SAND2021-0767J

True Load Balancing for Matricized Tensor Times Khatri-Rao Product

Nabil Abubaker, Seher Acer, and Cevdet Aykanat

Abstract—MTTKRP is the bottleneck operation in algorithms used to compute the CP tensor decomposition. For sparse tensors, utilizing the compressed sparse fibers (CSF) storage format and the CSF-oriented MTTKRP algorithms is important for both memory and computational efficiency on distributed-memory architectures. Existing intelligent tensor partitioning models assume the computational cost of MTTKRP to be proportional to the total number of nonzeros in the tensor. However, this is not the case for the CSF-oriented MTTKRP on distributed-memory architectures. We outline two deficiencies of nonzero-based intelligent partitioning models when CSF-oriented MTTKRP operations are performed locally: failure to encode processors' computational loads and increase in total computation due to fiber fragmentation. We focus on existing fine-grain hypergraph model and propose a novel vertex weighting scheme that enables this model encode correct computational loads of processors. We also propose to augment the fine-grain model by fiber nets for reducing the increase in total computational load via minimizing fiber fragmentation. In this way, the proposed model encodes minimizing the load of the bottleneck processor. Parallel experiments with real-world sparse tensors on up to 1024 processors prove the validity of the outlined deficiencies and demonstrate the merit of our proposed improvements in terms of parallel runtimes.

Index Terms—Load balancing, Sparse tensors, MTTKRP, CP Decomposition, Fine-grain hypergraph partitioning.

1 Introduction

Canonical polyadic decomposition (CPD) is an extension of singular value decomposition to tensors and a fundamental tool for the analysis of multiway data. It approximates a given tensor by the sum of multiple rank-one tensors so that each rank-one tensor corresponds to a structural feature in the data set. CPD is used for dimensionality reduction, data completion and compression, and finds application in various domains such as neuroscience [1, 2], machine learning [3, 4], chemistry [5], cybersecurity [6], signal processing [7], and network analysis [8].

The most popular algorithm that computes the CPD is based on the alternating least squares method and generally referred to as CP-ALS. Each iteration of the CP-ALS algorithm computes a new factor matrix for each mode by performing several computational steps. Among those steps, matricized tensor times Khatri-Rao product (MTTKRP) constitutes the biggest bottleneck because of its high computational cost.

When CP-ALS is performed on a sparse tensor and on a distributed-memory setting, the optimization of the MTTKRP operation becomes more tedious due to the irregular sparsity pattern of the tensor nonzeros. Practitioners usually perform tensor decomposition many times with different ranks, which makes the optimization of MTTKRP even more crucial for reducing the turnaround time of their analysis. For achieving a performant and scalable parallel decomposition, one should take the sparsity information into account in crucial design decisions associated with high communication and computational costs. These decisions involve

(i) how the input tensor is distributed among processors,
(ii) how the tensor nonzeros are stored in each processor,
(iii) and how MTTKRP is realized on the given storage.

To address (i), several successful partitioning models [9–12] have been proposed with the goal of reducing the communication cost of MTTKRP while maintaining a balance on its computational costs on all processors. To address (ii) and (iii), several storage formats [13–15] have been proposed, usually together with a new method to realize MTTKRP on the proposed format. Among those, compressed sparse fiber (CSF) proves to be the most commonly-used storage format due to its efficiency in terms of both memory and computation [11, 13, 14]. CSF is an extension of the compressed sparse row format to tensors and the total flop count in the CSF-oriented MTTKRP is proportional to the total number of nonzeros and fibers (along a specified mode) in the given tensor. The flop count of the CSF-oriented MTTKRP is significantly smaller than the flop count of the MTTKRP based on the coordinate-format [13, 14].

Besides the popularity of the CSF format, tensor partitioning models still assume the computational cost of MTTKRP to be proportional to the total number of nonzeros in the input sparse tensor. This creates a discrepancy when the tensor is stored in a CSF format and hence a CSF-oriented MTTKRP operation is performed. This discrepancy leads to a failure in balancing the computational loads of processors in the distributed-memory parallelization. This failure becomes more prominent as the variance on the nonzero counts of fibers becomes larger, that is, as the tensor becomes more irregular.

In this work, we propose a tensor partitioning model with true load balancing for MTTKRP operation when the CSF format is used. Our model is based on the fine-grain

• *N. Abubaker and C. Aykanat are with the Department of Computer Engineering, Bilkent University, Turkey.*
E-mail: nabil.abubaker@bilkent.edu.tr, aykanat@cs.bilkent.edu.tr
S. Acer is with Sandia National Labs, Albuquerque, NM, USA.
E-mail: sacer@sandia.gov

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

model [9], which is (theoretically and practically) the most successful model in reducing the total communication volume and balancing the number of tensor nonzeros in processors. Our contributions can be summarized as follows:

- We outline two deficiencies of the existing fine-grain model when the CSF scheme is used for local MTTKRP operations: failure to encode the correct computational loads of processors and the increase in the total amount of computation due to fiber fragmentation.
- We first propose a heuristic that leads to a novel vertex weighting scheme which helps the hypergraph model correctly encapsulate the computational loads of processors. We then utilize the well-known recursive bipartitioning framework for improving the accuracy of the heuristic.
- We also propose an augmentation to the fine-grain model by fiber nets that reduce the fiber fragmentation and help the weighting scheme achieve its potential.

The rest of the paper is organized as follows. In Section 2 the necessary backgrounds for CPD, CP-ALS, and hypergraph partitioning are given. The deficiencies of the HP-based fine-grain method are discussed in Section 3. In Section 4, our proposed framework is presented and discussed in details. Experimental results are given and discussed in Section 5. Related work is given in Section 6 and the paper is concluded in Section 7.

2 Background

2.1 Tensors and notations

We denote tensors by calligraphic letters ($\mathcal{X}$) and matrices by bold capital letters ($\mathbf{A}$). The number of dimensions of a tensor, denoted by N, is called the *mode* of the tensor. Note that matrices and vectors are 2-mode and 1-mode tensors, respectively. For the sake of simplicity, we assume 3-mode tensors of size $I \times J \times K$.

Fibers are analogous to matrix rows or columns, which can be obtained by fixing all but one indices of the tensor. In 3-mode tensors there are row, column and tube fibers which are respectively denoted by $\mathcal{X}(i,:,k)$, $\mathcal{X}(:,j,k)$ and $\mathcal{X}(i,j,:)$. *Slices* are analogous to matrices and can be obtained by fixing all but two indices. In 3-mode tensors, there are horizontal (e.g., $\mathcal{X}(i,:,:)$), lateral (e.g., $\mathcal{X}(:,j,:)$) and frontal (e.g., $\mathcal{X}(:,:,k)$) slices. *Matricization* of a tensor means unfolding it into a matrix shape along one of its modes. For instance, the matricization of $\mathcal{X}$ along the first mode, denoted by $\mathbf{X}_{(1)}$, is a matrix of size $I \times JK$. We refer the reader to the survey by Kolda and Bader [16] for more details on tensor decompositions.

2.2 The Canonical Polyadic Decomposition

The CPD, with R as the decomposition rank, approximates tensor $\mathcal{X}$ as the sum of R rank-one tensors: $\mathcal{X} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$. Here "$\circ$" denotes the outer product operation. The $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ components in each of the R rank-one tensors are assembled to respectively form factor matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$ and $\mathbf{C} \in \mathbb{R}^{K \times R}$.

The most commonly used algorithm to compute the CPD is CP-ALS, which uses the *Alternating Least Squares* approach. Algorithm 1 shows the steps of the CP-ALS algorithm. During each iteration, two factor matrices are fixed to find the remaining one by solving a linear alternating least squares problem. For instance, $min_{\mathbf{A}} ||\mathbf{X}_{(1)} - \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T||_R^2$ is solved to find $\mathbf{A}$ by computing $\mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})$. The columns of factor matrices are then normalized to length one, and the actual lengths are stored in λ. The operations $\odot$ and $*$ denote the Khatri-Rao and the Hadamard products, respectively.

The MTTKRP operation, which is the target operation in this work, takes place in lines 4, 7, and 10 of Algorithm 1, each of which is for computing a factor matrix along a different mode. Although it is shown as a multiply of an unfolded (matricized) tensor (e.g., $\mathbf{X}_{(1)}$) with a large matrix (e.g., $(\mathbf{B} \odot \mathbf{C})$), this is basically for simplicity and the corresponding multiply is impractical for sparse tensors. Many implementations prefer to realize the MTTKRP operation $\hat{\mathbf{A}} \leftarrow \mathbf{X}_{(1)}(\mathbf{B} \odot \mathbf{C})$ in a rowwise way for $\hat{\mathbf{A}}$, such as

$$\hat{\mathbf{A}}(i,:) = \sum_{\mathcal{X}(i,j,k) \neq 0} \mathcal{X}(i,j,k)[\mathbf{B}(j,:) * \mathbf{C}(k,:)]. \quad (1)$$

This computation style is preferred when the tensor is stored as a list of (i,j,k,val) coordinates, called the COO format. Note that in this formulation, the corresponding rows of $\mathbf{B}$ and $\mathbf{C}$ are retrieved and multiplied for each nonzero.

As a better alternative, the software toolkit SPLATT [13] uses the flops-reducing formulation

$$\hat{\mathbf{A}}(i,:) = \sum_{j \ni nnz(\mathcal{X}(i,j,:)) \neq 0} \mathbf{B}(j,:) * \sum_{k \ni \mathcal{X}(i,j,k) \neq 0} \mathcal{X}(i,j,k)\mathbf{C}(k,:) \quad (2)$$

which uses a fiber-centric data structure (to be discussed in the next subsection). Hereafter, $nnz(.)$ refers to the number of nonzeros in a (sub)tensor. In (2), the outer and inner summations respectively run over all nonzero fibers of slice $\mathcal{X}(i,:,:)$, and all nonzero entries of fiber $\mathcal{X}(i,j,:)$.

Algorithm 1 CP-ALS for 3-mode Tensors

```
1:  procedure CP-ALS(X)
2:      Initialize matrices A, B and C randomly
3:      while not converged do
4:          Â ← X_(1)(B ⊙ C)
5:          A ← Â(C^T C * B^T B)^-1
6:          Normalize columns of A into λ
7:          B̂ ← X_(2)(C ⊙ A)
8:          B ← B̂(C^T C * A^T A)^-1
9:          Normalize columns of B into λ
10:         Ĉ ← X_(3)(B ⊙ A)
11:         C ← Ĉ(B^T B * A^T A)^-1
12:         Normalize columns of C into λ
return [[λ; A, B, C]]
```

2.3 Efficient computation of MTTKRP

The efficient formulation in (2) can be realized using the Compressed Sparse Fibers (CSF) scheme, which was first introduced by Smith and Karypis [14]. The CSF storage scheme can be considered as a natural extension of the Compressed Sparse Rows/Columns schemes widely used for sparse matrices. Fig. 1 shows an illustration of the CSF storage format for a 3-mode sparse tensor. In the figure, the $pSlice$ and $pFiber$ arrays respectively represent the compressed slices and fibers. The $pSlice$ array consists of pointers to the starting indices of the compressed fibers

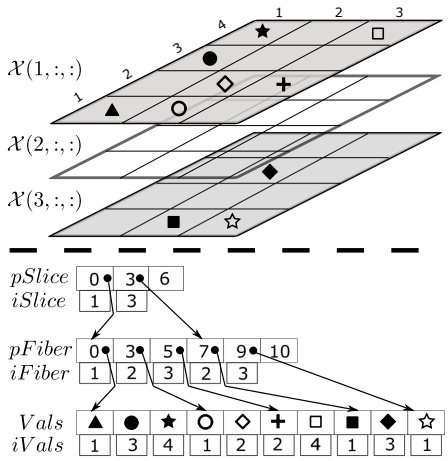


Fig. 1: A 3-mode tensor (top) and the corresponding CSF storage (bottom).

of the respective slices in the $pFiber$ array. Similarly, the $pFiber$ array consists of pointers to the starting indices of the nonzeros of the respective fibers in $Vals$. The $iSlice$, $iFiber$ and $iVals$ arrays respectively store the i, j and k indices of the respective nonzero slices, fibers and entries.

Two CSF-based computational schemes are used for computing the MTTKRP operations in Algorithm 1 using the formulation in (2). These two schemes will be referred to as CSF-S and CSF-D, where "-S" and "-D" refer to the use of Single and Double storage, as will be explained shortly.

The CSF-S scheme operates on a single CSF storage of the tensor, where the compressed fibers are the tensor's fibers along the longest mode. This scheme is proposed by Smith and Karypis [14] and currently used in SPLATT. CSF-S utilizes Algorithm 2 for computing the MTTKRP operations along all modes but the longest mode, while it utilizes Algorithm 3 to compute the MTTKRP operation along the longest mode.

The CSF-D scheme uses two different CSF storages of the tensor. The first storage s_1 is the same as of CSF-S, while the second storage s_2 utilizes the fibers along the second longest mode as the compressed $pFiber$ array. This scheme was used in several works that target computing the MTTKRP in distributed settings [11, 13, 17]. CSF-D utilizes Algorithm 2 for computing the MTTKRP operations along all modes but the longest mode by feeding s_1, and it utilizes the same algorithm to compute the MTTKRP operation along the longest mode by feeding s_2. Although CSF-D has a larger memory footprint compared to CSF-S, it has the advantage of avoiding the use of costly mutexes when used in hybrid (distributed + shared) settings [14].

2.4 Hypergraph Partitioning (HP)

A hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$. Each net $n\in\mathcal{N}$ connects a subset of vertices, which is denoted by $Pins(n)$. Each net n is assigned a cost $c(n)$, whereas each vertex v maybe assigned C weights denoted by $w^c(v)$ where $c \in \{1,2,..,C\}$. For $C>1$, the HP problem is commonly known as multi-constraint HP.

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, ..., \mathcal{V}_P\}$ denotes a P-way partition of $\mathcal{H}$ if the vertex parts are mutually exclusive and exhaustive. For

Algorithm 2 MTTKRP used by both CSF-D and CSF-S

Require: Tensor $\mathcal{X}$ stored in CSF, $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$
for $i \leftarrow 1$ to $size(pSlice)$ **do**
  $is \leftarrow iSlice[i]$
  for $j \leftarrow pSlice[i]$ to $pSlice[i+1]-1$ **do**
    $jf \leftarrow iFiber[j]$
    if $pFiber[j+1]-pFiber[j]=1$ **then**
      $k \leftarrow pFiber[j]$
      $\hat{\mathbf{A}}(is,:) += Vals[k] * \mathbf{C}(iVals[k]) * \mathbf{B}(if,:)$
    else
      $acc(:) \leftarrow 0$
      for $k \leftarrow pFiber[j]$ to $pFiber[j+1]-1$ **do**
        $acc(:) += acc(:) + Vals[k] * \mathbf{C}(iVals[k],:)$
      $\hat{\mathbf{A}}(is,:) += acc(:) * \mathbf{B}(jf,:)$

Algorithm 3 MTTKRP used by CSF-S

Require: Tensor $\mathcal{X}$ stored in CSF, $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$
for $i \leftarrow 1$ to $size(pSlice)$ **do**
  $is \leftarrow iSlice[i]$
  for $j \leftarrow pSlice[i]$ to $pSlice[i+1]-1$ **do**
    $jf \leftarrow iFiber[j]$
    if $pFiber[j+1]-pFiber[j]=1$ **then**
      $k \leftarrow pFiber[j]$
      $\hat{\mathbf{C}}(iVals[k],:) += Vals[k] * \mathbf{A}(is,:) * \mathbf{B}(jf,:)$
    else
      $acc(:) \leftarrow \mathbf{A}(is,:) * \mathbf{B}(jf,:)$
      for $k \leftarrow pFiber[j]$ to $pFiber[j+1]-1$ **do**
        $\hat{\mathbf{C}}(iVals[k],:) += acc(:) * Vals[k]$

a given partition Π,

$$W^c(\mathcal{V}_p) = \sum_{v_i \in \mathcal{V}_p} w^c(v_i), \forall c \in \{1,2,..,C\}. \quad (3)$$

denotes the c^{th} weight of part $\mathcal{V}_p$. In Π, a net is said to connect a part if it connects at least one vertex in that part. $con(n)$ denotes the number of parts that net n connects. Net n is called *cut* if it connects at least two parts, i.e., $con(n) > 1$, and called *internal* otherwise.

In the HP problem, the partitioning objective is to minimize the cutsize, which is defined as

$$cutsize(\Pi) = \sum_{n \in \mathcal{N}} (con(n)-1)c(n). \quad (4)$$

The partitioning constraint is to maintain balance on part weights

$$W^c(\mathcal{V}_p) \leq W^c_{avg}(1+\epsilon), \forall \mathcal{V}_p \in \Pi, \forall c \in \{1,2,..,C\}, \quad (5)$$

where $W^c_{avg} = \sum_{p=1}^{P} W^c(\mathcal{V}_p)/P$ denotes the average part weight for the c^{th} constraint and ϵ denotes the maximum allowed imbalance ratio.

The Recursive Bipartitioning (RB) paradigm is widely used in graph/hypergraph partitioning. In the RB paradigm, an input hypergraph is bipartitioned recursively in $\log_2 P$ steps to obtain P parts. Without loss of generality, we assume that P is an exact power of 2. A bipartition $\Pi_2 = \{\mathcal{V}_L, \mathcal{V}_R\}$ of a hypergraph $\mathcal{H}$ at some RB step is used to produce two new hypergraphs $\mathcal{H}_L = (\mathcal{V}_L, \mathcal{N}_L)$ and $\mathcal{H}_R = (\mathcal{V}_R, \mathcal{N}_R)$, where L and R to refer to the two parts as *Left* and *Right*, respectively. The net sets $\mathcal{N}_L$ and $\mathcal{N}_R$ are obtained by keeping all internal nets of each part and splitting the cut-nets using the cut-net splitting method [18].

2.5 Fine-Grain (FG) Partitioning for MTTKRP

In the work by Kaya and Uçar [9], a fine-grain task is defined as the multiplication of a tensor nonzero by the Hadamard product of the corresponding rows of the factor matrices along all but the mode of the factor matrix being computed (according to (1)).

A fine-grain hypergraph model $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is proposed [9] for fine-grain task partitioning. $\mathcal{H}$ contains a vertex v_{ijk} for each tensor nonzero $\mathcal{X}(i,j,k)$, and nets n_i^H, n_j^L and n_k^F for respectively each nonempty horizontal, lateral and frontal slice of the tensor. Each vertex v_{ijk} is connected by three nets n_i^H, n_j^L and n_k^F.

All vertices of $\mathcal{H}$ are assigned a unit weight under the assumption that every nonzero of $\mathcal{X}$ incurs the same amount of computation during the MTTKRP operations. All nets of $\mathcal{H}$ are assigned a cost of R since factor-matrix rows of size R words are communicated between processors. Then, the partitioning objective of minimizing the cutsize encodes minimizing the total volume of communication due to expand-type communications on input matrix rows as well as reduce-type communications on output matrix rows.

3 Deficiencies of the Fine-Grain Model

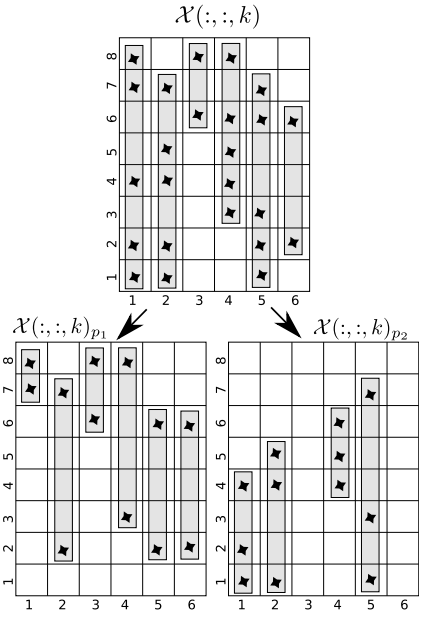


Fig. 2: A bipartition of slice $\mathcal{X}(:,:,k)$ to processors p_1 and p_2, having the same nonzero count but different flop counts.

3.1 Failure to encode processors' computational loads

The COO-based implementation of the MTTKRP operation according to (1) incurs $3Rm$ flops for tensor $\mathcal{X}$ with m nonzero elements. Here, $2Rm$ flops are performed for the initial products and Rm flops are performed for the summation operations. In other words, each tensor nonzero incurs $3R$ flops. So although vertices are assigned unit weight in the conventional FG model, we assume the vertices are assigned a weight of

$$w(v_{ijk}) = 3R. \tag{6}$$

In this way, the part weights computed by using (6) in (3) will correctly encapsulate processor's computational loads .

On the other hand, the CSF format enables reducing the amount of total computation from $3Rm$ flops to $2R(m+F)$ flops, where F denotes the number of fibers, using the formulation in (2). This is because, in (2), each nonzero $\mathcal{X}(i,j,k)$ incurs $2R$ flops due to $\mathbf{C}(k,:)$, whereas each fiber $\mathcal{X}(i,j,:)$ incurs $2R$ flops due to $\mathbf{B}(j,:)$. That is, the amount of computation associated with each nonzero may differ depending on the fiber fragmentation introduced by the partitioning algorithm. So, the part weights computed according to (6) fail to correctly encapsulate the computational loads of processors.

The top part of Fig. 2 shows a sample 8×6 frontal slice $\mathcal{X}(:,:,k)$ with 24 nonzeros. In the figure, stars represent nonzeros while shaded rectangles represent fibers along the longest mode which is the second mode. The bottom part shows a bipartition of the nonzeros of the slice between two processors p_1 and p_2. The subslices assigned to p_1 and p_2 are respectively denoted by $\mathcal{X}(:,:,k)_{p_1}$ and $\mathcal{X}(:,:,k)_{p_2}$. This bipartition shows an even nonzero partition since each subslice has 12 nonzeros. However, the nonzeros of the subslices $\mathcal{X}(:,:,k)_{p_1}$ and $\mathcal{X}(:,:,k)_{p_2}$ respectively belong to 6 and 4 subfibers. Thus, $\mathcal{X}(:,:,k)_{p_1}$ will incur $2R(12+6) = 36R$ flops associated with the MTTKRP operation on p_1, whereas $\mathcal{X}(:,:,k)_{p_2}$ will incur $32R$ flops on p_2. So, despite the even nonzero distribution, the partition incurs a significant amount of computational load imbalance.

3.2 Increase in total computation

As mentioned in Section 3.1, the number of flops performed using the fiber-centric MTTKRP formulation in (2) is equal to $2R(m+F)$ in serial and shared-memory settings. However, in distributed-memory settings, since the fibers are local to the processors, the number of flops increases as a result of fragmenting fibers among processors.

Assuming single precision floating-point values, the COO-based MTTKRP incurs $16m + 12Rm$ memory byte accesses [19], whereas the CSF-based MTTKRP with S slices incurs $8(S+F+m) + 12R(m+F)$ accesses . Note that fiber fragmentation increases the number of flops as well as the number of memory accesses at the same rate, thus it does not affect the arithmetic intensity (flops per byte) of the CSF-based MTTKRP. Therefore, any further discussion on increasing/decreasing flop counts also applies to the associated number of memory accesses.

In an ideal situation, each fiber is assigned to a single processor as a whole without any fragmentation thus resulting in no increase in the number of flops, which can be set as the lower bound for distributed-memory settings. However, any fiber whose nonzeros are fragmented among λ processors will incur $2R(\lambda-1)$ additional flops. In the worst-case, if every nonzero of each fiber is assigned to a different processor, then each fiber will have a single nonzero, resulting in a loose upper bound of $3Rm$ total flops following the *if*-statement in Algorithm 2.

The bipartition shown in the bottom part of Fig. 2 incurs the fragmentation of 4 out of 6 fibers of $\mathcal{X}(:,:,k)$. So, this bipartition incurs an increase in the total number of fibers from 6 to 10 thus increasing the total number of flops from

$2R(24+6) = 60R$ to $36R+32R = 68R$ during the MTTKRP operations associated with $\mathcal{X}(:,:,k)$.

Since the fine-grain HP-based method described in Section 2.5 is not aware of the role of tensor fibers while partitioning the HP model, it may incur a significant amount of fiber fragmentation leading to significant increase in the total computational load.

4 Improving Fine-Grain HP Model

4.1 A novel vertex weighting scheme

As mentioned earlier, balancing on the flop counts of processors cannot be enforced during partitioning the fine-grain HP model. This is because of the vertex weighting scheme that only encodes balancing nonzero counts of processors while failing to encode the fiber counts.

Here, we propose a novel vertex weighting scheme for estimating correct flop counts of processors during partitioning the fine-grain model. For this purpose, we propose an Inverse-Fiber-Size (IFS) heuristic for estimating the fiber counts of processors. In the IFS scheme, the $2R$ flop contribution of a fiber is distributed uniformly, as vertex weights, among the vertices representing the tensor nonzeros constituting that fiber. That is, a fiber $\mathcal{X}(i,j,:)$ of size $nnz(\mathcal{X}(i,j,:))$ contributes $2R/nnz(\mathcal{X}(i,j,:))$, as a weight, to each vertex representing its constituent nonzeros.

In a given nonzero partition, if the nonzeros of a given fiber are all assigned to the same part, the IFS scheme correctly encodes the contribution of a fiber count ($2R$) to the respective part. If, however, the nonzeros of a given fiber are fragmented between two parts, then the IFS scheme will incur fractional fiber count contributions to these two parts with a sum of $2R$.

Since the two efficient schemes described in Section 2.3 (CSF-S and CSF-D) for computing the MTTKRP have different algorithms and different fiber types, we describe how the IFS scheme is applied to each of them separately. Without loss of generality, we assume that tube fibers ($\mathcal{X}(i,j,:)$) and row fibers ($\mathcal{X}(i,:,k)$) are the tensor's fibers along the longest and second longest modes, respectively.

4.1.1 IFS scheme for CSF-S

In this scheme, the tensor is stored only once as fibers of the longest mode. While computing the MTTKRP for $N-1$ modes as described in Algorithm 2, the number of fibers times $2R$ correctly encapsulate flop count of the Hadamard product and the addition operation involving $\mathbf{B}(jf,:)$ (line 12). On the other hand, while computing the MTTKRP for the longest mode using Algorithm 3, the number of fibers times $2R$ correctly encapsulates the flop count of only the Hadamard product operations (line 9). Therefore, we use the IFS scheme for updating the weights of the vertices as follows: for each $v_{ijk} \in \mathcal{V}$

$$w(v_{ijk}) = 2R + \frac{2R}{nnz(\mathcal{X}(i,j,:))}. \tag{7}$$

Here, "$2R$" refers to the number of flops associated with the respective nonzero, whereas $2R/nnz(\mathcal{X}(i,j,:))$ refers to the number of flops associated with the fiber that contains the respective nonzero.

4.1.2 IFS scheme for CSF-D

In this scheme, the tensor is stored twice in fiber-centric fashion. As discussed in Section 2.3, the first storage utilizes the fibers of the longest mode, while the second storage utilizes the fibers of the second longest mode. For both fiber types, the number of fibers times $2R$ correctly encapsulate flop count of the Hadamard product and addition operations (Algorithm 2 line 12). The distinction is, the number of fibers along the longest mode correctly encapsulates the number of flops during each of the $N-1$ MTTKRP operations performed along all but the longest mode, whereas number of fibers along the second longest mode correctly encapsulates the number of flops during the MTTKRP operation along the longest mode.

A two-constraint formulation is needed for balancing the computational loads of processors in the computational scheme that utilizes CSF-D. This is because, in the CP-ALS algorithm, MTTKRP operations are performed in different phases interleaved with synchronizing communication operations, and the MTTKRP operations are performed with two different types of fibers.

We use the IFS scheme to compute the two weights of the vertices for the two-constraint formulation as follows: for each $v_{ijk} \in \mathcal{V}$

$$w^1(v_{ijk}) = 2R + \frac{2R}{nnz(\mathcal{X}(i,j,:))} \tag{8a}$$

$$w^2(v_{ijk}) = 2R + \frac{2R}{nnz(\mathcal{X}(i,:,k))}. \tag{8b}$$

At each iteration, $W^1(\mathcal{V}_p)$ (computed using (3)) encodes the computational load of processor p during $N-1$ MTTKRP operations, whereas $W^2(\mathcal{V}_p)$ encodes the computational load of processor p during only one MTTKRP operation. So, the success of this two-constraint scheme depends on giving more importance to the first over second constraint. This can only be achieved by relaxing the maximum allowed imbalance ratio (ϵ) of the second constraint. Unfortunately, the state-of-the-art HP tools do not support different ϵ values for different constraints. For this reason, we propose the following alternative single-constraint weighting scheme that can emulate the above mentioned two-constraint scheme:

$$w(v_{ijk}) = (N-1)w^1(v_{ijk}) + w^2(v_{ijk}) \tag{9a}$$

$$= 2RN + \frac{2R(N-1)}{nnz(\mathcal{X}(i,j,:))} + \frac{2R}{nnz(\mathcal{X}(i,:,k))} \tag{9b}$$

for each $v_{ijk} \in \mathcal{V}$. In (9a), the relative importance of $w^1(v_{ijk})$ over $w^2(v_{ijk})$ is modeled by multiplying $w^1(v_{ijk})$ by $N-1$ as the CSF storage along the longest mode is used in $N-1$ MTTKRP operations at each CP-ALS iteration. Note that in (9a) and (9b) the value of N should be set to 3 in case of a 3-mode tensor, but we prefer to use N for a more general presentation.

4.2 Improving IFS through utilizing RB

The accuracy of the IFS heuristic depends on keeping track of the correct fibers sizes, which could change significantly as a result of fiber fragmentation during partitioning. We propose to utilize the RB scheme to increase the accuracy of the IFS heuristic in estimating the fiber counts of parts. After each bipartitioning step, the sizes of the fragmented fibers

Algorithm 4 RB-based FG HP with IFS scheme

Require: Sparse tensor $\mathcal{X}$
1: $\mathcal{H} \leftarrow$ Fine-grain hypergraph of $\mathcal{X}$
2: ▷ $\mathcal{F}$ is the set of nonzero fibers along the longest mode.
3: $\mathcal{F} \leftarrow \{f_{ij} = \mathcal{X}(i,j,:) : nnz(\mathcal{X}(i,j,:)) \neq 0\}$
4: **if** $\mathcal{X}$ is stored as CSF-S **then**
5: RB-STEP-S($\mathcal{H}$, $\mathcal{F}$)
6: **else** ▷ $\mathcal{X}$ is stored as CSF-D
7: ▷ $\mathcal{F}2$ is the set of nonzero fibers along 2^{nd} longest mode.
8: $\mathcal{F}2 \leftarrow \{f_{ik} = \mathcal{X}(i,:,k) : nnz(\mathcal{X}(i,:,k)) \neq 0\}$
9: RB-STEP-D($\mathcal{H}$, $\mathcal{F}$, $\mathcal{F}2$)
10: **function** RB-STEP-S($\mathcal{H}$, $\mathcal{F}$)
11: $\Pi_2 = (\mathcal{V}_L, \mathcal{V}_R) \leftarrow$ BIPARTITION($\mathcal{H}$)
12: Form $\mathcal{H}_L = (\mathcal{V}_L, \mathcal{N}_L)$ and $\mathcal{H}_R = (\mathcal{V}_R, \mathcal{N}_R)$
13: $(\mathcal{F}_L, \mathcal{F}_R)$=SPLIT-FIBERS($\Pi_2$, $\mathcal{F}$)
14: UPDATE-WEIGHTS-S($\Pi_2, \mathcal{F}_L, \mathcal{F}_R$)
15: RB-STEP-S($\mathcal{H}_L$, $\mathcal{F}_L$)
16: RB-STEP-S($\mathcal{H}_R$, $\mathcal{F}_R$)
17: **function** RB-STEP-D($\mathcal{H}$, $\mathcal{F}$, $\mathcal{F}2$)
18: $\Pi_2 = (\mathcal{V}_L, \mathcal{V}_R) \leftarrow$ BIPARTITION($\mathcal{H}$)
19: Form $\mathcal{H}_L = (\mathcal{V}_L, \mathcal{N}_L)$ and $\mathcal{H}_R = (\mathcal{V}_R, \mathcal{N}_R)$
20: $(\mathcal{F}_L, \mathcal{F}_R)$=SPLIT-FIBERS($\mathcal{F}, \Pi_2$)
21: $(\mathcal{F}2_L, \mathcal{F}2_R)$=SPLIT-FIBERS($\mathcal{F}2$,$\Pi_2$)
22: UPDATE-WEIGHTS-D(Π_2, $\mathcal{F}_L$, $\mathcal{F}_R$, $\mathcal{F}2_L$, $\mathcal{F}2_R$)
23: RB-STEP-D($\mathcal{H}_L$, $\mathcal{F}_L$, $\mathcal{F}2_L$)
24: RB-STEP-D($\mathcal{H}_R$, $\mathcal{F}_R$, $\mathcal{F}2_R$)

Algorithm 5 SPLIT-FIBERS

Require: (Π_2, $\mathcal{F}$)
1: $\mathcal{F}_L, \mathcal{F}_R \leftarrow \emptyset$
2: **for all** $\mathcal{X}(i,j,:) = f_{ij} \in \mathcal{F}$ **do**
3: $f_{ij}^L = \mathcal{X}^L(i,j,:) = \mathcal{X}(i,j,:) \cap \{\mathcal{X}(i,j,k) : v_{ijk} \in \mathcal{V}_L\}$
4: $f_{ij}^R = \mathcal{X}^R(i,j,:) = \mathcal{X}(i,j,:) \cap \{\mathcal{X}(i,j,k) : v_{ijk} \in \mathcal{V}_R\}$
5: **if** $nnz(f_{ij}^L) > 0$ **then**
6: $\mathcal{F}_L \leftarrow \mathcal{F}_L \cup \{f_{ij}^L\}$
7: **if** $nnz(f_{ij}^R) > 0$ **then**
8: $\mathcal{F}_R \leftarrow \mathcal{F}_R \cup \{f_{ij}^R\}$
return $(\mathcal{F}_L, \mathcal{F}_R)$

Algorithm 6 UPDATE-WEIGHTS-S

Require: Π_2 , $\mathcal{F}_L$, $\mathcal{F}_R$
1: **for all** $v_{ijk} \in \mathcal{V}_L$ **do**
2: $w(v_{ijk}) \leftarrow 2R + \dfrac{2R}{nnz(\mathcal{X}^L(i,j,:))}$
3: **for all** $v_{ijk} \in \mathcal{V}_R$ **do**
4: $w(v_{ijk}) \leftarrow 2R + \dfrac{2R}{nnz(\mathcal{X}^R(i,j,:))}$

Algorithm 7 UPDATE-WEIGHTS-D

Require: Π_2, $\mathcal{F}_L$, $\mathcal{F}_R$, $\mathcal{F}2_L$, $\mathcal{F}2_R$
1: **for all** $v_{ijk} \in \mathcal{V}_L$ **do**
2: $w(v_{ijk}) = 2RN + \dfrac{2R(N-1)}{nnz(\mathcal{X}^L(i,j,:))} + \dfrac{2R}{nnz(\mathcal{X}^L(i,:,k))}$
3: **for all** $v_{ijk} \in \mathcal{V}_R$ **do**
4: $w(v_{ijk}) = 2RN + \dfrac{2R(N-1)}{nnz(\mathcal{X}^R(i,j,:))} + \dfrac{2R}{nnz(\mathcal{X}^R(i,:,k))}$

are updated for recomputing the vertex weights according to the IFS heuristic.

Algorithm 4 shows the proposed RB-based IFS scheme. In the algorithm, $\mathcal{H}$ refers to the current hypergraph to be bipartitioned, whereas $\mathcal{F}$ and $\mathcal{F}2$ refer to the current set of nonzero fibers along the first and second longest modes, respectively. The sets of (fragmented) fibers are maintained during the RB scheme for recomputing the vertex weights according to the correct fiber sizes. Note that both $\mathcal{F}$ and $\mathcal{F}2$ are used for the CSF-D scheme while only $\mathcal{F}$ is required for the CSF-S scheme. The algorithm checks whether CSF-S or CSF-D is used and respectively invokes RB-STEP-S or RB-STEP-D accordingly.

In lines 11 and 18 of Algorithm 4, the hypergraph partitioning tool is invoked to obtain a bipartition Π_2 on the vertices of $\mathcal{H}$. In lines 12 and 19, the left hypergraph $\mathcal{H}_L$ and right hypergraph $\mathcal{H}_R$ are constructed according to the net-splitting strategy mentioned in Section 2.4. In line 13, SPLIT-FIBERS function is invoked to form the fiber sets $\mathcal{F}_L$ and $\mathcal{F}_R$ of the left and right parts, for the CSF-S scheme. In lines 20 and 21, SPLIT-FIBERS is invoked to compute $\mathcal{F}_L$ and $\mathcal{F}_R$ as well as $\mathcal{F}2_L$ and $\mathcal{F}2_R$ of the left and right parts, respectively, for the CSF-D scheme.

The SPLIT-FIBERS function (Algorithm 5) implements the fiber fragmentation strategy as follows. The *for*-loop in lines 2-8 computes the intersection of each fiber of the current fiber set $\mathcal{F}$ with the nonzeros corresponding to the vertices of the left and right parts. Then, it assigns an unfragmented fiber to either $\mathcal{F}_L$ or $\mathcal{F}_R$, whereas it adds the subfibers of a fragmented fiber to both $\mathcal{F}_L$ and $\mathcal{F}_R$.

Then, in lines 14 and 22 of Algorithm 4, the vertex weighting scheme is invoked in order to recompute the weights of vertices according to the IFS scheme with correct (fragmented) fiber sizes. Algorithm 6 (UPDATE-WEIGHTS-S) is used to update the weights for CSF-S according to (7), whereas Algorithm 7 (UPDATE-WEIGHTS-D) is used to update the weights for CSF-D according to (9).

4.3 Fiber-net augmentation for reducing total flops

In conventional graph/hypergraph partitioning formulations used for irregular scientific applications in distributed settings, the total amount of computational work is constant. So, in these formulations the partitioning constraint of balancing the part weights correctly corresponds to reducing the computational load of the maximally loaded processor (bottleneck processor). This correspondence will refer to minimizing the computational load of the bottleneck processor as the maximum allowed imbalance ratio (ϵ) is reduced. This is in fact the case for finding a fine-grain partitioning formulation for parallel tensor decomposition which utilizes the COO format for local MTTKRP computations (formulation (1)). However, the total amount of computational work is not constant in the fine-grain partitioning formulation that utilizes the CSF format for local MTTKRP computations (formulation (2)). Hence, the partitioning constraint of balancing the part weights loosely relates to reducing the computational load of the bottleneck processor.

The partitioning constraint of balancing part weights correctly refers to reducing the computational load of the bottleneck processor if the partitioning formulation targets at reducing the increase in the total computational load due to fiber fragmentation while minimizing the total communication volume. For this purpose, the standard fine-grain hypergraph model, which contains slice nets that encode communication volume, is augmented with fiber nets. Each

fiber net connects all vertices corresponding to the nonzeros constituting the fiber.

For the CSF-D scheme, a net n^f_{ij} is created for each nonzero fiber $\mathcal{X}(i,j,:)$ along the longest mode. Similarly, a net n^f_{ik} is created for each nonzero fiber $\mathcal{X}(i,:,k)$ along the second longest mode. The sets of vertices connected by n^f_{ij} and n^f_{ik} are respectively defined as:

$$Pins(n^f_{ij}) = \{v_{ijk} : \mathcal{X}(i,j,k) \neq 0 \ \ \forall k \in \{1,..,K\}\} \quad (10a)$$

$$Pins(n^f_{ik}) = \{v_{ijk} : \mathcal{X}(i,j,k) \neq 0 \ \ \forall j \in \{1,..,J\}\} \quad (10b)$$

For the CSF-S scheme, constructing the nets for the longest-mode fibers suffices, and the set of vertices connected by each fiber net is the same as in (10a).

The fiber-net augmentation can be easily integrated into the RB-based framework given in Algorithm 4. After constructing the hypergraph model, the sets of fibers $\mathcal{F}$ (line 3) and $\mathcal{F}2$ (line 8) provide the sufficient nonzero-to-fiber relations that can be used to construct the fiber nets. No other modifications are needed in the RB-STEP routines.

Consider a partition Π of an augmented hypergraph for the CSF-S scheme. In Π, a cut slice net n^s with connectivity $con(n^s)$ will incur a communication of $R(con(n^s)-1)$ words during each MTTKRP operation as in the standard fine-grain hypergraph model. In Π, internal fiber nets do not incur any increase in the total number of flops. However, a cut fiber net n^f with connectivity $con(n^f)$ encodes an increase of $2R(con(n^f)-1)$ flops during each MTTKRP operation. A similar discussion holds for the CSF-D scheme.

For the CSF-S scheme, the cost of fiber nets along the longest mode is set to $2R$, whereas the cost of slice nets is set to αR. For the CSF-D scheme, the cost of fiber nets along the longest and second longest modes are set to $2R(N-1)$ and $2R$ respectively, whereas the cost of slice nets are set to αRN. Here, α refers to the scaling factor between the cost of increasing the communication volume by R words and the cost of increasing the total flop count by $2R$.

In the augmented fine-grain hypergraph model, the partitioning objective of minimizing the cut size will simultaneously encode minimizing both the communication volume and the increase in total flop count. The partitioning constraint of maintaining balance on part weighs (according to the proposed vertex weighting schemes described in Section 4.1) will encode minimizing the flop count of the bottleneck processor with decreasing ϵ because of the proposed fiber-net augmentation.

The augmentation of fiber nets is also expected to contribute to improving the accuracy of the IFS scheme. Reducing the number of cut fiber nets relates to maximizing the number of internal nets, where internal nets correspond to unfragmented fibers. So, increasing the number of unfragmented fibers enables the IFS scheme to correctly encode the contribution of larger number of fibers to the part weights. So, the objective of reducing fiber fragmentation decreases the number of erroneous vertex weight contributions incurred by fragmented fibers. This decrease is expected to improve the accuracy of the IFS scheme thus leading to better load balancing.

5 Experiments

5.1 Setting

There are several successful hypergraph partitioning tools [18, 20, 21]. We use PaToH [18] (version 3.2) in speed mode and the value of ϵ is set to 0.10. Since PaToH contains randomized algorithms, we partition each tensor three times for each partitioning method, and we report the average of the three instances.

The topologies of the hypergraph models are orthogonal to the value of R. On the other hand, the vertex weighting schemes as well as the net costs presented in this paper involve R, which acts as a scaling factor, for the sake of clarity of presentation. Thus, removing this scaling factor affects neither the cutsize nor the balancing qualities, so in our partitioning implementation the R value is set to one.

For the parallel experiments, we use the parallel CP-ALS code developed and used in the work by Acer et al. [11]. The code is implemented in C, uses MPI for interprocess communication and compiled with `gcc` version 8.3.0 using `O3` optimization flag. The MTTKRP implementation in the code is based on the flop-efficient formulation in (2), which is identical to CSF-D. We have modified the code to include CSF-S. The runtimes of CP-ALS are reported as per-iteration times by taking the average of total runtime of 50 iterations.

Our parallel experiments are conducted on Bull Sequana X1000 system. A node in this system operates on dual Intel Xeon Skylake 8168 with total of 48 cores, 96 GB of memory and 2.70 GHz clock frequency. The nodes are connected with the high speed network EDR-Infiniband (Connect-X4).

5.2 Dataset

Our dataset is composed of six real-world sparse tensors commonly used as a benchmark for parallel sparse tensor research. Table 1 shows the properties of the tensors. `Enron` [22] consists of words of email exchanges in the form of *sender-receiver-word-date* quadruplets. It has been used with tensor decomposition methods for social network analysis and link prediction [23]. `Flickr` is a binary tensor representing *user-image-tag-date* quadruplets, which was first crawled by Görlitz et al. [24] from flickr.com.

Bhargava et al. [25] factorize `Flickr` using CP-ALS for forming multi-dimensional collaborative recommendations. `Movies-amazon` represents *user-movie-word* triplets extracted from the user reviews of movies in Amazon [26]. `Movies-amazon` is one of the datasets used for evaluating recommender systems research, including CPD-based systems. `Nell-1` and `Nell-2` [27] represent *entity-relation-entity* tuples of the Never Ending Language Learner knowledge base. Kang et al. [28] used both tensors for concept discovery and contextual synonym detection using CP-ALS. `Yelp` contains *user-business-word* triplets obtained from business reviews in Yelp academic dataset [1]. Yelp is generally used in the context of tensor decomposition for community detection and recommender systems.

5.3 Performance comparison

We compare the performance of the proposed improvement schemes against the baseline FG method in terms of computational and communication cost metrics as well as

1. https://www.yelp.com/dataset/challenge

TABLE 1: Properties of Test Tensors

Tensor	size of dimensions I	J	K	L	nnz	Density
Enron	6.0K	5.6K	244.2K	1.1K	54.2M	$5.5{\cdot}10^{-9}$
Flickr	319.6K	28.1M	1.6M	730	112.9M	$1.1{\cdot}10^{-14}$
Movies-amazon	87.8K	4.4K	226.5K	—	15.0M	$1.7{\cdot}10^{-7}$
Nell-1	2.9M	2.1M	25.5M	—	143.6M	$9.1{\cdot}10^{-13}$
Nell-2	12.1K	9.2K	28.8K	—	76.8M	$2.4{\cdot}10^{-5}$
Yelp	686.5K	85.5K	773.2K	—	185.5M	$4.1{\cdot}10^{-9}$

parallel MTTKRP and CP-ALS times on the 6 tensors given in Table 1. We use $P = 512$ and $\alpha = 10$ in all tables unless specified otherwise.

The computational cost metrics consist of maximum and average number of flops performed by a processor. The communication cost metrics consist of maximum and average send volume handled by a processor. The latency-based communication cost metrics regarding maximum and average number of messages sent by a processor are not reported as all methods display almost the same performance on these metrics. Here, average flop count and average volume values refer to the total flop count and total volume values, respectively, divided by the number of processors. We prefer to report average values instead of total values because the former give a better view on the deviation of maximum from average. When a normalized value is presented, it means the value of the respective method divided by that of other method (usually the baseline). Since we aim at minimizing all performance metrics considered in this paper, a normalized value of < 1 means an improvement over the baseline, and deterioration otherwise.

5.3.1 Results of CSF-S experiments

Table 2 displays the performance improvement rates attained by the optimization schemes in Section 4 in an incremental way. Note that "Avg." row at the bottom of the table and all other tables refers to the geometric mean. As seen in the table, on average, utilizing IFS for vertex weighting (Algorithm 4) in FG+ improves the maximum and average flop counts by 8.0% and 4.0%, respectively, compared to FG. Fiber-net augmentation used in FG++ significantly decreases maximum and average flop counts respectively by 16.3% and 14.6% compared to FG+. As seen in the table, utilizing the two optimization schemes in FG++ leads to a significant decrease in the maximum and average flop counts respectively by 23.2% and 17.8% compared to the baseline FG method.

As seen in Table 2, in terms of communication volume metrics, FG+ attains slightly better performance compared to FG. That is, FG+ reduces the maximum and average communication volumes by 6.2% and 7.0% compared to FG, on average. Comparing FG++ against FG+ shows that although they display comparable performance in terms of average communication volume, FG++ achieves considerably better performance in terms of maximum communication volume by an amount of 9.6%. As seen in the table, FG++ achieves a considerable decrease in the maximum and average communication volume respectively by 15.5% and 7.3% compared to the baseline FG method. These findings show that the use of fiber nets do not lead to performance degradation in communication volume metrics. This can be attributed to the fact that fiber nets are subnets of the slice nets. Relatively better performance obtained by FG++ against FG in terms of maximum volume compared to average volume can be attributed to the expectation that better computational load balancing achieved by FG++ leads to a better communication volume balancing. Here and hereafter, the proposed FG++ will be referred to as impFG.

Table 3 shows how the above-mentioned performance improvements lead to improving the actual parallel runtimes. In the table, the values under FG are actual runtimes, while those under impFG are normalized with respect to those of FG. Under MTTKRP tab, the "comp" column refers to the computational part of the MTTKRP operation, whereas "tot" refers to the total runtime of the MTTKRP operation including communication. Comparing "max flops" column of impFG in Table 2 with the "comp" column of impFG in Table 3 show that there exist close correlation between the amount of improvement in maximum flop count and the amount of improvement in parallel MTTKRP computation time. That is, the 23% improvement attained by impFG in maximum flop count reflects as approximately 22% improvement in parallel MTTKRP computation time. In fact, this close relation also applies to individual tensors except for Nell-2. For example, 28%, 25%, 13%, 18% and 29% reduction in max flop counts obtained by impFG for the tensors Enron, Flickr, Amazon, Nell-1 and Yelp respectively reflect as approximately 36%, 23%, 11%, 15% and 32% improvement in parallel MTTKRP computation times. This confirms the validity of the maximum flop count metric in determining the parallel computation time.

Table 3 also shows relative runtime performance variation of impFG over FG with increasing R. We use the same partitioned tensor, for each tensor, to obtain the parallel running times with different R values. Keep in mind that with increasing R value, the improvement ratios of impFG over FG remain the same in terms of computational and communication cost metrics (as in Table 2). As seen in the normalized columns of Table 3, the relative performance of impFG over FG slightly increases with increasing R in terms of both parallel MTTKRP and CP-ALS runtimes. This is expected because, with increasing R, while latency-based communication costs remain the same, communication volume and computational costs increase.

Table 4 shows the effect of augmenting fiber nets on the total communication volume along the longest mode as well as the other $N-1$ modes. In the table, the values under FG are actual communication volume values (in words), while those under impFG are normalized with respect to those of FG. Comparing the relative performance of impFG over FG, the fiber-net augmentation along the longest mode incurs an increase in communication volume during the MTTKRP operation along that mode, whereas it achieves a decrease in communication volume during MTTKRP operations along all other $N-1$ modes. As seen in the table, on average, impFG incurs 16% increase in total volume during MTTKRP along the longest mode, whereas it achieves 20% decrease in that along all other $N-1$ modes.

The above-mentioned experimental finding can be attributed to the fact that the nets representing the fibers along the longest mode are subnets of the nets that represent slices

TABLE 2: Performance comparison in terms of computational and communication cost metrics on $P = 512$ processors for CSF-S.

	Actual values (in terms of R)				Normalized with respect to FG							
	FG				FG+IFS (FG+)				FG+IFS+FNA (FG++ ≡ impFG)			
	flops		comm. vol.		flops		comm. vol.		flops		comm. vol.	
Tensor	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg
Enron	718,123	648,890	43,723	22,436	0.99	0.91	1.00	0.93	0.72	0.70	0.86	0.81
Flickr	1,477,230	1,157,132	105,871	52,912	0.80	0.99	0.86	0.96	0.75	0.93	0.75	0.85
Movies-amazon	126,153	107,987	40,290	22,137	0.93	0.96	1.03	0.89	0.87	0.92	1.00	0.94
Nell-1	1,491,700	1,421,460	409,820	279,581	0.98	0.98	0.98	0.98	0.82	0.84	0.92	1.04
Nell-2	769,255	734,175	82,056	40,171	1.00	0.98	0.90	0.92	0.74	0.73	0.79	1.15
Yelp	1,689,961	1,320,840	384,328	113,941	0.85	0.94	0.86	0.93	0.71	0.85	0.79	0.82
Avg.	**798,660**	**694,041**	**115,793**	**56,814**	**0.92**	**0.96**	**0.94**	**0.93**	**0.77**	**0.82**	**0.85**	**0.93**

IFS: Inverse-Fiber-Size for vertex weighting with RB (Secs. 4.1 & 4.2); FNA: Fiber Net Augmentation (Sec. 4.3) with $\alpha = 10$.

TABLE 3: Performance comparison in terms of parallel runtimes on $P = 512$ processors for CSF-S.

Tensor	R	FG (times in ms)			impFG (normalized)		
		MTTKRP		CP-ALS	MTTKRP		CP-ALS
		comp	tot		comp	tot	
Enron	32	55.9	152.4	156.8	0.68	0.83	0.83
	64	124.0	245.0	260.9	0.64	0.77	0.77
	128	302.2	477.5	527.6	0.58	0.69	0.72
Flickr	32	128.5	246.6	330.1	0.77	0.84	0.93
	64	246.3	435.3	628.8	0.77	0.80	0.92
	128	485.0	824.9	1,312.1	0.76	0.80	0.95
Movies-a	32	12.5	92.2	100.0	0.91	0.93	0.93
	64	28.2	135.5	153.4	0.89	0.93	0.93
	128	60.4	218.7	276.3	0.89	0.93	0.92
Nell-1	32	295.3	718.7	888.4	0.86	0.81	0.85
	64	570.1	1,422.2	1,772.8	0.85	0.82	0.87
	128	1,249.0	3,106.2	3,975.8	0.84	0.85	0.89
Nell-2	32	67.7	165.7	176.4	0.99	1.02	1.03
	64	159.7	295.4	325.2	0.95	0.99	0.99
	128	419.8	643.6	733.1	0.89	0.93	0.94
Yelp	32	202.8	386.2	457.6	0.63	0.75	0.74
	64	379.3	716.5	891.3	0.66	0.73	0.72
	128	725.8	1,365.8	1,861.1	0.72	0.76	0.74
Avg.	32	**84.5**	**232.9**	**268.1**	**0.79**	**0.86**	**0.88**
	64	**176.0**	**404.0**	**484.5**	**0.78**	**0.84**	**0.86**
	128	**387.2**	**785.7**	**1,006.2**	**0.77**	**0.82**	**0.85**

TABLE 4: Performance comparison in terms of total volume during MTTKRP along the longest mode and other $N-1$ modes on $P = 512$ processors for CSF-S.

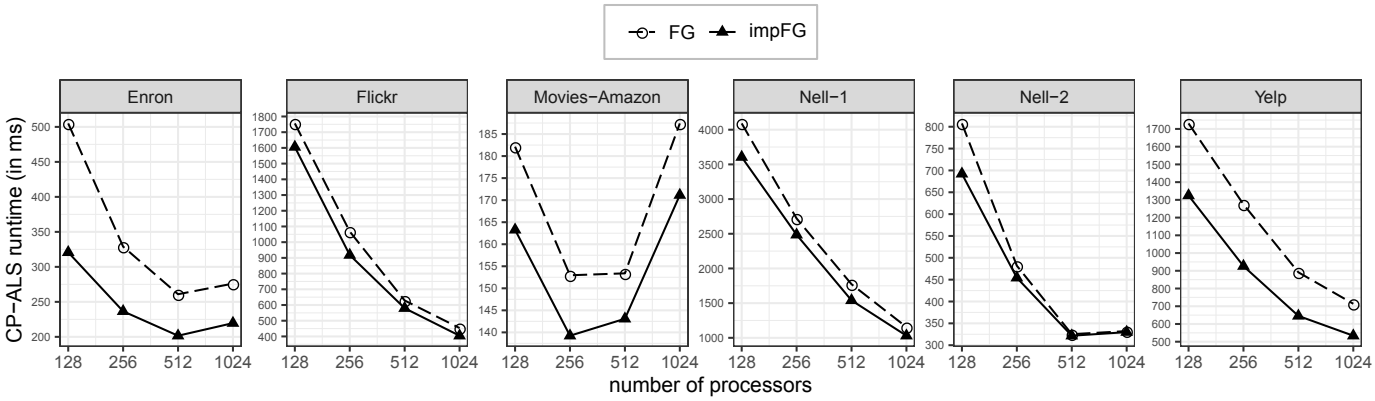
Tensor	FG (total volume, in terms of R)		impFG (normalized)	
	Longest mode	$N-1$ modes	Longest mode	$N-1$ modes
Enron	4,476,252	7,006,890	1.06	0.65
Flickr	968,022	26,119,510	1.11	0.84
Movies-a	4,926,116	6,405,454	1.09	0.83
Nell-1	55,605,248	87,536,435	1.24	0.91
Nell-2	10,726,748	9,837,220	1.37	0.91
Yelp	18,634,076	39,700,541	1.10	0.68
Avg.	**7,868,030**	**18,499,059**	**1.16**	**0.80**

of the other $N-1$ modes. That is, trying to keep fiber nets along the longest mode internal can be expected to increase the possibility of keeping the nets representing slices along other $N-1$ modes internal as well. Recall that the communication volume during MTTKRP operations along different modes differ depending on the number and connectivities of the cut nets representing slices along those modes. As seen in the last column of Table 3, impFG achieves an average decrease of 7% compared to FG in total volume during all MTTKRP operations. However, for some tensors such as Nell-1 and Nell-2, fiber-net augmentation respectively incurs overall communication volume increase of 4% and 15%. This is because for FG on Nell-1 and Nell-2, the total volume along the longest mode is larger than or very close to that of all other N–1 modes. In other tensors, such as Flickr, the communication volume along all N–1 modes is significantly larger than that of the longest mode. Therefore, the overall improvement achieved by fiber-net augmentation depends on two factors; the relative communication volume during the longest and $N-1$ modes, and the increase/decrease incurred/achieved along the longest and other $N-1$ modes.

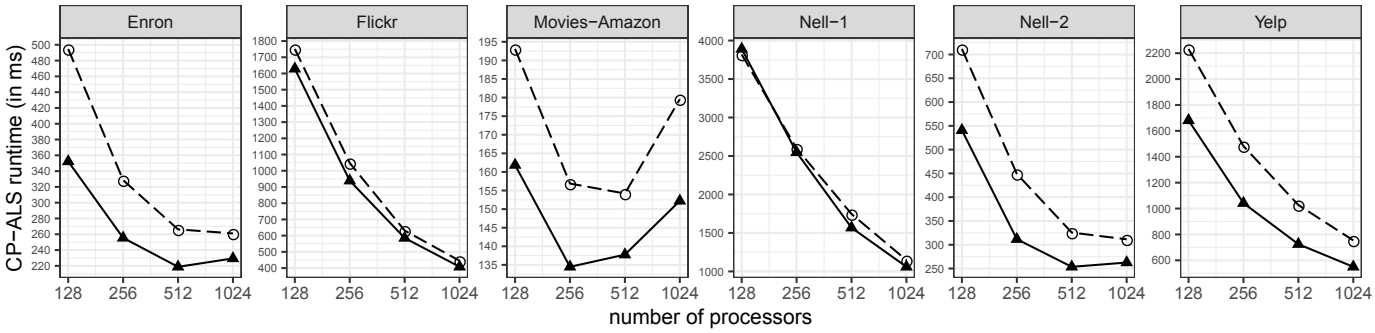
TABLE 5: Computational and communication cost metrics (in terms of R) of the impFG method with different α values on $P = 512$ for CSF-S.

	α	flops		comm. vol.	
		max	avg	max	avg
Avg. of all tensors	5	600,591	564,647	99,463	54,371
	10	613,372	572,075	97,739	52,903
	50	623,855	578,992	94,929	51,453
	100	626,631	580,123	95,516	51,085

Table 5 shows the effect of different α values on the performance of fiber-net augmentation in impFG. We ran the impFG method with $\alpha = 5, 10, 50$ and 100. We report the average flop count and communication volume statistics in Table 5 as actual values. As seen in the table, increasing the α value (giving more importance to decreasing total communication volume over decreasing fiber fragmentation) results in increasing the total flops while the communication volume is decreased. As a trade-off between total flops and communication volume, we choose $\alpha = 10$ for the rest of tables and figures in this section.



(a) CSF-S



(b) CSF-D

Fig. 3: Strong scaling curves for parallel CP-ALS obtained by FG and impFG using (a) CSF-S and (b) CSF-D

TABLE 6: Performance comparison in terms of computational and total volume metrics on $P = 512$ processors for CSF-D.

	FG + IFS^{S} +FNA^{S} (impFG^{S})									FG+IFS^{D}+FNA^{D} (impFG^{D})								
	Longest mode			$N-1$ modes			All modes			Longest mode			$N-1$ modes			All modes		
	flops		tot. vol.	flops		tot. vol.	flops		tot. vol.	flops		tot. vol.	flops		tot. vol.	flops		tot. vol.
Tensor	max	avg		max	avg		max	avg		max	avg		max	avg		max	avg	
Enron	1.06	1.00	1.06	0.72	0.70	0.65	0.82	0.79	0.81	1.01	0.97	1.07	0.74	0.71	0.65	0.82	0.79	0.81
Flickr	1.23	1.00	1.11	0.75	0.93	0.84	0.86	0.95	0.85	1.10	1.00	0.46	0.81	0.94	0.85	0.88	0.95	0.84
Movies-amazon	1.04	1.00	1.09	0.87	0.92	0.83	0.94	0.95	0.94	0.87	0.72	0.94	0.85	0.92	0.76	0.86	0.83	0.84
Nell-1	1.18	1.00	1.24	0.82	0.84	0.91	0.95	0.89	1.04	1.05	0.95	1.21	0.84	0.86	0.92	0.91	0.89	1.04
Nell-2	1.01	1.02	1.37	0.74	0.73	0.91	0.84	0.83	1.15	0.81	0.69	0.96	0.74	0.71	0.66	0.77	0.70	0.81
Yelp	1.01	1.00	1.10	0.71	0.85	0.68	0.83	0.92	0.82	0.81	0.71	0.97	0.73	0.87	0.74	0.76	0.80	0.81
Avg.	**1.09**	**1.00**	**1.16**	**0.77**	**0.82**	**0.80**	**0.87**	**0.89**	**0.93**	**0.93**	**0.83**	**0.90**	**0.78**	**0.83**	**0.76**	**0.83**	**0.82**	**0.86**

IFS^{S}, FNA^{S} and impFG^{S} denote the improvement schemes denoted at the bottom of Table 2

TABLE 7: Performance comparison in terms of parallel runtimes on $P = 512$ processors for CSF-D with $R = 64$.

	FG (in ms)			impFG^{D} (normalized)		
	MTTKRP		CP-ALS	MTTKRP		CP-ALS
Tensor	comp	tot		comp	tot	
Enron	131.2	252.7	266.2	0.73	0.81	0.82
Flickr	252.0	443.6	631.5	0.81	0.84	0.93
Movies-amazon	29.5	136.1	154.2	0.90	0.88	0.89
Nell-1	521.6	1,393.8	1,746.0	0.92	0.83	0.90
Nell-2	155.7	296.2	325.6	0.71	0.78	0.78
Yelp	502.7	851.8	1,027.2	0.69	0.73	0.71
Avg.	**184.78**	**418.40**	**497.36**	**0.79**	**0.81**	**0.83**

5.3.2 *Results of CSF-D experiments*

The performance comparisons in the previous section regarding the incremental performance improvement attained by the optimization schemes in Section 4, the effect of different R values as well as the effect of α value apply to the CSF-D scheme as well. In order to present a wider spectrum of results, here we study the effect of applying the CSF-S-based and the CSF-D-based optimization schemes on the computational and total volume cost metrics for the CSF-D-based MTTKRP. We perform this in order to justify proposing a separate optimization techniques for CSF-D. Here and hereafter, the superscripts 'S' and 'D' will be used to distinguish the CSF-S-based and CSF-D-based optimization schemes on the FG method. All the experiments in this

section utilize the CSF-D-based MTTKRP regardless of the type of optimization scheme applied.

Table 6 compares impFG$^{\mathrm{D}}$ against impFG$^{\mathrm{S}}$ in terms of computational and total communication volume statistics normalized with respect to those of FG. In the table, these statistics are detailed along the longest mode, remaining $N-1$ modes, and all modes. As seen in the table, on average, utilizing the IFS scheme and fiber-net augmentation for the longest mode only (CSF-S-based improvements) in the impFG$^{\mathrm{S}}$ method improves the maximum and average flop counts by 13.0% and 11.0%, respectively, compared to the FG method in all modes. Utilizing the IFS scheme and fiber-net augmentation for both longest and second longest modes (CSF-D-based improvements) in the impFG$^{\mathrm{D}}$ method improves the maximum and average flop counts by 4.6% and 7.8%, respectively, compared to the impFG$^{\mathrm{S}}$ method in all modes. Although the maximum and average flop counts along $N-1$ modes are almost the same for both impFG$^{\mathrm{S}}$ and impFG$^{\mathrm{D}}$, the relative improvements in all modes come from improving the maximum and average flop counts along the longest mode by 14.6% and 17.0%, respectively. As also seen in the table, the impFG$^{\mathrm{D}}$ method respectively achieves 17.0% and 18.0% improvements in maximum and average flop counts compared to the baseline FG method.

Comparison in terms of total volume metric shows a similar behavior as the comparison in terms of computational cost metrics discussed above. That is, utilizing the CSF-S-based improvements for the CSF-D-based MTTKRP in the impFG$^{\mathrm{S}}$ method incurs an increase of 16.0%, on average, in total volume along the longest mode compared to FG. On the other hand, impFG$^{\mathrm{D}}$ achieves an improvement of 10.0% in total volume during the MTTKRP along the longest mode as a result of augmenting the fiber nets along the second longest mode. The effect of this improvement can be seen in the table as 7.5% improvement in terms of total volume of impFG$^{\mathrm{D}}$ compared to impFG$^{\mathrm{S}}$ along all modes. As seen in the table, impFG$^{\mathrm{D}}$ achieves 14% improvement in terms of total volume compared to FG along all modes.

Table 7 shows how the performance improvement achieved by the proposed impFG$^{\mathrm{D}}$ method in computational and total volume metrics lead to improvements in actual parallel runtimes. In the table, the values under FG are actual runtimes, while those under impFG$^{\mathrm{D}}$ are normalized with respect to those of FG. Comparing "max flop" column of impFG$^{\mathrm{D}}$ in Table 6 (along all modes) with the "MTTKRP comp" column of impFG$^{\mathrm{D}}$ in Table 7 shows the close correlation between the amount of improvement in maximum flop count and the amount of improvement in parallel MTTKRP computation time. That is, the 17.0% improvement attained by impFG$^{\mathrm{D}}$ in maximum flop count reflects as approximately 21.0% improvement in parallel MTTKRP computation time, on average. As also seen in the table, the CP-ALS runtime improves, on average, by 17.0% as a result of applying the optimization schemes for CSF-D.

Figures 3a and 3b respectively display the strong scaling curves of impFG vs FG and impFG$^{\mathrm{D}}$ vs FG. Note that in 3a the CSF-S scheme is utilized for computing the MTTKRP, whereas in 3b the CSF-D is utilized instead. The curves display parallel runtimes of the CP-ALS algorithm on $P = 128$ up to $P = 1024$ processors with $R = 64$. As seen in the figure, impFG increases the scalability of FG for both CSF-S and CSF-D schemes on all tensors. The relative scalability between impFG and FG for CSF-S and CSF-D schemes shows similar trend for all tensors, except for `Nell-2` which favors the CSF-D scheme. That is, for `Nell-2`, although impFG and FG show very close scaling performance for CSF-S scheme, impFG$^{\mathrm{D}}$ displays significantly better performance than FG for CSF-D. A grasp of the actual runtime values can be taken from comparing the values of the CP-ALS column in Table 3 for $R = 64$ with the same column values in Table 7.

6 Related Work

In the literature, there are various CP-ALS implementations adopting different parallelism paradigms [13, 17, 28–32]. On distributed-memory systems, DMS [17] is the most commonly-used implementation. DMS adopts a multi-dimensional cartesian partitioning approach, however it does not support different partitioning techniques coming in more irregular forms.

To devise intelligent tensor partitioning models, sparse matrix partitioning community adapted well-known sparse matrix partitioning models for tensors. These models came in different granularities: coarse-grain [9], multi-dimensional cartesian model [11], fine-grain [9] and medium-grain [12]. The multidimensional cartesian model is derived from the hypergraph model proposed earlier for 2D checkerboard partitioning of sparse matrices [33, 34]. The fine-grain model can be considered as an extension of the fine-grain hypergraph model for 2D nonzero-based sparse matrix partitioning [33, 35, 36] to multi-dimensional tensor partitioning. The recent general medium-grain model [12] can be considered as an extension of the medium-grain model for 2D sparse matrix partitioning [37] to tensors. Among these, fine-grain model achieves the minimum communication volume as well as the best computational balance on the tensor nonzeros assigned to processors.

Sparse tensor storage formats include COO (coordinate) [9], CSF (compressed sparse fiber) [14], and HiCOO (hierarchical coordinate) [15]. COO corresponds to a list of tensor nonzeros, where each nonzero represented by a list of indices and the value. Besides its simplicity, COO stores repeated indices (within a fiber or a slice) redundantly and the MTTKRP on COO performs redundant flops (see section 3.1). CSF and HiCOO are motivated by reducing the storage used by COO, due to the limited memory in shared-memory architectures. While the (sequential) MTTKRP algorithm on HiCOO has the same flop count as that on COO, the algorithm on CSF achieves a much better flop count compared to those on COO-based formats. This improvement in the flop count makes CSF the most favorable alternative for the local MTTRKP computation on distributed-memory systems.

7 Conclusion

We proposed two improvement schemes to the existing fine-grain hypergraph model in order to address the deficiencies introduced by utilizing the CSF-oriented MTTKRP for distributed-memory CP-ALS computation. The improvement schemes target at achieving true computational load balancing among processors, thus leading to faster parallel

runtime. The improvement schemes do not deteriorate the communication overhead. In fact, the total volume overhead decreases as a result of better load balancing, while the latency overhead stays the same as that of the FG method. On average, applying the proposed improvement schemes to the FG method improves the parallel MTTKRP computation time and the overall CP-ALS time respectively by 22.0% and 14.0% on 512 processors, and with similar percentages on 128, 256 and 1024. As future work, we plan to extend the proposed true balancing method for other nonzero-based tensor partitioning models.

Acknowledgement

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under project EEEAG-116E043. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

[1] E. Acar, C. Aykut-Bingol, H. Bingol, R. Bro, and B. Yener, "Multiway analysis of epilepsy tensors," *Bioinformatics*, vol. 23, no. 13, pp. i10–i18, 07 2007.

[2] I. Davidson, S. Gilpin, O. Carmichael, and P. Walker, "Network discovery via constrained tensor analysis of fMRI data," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13, New York, NY, USA, 2013, pp. 194–202.

[3] E. Acar and B. Yener, "Unsupervised multiway data analysis: A literature survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 1, pp. 6–20, 2009.

[4] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[5] K. R. Murphy, C. A. Stedmon, D. Graeber, and R. Bro, "Fluorescence spectroscopy and multi-way techniques. PARAFAC," *Analytical Methods*, vol. 5, no. 23, pp. 6557–6566, 2013.

[6] K. Maruhashi, F. Guo, and C. Faloutsos, "MultiAspectForensics: Pattern mining on large-scale heterogeneous networks with tensor analysis," in *2011 International Conference on Advances in Social Networks Analysis and Mining*, 2011, pp. 203–210.

[7] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[8] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *ACM Trans. Knowl. Discov. Data*, vol. 5, no. 2, Feb. 2011.

[9] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2015, pp. 1–11.

[10] O. Kaya and B. Uçar, "Parallel CANDE-COMP/PARAFAC decomposition of sparse tensors using dimension trees," *SIAM Journal on Scientific Computing*, vol. 40, no. 1, pp. C99–C130, 2018.

[11] S. Acer, T. Torun, and C. Aykanat, "Improving medium-grain partitioning for scalable sparse tensor decomposition," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2814–2825, Dec 2018.

[12] M. O. Karsavuran, S. Acer, and C. Aykanat, "Partitioning models for general medium-grain parallel sparse tensor decomposition," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 147–159, 2021.

[13] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "SPLATT: Efficient and parallel sparse tensor-matrix multiplication," in *2015 IEEE International Parallel and Distributed Processing Symposium*, May 2015, pp. 61–70.

[14] S. Smith and G. Karypis, "Tensor-matrix products with a compressed sparse tensor," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, ser. IA3 '15. New York, NY, USA: ACM, 2015, pp. 5:1–5:7.

[15] J. Li, J. Sun, and R. Vuduc, "HiCOO: Hierarchical storage of sparse tensors," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 238–252.

[16] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[17] S. Smith and G. Karypis, "A medium-grained algorithm for sparse tensor factorization," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 902–911.

[18] U. V. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, July 1999.

[19] J. Li, Y. Ma, X. Wu, A. Li, and K. Barker, "PASTA: a parallel sparse tensor algorithm benchmark suite," *CCF Transactions on High Performance Computing*, vol. 1, no. 2, pp. 111–130, 2019.

[20] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 69–79, 1999.

[21] B. Vastenhouw and R. H. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM review*, vol. 47, no. 1, pp. 67–95, 2005.

[22] J. Shetty and J. Adibi, "The enron email dataset database schema and brief statistical report," *Information sciences institute technical report, University of Southern California*, vol. 4, 2004.

[23] S. Fernandes, H. Fanaee-T, and J. Gama, "Tensor decomposition for analysing time-evolving social networks: an overview," *Artificial Intelligence Review*, pp.

1–26, 2020.
[24] O. Görlitz, S. Sizov, and S. Staab, "PINTS: peer-to-peer infrastructure for tagging systems." in *IPTPS*, 2008, p. 19.
[25] P. Bhargava, T. Phan, J. Zhou, and J. Lee, "Who, what, when, and where: Multi-dimensional collaborative recommendations using tensor factorization on sparse user-generated data," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2015, p. 130?140.
[26] J. McAuley and J. Leskovec, "Hidden factors and hidden topics: understanding rating dimensions with review text," in *Proceedings of the 7th ACM conference on Recommender systems*, 2013, pp. 165–172.
[27] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning." in *AAAI*, vol. 5, 2010, p. 3.
[28] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 316–324.
[29] B. W. Bader and T. G. Kolda, "Efficient MATLAB computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 205–231, 2007.
[30] J. H. Choi and S. Vishwanathan, "DFacTo: Distributed factorization of tensors," in *Advances in Neural Information Processing Systems*, 2014, pp. 1296–1304.
[31] E. T. Phipps and T. G. Kolda, "Software for sparse tensor decomposition on emerging computing architectures," *SIAM Journal on Scientific Computing*, vol. 41, no. 3, pp. C269–C290, 2019.
[32] J. Li, Y. Ma, and R. Vuduc, "ParTI! : A parallel tensor infrastructure for multicore CPUs and GPUs," Oct 2018.
[33] Ü. V. Çatalyürek, C. Aykanat, and B. Uçar, "On two-dimensional sparse matrix partitioning: Models, methods, and a recipe," *SIAM Journal on Scientific Computing*, vol. 32, no. 2, pp. 656–683, 2010.
[34] U. Catalyurek and C. Aykanat, "A hypergraph-partitioning approach for coarse-grain decomposition," in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, 2001, pp. 28–28.
[35] Ü. V. Çatalyürek and C. Aykanat, "A fine-grain hypergraph model for 2D decomposition of sparse matrices." in *IPDPS*, vol. 1, 2001, p. 118.
[36] B. Uçar and C. Aykanat, "Minimizing communication cost in fine-grain partitioning of sparse matrices," in *International Symposium on Computer and Information Sciences*. Springer, 2003, pp. 926–933.
[37] D. M. Pelt and R. H. Bisseling, "A medium-grain method for fast 2D bipartitioning of sparse matrices," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 2014, pp. 529–539.

Nabil Abubaker received the BS degree from An-Najah National University, Nablus, Palestine, and the MS degree from Bilkent University, Ankara, Turkey where he is currently pursuing his PhD degree, all in Computer Engineering. His research interests include parallel and scientific computing, with focus on graph/hypergraph-partitioning-based decompositions for irregular applications.

Seher Acer received her BS, MS and PhD degrees in Computer Engineering from Bilkent University, Turkey. She is currently a post-doctoral researcher at Center for Computing Research, Sandia National Laboratories, Albuquerque, New Mexico, USA. Her research interests include parallel computing and combinatorial scientific computing with a focus on partitioning sparse irregular computations.

Cevdet Aykanat received the BS and MS degrees from Middle East Technical University, Turkey, both in electrical engineering, and the PhD degree from Ohio State University, Columbus, in electrical and computer engineering. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Turkey, where he is currently a professor. His research interests mainly include parallel computing and its combinatorial aspects. He is the recipient of the 1995 Investigator Award of The Scientific and Technological Research Council of Turkey and 2007 Parlar Science Award. He has served as an Associate Editor of IEEE Transactions of Parallel and Distributed Systems between 2009 and 2013.