to healthcare. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'15)*. IEEE Computer Society, Washington, DC, 943–948. DOI: https://doi.org/10.1109/ICDM.2015.29

[31] Ioakeim Perros, Evangelos E. Papalexakis, Fei Wang, Richard Vuduc, Elizabeth Searles, Michael Thompson, and Jimeng Sun. 2017. SPARTan: Scalable PARAFAC2 for large & sparse data. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'17)*. ACM, New York, NY, 375–384. DOI: https://doi.org/10.1145/3097983.3098014

[32] Thomas B. Rolinger, Tyler A. Simon, and Christopher D. Krieger. 2019. Performance considerations for scalable parallel tensor decomposition. *J. Parallel and Distrib. Comput.* 129 (2019), 83–98.
[33] Martin D. Schatz, Robert A. van de Geijn, and Jack Poulson. 2016. Parallel matrix multiplication: A systematic journey. *SIAM J. Scient. Comput.* 38, 6 (2016), C748–C781.
[34] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis. 2017. FROSTT: The Formidable Repository of Open Sparse Tensors and Tools. Retrieved from: http://frostt.io/.
[35] Shaden Smith and George Karypis. 2016. A medium-grained algorithm for distributed sparse tensor factorization. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE.
[36] Shaden Smith, Niranjay Ravindran, Nicholas Sidiropoulos, and George Karypis. 2016. SPLATT: The Surprisingly ParalleL spArse Tensor Toolkit (Version 1.1.1). Retrieved from: https://github.com/ShadenSmith/splatt.
[37] Edgar Solomonik and James Demmel. 2011. Communication-optimal parallel 2.5 D matrix multiplication and LU factorization algorithms. In *Proceedings of the European Conference on Parallel Processing*. Springer, 90–109.
[38] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *Int. J. High Perform. Comput. Applic.* 19, 1 (2005), 49–66.
[39] L. R. Tucker. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31 (1966), 279–311.
[40] Robert A. van De Geijn and Jerrell Watts. 1997. SUMMA: Scalable universal matrix multiplication algorithm. *Concurr.: Pract. Exper.* 9, 4 (1997), 255–274.

Received 14 September 2021; accepted 11 January 2023