Partitioning Models for General Medium-Grain Parallel Sparse Tensor Decomposition

M. Ozan Karsavuran, Seher Acer, and Cevdet Aykanat

Abstract—The focus of this article is efficient parallelization of the canonical polyadic decomposition algorithm utilizing the alternating least squares method for sparse tensors on distributed-memory architectures. We propose a hypergraph model for general medium-grain partitioning which does not enforce any topological constraint on the partitioning. The proposed model is based on splitting the given tensor into nonzero-disjoint component tensors. Then a mode-dependent coarse-grain hypergraph is constructed for each component tensor. A net amalgamation operation is proposed to form a composite medium-grain hypergraph from these mode-dependent coarse-grain hypergraphs to correctly encapsulate the minimization of the communication volume. We propose a heuristic which splits the nonzeros of dense slices to obtain sparse slices in component tensors. So we partially attain slice coherency at (sub)slice level since partitioning is performed on (sub)slices instead of individual nonzeros. We also utilize the well-known recursive-bipartitioning framework to improve the quality of the splitting heuristic. Finally, we propose a medium-grain tripartite graph model with the aim of a faster partitioning at the expense of increasing the total communication volume. Parallel experiments conducted on 10 real-world tensors on up to 1024 processors confirm the validity of the proposed hypergraph and graph models.

Index Terms—sparse tensor, tensor decomposition, canonical polyadic decomposition, communication cost, communication volume, medium-grain partitioning,recursive bipartitioning, hypergraph partitioning,graph partitioning

1 Introduction

The canonical polyadic decomposition (CPD) is a common tensor decomposition method [1], [2]. It decomposes a given tensor into a sum of vector outer-products. Many applications from different domains utilize CPD such as machine learning [2], [3], [4], recommender systems [5], [6], among many others. The alternating least squares (ALS) method [7] is a common method to compute the CPD. We refer the reader to [2] for the details of tensor decomposition methods and their applications.

The bottleneck operation in CPD-ALS is the Matricized Tensor Times Khatri-Rao Product (MTTKRP) operation performed for each mode. In distributed-memory parallel CPD-ALS, each MTTKRP operation incurs a high volume of irregular communication for sparse tensors. The communication requirement of MTTKRP scales with both tensor size and decomposition rank. That is, CPD-ALS becomes bandwidth bound as rank increases, and reducing the communication volume is crucial to attain high performance.

1.1 Related Work

There are many shared- and distributed-memory parallel CPD-ALS algorithms and partitioning methods [8], [9], [10], [11], [12], [13], [14], [15], [16]. Here we focus on MPI-based distributed-memory parallel models and methods.

Kaya and Uçar [10] propose parallel CPD-ALS algorithms based on coarse- and fine-grain partitioning of sparse tensors. In the coarse-grain algorithm, an atomic task is defined as the MTTKRP operation associated with all nonzeros within a slice to compute the respective factor-matrix row. In the fine-grain algorithm, an atomic task is defined as the operation associated with a nonzero of the tensor.

Smith and Karypis [11] propose a parallel CPD-ALS algorithm based on cartesian partitioning of sparse tensors. Cartesian partitioning is obtained through applying block partitioning on each mode, which is randomly permuted beforehand to maintain balance on the number of tensor nonzeros assigned to processors, hence their computational loads. Cartesian partitioning has the nice property of attaining a natural upper-bound on the number of messages.

There are two recent works [10], [12] which propose intelligent partitioning models for improving the performance of the CPD-ALS algorithm. These models exploit the sparsity pattern of the tensor for reducing communication overhead while maintaining computational load balance during the MTTKRP operations.

Kaya and Uçar [10] propose two distinct hypergraph models for coarse-grain and fine-grain tensor partitioning. The coarse-grain hypergraph model contains one vertex and one net for each slice in each mode. Vertices are assigned M weights for an M-mode tensor. The mth weight of a vertex representing a slice along mode m is set to the number of nonzeros in that slice, whereas all other weights are set to zero. An M-constraint formulation is required to achieve computational balance during the MTTKRP operations along each mode. Partitioning of this model may incur nonzero replication, where a nonzero is replicated at most M times. The

- M. Ozan Karsavuran and Cevdet Aykanat are with the Computer Engineering Department, Bilkent University, Turkey 06800. E-mail: {ozan.karsavuran, aykanat}@cs.bilkent.edu.tr.
- Seher Acer is with the Center for Computing Research, Sandia National Laboratories, Albuquerque, NM 87123. E-mail: sacer@sandia.gov.

Manuscript received 8 Oct. 2019; revised 15 June 2020; accepted 24 July 2020. Date of publication 4 Aug. 2020; date of current version 14 Aug. 2020. (Corresponding author: Cevdet Aykanat.) Recommended for acceptance by K. Madduri. Digital Object Identifier no. 10.1109/TPDS.2020.3012624

fine-grain hypergraph model contains one vertex for each nonzero and one net for each slice and it can be considered as an extension of fine-grain hypergraph model for 2D nonzero-based sparse matrix partitioning [17], [18], [19] to multi-dimensional tensor partitioning. Since individual nonzeros are partitioned, vertices are assigned unit weight, so a single-constraint formulation suffices to achieve computational balance during MTTKRP operations along all modes. In both models, nets are assigned a cost of the rank of the decomposition. In this way, the partitioning objective of minimizing the cutsize encodes the minimization of total communication volume. The fine-grain model is reported to perform much better than the coarse-grain model in [10].

Acer *et al.* [12] propose a hypergraph model for cartesian partitioning of sparse tensors. This model consists of M phases for partitioning an M-mode tensor. In each partitioning phase, the corresponding hypergraph contains one vertex for each slice of the current mode, and one net for each slice of each other mode. Slices partitioned in earlier phases incur different nets for their subslices in the following phases. Those slices also incur multiple vertex weights, as many as the product of the number of parts in the earlier phases. In each phase, multiple weights of a vertex represent the distribution of the nonzeros of the respective slice among the partitioned slices in earlier phases. So this model necessitates multi-constraint partitioning at each phase except the first phase in order to balance the number of nonzeros in the parts of the resulting cartesian partition. Nets are assigned a cost of the rank of the decomposition. This model is derived from the hypergraph model proposed earlier for 2D cartesian (known as checkerboard) partitioning of sparse matrix [19], [20]. The hypergraph model for tensor partitioning mainly differs from matrix partitioning by nets incurred by subslices which is not the case in 2D matrix partitioning.

1.2 Contributions

We propose a medium-grain hypergraph model to address the following drawbacks of the hypergraph models proposed for coarse-grain [10], fine-grain [10] and cartesian [12] partitioning methods.

- Although the coarse-grain method attains slice coherency in each mode, it suffers from the restriction of assigning all nonzeros in possibly dense slices to the same processor.
- Although the fine-grain model attains very good performance in reducing total communication volume, it may incur a large number of messages.
- The fine-grain model incurs large hypergraphs which constitute hard partitioning instances for current hypergraph partitioning (HP) tools as also reported in [10].
- Although the cartesian partitioning method utilizes an upper bound on the number of messages, it suffers from a large number of constraints during partitioning which may limit the solution space.

The proposed model is based on splitting the given tensor into nonzero-disjoint component tensors. Then a mode-dependent coarse-grain hypergraph is constructed for each component tensor. This coarse-grain hypergraph model is different than the one in [10] since in each of these hypergraphs, each slice along one of the modes is represented with both a vertex and a net, whereas each slice along the other modes is represented with only a net. Thus this model avoids the nonzero replication problem of the original model. A net amalgamation operation is proposed to form a composite medium-grain hypergraph from these mode-dependent coarse-grain hypergraphs. This operation enables the hypergraph model to correctly encapsulate the minimization of total communication volume.

We propose a heuristic which splits the nonzeros of dense slices to obtain sparse slices in component tensors thus avoiding the above-mentioned drawback of the coarse-grain model. So we partially attain slice coherency at (sub)slice level since partitioning is performed on (sub)slices instead of individual nonzeros of the fine-grain model. In this way, the proposed model avoids the above-mentioned drawbacks of the fine-grain model incurring a large number of messages and incurring large hypergraphs.

We also utilize the well-known recursive-bipartitioning (RB) framework to improve the quality of the splitting heuristic. Finally, we propose a medium-grain tripartite graph model with the aim of a faster partitioning at the expense of increasing the total communication volume. Parallel experiments conducted on 10 real-world tensors on up to 1024 processors confirm the validity of the proposed hypergraph and graph models.

We should mention here that parallelization based on random cartesian partitioning [11] as well as the one based on HP [12] utilize the phrase "medium-grain" partitioning for their methods, since they remain between the coarse- and fine-grain methods. On the other hand, the medium-grain method proposed in this work introduces a much more general framework between the coarse- and fine-grain methods as it does not enforce any topological constraint on the partitioning. The medium-grain method proposed in this work is an extension of the medium-grain matrix partitioning proposed for parallel sparse matrix-vector multiply (SpMV) by Pelt and Bisseling [21].

The rest of the paper is organized as follows: Section 2 provides background information. In Sections 3 and 4, we propose medium-grain hypergraph and graph models for tensor partitioning, respectively. Section 5 discusses experimental results. Finally, Section 6 concludes the paper.

2 Background Information

2.1 Tensor Notation

We denote tensors and matrices respectively by calligraphic ($\mathcal{X}$) and bold uppercase ($\mathbf{A}$) letters. To refer to a varying index, we use a colon as in Matlab notation, e.g., $\mathbf{A}(i,:)$ denotes the ith row of $\mathbf{A}$.

A tensor with M dimensions is called an M-mode tensor and mode m refers to the mth dimension. Without loss of generality we assume $\mathcal{X}$ is a 3-mode tensor of size $I \times J \times K$. $\mathcal{X}(i,j,k)$ denotes the tensor element with indices i, j, and k. Subtensors obtained by keeping one and two indices constant are called slices and fibers, respectively. Slices are called mode-m slice where m denotes the constant mode, i.e., $\mathcal{X}(i,:,:)$ is the ith mode-1 slice, $\mathcal{X}(:,j,:)$ is the jth mode-2 slice, and $\mathcal{X}(:,:,k)$ is the kth mode-3 slice. These are also called horizontal, lateral, and frontal slices, respectively.

2.2 Canonical Polyadic Decomposition

Algorithm 1 displays the CPD-ALS algorithm that finds a rank-R decomposition of a 3-mode tensor as a sum of R outer-products of three vectors. These R column-vectors constitute factor matrices. Then, the CPD of $\mathcal{X}$ is written in short as $\mathcal{X} \approx [\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$.

Generally, MTTKRP operations in lines 3, 5, and 7, are the bottleneck steps of the CPD-ALS algorithm due to large matrices $\mathbf{X}_{(1)}$, $\mathbf{X}_{(2)}$, and $\mathbf{X}_{(3)}$, respectively. Here "$*$" denotes the Hadamard product which is element-wise multiplication of two matrices, whereas "$\odot$" denotes the Khatri–Rao product.

Algorithm 1. CPD-ALS($\mathcal{X}$)

```
1: Randomly initialize factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$
2: while not converged do
3:     $\mathbf{A} \leftarrow \mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{-1}$
4:     Normalize columns of $\mathbf{A}$ into $\lambda$
5:     $\mathbf{B} \leftarrow \mathbf{X}_{(2)}(\mathbf{C}\odot\mathbf{A})(\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A})^{-1}$
6:     Normalize columns of $\mathbf{B}$ into $\lambda$
7:     $\mathbf{C} \leftarrow \mathbf{X}_{(3)}(\mathbf{A}\odot\mathbf{B})(\mathbf{A}^T\mathbf{A} * \mathbf{B}^T\mathbf{B})^{-1}$
8:     Normalize columns of $\mathbf{C}$ into $\lambda$
9: return $[\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$
```

2.3 Parallel Medium-Grain CPD-ALS Algorithm

The parallel CPD-ALS algorithm utilizes a nonzero-based tensor partitioning and the rowwise partitioning of the factor matrices in a distributed-memory setting. Let $\mathcal{X}_k$ denote the local nonzeros assigned to processor p_k. Let $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$ denote the local factor-matrix rows assigned to processor p_k. A factor-matrix row is said to be shared if more than one processor produce a partial result for it. A shared matrix row is said to be local for the sharing processor it is assigned to, whereas it is said to be nonlocal for the other sharing processors. The number of processors that share a matrix row is said to be the connectivity of that row.

Algorithm 2 displays the parallel CPD-ALS algorithm utilizing the above mentioned partitioning scheme. Lines 3, 8, and 13 correspond to local MTTKRP operations. The MTTKRP operation for the ith row of factor matrix $\hat{\mathbf{A}}$ can be rewritten as $\hat{\mathbf{A}}(i,:) = \sum_{\mathcal{X}(i,j,k)\neq 0} \mathcal{X}(i,j,k)(\mathbf{B}(j,:) * \mathbf{C}(k,:))$. In these lines, $\hat{\mathbf{A}}_k$, $\hat{\mathbf{B}}_k$, and $\hat{\mathbf{C}}_k$ denote intermediate results of the corresponding local factor matrices. $\mathbf{A}'_k$, which appears as an input in a local MTTKRP, refers to $\mathbf{A}_k$ augmented with the nonlocal rows that are needed by the local MTTKRP. $\hat{\mathbf{A}}'_k$, which appears as an output in a local MTTKRP, refers to $\hat{\mathbf{A}}_k$ augmented with the nonlocal rows for which the local MTTKRP produces partial results. The same notation applies to $\mathbf{B}'_k$ and $\mathbf{C}'_k$ matrices.

As seen in Algorithm 2, nonzero-partitioning-based parallel CPD-ALS requires communication both before and after the local MTTKRP operation in each mode. After the distributed MTTKRP operation along each mode, reduce operations (i.e., lines 4, 9, and 14) are performed on the local partial results to compute $\hat{\mathbf{A}}_k$, $\hat{\mathbf{B}}_k$, and $\hat{\mathbf{C}}_k$. Then, local pseudoinverse operations and column normalization operations are performed on the local factor-matrices. Then, expand operations (i.e., lines 7, 12, and 17) are performed on the normalized shared factor matrices for the sake of the MTTKRP operations along the other modes.

Reduce and expand operations before and after each distributed MTTKRP are the dual of each other. Consider a shared factor-matrix row $\mathbf{A}(i,:)$ assigned to a sharing processor p_k. For the reduce operation, all processors except p_k in the connectivity set of $\mathbf{A}(i,:)$ send their partial results for nonlocal $\hat{\mathbf{A}}'(i,:)$ to p_k. For the expand operation, p_k sends its local normalized $\mathbf{A}_k(i,:)$ to all processors in the connectivity set of $\mathbf{A}(i,:)$. Since each shared factor-matrix row is assigned to one of the sharing processors, the total volume of communication along each mode is equal to $2R$ times the sum of the "connectivity -1' values of the shared factor-matrix rows along that mode.

Algorithm 2. Parallel CPD-ALS ($\mathcal{X}$)

```
1: Randomly initialize factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$
2: while not converged do
3:     $\hat{\mathbf{A}}'_k(i,:) \leftarrow \sum_{\mathcal{X}_k(i,j,k)\neq 0} \mathcal{X}_k(i,j,k)(\mathbf{B}'_k(j,:) * \mathbf{C}'_k(k,:))$
4:     Sparse REDUCE on shared $\hat{\mathbf{A}}$-matrix rows
         ▷ SEND nonlocal $\hat{\mathbf{A}}'_k$ rows, RECV local-shared $\hat{\mathbf{A}}_k$ rows to form $\hat{\mathbf{A}}_k$
5:     $\mathbf{A}_k \leftarrow \hat{\mathbf{A}}_k(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{-1}$ and norm. cols. of $\mathbf{A}$ into $\lambda$
6:     ALL-REDUCE to compute $\mathbf{A}^T\mathbf{A}$
7:     Sparse EXPAND on shared $\mathbf{A}$-matrix rows
         ▷ SEND local-shared $\mathbf{A}_k$ rows, RECV nonlocal $\mathbf{A}'_k$ rows
8:     $\hat{\mathbf{B}}'_k(j,:) \leftarrow \sum_{\mathcal{X}_k(i,j,k)\neq 0} \mathcal{X}_k(i,j,k)(\mathbf{A}'_k(i,:) * \mathbf{C}'_k(k,:))$
9:     Sparse REDUCE on shared $\hat{\mathbf{B}}$-matrix rows
         ▷ SEND nonlocal $\hat{\mathbf{B}}'_k$ rows, RECV local-shared $\hat{\mathbf{B}}_k$ rows to form $\hat{\mathbf{B}}_k$
10:    $\mathbf{B}_k \leftarrow \hat{\mathbf{B}}_k(\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A})^{-1}$ and norm. cols. of $\mathbf{B}$ into $\lambda$
11:    ALL-REDUCE to compute $\mathbf{B}^T\mathbf{B}$
12:    Sparse EXPAND on shared $\mathbf{B}$-matrix rows
         ▷ SEND local-shared $\mathbf{B}_k$ rows, RECV nonlocal $\mathbf{B}'_k$ rows
13:    $\hat{\mathbf{C}}'_k(k,:) \leftarrow \sum_{\mathcal{X}_k(i,j,k)\neq 0} \mathcal{X}_k(i,j,k)(\mathbf{A}'_k(i,:) * \mathbf{B}'_k(j,:))$
14:    Sparse REDUCE on shared $\hat{\mathbf{C}}$-matrix rows
         ▷ SEND nonlocal $\hat{\mathbf{C}}'_k$ rows, RECV local-shared $\hat{\mathbf{C}}_k$ rows to form $\hat{\mathbf{C}}_k$
15:    $\mathbf{C}_k \leftarrow \hat{\mathbf{C}}_k(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A})^{-1}$ and norm. cols. of $\mathbf{C}$ into $\lambda$
16:    ALL-REDUCE to compute $\mathbf{C}^T\mathbf{C}$
17:    Sparse EXPAND on shared $\mathbf{C}$-matrix rows
         ▷ SEND local-shared $\mathbf{C}_k$ rows, RECV nonlocal $\mathbf{C}'_k$ rows
18: return $[\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$
```

With the above-mentioned parallelization scheme, factor-matrix-transpose and factor-matrix multiplication operations (i.e., $\mathbf{A}^T\mathbf{A}$, $\mathbf{B}^T\mathbf{B}$, and $\mathbf{C}^T\mathbf{C}$) can easily and efficiently be parallelized so that they incur all-reduce type global communication on $R\times R$ local dense matrix results. Similarly, column normalization operations can also be parallelized through performing all-reduce type operations on R local matrix normalization results. Note that in the current parallelization, Hadamard products as well as pseudoinverse operations associated with these resulting $R\times R$ matrices (e.g., $(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{-1}$) are performed sequentially and redundantly by each processor.

The communication volume incurred by dense matrix-matrix multiplication and column normalization operations scale only with R^2 and R, respectively, whereas they do not scale with tensor size (i.e., I, J, K, and $nnz(\mathcal{X})$). However, communication volumes incurred by the reduce and expand operations scale with both the tensor size and rank R. So reducing the communication volume overhead incurred by sparse communication operations is crucial for scaling parallel CPD-ALS algorithm.

2.4 Hypergraph Partitioning (HP) Problem

A hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{N})$ is defined as the set $\mathcal{V}$ of vertices and the set $\mathcal{N}$ of nets. Each net n connects a subset of vertices denoted by $Pins(n)$. The set of nets that connect vertex v is denoted by $Nets(v)$. Each vertex v is assigned a weight $w(v)$ and each net n is assigned a cost $c(n)$.

Let $\Pi=\{\mathcal{V}_1,\mathcal{V}_2,\ldots,\mathcal{V}_P\}$ denote a P-way vertex partition of $\mathcal{H}$. In Π, the weight of part $\mathcal{V}_k$ is defined as $W(\mathcal{V}_k)=\sum_{v\in\mathcal{V}_k} w(v)$. Π is said to satisfy the partitioning constraint if $W(\mathcal{V}_k)\le W_{avg}(1+\epsilon)$ for each part $\mathcal{V}_k$ in Π, for a given maximum allowed imbalance ratio ϵ. Here $W_{avg}=W_{tot}/P$, where $W_{tot}=\sum_{k=1}^{P} W(\mathcal{V}_k)$.

In a given partition Π, net n is said to connect part $\mathcal{V}_k$ if it connects at least one vertex in $\mathcal{V}_k$. The connectivity set of n is defined as the set of parts connected by n. The connectivity of n, $con(n)$, denotes the number of parts connected by n. n is said to be cut if it connects more than one part, i.e., $con(n)>1$, and uncut otherwise. Then the connectivity cutsize is defined as $cutsize(\Pi)=\sum_{n\in\mathcal{N}}(con(n)-1)c(n)$. In the HP problem, the partitioning objective is to minimize the cutsize while maintaining the partitioning constraint.

3 Medium-Grain Hypergraph Model

3.1 Splitting Framework

The given sparse tensor $\mathcal{X}$ is split into three nonzero-disjoint component tensors as follows:

$$\mathcal{X}=\mathcal{X}^H+\mathcal{X}^L+\mathcal{X}^F. \tag{1}$$

In Eq. (1), all three component tensors $\mathcal{X}^H$, $\mathcal{X}^L$, and $\mathcal{X}^F$ have the same size of $I\times J\times K$ as the original tensor $\mathcal{X}$. Here, $\mathcal{X}^H$, $\mathcal{X}^L$, and $\mathcal{X}^F$ are considered as composed of horizontal, lateral, and frontal slices, respectively. In this splitting, we enforce each nonzero to be assigned to the respective slice of one of the component tensors according to the selection of one of the three slices that contains it. That is, nonzero $\mathcal{X}(i,j,k)$ is placed in either the ith horizontal slice of $\mathcal{X}^H$ or the jth lateral slice of $\mathcal{X}^L$ or the kth frontal slice of $\mathcal{X}^F$. This placement ensures Eq. (1).

Then a mode-dependent coarse-grain hypergraph (CGH) is constructed for each component tensor as follows: In each of these hypergraphs, slices along one of the modes are represented as vertices and nets, whereas the slices along the other two modes are represented as nets. Vertices are assigned weights equal to the number of nonzeros in the respective slices, whereas nets are assigned a cost of R. The topology of each component hypergraph is described below.

For mode-1, CGH $\mathcal{H}^1_{CG}(\mathcal{X}^H)=(\mathcal{V}^H,\mathcal{N}^H\cup\mathcal{N}^L\cup\mathcal{N}^F)$ contains one vertex v_i^H for each horizontal slice $\mathcal{X}^H(i,:,:)$. It contains nets n_i^H, n_j^L, and n_k^F for each horizontal slice $\mathcal{X}^H(i,:,:)$, lateral slice $\mathcal{X}^H(:,j,:)$, and frontal slice $\mathcal{X}^H(:,:,k)$, respectively. Net n_j^L connects vertex v_i^H if intersection of $\mathcal{X}^H(i,:,:)$ and $\mathcal{X}^H(:,j,:)$ is a nonzero fiber. Similarly, net n_k^F connects vertex v_i^H if intersection of $\mathcal{X}^H(i,:,:)$ and $\mathcal{X}^H(:,:,k)$ is a nonzero fiber. That is,

$$\begin{aligned}
&Pins(n_i^H)=\{v_i^H\},\\
&Pins(n_j^L)=\{v_i^H:\exists\mathcal{X}^H(i,j,k)\in\mathcal{X}^H(i,:,:)\cap\mathcal{X}^H(:,j,:)\},\\
&Pins(n_k^F)=\{v_i^H:\exists\mathcal{X}^H(i,j,k)\in\mathcal{X}^H(i,:,:)\cap\mathcal{X}^H(:,:,k)\}.
\end{aligned}$$

For mode-2, CGH $\mathcal{H}^2_{CG}(\mathcal{X}^L)=(\mathcal{V}^L,\mathcal{N}^H\cup\mathcal{N}^L\cup\mathcal{N}^F)$ contains vertex v_j^L for each lateral slice $\mathcal{X}^L(:,j,:)$. It contains nets n_i^H, n_j^L, and n_k^F for slices $\mathcal{X}^L(i,:,:)$, $\mathcal{X}^L(:,j,:)$, and $\mathcal{X}^L(:,:,k)$, respectively, where

$$\begin{aligned}
&Pins(n_i^H)=\{v_j^L:\exists\mathcal{X}^L(i,j,k)\in\mathcal{X}^L(:,j,:)\cap\mathcal{X}^L(i,:,:)\},\\
&Pins(n_j^L)=\{v_j^L\},\\
&Pins(n_k^F)=\{v_j^L:\exists\mathcal{X}^L(i,j,k)\in\mathcal{X}^L(:,j,:)\cap\mathcal{X}^L(:,:,k)\}.
\end{aligned}$$

For mode-3, CGH $\mathcal{H}^3_{CG}(\mathcal{X}^F)=(\mathcal{V}^F,\mathcal{N}^H\cup\mathcal{N}^L\cup\mathcal{N}^F)$ contains one vertex v_k^F for each frontal slice $\mathcal{X}^F(:,:,k)$. It contains nets n_i^H, n_j^L, and n_k^F for slices $\mathcal{X}^F(i,:,:)$, $\mathcal{X}^F(:,j,:)$, and $\mathcal{X}^F(:,:,k)$, respectively.

The three component hypergraphs $\mathcal{H}^1_{CG}(\mathcal{X}^H)$, $\mathcal{H}^2_{CG}(\mathcal{X}^L)$, and $\mathcal{H}^3_{CG}(\mathcal{X}^F)$ are both vertex and pin disjoint, whereas they share nets. From these three component hypergraphs, we form a composite hypergraph

$$\mathcal{H}_{MG}(\mathcal{X})=(\mathcal{V}^H\cup\mathcal{V}^L\cup\mathcal{V}^F,\mathcal{N}^H\cup\mathcal{N}^L\cup\mathcal{N}^F),$$

through a net amalgamation operation which is defined as combining disjoint pin sets of the nets corresponding to the same slice in three component hypergraphs. That is,

$$\begin{aligned}
Pins(n_i^H(\mathcal{X}))&=\{v_i^H\}\cup Pins(n_i^H(\mathcal{X}^L))\cup Pins(n_i^H(\mathcal{X}^F)),\\
Pins(n_j^L(\mathcal{X}))&=Pins(n_j^L(\mathcal{X}^H))\cup\{v_j^L\}\cup Pins(n_j^L(\mathcal{X}^F)),\\
Pins(n_k^F(\mathcal{X}))&=Pins(n_k^F(\mathcal{X}^H))\cup Pins(n_k^F(\mathcal{X}^L))\cup\{v_k^F\}.
\end{aligned}$$

The motivation behind the net amalgamation operation is the fact that the sum of connectivities of the nets that represent the same slice in three component hypergraphs overestimates the communication volume due to the respective factor-matrix row, whereas the connectivity of the amalgamated net exactly encodes the total communication volume as discussed below.

Net n_i^H represents the same factor-matrix row $\mathbf{A}(i,:)$ in each of the mode-dependent coarse-grain hypergraphs $\mathcal{H}^1_{CG}(\mathcal{X}^H)$, $\mathcal{H}^2_{CG}(\mathcal{X}^L)$, and $\mathcal{H}^3_{CG}(\mathcal{X}^F)$. Each vertex connected by n_i^H in each of these hypergraphs denotes a distinct atomic task that contributes to $\mathbf{A}(i,:)$. Consider an atomic task represented by vertex v_j^L of mode-2 coarse-grain hypergraph $\mathcal{H}^2_{CG}(\mathcal{X}^L)$, where $v_j^L\in Pins(n_i^H(\mathcal{X}^L))$. This means that at least one nonzero of the mode-3 fiber $\mathcal{X}(i,j,:)$ is assigned to slice $\mathcal{X}^L(:,j,:)$ during the splitting process (Eq. (1)). So, the atomic task represented by vertex v_j^L contributes to the factor-matrix row $\mathbf{A}(i,:)$. Assume that atomic tasks represented by vertices v_i^H and v_j^L are assigned to different processors p_q and p_r, respectively, during the partitioning of composite hypergraph. This will incur reduce (fold) type of communication on the partial results computed by processors p_q and p_r for factor-matrix row $\mathbf{A}(i,:)$ during the MTTKRP operation along mode-1. This will also incur a dual expand type of communication during the MTTKRP operation along mode-2 because of p_q sending $\mathbf{A}(i,:)$ to p_r. A similar discussion holds for an atomic task represented by vertex v_k^F of mode-3 coarse-grain hypergraph $\mathcal{H}^3_{CG}(\mathcal{X}^F)$, where $v_k^F\in Pins(n_i^H(\mathcal{X}^F))$. So, the parts in the connectivity set of amalgamated net n_i^H denote the set of distinct processors that contribute to $\mathbf{A}(i,:)$. This explains the validity of pin-wise amalgamation of nets $n_i^H(\mathcal{X}^H)$, $n_i^H(\mathcal{X}^L)$, and $n_i^H(\mathcal{X}^F)$ into $n_i^H(\mathcal{X})$, where $2R(con(n_i^H(\mathcal{X}))-1)$ will correctly encode the volume of communication due to factor-matrix row $\mathbf{A}(i,:)$.

The medium-grain hypergraph model presented above can also be viewed as a nonzero-to-slice assignment for producing the medium-grain model from the fine-grain model. Each tensor nonzero belongs to three slices, one along each mode. We assign each nonzero to one of those slices utilizing a greedy splitting heuristic. We then construct the medium-grain hypergraph by representing each slice in each mode as a vertex as well as a net connecting that vertex along that mode. These one-to-one net-to-vertex connections constitute the pins of those nets along the same mode. Then we construct pins of these nets along other modes as follows: Let u represent slice i along the first mode and v represent slice j along the second mode. Then, if there exists a nonzero in fiber $\mathcal{X}(i,j,:)$ assigned to v, u is added to the pin list of the net corresponding to the slice represented by vertex v. Pin additions for all pairs of modes are defined similarly.

3.2 Splitting Heuristic

The motivation behind the splitting given in Eq. (1) is to address the drawback of the coarse-grain model [10] incurring high communication volume due to dense slices. For this purpose, splitting should be performed in such a way to attain sparse mode slices in the respective component tensors as much as possible. In particular, the objective should be to attain sparse horizontal, lateral, frontal slices in component tensors $\mathcal{X}^H$, $\mathcal{X}^L$, $\mathcal{X}^F$, respectively.

The proposed splitting heuristic works as follows: Each nonzero is assigned to a component tensor according to the sparsest slice containing that nonzero. That is, for $\mathcal{X}(i,j,k)$ we look for the sparsest slice among the three slices $\mathcal{X}(i,:,:)$, $\mathcal{X}(:,j,:)$, and $\mathcal{X}(:,:,k)$. For example, if $nnz(\mathcal{X}(i,:,:))$ is smaller than both $nnz(\mathcal{X}(:,j,:))$ and $nnz(\mathcal{X}(:,:,k))$, then we assign $\mathcal{X}(i,j,k)$ to the ith slice of $\mathcal{X}^H$.

The special case, which occurs when a slice contains a single nonzero, is handled as follows: For example, if the ith horizontal slice of $\mathcal{X}$ contains a single nonzero $\mathcal{X}(i,j,k)$, then the splitting heuristic assigns $\mathcal{X}(i,j,k)$ to the ith slice of $\mathcal{X}^H$. However, this will create a single-pin net n_i^H in both $\mathcal{H}_{CG}(\mathcal{X}^H)$ and $\mathcal{H}_{MG}(\mathcal{X})$. Since a single-pin net cannot be cut, we assign nonzero $\mathcal{X}(i,j,k)$ to either $\mathcal{X}^L(:,j,:)$ or $\mathcal{X}^F(:,:,k)$ according to number of nonzeros in $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$. Tie cases occur when more than one slice having the same number of nonzeros constitute the most sparse slice for a nonzero. Such tie cases are resolved by assigning the nonzero to the slice of the longer mode. The algorithm for this splitting heuristic is given in Algorithm 3.

The leftmost part of Fig. 1 displays a sample $4\times3\times2$ tensor $\mathcal{X}$ with 10 nonzeros. The second part of the figure displays the component tensors $\mathcal{X}^H$, $\mathcal{X}^L$, $\mathcal{X}^F$ obtained after running the splitting heuristic. In the figure, each nonzero is displayed with a different empty/filled symbol for a better understanding of the splitting heuristic. The third part of the figure displays the component hypergraphs $\mathcal{H}^1_{CG}(\mathcal{X}^H)$, $\mathcal{H}^2_{CG}(\mathcal{X}^L)$, $\mathcal{H}^3_{CG}(\mathcal{X}^F)$. Finally, the rightmost part displays the medium-grain composite hypergraph $\mathcal{H}_{MG}(\mathcal{X})$. As seen in the figure, the heuristic splits eight nonzeros of the dense frontal slice $\mathcal{X}(:,:,1)$ to sparse slices $\mathcal{X}^H(2,:,:)$, $\mathcal{X}^H(3,:,:)$, $\mathcal{X}^H(4,:,:)$, $\mathcal{X}^L(:,1,:)$, $\mathcal{X}^L(:,2,:)$, and $\mathcal{X}^L(:,3,:)$ so that the component-frontal slice $\mathcal{X}^F(:,:,1)$ becomes empty.

Algorithm 3. SPLIT-TENSOR($\mathcal{X}$)

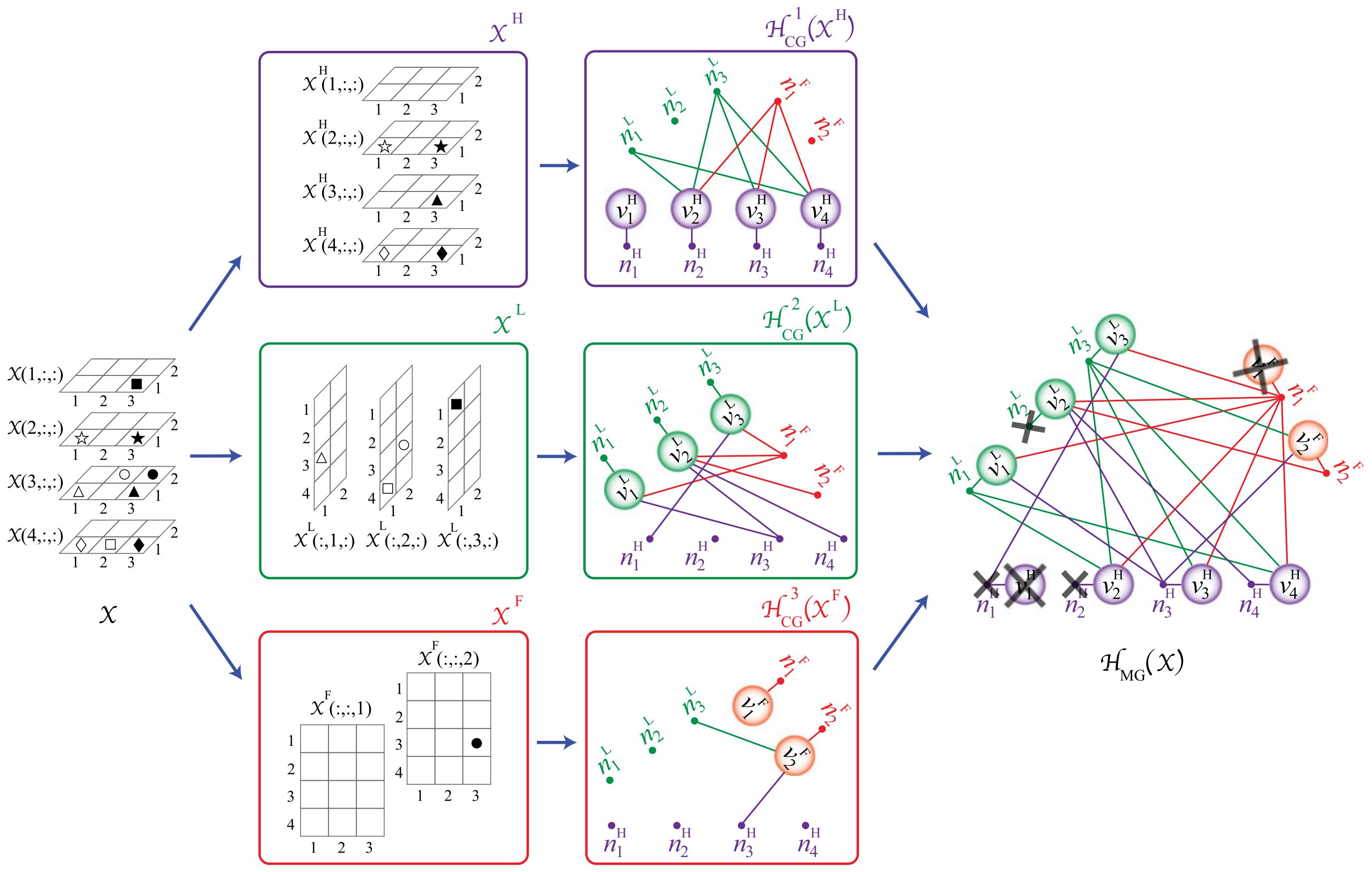
```
1:  $\mathcal{X}^H \leftarrow 0, \mathcal{X}^L \leftarrow 0, \mathcal{X}^F \leftarrow 0,$
2:  for each nonzero $\mathcal{X}(i,j,k) \in \mathcal{X}$ do
3:    $nzH \leftarrow nnz(\mathcal{X}(i,:,:))$
4:    $nzL \leftarrow nnz(\mathcal{X}(:,j,:))$
5:    $nzF \leftarrow nnz(\mathcal{X}(:,:,k))$
6:    if $nnz(\mathcal{X}(i,:,:)) = 1$ then
7:      $nzH \leftarrow \infty$
8:    if $nnz(\mathcal{X}(:,j,:)) = 1$ then
9:      $nzL \leftarrow \infty$
10:   if $nnz(\mathcal{X}(:,:,k)) = 1$ then
11:     $nzF \leftarrow \infty$
12:   $nzM \leftarrow 0$
13:   if $nzH \le nzL$ then
14:     if $nzH = nzL$ then
15:       $m \leftarrow \arg\max(I,J)$
16:     else
17:       $m \leftarrow H$
18:     $nzM \leftarrow nzH$
19:   else
20:     $m \leftarrow L$
21:     $nzM \leftarrow nzL$
22:   if $nzF \le nzM$ then
23:     if $nzF = nzM$ then
24:       $m \leftarrow \arg\max(K, dim(m))$
25:     else
26:       $m \leftarrow F$
27:   assign $\mathcal{X}(i,j,k)$ to $\mathcal{X}^m$
```

3.3 Discussion

The proposed medium-grain model can be considered as an in-between model that tries to attain merits of both coarse- and fine-grain models while trying to avoid their drawbacks. The medium-grain model applies mode-dependent coarse-grain models on the component tensors obtained by splitting in Eq. (1). In this way, it partially exploits the computational coherence of nonzeros in the same slice while avoiding the drawbacks of the coarse-grain model on nonzero replication and assignment of dense slices as a whole to individual processors. The medium-grain model can also be considered as clustering individual nonzeros in the slices of the component tensors via the splitting heuristic. In this way, it partially exploits the flexibility of assigning individual nonzeros to different processors while avoiding the drawbacks of the fine-grain model on partitioning very large hypergraphs as well as disturbing message coherence.

We introduce Table 1 on the size of the hypergraphs utilized by coarse-grain (CG), fine-grain (FG), and medium-grain (MG) models. $\mathrm{n}f(\mathcal{X})$ denotes the sum of the numbers of nonzero fibers along all modes and $\delta(\mathcal{X})$ denotes the number of slices that contain a single nonzero. "$\le I+J+K-\delta(\mathcal{X})$" refers to the possibility of vanishing vertices and/or nets because of the following cases.

Although the original tensor does not contain empty slices, component tensors may contain empty slices because of the following two cases. The first case occurs when the original tensor contains a relatively dense slice along one mode in such a way that each nonzero of that slice belongs to a sparser



$\mathcal{X}$
$\mathcal{X}^H$
$\mathcal{X}^L$
$\mathcal{X}^F$
$\mathcal{H}^1_{CG}(\mathcal{X}^H)$
$\mathcal{H}^2_{CG}(\mathcal{X}^L)$
$\mathcal{H}^3_{CG}(\mathcal{X}^F)$
$\mathcal{H}_{MG}(\mathcal{X})$

Fig. 1. Split tensor, coarse-grain mode-dependent hypergraphs and medium-grain hypergraph. "×" denotes the vanishing vertices and nets.

slice along another mode. The second case occurs because of the special case handling of tensor slices which contain a single nonzero by the splitting heuristic as mentioned earlier. In such cases, a vertex as well as a net corresponding to an empty slice of a component tensor vanish in the respective component hypergraph. In the rightmost part of Fig. 1, vertex v_1^F exemplifies the first case, whereas vertex v_1^H and net n_1^H exemplify the second case with crossed out vertices and nets.

Consider another case where the original tensor contains a relatively sparse slice along one mode (say mode-m) in such a way that each nonzero of that slice belongs to a denser slice along another mode. In such a case, the splitting heuristic assigns all nonzeros of that mode-m slice to the mode-m component tensor so that the net corresponding to that mode-m slice becomes a single-pin net in the component hypergraph. These single-pin nets vanish as they cannot become cut. In the rightmost part of Fig. 1, crossed out nets n_2^H and n_2^L exemplify such nets.

TABLE 1
Size Comparison of Coarse-, Fine-, and Medium-Grain Models

	Number of		
	vertices	nets	pins
CG	$I+J+K$	$I+J+K$	$I+J+K+2nf(\mathcal{X})$
FG	$nnz(\mathcal{X})$	$I+J+K-\delta(\mathcal{X})$	$3nnz(\mathcal{X})-\delta(\mathcal{X})$
MG	$\leq I+J+K-\delta(\mathcal{X})$	$\leq I+J+K-\delta(\mathcal{X})$	$\leq 3nnz(\mathcal{X})-2\delta(\mathcal{X})$
CG	0.15	1.64	1.12
FG	1.00	1.00	1.00
MG	0.10	0.12	0.36

Here the numbers are empirical values averaged over the tensors used in the experiments and normalized with respect to those of FG.

As seen in top part of Table 1, CG incurs a smaller number of vertices than FG because of fine-grain atomic task definition of FG, whereas MG incurs a smaller number of vertices than CG because of the vanishing vertices. FG incurs a smaller number of nets than CG because of the slices that contain a single nonzero, whereas MG incurs a smaller number of nets than FG because of the vanishing nets. Size comparison of CG and FG, in terms of pins, depends on topological properties of the original tensor. On the other hand, MG incurs a smaller number of pins than FG because of the vanishing vertices and nets.

The bottom part of Table 1 shows the comparison of hypergraph sizes of these three methods normalized with respect to those of FG model averaged over 10 test tensors. Size comparison of MG against FG is as follows: MG incurs a significantly smaller number of vertices, nets, and pins than FG as expected. On average, MG incurs 90, 88, and 64 percent smaller number of vertices, nets, and pins than FG, respectively. Size comparison of MG against CG is as follows: MG incurs a significantly smaller number of nets and pins than CG as also expected, whereas MG incurs a considerably smaller number of vertices than CG. On average, MG incurs 36, 93, and 68 percent smaller number of vertices, nets, and pins than CG, respectively. Relatively smaller percentage of decrease in the number of vertices than those of nets can be explained by a larger number of vanishing nets than the number of vanishing vertices.

Here, we discuss similarities and differences between medium-grain tensor partitioning with the medium-grain matrix partitioning proposed by Pelt and Bisseling [21] for parallel SpMV. Tensor splitting framework given in Eq. (1) is the generalization of the composite model given in [21] for matrices. The proposed net amalgamation operation is established by the identity matrices of [21, Eq. (4)]. The tensor splitting heuristic utilized in this work is an extension of the matrix splitting heuristic utilized in [21]. The vanishing vertices and nets discussed here generalize the missing entries of the identity matrices in [21]. The RB scheme we utilized here is also similarly utilized in [21]. The main difference of [21] is that iterative refinement for matrix bipartitioning cannot be directly applied for tensors, whereas the outcome of a matrix bipartitioning can be used as a new split to feed back into the refinement part of a bipartitioning. Obviously with tensors being split into three parts this is not directly possible.

3.4 Recursive-Bipartitioning Scheme

A naive implementation of the proposed splitting framework is to apply a splitting heuristic to the original tensor at the topmost level to form medium-grain hypergraph $\mathcal{H}_{MG}(\mathcal{X})$ and then invoke a P-way HP tool to obtain P subtensors to be assigned to P processors.

Here we try to improve the quality of the splitting heuristic by utilizing the RB paradigm which is commonly used by hypergraph/graph partitioning tools for obtaining multi-way partitions. Applying the splitting heuristic to each of the $P-2$ intermediate subtensors in addition to the original tensor at the topmost level is expected to improve the quality of the partitioning. This is because the subtensors obtained after each RB step are likely to have different sparsity patterns which justifies the possible benefit of using the splitting heuristic on subtensors individually.

In the proposed RB paradigm, the given tensor is bipartitioned into two subtensors, which are further bipartitioned recursively until P subtensors are obtained, where each subtensor at the last level is assigned to a different processor. Here, without loss of generality, we assume that the number P of processors is an exact power of 2. This procedure produces a complete binary tree with $\log_2 P$ levels which is referred as the RB tree. 2^ℓ tensors in the ℓth level are denoted by $\mathcal{X}_1^\ell, \ldots, \mathcal{X}_{2^\ell}^\ell$ from left to right for $0 \le \ell \le \log_2 P$.

A vertex bipartition $\Pi_2 = \{\mathcal{V}_L, \mathcal{V}_R\}$ of the medium-grain hypergraph $\mathcal{H}_{MG}(\mathcal{X}_k^\ell)$ of the ℓth level tensor $\mathcal{X}_k^\ell$ is utilized to form two new vertex-induced subtensors $\mathcal{X}_{2k-1}^{\ell+1}$ and $\mathcal{X}_{2k}^{\ell+1}$, both in level $\ell+1$. Here, $\mathcal{V}_L$ and $\mathcal{V}_R$ respectively refer to the left and right parts of the vertex bipartition of $\mathcal{H}_{MG}(\mathcal{X}_k^\ell)$. Because of the nature of the splitting heuristic applied to $\mathcal{X}_k^\ell$, any vertex v_i^m in $\mathcal{V}_L$ incurs the assignment of the respective mode-m (sub)slice of $\mathcal{X}_k^\ell$ as a whole mode-m slice to left subtensor $\mathcal{X}_L = \mathcal{X}_{2k-1}^{\ell+1}$, for any mode-$m$. The right subtensor $\mathcal{X}_R = \mathcal{X}_{2k}^{\ell+1}$ is formed in a dual manner. Then the splitting heuristic is applied to the left and the right subtensors separately to construct the left and right medium-grain hypergraphs $\mathcal{H}_{MG}(\mathcal{X}_{2k-1}^{\ell+1})$ and $\mathcal{H}_{MG}(\mathcal{X}_{2k}^{\ell+1})$, respectively, for further bipartitioning. Algorithm 4 shows the steps of the proposed RB scheme.

4 Medium-Grain Tripartite Graph Model

We derive the medium-grain tripartite graph from the medium-grain hypergraph: Each vertex and net pair (v_i^H, n_i^H) in $\mathcal{H}_{MG}(\mathcal{X})$ that represents the same slice is replaced by a node u_i^H in $\mathcal{G}_{MG}(\mathcal{X})$ with the adjacency list defined as follows:

$$\begin{aligned} dj(u_i^H) = &\{u_j^L : v_j^L \in Pins(n_i^H)\} \cup \{u_k^F : v_k^F \in Pins(n_i^H)\} \cup \\ &\{u_x^L : n_x^L \in Nets(v_i^H)\} \cup \{u_y^F : n_y^F \in Nets(v_i^H)\}. \end{aligned}$$

The two sets in the upper line come from the vertices connected by n_i^H, whereas the two sets in the lower line come from the nets that connect v_i^H.

Fig. 2 displays medium-grain tripartite graph $\mathcal{G}_{MG}(\mathcal{X})$ derived from the $\mathcal{H}_{MG}(\mathcal{X})$ displayed in Fig. 1. For example, v_1^L and n_1^L in Fig. 1 is replaced by node u_1^L with adjacency list $Adj(u_1^L) = \{u_2^H, u_3^H, u_4^H, u_1^F\}$. $Adj(u_1^L)$ contains: u_2^H and u_4^H because of n_1^L connecting v_2^H and v_4^H; u_3^H because of n_3^H connecting v_1^L; u_1^F beacuse of n_1^F connecting v_1^L.

The topology of the tripartite graph resembles that of the tripartite graph model proposed in [8] for mode-independent slice reorderings for shared-memory parallelism. The topology of the medium-grain tripartite graph model utilized in this work differs because of the vanishing nodes incurred by the special case handling of the splitting heuristic. The two graph models also differ in terms of the weights associated with the nodes and edges. In the graph model of [8], nodes are assigned unit weights, whereas each edge is assigned a weight equal to the number of nonzeros in the fiber constituting the respective edge. In the medium-grain graph model, edges are assigned a weight of R, whereas each node is assigned a weight equal to the number of nonzeros in the respective slice of the split tensor.

The two tripartite graph models also differ because of the RB framework proposed in this work. The work in [8] performs P-way partitioning on their tripartite graph model for a predetermined sufficiently large P value to find a top-down partial slice ordering. We utilize the same RB framework used in Algorithm 4 for the tripartite graph model. In

Algorithm 4. RB-Based Medium-Grain Partitioning($\mathcal{X}$, P)

```
1:  $\mathcal{X}_1^0 = \mathcal{X}$
2:  for $\ell \leftarrow 0$ to $\log_2 P - 1$ do
3:      for $k \leftarrow 1$ to $2^\ell$ do
4:          $\{\mathcal{X}^H, \mathcal{X}^L, \mathcal{X}^F\} \leftarrow$ SPLIT-TENSOR($\mathcal{X}_k^\ell$)
5:          Form $\mathcal{H}_{MG}(\mathcal{X}_k^\ell)$ from $\mathcal{X}^H, \mathcal{X}^L, \mathcal{X}^F$
6:          $\Pi_2 \leftarrow$ BIPARTITION($\mathcal{H}_{MG}(\mathcal{X}_k^\ell)$)  $\triangleright$ $\Pi_2 = \{\mathcal{V}_L, \mathcal{V}_R\}$
7:          Form $\mathcal{X}_L = \mathcal{X}_{2k-1}^{\ell+1}$ induced by $\mathcal{V}_L$
8:          Form $\mathcal{X}_R = \mathcal{X}_{2k}^{\ell+1}$ induced by $\mathcal{V}_R$
9:  for $k \leftarrow 1$ to $P$ do
10:     assign subtensor $\mathcal{X}_k^{\log_2 P}$ to processor $k$
```

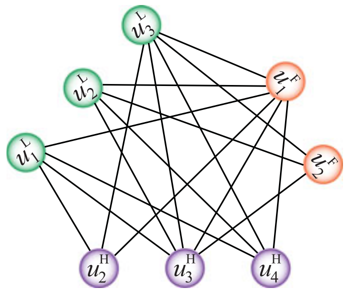


Fig. 2. Medium-grain tripartite graph obtained from $\mathcal{H}_{MG}(\mathcal{X})$ in Fig. 1.

this way, by applying the splitting heuristic at each RB step, we try to maximize its improvement.

The partitioning objective of minimizing the cutsize in $\mathcal{H}_{MG}(\mathcal{X})$ and in $\mathcal{G}_{MG}(\mathcal{X})$ differs as the hypergraph model correctly encapsulates the volume of communication incurred during parallel MTTKRP operations, while the tripartite graph model encapsulates an approximation of the same metric. Consider a case where a processor p_q produces multiple partial results for the same factor-matrix row $\mathbf{A}(i,:)$ which is assigned to another processor p_r. The hypergraph model correctly encodes communication volume considering the fact that p_q will sum these partial results and send only one partial result for $\mathbf{A}(i,:)$ to p_r. The tripartite graph model overestimates the communication volume as if processor p_q will send multiple partial results for $\mathbf{A}(i,:)$ to p_r without any local summation. The tripartite graph model displays a dual communication volume overestimation in expand type of communications where processor p_r should send $\mathbf{A}(i,:)$ only once to p_q.

5 Experiments

We evaluate the performance of the proposed medium-grain (MG) tensor partitioning method against the fine-grain (FG) tensor partitioning method [10] as well as the state-of-the-art cartesian methods [11], [12].

5.1 Setup

We use PaToH [22], [23] for partitioning hypergraph models in Cart [12], FG [10] and MG methods, which will be referred to as $\mathrm{Cart}_{\mathrm{HP}}$, $\mathrm{FG}_{\mathrm{HP}}$ and $\mathrm{MG}_{\mathrm{HP}}$, respectively. For $\mathrm{Cart}_{\mathrm{HP}}$, we used the same parameters described in [12]. For $\mathrm{MG}_{\mathrm{HP}}$ and $\mathrm{FG}_{\mathrm{HP}}$, we use PaToH [22] with SPEED parameters and maximum imbalance ratio set to $\epsilon = 0.10$. We use METIS [24] with the default parameters and maximum imbalance ratio set to $\epsilon = 0.10$ for partitioning the tripartite graph model in the MG method, which will be referred to as $\mathrm{MG}_{\mathrm{TGP}}$. Since PaToH and METIS use randomized algorithms, we partition each instance three times and report average results.

Recall that correctness of encapsulating the total communication volume depends on the consistency condition of assigning each factor-matrix row to one of the processors that contributes to that row. Here we adopt the best-fit increasing heuristic used for solving the P-feasible bin-packing problem [25]. We assign factor-matrix rows to processors independently for each mode. First, we sort the factor-matrix rows in that mode in increasing order of their connectivities. Then, we assign them to the processors according to the best-fit criterion which corresponds to assigning a row to a processor that currently has the minimum communication volume among the processors that own a nonzero in the corresponding slice. Here communication volume refers to total send-volume of communication in fold and expand steps. After assigning a row to a processor, the volumes of the respective processors are increased accordingly.

For parallel CPD-ALS experiments, we used the parallel CPD-ALS code developed and used in [12]. The source code is implemented in C using MPI for interprocess communication and compiled with gcc version 8.3.0 using O3 optimization flag and ParaStation MPI (version 5.2.2, based on MPICH v3). Compressed fiber-based multiplication and storage scheme is used for MTTKRP as proposed in [8] to reduce the number of FLOPs and memory footprint. CBLAS routines from Intel MKL library (version 2019) are used for the rest of the computations on factor-matrix rows.

We conducted our parallel experiments on the JUWELS cluster from Jülich Supercomputing Centre. A node of this cluster has 48 cores (two Intel Xeon Platinum 8168 CPUs) running at 2.70 GHz clock frequency and 96 GB memory. The interconnection network of this cluster is EDR-Infiniband (Connect-X4).

TABLE 2
Properties of the Test Tensors

tensor	size of dimensions				nnz	density
	I	J	K	L		
1998DARPA	23.7M	13.5K	17.9K	–	28.4M	4.92e-09
brightkite	772.9K	942	51.4K	–	2.6M	7.16e-08
delicious	17.2M	532.9K	2.5M	1.4K	140.1M	4.27e-15
enron	244.2K	5.7K	6.0K	1.1K	54.2M	5.46e-09
facebook	42.4K	1.5K	39.9K	–	738.0K	2.89e-07
flickr	28.1M	319.6K	1.6M	731	112.9M	1.07e-14
freebase_music	22.6M	166	23.3M	–	99.5M	1.14e-09
lbnl-network	868.1K	4.2K	4.2K	–	1.7M	1.11e-07
movies-amazon	226.5K	4.4K	87.8K	–	15.0M	1.72e-07
nell-1	25.4M	2.1M	2.9M	–	143.6M	9.05e-13

5.2 Dataset

The dataset given in Table 2 contains 10 sparse tensors which arise from real-world applications. The density of each tensor $\mathcal{X}$ is computed as $nnz(\mathcal{X})/(I \times J \times K \times L)$. delicious, flickr, enron, lbnl-network, and nell-1 are acquired from the FROSTT tensor library [26]. The details of these five tensors are already given in [26], whereas we summarize the remaining five tensors as follows: 1988DARPA consists of time–IP address–IP address triplets from the network of MIT Lincoln laboratory [27]. brightkite consists of user–date–location triplets of check-in information shared to location based social networks [28]. facebook consists of owner–poster–date triplets extracted from the wall posts of Facebook New Orleans networks [29]. freebase_music consists of subject–object–relation triplets from freebase.com [27]. movies-amazon consists of user–movie–word triplets extracted from the user reviews in Amazon [30].

5.3 Parallel Performance Comparison

We compare the proposed methods in terms of communication cost metrics and parallel CPD-ALS runtimes on 10 tensors on $P = 512$ processors. The communication cost metrics consist of maximum and average send volume handled by a processor and maximum and average number of messages sent by a processor. Average volume and average message count values refer to the total communication volume and total number of messages divided by P. We prefer to report average values instead of total values, because average values give a better view on the amount of deviation of maximum values from the average values.

In all experiments, $\mathrm{MG}_{\mathrm{HP}}$ and $\mathrm{MG}_{\mathrm{TGP}}$ are run through utilizing the RB scheme (see Section 3.4). On average, RB-based implementation achieves about 4 and 12 percent decrease in

TABLE 3
Comparison of Communication Metrics and Parallel CPD-ALS Time of $\mathrm{FG_{HP}}$ and $\mathrm{MG_{HP}}$ for $P = 512$ and $R = 64$

tensor	$\mathrm{FG_{HP}}$							$\mathrm{MG_{HP}}$ normalized with respect to $\mathrm{FG_{HP}}$						
	part. time (s)	max nnz	comm. vol. $\times R$		# of mssgs.		CPD-ALS time (s)	part. time	max nnz	comm. vol.		# of mssgs.		CPD-ALS time
			max	avg	max	avg				max	avg	max	avg	
`1998DARPA`	454	56,010	7,572	1,646	1,335	227	0.120	0.63	1.00	1.24	0.74	0.96	0.83	1.01
`brightkite`	65	5,237	5,044	3,392	2,182	1,060	0.038	0.63	1.03	1.13	0.95	0.95	0.89	0.91
`delicious`	12,259	277,391	226,452	212,460	4,088	3,772	1.008	0.38	1.01	0.93	0.89	1.00	0.98	0.90
`enron`	1,053	106,642	35,238	13,788	4,026	2,517	0.135	0.30	1.07	1.02	0.68	0.99	0.75	0.82
`facebook`	20	1,443	3,968	3,168	1,679	1,112	0.034	0.59	1.03	0.99	1.00	1.04	0.94	0.98
`flickr`	3,040	222,377	59,018	48,952	3,505	2,770	0.406	0.54	1.00	1.07	1.06	1.03	1.05	0.96
`freebase_music`	4,706	197,101	52,614	14,686	2,602	935	0.724	0.51	1.00	0.89	1.01	1.00	1.00	1.00
`lbnl-network`	9	3,349	2,074	68	1,123	32	0.007	0.73	1.00	0.99	1.06	1.00	1.03	1.00
`movies-amazon`	410	29,651	34,260	15,344	2,850	1,500	0.089	0.41	1.03	1.03	0.94	1.00	0.98	1.01
`nell-1`	14,675	283,294	306,766	284,024	3,066	2,807	1.447	0.53	1.02	1.03	0.97	0.99	0.95	1.02
geo mean	**598**	**41,578**	**24,621**	**10,099**	**2,431**	**988**	**0.149**	**0.51**	**1.02**	**1.03**	**0.92**	**1.00**	**0.94**	**0.96**

Communication volume values are given in terms of the number of the factor-matrix rows sent by processors.

the total communication volume compared to the naive implementation for $\mathrm{MG_{HP}}$ and $\mathrm{MG_{TGP}}$, respectively.

We first discuss the results of the experiments performed for comparing MG against FG. We restrict the performance comparison on hypergraph models since the graph model on FG performs significantly worse than on MG. For example, on average, FG partitioning using the graph model decreases the partitioning time by 17 percent at the expense of 3.74× total communication volume, compared to using the hypergraph model. We do not include the coarse-grain model in the experiments since the fine-grain model is reported to perform much better [10].

Table 3 shows the performance improvement attained by $\mathrm{MG_{HP}}$ against $\mathrm{FG_{HP}}$ on $P = 512$ processor with $R = 64$. The left part of the table displays the actual values for $\mathrm{FG_{HP}}$, whereas the right part displays the values of $\mathrm{MG_{HP}}$ normalized with respect to those of $\mathrm{FG_{HP}}$. The last row shows the geometric averages.

As seen in Table 3, $\mathrm{MG_{HP}}$ and $\mathrm{FG_{HP}}$ display comparable performance in computational load balance ("max nnz"). Note that $\mathrm{FG_{HP}}$ has a large number of vertices with unit weights thus enabling good load balance. Thus, the comparable performance of $\mathrm{MG_{HP}}$ and $\mathrm{FG_{HP}}$ in load balancing shows the success of the splitting heuristic in reducing the variance on vertex weights.

$\mathrm{MG_{HP}}$ achieves considerably better performance than $\mathrm{FG_{HP}}$ in both average communication volume and average message count metrics, whereas it displays slightly worse performance in maximum communication volume and maximum message counts metrics. These findings in the communication cost metrics lead to 4.2 percent reduction in parallel CPD-ALS time. $\mathrm{MG_{HP}}$ achieves this improvement with about two times less preprocessing overhead.

Table 4 compares the performance of $\mathrm{MG_{HP}}$ and $\mathrm{MG_{TGP}}$ against the state-of-the-art $\mathrm{Cart_{HP}}$ algorithm for $P = 512$ and $R = 64$. In the table, a block of three rows displays the values for three different methods for the respective tensor, whereas the bottom block shows the average values. The left part of the table displays the actual values, whereas the right part displays the values of $\mathrm{MG_{HP}}$ and $\mathrm{MG_{TGP}}$ normalized with respect to those of $\mathrm{Cart_{HP}}$.

As seen in Table 4, $\mathrm{MG_{HP}}$ achieves considerably better computational load balance than $\mathrm{Cart_{HP}}$. On average, $\mathrm{MG_{HP}}$ reduces the maximum computational load by 13 percent compared to $\mathrm{Cart_{HP}}$. This experimental finding is because of the following reasons: As discussed above, $\mathrm{MG_{HP}}$ performs as good as $\mathrm{FG_{HP}}$ in load balancing. However, in $\mathrm{Cart_{HP}}$, vertices representing whole slices in the first partitioning phase as well as multi-constraint partitioning in the second and third partitioning phases bring difficulty in attaining good load balance. $\mathrm{MG_{HP}}$ achieves significantly better performance in both bandwidth metrics as expected. However, $\mathrm{Cart_{HP}}$ achieves better performance in both latency metrics because of the nice upper bounds provided by the cartesian partitioning in the maximum and average number of messages. The improvements attained by $\mathrm{MG_{HP}}$ against $\mathrm{Cart_{HP}}$ in computational load balance and communication bandwidth metrics lead to an average of 22 percent improvement in parallel CPD-ALS time.

As seen in Table 4, $\mathrm{MG_{TGP}}$ and $\mathrm{MG_{HP}}$ display comparable performance in load balance, whereas $\mathrm{MG_{TGP}}$ displays significantly worse performance in all communication cost metrics. The performance degradation of the graph model compared to the hypergraph model in communication cost metrics leads to an average of 9 percent increase in the parallel CPD-ALS time. Despite this, $\mathrm{MG_{TGP}}$ still performs better than $\mathrm{Cart_{HP}}$ by an amount of 14 percent on average in terms of parallel CPD-ALS time.

In Fig. 3, we provide performance profiles for the parallel CPD-ALS and partitioning times of $\mathrm{Cart_{HP}}$, $\mathrm{MG_{HP}}$, and $\mathrm{MG_{TGP}}$ as well as $\mathrm{FG_{HP}}$ for a more detailed comparison. Performance profiles capture the relative performance of the compared methods more accurately, thus provide a better understanding of the characteristics of the methods [31]. A point (x, y) in a profile means that the respective model is within an x factor of the best result in a fraction y of the dataset. For example, the point $(x = 1.04, y = 0.90)$ on the profile for $\mathrm{MG_{HP}}$ means that $\mathrm{MG_{HP}}$ incurs 4 percent more parallel CPD-ALS time than the smallest running time achieved in 90 percent of the dataset. Hence, the method closest to the top left corner is the best method.

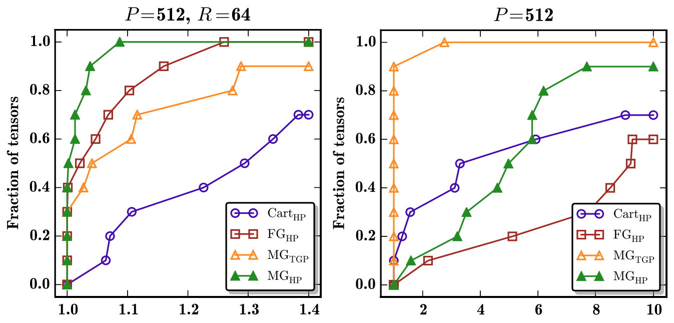
As seen in Fig. 3a, in terms of parallel runtime, $\mathrm{MG_{HP}}$ is the clear winner, where $\mathrm{FG_{HP}}$, $\mathrm{MG_{TGP}}$ display close performance

TABLE 4
Comparison of Communication Metrics and Parallel CPD-ALS time of $\mathrm{Cart}_{\mathrm{HP}}$, $\mathrm{MG}_{\mathrm{HP}}$, and $\mathrm{MG}_{\mathrm{TGP}}$ for $P=512$ and $R=64$

		actual values							normalized with respect to $\mathrm{Cart}_{\mathrm{HP}}$						
tensor	method	part. time (s)	max nnz	comm. vol. $\times R$ max	comm. vol. $\times R$ avg	# of mssgs. max	# of mssgs. avg	CPD-ALS time (s)	part. time	max nnz	comm. vol. max	comm. vol. avg	# of mssgs. max	# of mssgs. avg	CPD-ALS time
	$\mathrm{Cart}_{\mathrm{HP}}$	139	55,544	14,232	3,490	1,589	565	0.127	–	–	–	–	–	–	–
1998DARPA	$\mathrm{MG}_{\mathrm{HP}}$	284	56,066	9,386	1,222	1,284	188	0.121	2.04	1.01	0.66	0.35	0.81	0.33	0.95
	$\mathrm{MG}_{\mathrm{TGP}}$	312	56,941	13,190	2,146	1,103	230	0.123	2.24	1.03	0.93	0.61	0.69	0.41	0.97
	$\mathrm{Cart}_{\mathrm{HP}}$	22	18,316	6,000	3,990	1,162	1,042	0.050	–	–	–	–	–	–	–
brightkite	$\mathrm{MG}_{\mathrm{HP}}$	41	5,403	5,682	3,220	2,079	942	0.035	1.86	0.29	0.95	0.81	1.79	0.90	0.69
	$\mathrm{MG}_{\mathrm{TGP}}$	26	5,328	7,464	5,554	2,512	1,294	0.044	1.17	0.29	1.24	1.39	2.16	1.24	0.89
	$\mathrm{Cart}_{\mathrm{HP}}$	5,418	285,764	524,310	380,112	2,184	2,183	1.443	–	–	–	–	–	–	–
delicious	$\mathrm{MG}_{\mathrm{HP}}$	4,617	279,392	210,976	189,010	4,079	3,701	0.903	0.85	0.98	0.40	0.50	1.87	1.70	0.63
	$\mathrm{MG}_{\mathrm{TGP}}$	3,395	276,311	259,798	203,784	4,083	3,842	0.869	0.63	0.97	0.50	0.54	1.87	1.76	0.60
	$\mathrm{Cart}_{\mathrm{HP}}$	54	112,544	29,770	18,246	2,048	2,021	0.118	–	–	–	–	–	–	–
enron	$\mathrm{MG}_{\mathrm{HP}}$	315	113,913	35,982	9,350	3,978	1,876	0.110	5.79	1.01	1.21	0.51	1.94	0.93	0.93
	$\mathrm{MG}_{\mathrm{TGP}}$	235	108,573	32,012	13,884	3,875	2,271	0.107	4.33	0.96	1.08	0.76	1.89	1.12	0.90
	$\mathrm{Cart}_{\mathrm{HP}}$	9	1,456	4,504	3,584	1,063	918	0.043	–	–	–	–	–	–	–
facebook	$\mathrm{MG}_{\mathrm{HP}}$	12	1,491	3,936	3,178	1,739	1,050	0.033	1.39	1.02	0.87	0.89	1.64	1.14	0.77
	$\mathrm{MG}_{\mathrm{TGP}}$	7	1,472	4,492	3,866	1,956	1,216	0.042	0.83	1.01	1.00	1.08	1.84	1.32	0.98
	$\mathrm{Cart}_{\mathrm{HP}}$	1,941	247,494	228,646	204,654	2,304	2,295	0.520	–	–	–	–	–	–	–
flickr	$\mathrm{MG}_{\mathrm{HP}}$	1,630	222,643	63,222	52,052	3,602	2,919	0.388	0.84	0.90	0.28	0.25	1.56	1.27	0.75
	$\mathrm{MG}_{\mathrm{TGP}}$	2,688	220,496	125,080	67,210	3,895	3,176	0.404	1.39	0.89	0.55	0.33	1.69	1.38	0.78
	$\mathrm{Cart}_{\mathrm{HP}}$	5,423	198,181	250,472	55,234	703	191	0.775	–	–	–	–	–	–	–
freebase_music	$\mathrm{MG}_{\mathrm{HP}}$	2,378	196,738	47,006	14,792	2,602	936	0.723	0.44	0.99	0.19	0.27	3.70	4.90	0.93
	$\mathrm{MG}_{\mathrm{TGP}}$	2,423	199,864	780,312	161,370	2,425	1,076	1.039	0.45	1.01	3.12	2.92	3.45	5.63	1.34
	$\mathrm{Cart}_{\mathrm{HP}}$	753	3,456	5,860	3,796	985	203	0.011	–	–	–	–	–	–	–
lbnl-network	$\mathrm{MG}_{\mathrm{HP}}$	6	3,354	2,060	72	1,121	33	0.007	0.01	0.97	0.35	0.02	1.14	0.16	0.61
	$\mathrm{MG}_{\mathrm{TGP}}$	10	3,389	2,082	170	1,237	49	0.008	0.01	0.98	0.36	0.04	1.26	0.24	0.68
	$\mathrm{Cart}_{\mathrm{HP}}$	62	31,464	73,956	26,394	1,114	1,092	0.124	–	–	–	–	–	–	–
movies-amazon	$\mathrm{MG}_{\mathrm{HP}}$	170	30,688	35,374	14,442	2,852	1,470	0.091	2.73	0.98	0.48	0.55	2.56	1.35	0.73
	$\mathrm{MG}_{\mathrm{TGP}}$	94	29,749	45,284	16,392	2,901	1,545	0.099	1.51	0.95	0.61	0.62	2.60	1.41	0.80
	$\mathrm{Cart}_{\mathrm{HP}}$	7,507	288,684	781,352	458,188	538	526	1.661	–	–	–	–	–	–	–
nell-1	$\mathrm{MG}_{\mathrm{HP}}$	7,726	289,355	315,008	275,958	3,045	2,669	1.473	1.03	1.00	0.40	0.60	5.66	5.07	0.89
	$\mathrm{MG}_{\mathrm{TGP}}$	3,449	280,660	358,938	321,158	3,066	2,994	1.355	0.46	0.97	0.46	0.70	5.70	5.69	0.82
	$\mathrm{Cart}_{\mathrm{HP}}$	**351**	**48,589**	**50,886**	**26,634**	**1,237**	**810**	**0.183**	**–**	**–**	**–**	**–**	**–**	**–**	**–**
geo mean	$\mathrm{MG}_{\mathrm{HP}}$	**304**	**42,394**	**25,329**	**9,308**	**2,419**	**925**	**0.143**	**0.86**	**0.87**	**0.50**	**0.35**	**1.96**	**1.14**	**0.78**
	$\mathrm{MG}_{\mathrm{TGP}}$	**249**	**41,907**	**40,593**	**16,226**	**2,483**	**1,094**	**0.157**	**0.71**	**0.86**	**0.80**	**0.61**	**2.01**	**1.35**	**0.86**

Communication volume values are given in terms of the number of the factor-matrix rows sent by processors.

for the 40 and 30 percent of the dataset respectively, and $\mathrm{Cart}_{\mathrm{HP}}$ displays significantly worse performance. As seen in Fig. 3b, in terms of partitioning time, $\mathrm{MG}_{\mathrm{TGP}}$ is the clear winner, where $\mathrm{FG}_{\mathrm{HP}}$ perfoms significantly worse. These experimental findings conform to the expectations.



(a) parallel CPD-ALS time relative to the best (b) partitioning time relative to the best

Fig. 3. Performance profile for parallel CPD-ALS and partitioning times.

Fig. 4 displays the strong scaling results for $\mathrm{Cart}_{\mathrm{HP}}$, $\mathrm{MG}_{\mathrm{HP}}$, and $\mathrm{MG}_{\mathrm{TGP}}$ as speedup curves for eight out of 10 tensors up to $P=1024$ processors. Speedup curves for the smallest tensors `facebook` and `brightkite` are not given. As seen in the figure, both MG models scale better than $\mathrm{Cart}_{\mathrm{HP}}$ in all instances, whereas $\mathrm{MG}_{\mathrm{HP}}$ scales better than $\mathrm{MG}_{\mathrm{TGP}}$ in six out of eight instances.

We provide Table 5 to compare the relative parallel runtime performance of both MG models against those of $\mathrm{Cart}_{\mathrm{HP}}$ with decreasing R values. The application becomes latency bound with decreasing R values, whereas it becomes bandwidth bound with increasing R values.

As seen in Table 5, relative performance of the both MG models against $\mathrm{Cart}_{\mathrm{HP}}$ decreases as R decreases, as expected. For example, 22 percent better performance of $\mathrm{MG}_{\mathrm{HP}}$ for $R=64$ reduces to 20, 17, and 15 percent for $R=32$, 16, and 8, respectively. For $R=64$, while $\mathrm{MG}_{\mathrm{HP}}$ achieves better runtime performance than $\mathrm{Cart}_{\mathrm{HP}}$ for all of the 10 instances, for $R=8$, $\mathrm{Cart}_{\mathrm{HP}}$ achieves better runtime performance

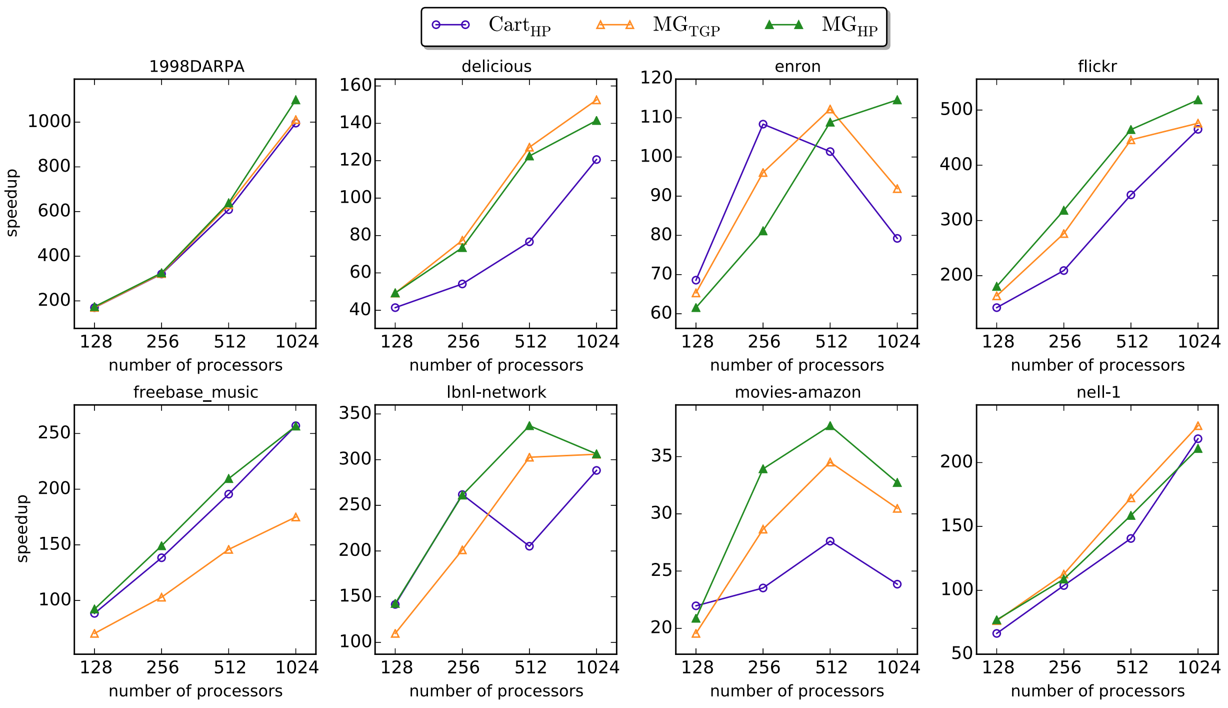


Fig. 4. Strong scaling curves for CPD-ALS obtained by $\mathrm{Cart_{HP}}$, $\mathrm{MG_{HP}}$, and $\mathrm{MG_{TGP}}$ up to $P = 1024$ processors and $R = 64$.

for three of the 10 instances. A similar relative parallel runtime performance variation for $\mathrm{MG_{TGP}}$ against those of $\mathrm{Cart_{HP}}$ can be observed in the table.

5.4 Preprocessing Overhead

Tables 6 and 7 display the partitioning times of $\mathrm{Cart_{HP}}$, $\mathrm{MG_{HP}}$, and $\mathrm{MG_{TGP}}$ normalized with respect to those of sequential CPD-ALS times. Table 6 displays partitioning times for each tensor for $R = 64$, whereas Table 7 displays average partitioning times for $R = \{64, 32, 16\}$. We report the time of one iteration of CPD-ALS and average number of iterations to complete a factorization with 10^{-5} tolerance [8].

As seen in Table 7, on average, for $R = 64$ partitioning time of $\mathrm{Cart_{HP}}$ is 63 percent of the factorization time, whereas partitioning times of $\mathrm{MG_{HP}}$ and $\mathrm{MG_{TGP}}$ are 54 percent and 12 percent of the factorization time, respectively. Since CPD-ALS time depends on the R value, while partitioning time does not, normalized partitioning times increase with decreasing R. For example, partitioning times of $\mathrm{MG_{TGP}}$ are 12, 40 and 121 percent of the factorization times for $R = 64, 32$ and 16, respectively. Note that in many real-world applications users use constraints (e.g., non-negativity) that result in more iterations of CPD-ALS. Thus, these preprocessing overheads may be pessimistic in those scenarios.

We introduce Fig. 5 in order to better visualize the partitioning overhead for P=512. Each bar shows the breakdown

TABLE 5
Comparison of Parallel CPD-ALS time of $\mathrm{Cart_{HP}}$, $\mathrm{MG_{HP}}$, and $\mathrm{MG_{TGP}}$ for $R = \{64, 32, 16, 8\}$ for $P = 512$

tensor	$R = 64$		$R = 32$		$R = 16$		$R = 8$	
	HP	TGP	HP	TGP	HP	TGP	HP	TGP
1998DARPA	0.95	0.97	0.89	0.95	0.84	0.90	0.68	0.73
brightkite	0.69	0.89	0.79	1.05	0.77	1.07	0.76	1.01
delicious	0.63	0.60	0.75	0.70	0.97	0.95	0.90	0.93
enron	0.93	0.90	0.77	0.86	0.82	0.98	0.85	1.01
facebook	0.77	0.98	0.92	1.11	0.96	1.11	0.95	1.08
flickr	0.75	0.78	0.84	0.89	0.91	0.94	0.91	0.97
freebase_music	0.93	1.34	0.94	1.41	0.97	1.54	1.05	1.62
lbnl-network	0.61	0.68	0.54	0.54	0.46	0.47	0.44	0.47
movies-amazon	0.73	0.80	0.77	0.86	0.84	0.92	1.05	1.17
nell-1	0.89	0.82	0.91	0.89	0.96	1.00	1.15	1.32
geo mean	**0.78**	**0.86**	**0.80**	**0.90**	**0.83**	**0.95**	**0.85**	**0.98**

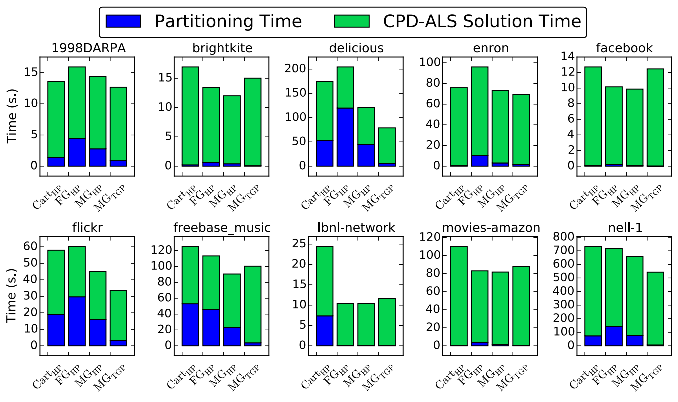
HP and TGP values are given in terms of parallel CPD-ALS times normalized with respect to those of $\mathrm{Cart_{HP}}$.

TABLE 6
Sequential 512-Way Partitioning Overhead in Terms of Sequential CPD-ALS Factorization Time for $R = 64$

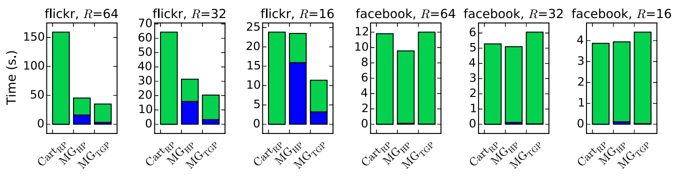
tensor	iteration time (s)	# of iters.	$\mathrm{Cart_{HP}}$	$\mathrm{MG_{HP}}$	$\mathrm{MG_{TGP}}$
1998DARPA	77.443	11	0.17	0.34	0.11
brightkite	2.790	37	0.21	0.39	0.07
delicious	115.719	9	5.02	4.27	0.56
enron	12.005	71	0.06	0.37	0.18
facebook	0.349	33	0.74	1.03	0.23
flickr	181.528	8	1.28	1.08	0.22
freebase_music	151.535	10	3.46	1.52	0.25
lbnl-network	2.297	169	1.94	0.02	0.01
movies-amazon	3.412	98	0.19	0.51	0.14
nell-1	233.521	44	0.73	0.75	0.06

TABLE 7
Average Sequential 512-Way Partitioning Overhead in Terms of Sequential CPD-ALS Factorization Time for $R = \{64, 32, 16\}$

R	# of iters.	$\mathrm{Cart_{HP}}$	$\mathrm{MG_{HP}}$	$\mathrm{MG_{TGP}}$
64	30	0.63	0.54	0.12
32	21	2.05	1.77	0.40
16	14	6.18	5.34	1.21



(a) $\mathrm{MG_{HP}}$ and $\mathrm{MG_{TGP}}$ against $\mathrm{Cart_{HP}}$ and $\mathrm{FG_{HP}}$ for $R=64$.



(b) $\mathrm{MG_{HP}}$ and $\mathrm{MG_{TGP}}$ against $\mathrm{Cart_{RP}}$ [11] for $R=\{64, 32, 16\}$.

Fig. 5. Breakdown of overall runtime for $P=512$.

of the overall parallel runtime into partitioning and CPD-ALS solution times with blue and green colors, respectively. Here, parallel partitioning times are computed assuming a 20 percent efficiency. A CPD-ALS solution refers to factorizing the tensor with three different sets of initial factor matrices [32] and three different R values [33].

Fig. 5a compares proposed $\mathrm{MG_{HP}}$ and $\mathrm{MG_{TGP}}$ models against $\mathrm{Cart_{HP}}$ and $\mathrm{FG_{HP}}$, all of which utilize intelligent partitioning methods. Fig. 5b compares $\mathrm{MG_{HP}}$ and $\mathrm{MG_{TGP}}$ against $\mathrm{Cart_{RP}}$ [11], which utilizes random cartesian partitioning and hence involves negligible preprocessing overhead. For $\mathrm{Cart_{RP}}$ [11], we used the same parallel code as in [12] for a fair comparison. Fig. 5a displays the comparison for each tensor and for $R=64$, whereas Fig. 5b displays the comparison for $R=\{64, 32, 16\}$ on `flickr` and `facebook`, which have quite different partitioning overheads.

As seen in Fig. 5a, the proposed methods amortize the preprocessing overhead against other intelligent partitioning methods for $R=64$. As seen in Fig. 5b, the proposed intelligent partitioning methods amortize the preprocessing overhead even against $\mathrm{Cart_{RP}}$ [11], for which no partitioning overhead was considered, while the performance gap between the proposed methods and $\mathrm{Cart_{RP}}$ [11] closes with decreasing R value.

6 Conclusion

We introduced a general medium-grain partitioning framework for sparse tensor decomposition on distributed-memory architectures. The proposed framework avoids the drawbacks of the fine-grain model incurring a large number of messages and large hypergraphs. It also avoids the drawback of the cartesian partitioning method incurring a large number of constraints during partitioning which may limit the solution space. On average, the proposed medium-grain hypergraph and graph models respectively achieve 22 and 14 percent better parallel runtime on 512 processors with significantly less partitioning overhead compared to the state-of-the-art cartesian partitioning model.

Recent contributions in the literature towards improving MTTKRP computations such as CSF [9], HiCOO [13], dimension trees [14] and reordering methods [16], as well as increasing level of parallelism by distributing factor matrices in the dimension of rank [15] are orthogonal to the proposed method here and can be adopted for further performance improvement.

Acknowledgments

The authors would like to thank the referees for their valuable comments and efforts. They believe that their suggestions improved the quality and presentation of the article. This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under project EEEAG-116E043. They would also like to acknowledge PRACE for awarding us access to resource JUWELS based in Germany at Julich Supercomputing Centre. 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] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an explanatory multimodal factor analysis," *UCLA Working Papers Phonetics*, vol. 16, pp. 1–84, 1970.
[2] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, 2009.
[3] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Proc. Advances Neural Inf. Process. Syst.*, 2015, pp. 442–450.
[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 Trans. Signal Process.*, vol. 65, no. 13, pp. 3551–3582, Jul. 2017.
[5] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos, "Tag recommendations based on tensor dimensionality reduction," in *Proc. ACM Conf. Recommender Syst.*, 2008, pp. 43–50.
[6] S. Rendle and L. Schmidt-Thieme, "Pairwise interaction tensor factorization for personalized tag recommendation," in *Proc. 3rd ACM Int. Conf. Web Search Data Mining*, 2010, pp. 81–90.
[7] N. K. M. Faber, R. Bro, and P. K. Hopke, "Recent developments in CANDECOMP/PARAFAC algorithms: A critical review," *Chemometrics Intell. Lab. Syst.*, vol. 65, no. 1, pp. 119–137, 2003.
[8] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "SPLATT: Efficient and parallel sparse tensor-matrix multiplication," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 61–70.
[9] S. Smith and G. Karypis, "Tensor-matrix products with a compressed sparse tensor," in *Proc. 5th Workshop Irregular Appl.: Architectures Algorithms*, 2015, pp. 5:1–5:7.
[10] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2015, pp. 1–11.
[11] S. Smith and G. Karypis, "A medium-grained algorithm for sparse tensor factorization," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 902–911.
[12] S. Acer, T. Torun, and C. Aykanat, "Improving medium-grain partitioning for scalable sparse tensor decomposition," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 12, pp. 2814–2825, Dec. 2018.
[13] J. Li, J. Sun, and R. Vuduc, "HiCOO: Hierarchical storage of sparse tensors," in *Proc. Int. Conf. High Performance Comput. Netw. Storage Anal.*, 2018, pp. 238–252.

[14] O. Kaya and B. Uçar, "Parallel CANDECOMP/PARAFAC decomposition of sparse tensors using dimension trees," *SIAM J. Sci. Comput.*, vol. 40, no. 1, pp. C99–C130, 2018.
[15] J. Choi, X. Liu, S. Smith, and T. Simon, "Blocking optimization techniques for sparse tensor computation," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2018, pp. 568–577.
[16] J. Li, B. Uçar, Ü. V. Çatalyürek, J. Sun, K. Barker, and R. Vuduc, "Efficient and effective sparse tensor reordering," in *Proc. ACM Int. Conf. Supercomputing*, 2019, pp. 227–237.
[17] Ü. V. Çatalyürek and C. Aykanat, "A fine-grain hypergraph model for 2D decomposition of sparse matrices," in *Proc. 15th Int. Parallel Distrib. Process. Symp.*, 2001, pp. 1199–1204.
[18] B. Uçar and C. Aykanat, "Minimizing communication cost in fine-grain partitioning of sparse matrices," in *Proc. Int. Symp. Comput. Inf. Sci.*, 2003, pp. 926–933.
[19] Ü. V. Çatalyürek, C. Aykanat, and B. Uçar, "On two-dimensional sparse matrix partitioning: Models, methods, and a recipe," *SIAM J. Sci. Comput.*, vol. 32, no. 2, pp. 656–683, 2010.
[20] Ü. V. Çatalyürek and C. Aykanat, "A hypergraph-partitioning approach for coarse-grain decomposition," in *Proc. ACM/IEEE Conf. Supercomputing*, 2001, Art. no. 28.
[21] D. M. Pelt and R. H. Bisseling, "A medium-grain method for fast 2D bipartitioning of sparse matrices," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, 2014, pp. 529–539.
[22] Ü. V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 7, pp. 673–693, Jul. 1999.
[23] Ü. V. Çatalyürek and C. Aykanat, *PaToH (Partitioning Tool for Hypergraphs)*. Boston, MA, USA: Springer, 2011, pp. 1479–1487.
[24] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.
[25] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*. Rockville, MD, USA: Comput. Sci. Press, 1978.
[26] S. Smith *et al.*, FROSTT: The formidable repository of open sparse tensors and tools. 2017. [Online]. Available: http://frostt.io/
[27] I. Jeon, E. E. Papalexakis, C. Faloutsos, L. Sael, and U. Kang, "Mining billion-scale tensors: Algorithms and discoveries," *VLDB J.*, vol. 25, no. 4, pp. 519–544, Aug. 2016.
[28] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 1082–1090.
[29] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *Proc. 2nd ACM Workshop Online Social Netw.*, 2009, pp. 37–42.
[30] J. McAuley and J. Leskovec, "Hidden factors and hidden topics: understanding rating dimensions with review text," in *Proc. 7th ACM Conf. Recommender Syst.*, 2013, pp. 165–172.
[31] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Math. Program.*, vol. 91, no. 2, pp. 201–213, 2002.
[32] R. A. Harshman and M. E. Lundy, "The PARAFAC model for three-way factor analysis and multidimensional scaling," *Res. Methods Multimode Data Anal.*, vol. 46, pp. 122–215, 1984.
[33] N. Zheng, Q. Li, S. Liao, and L. Zhang, "Flickr group recommendation based on tensor decomposition," in *Proc. 33rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2010, pp. 737–738.

M. Ozan Karsavuran received the BS and MS degrees in computer engineering from Bilkent University, Turkey, in 2012 and 2014, respectively. He is currently working toward the PhD degree at Bilkent University. His research interests include combinatorial scientific computing, graph and hypergraph partitioning for sparse matrix and tensor computations, and parallel computing in distributed and shared memory systems.

Seher Acer received the BS, MS, and PhD degrees in computer engineering from Bilkent University, Turkey. She is currently a postdoctoral researcher with Center for Computing Research, Sandia National Laboratories, Albuquerque, New Mexico. 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 in electrical engineering from Middle East Technical University, Turkey, and the PhD degree in electrical and computer engineering from Ohio State University, Columbus. 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 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 the *IEEE Transactions of Parallel and Distributed Systems* between 2009 and 2013.