Improving Medium-Grain Partitioning for Scalable Sparse Tensor Decomposition

Seher Acer, Tugba Torun, and Cevdet Aykanat

Abstract—Tensor decomposition is widely used in the analysis of multi-dimensional data. The canonical polyadic decomposition (CPD) is one of the most popular decomposition methods and commonly found by the CPD-ALS algorithm. High computational and memory costs of CPD-ALS necessitate the use of a distributed-memory-parallel algorithm for efficiency. The medium-grain CPD-ALS algorithm, which adopts multi-dimensional cartesian tensor partitioning, is one of the most successful distributed CPD-ALS algorithms for sparse tensors. This is because cartesian partitioning imposes nice upper bounds on communication overheads. However, this model does not utilize the sparsity pattern of the tensor to reduce the total communication volume. The objective of this work is to fill this literature gap. We propose a novel hypergraph-partitioning model, CartHP, whose partitioning objective correctly encapsulates the minimization of total communication volume of multi-dimensional cartesian tensor partitioning. Experiments on twelve real-world tensors using up to 1024 processors validate the effectiveness of the proposed CartHP model. Compared to the baseline medium-grain model, CartHP achieves average reductions of 52, 43 and 24 percent in total communication volume, communication time and overall runtime of CPD-ALS, respectively.

Index Terms—Sparse tensor, canonical polyadic decomposition, cartesian partitioning, load balancing, communication volume, hypergraph partitioning

1 Introduction

Tensors are multi-dimensional arrays consisting of zero or more dimensions (modes). The applications that make use of tensors often benefit from tensor decomposition to discover the latent features of the modes. The most popular tensor decomposition method achieving this feat is the canonical polyadic decomposition (CPD) [1], [2], [3]. CPD is an extension of singular value decomposition for tensors and approximates a given tensor as a sum of rank-one tensors. CPD is successfully utilized in a large variety of applications from different domains, such as chemometrics [4], telecommunications [5], medical imaging [6], [7], image compression and analysis [8], text mining [9], [10], knowledge bases [11] and recommendation systems [12]. Kolda and Bader [3] provide an extensive survey on tensor decomposition methods and their applications.

One common method for computing CPD is the CPD-ALS algorithm, which exploits the alternating least squares method [13]. CPD-ALS includes a bottleneck operation called Matricized Tensor Times Khatri-Rao Product (MTTKRP), which requires significantly large amounts of computation and memory. This necessitates an efficient distributed-memory implementation for the CPD-ALS algorithm.

• *The authors are with the Computer Engineering Department, Bilkent University, Ankara 06800, Turkey.*
E-mail: {acer, tugba.uzluer}@bilkent.edu.tr, aykanat@cs.bilkent.edu.tr.

Manuscript received 26 Aug. 2017; revised 21 May 2018; accepted 23 May 2018. Date of publication 29 May 2018; date of current version 9 Nov. 2018.
(Corresponding author: Cevdet Aykanat.)
Recommended for acceptance by A. Kalyanaraman.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2841843

Recently, Smith and Karypis [14] have proposed a successful distributed-memory implementation of CPD-ALS algorithm. Their algorithm adopts a medium-grain model, in which a cartesian partition of the input tensor is utilized. Cartesian partitioning has the nice property of confining the communications to the layers of a virtual multi-dimensional processor mesh, thus providing upper bounds on communication overheads. Hence, this algorithm outperforms the earlier CPD-ALS implementations by achieving smaller parallel runtimes and better scalability.

In order to obtain a cartesian partition of the tensor, the medium-grain algorithm applies 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. However, this algorithm does not utilize the sparsity pattern of the tensor to minimize the total communication volume. The objective of this work is to fill this literature gap by proposing an intelligent partitioning algorithm that utilizes the sparsity pattern for minimizing the total communication volume of the medium-grain model. For this purpose, we exploit the conceptual similarity between MTTKRP and sparse matrix vector multiplication (SpMV), for which many partitioning models and methods with different granularities are well-studied [15], [16], [17], [18]. The 2D cartesian partitioning for parallel SpMV, which is known as checkerboard partitioning, was first introduced by Hendrickson et al. [19] and its total communication volume is minimized by a hypergraph partitioning (HP) model, CBHP, proposed by Çatalyürek and Aykanat [16], [20]. Relying on the similarity between MTTKRP and SpMV, extending CBHP for cartesian partitioning of tensors with more than two dimensions seems promising for minimizing the total communication volume of the medium-grain CPD-ALS.

CBHP is a two-phase HP model, where row and column partitions are respectively obtained in the first and second phases. The row partition obtained in the first phase implies a division information in each column. However, this column division information is not utilized in the topology of the hypergraph formed in the second phase. On the contrary, in the case of more than two dimensions, a slice's division information obtained in a phase needs to be utilized in each of the subsequent phases which further divide that slice. Note that this need does not arise for the two-dimensional case since each row/column is divided in exactly one phase. Since the direct extension of the CBHP model for tensor partitioning does not keep division history, it fails to correctly encapsulate the objective of minimizing the total communication volume.

In order to overcome the above-mentioned problem on extending the CBHP model for more than two dimensions, we propose a new hypergraph partitioning model in which hypergraph topologies contain the priori division information of slices. The partitioning objective of our model encapsulates the minimization of the total communication volume of the medium-grain CPD-ALS. To validate the proposed model, we conduct parallel experiments on 12 real-world tensors for up to 1,024 processors. Compared to the baseline medium-grain model [14], the proposed model achieves average reductions of 52, 43 and 24 percent in total communication volume, communication time and overall runtime of CPD-ALS, respectively.

The rest of the paper is organized as follows. Sections 2 and 3 provide the background information and related work, respectively. In Section 4, we propose a novel HP model, CartHP, for minimizing the total communication volume of medium-grain CPD-ALS. Section 5 provides the experimental results and Section 6 concludes. A discussion on the direct extension of CBHP for tensors and detailed performance results are given in the supplemental material as appendices, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2018.2841843.

2 Background

We denote tensors, matrices and vectors respectively by calligraphic ($\mathcal{X}$), bold capital ($\mathbf{A}$) and bold lowercase ($\mathbf{a}$) letters. To denote indices, we use lowercase letters ranging from 1 to their capital version, e.g., $q = 1, \ldots, Q$. To refer to a varying index, we use a semicolon as in Matlab notation, e.g., $\mathbf{A}(i,:)$.

2.1 Tensors

A tensor with M dimensions is called an M-mode tensor and mode m refers to the mth dimension. Unless specified, $\mathcal{X}$ is assumed to be a three-mode tensor of size $I \times J \times K$. The tensor element with indices i, j, k is denoted by $\mathcal{X}(i,j,k)$. Slices and fibers are defined as the subtensors obtained by holding one and two indices constant, respectively. $\mathcal{X}(i,:,:)$, $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$ respectively denote the ith horizontal (mode-1), jth lateral (mode-2) and kth frontal (mode-3) slices. The intersection of two slices along different modes (e.g., $\mathcal{X}(i,:,:)$ and $\mathcal{X}(:,j,:)$) constitutes a fiber (e.g., $\mathcal{X}(i,j,:)$). An M-mode tensor is called rank-one if it can be written as an outer product of M vectors. For instance, $(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})$ is a rank-one tensor. The matricization of $\mathcal{X}$ in mode m is denoted by $\mathbf{X}_{(m)}$.

2.2 Canonical Polyadic Decomposition

Canonical polyadic decomposition (CPD) with F components factorizes a given tensor $\mathcal{X}$ as a sum of F rank-one tensors: $\mathcal{X} \approx \sum_{f=1}^{F}(\mathbf{a}_f \circ \mathbf{b}_f \circ \mathbf{c}_f)$, where $\mathbf{a}_f$, $\mathbf{b}_f$ and $\mathbf{c}_f$ are column vectors of size I, J and K, respectively. Then, the factor matrices are defined as $\mathbf{A} = [\mathbf{a}_1 \ldots \mathbf{a}_F]$, $\mathbf{B} = [\mathbf{b}_1 \ldots \mathbf{b}_F]$ and $\mathbf{C} = [\mathbf{c}_1 \ldots \mathbf{c}_F]$. The columns of the factor matrices are stored as normalized to length one, where the actual lengths are stored in vector λ. Then, CPD of $\mathcal{X}$ is written in short as $\mathcal{X} \approx [\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$.

CPD-ALS, which is given in Algorithm 1, is an iterative algorithm. At each iteration, it solves a linear least squares problem to find a factor matrix, by fixing the other two factor matrices. For example in order to find $\mathbf{A}$, CPD-ALS solves $min_{\mathbf{A}}||\mathbf{X}_{(1)} - \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T||_F^2$ for fixed $\mathbf{B}$ and $\mathbf{C}$ by computing $\mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{-1}$. Here, $\odot$ and $*$ denote Khatri-Rao and Hadamard products, respectively. MTTKRP operations $\hat{\mathbf{A}} = \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})$, $\hat{\mathbf{B}} = \mathbf{X}_{(2)}(\mathbf{C} \odot \mathbf{A})$ and $\hat{\mathbf{C}} = \mathbf{X}_{(3)}(\mathbf{B} \odot \mathbf{A})$ constitute the bottleneck operations of CPD-ALS due to large sizes of matrices involved. In MTTKRP operation $\hat{\mathbf{A}} = \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})$, each row $\hat{\mathbf{A}}(i,:)$ can be computed as

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

The computation of $\hat{\mathbf{A}}(i,:)$ only involves the nonzeros in slice $\mathcal{X}(i,:,:)$ and for each nonzero $\mathcal{X}(i,j,k)$ in that slice, it requires rows $\mathbf{B}(j,:)$ and $\mathbf{C}(k,:)$.

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

```
1: Initialize matrices A, B and C randomly
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 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 B into $\lambda$
7:     $\mathbf{C} \leftarrow \mathbf{X}_{(3)}(\mathbf{B} \odot \mathbf{A})(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A})^{-1}$
8:     Normalize columns of C into $\lambda$
9: return $[\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$
```

2.3 Medium-Grain CPD-ALS Algorithm

The medium-grain CPD-ALS algorithm [14] is based on a 3D cartesian partition of a given tensor $\mathcal{X}$ for a virtual 3D mesh of $P = Q \times R \times S$ processors. In this partition, horizontal, lateral and frontal slices of $\mathcal{X}$ are partitioned among Q, R and S parts, respectively. These partitions are used for reordering the slices into Q horizontal, R lateral and S frontal chunks in such a way that the slices belonging to the same part are ordered consecutively (in any order) to form a chunk. The qth horizontal, rth lateral and sth frontal chunks are respectively denoted by $\mathcal{X}_{q,:,:}$, $\mathcal{X}_{:,r,:}$ and $\mathcal{X}_{:,:,s}$. The intersection of $\mathcal{X}_{q,:,:}$, $\mathcal{X}_{:,r,:}$ and $\mathcal{X}_{:,:,s}$ forms subtensor $\mathcal{X}_{q,r,s}$. Similarly, the qth horizontal, rth lateral and sth frontal layers of the virtual processor mesh are respectively denoted by $p_{q,:,:}$, $p_{:,r,:}$ and $p_{:,:,s}$. Chunks $\mathcal{X}_{q,:,:}$, $\mathcal{X}_{:,r,:}$ and $\mathcal{X}_{:,:,s}$ are respectively distributed among the processors of layers

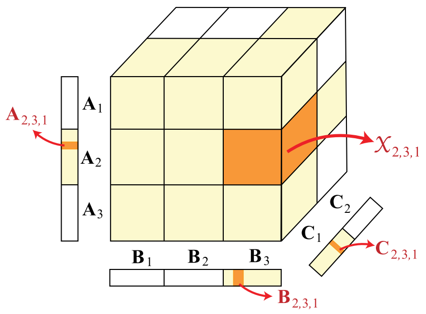


Fig. 1. A medium-grain partition for a $3 \times 3 \times 2$ virtual mesh of processors.

$p_{q,:,:}$, $p_{:,r,:}$ and $p_{:,:,s}$ in such a way that subtensor $\mathcal{X}_{q,r,s}$ is assigned to $p_{q,r,s}$.

A cartesian tensor partition induces a conformal partition of the rows of each factor matrix into chunks, e.g., $\mathbf{A}_1, \ldots, \mathbf{A}_Q$. The rows in the chunks $\mathbf{A}_q$, $\mathbf{B}_r$ and $\mathbf{C}_s$ are exclusively needed and updated by the processors in layers $p_{q,:,:}$, $p_{:,r,:}$ and $p_{:,:,s}$, respectively. The factor-matrix rows owned by processor $p_{q,r,s}$ are assumed to be contiguous and denoted by $\mathbf{A}_{q,r,s}$, $\mathbf{B}_{q,r,s}$ and $\mathbf{C}_{q,r,s}$.

Fig. 1 displays an example medium-grain partition with 3 horizontal, 3 lateral and 2 frontal chunks. Subtensor $\mathcal{X}_{2,3,1}$ as well as factor-matrix rows in $\mathbf{A}_{2,3,1}$, $\mathbf{B}_{2,3,1}$ and $\mathbf{C}_{2,3,1}$, which are all assigned to processor $p_{2,3,1}$, are highlighted with a darker shade. Note that $p_{2,3,1}$ may need to use the rest of the rows in $\mathbf{A}_2$, $\mathbf{B}_3$ and $\mathbf{C}_1$ during the MTTKRP operations.

The parallel medium-grain CPD-ALS algorithm consists of three phases at each iteration. The mth phase involves the computations and communications performed for computing the factor matrix along mode m. We only summarize the first phase since the other phases are similar. First, the MTTKRP operation is performed in a distributed fashion where each processor multiplies its nonzeros with the corresponding $\mathbf{B}$- and $\mathbf{C}$-matrix rows and produces partial results for the corresponding $\hat{\mathbf{A}}$-matrix rows as given in Equation (1). Here, $\hat{\mathbf{A}}$ and $\mathbf{A}$ have conformal partitions.

After performing the local MTTKRP operation, each processor $p_{q,r,s}$ sends its partial results for non-local $\hat{\mathbf{A}}$-matrix rows to their owner processors, which reside in layer $p_{q,:,:}$. In a dual manner, $p_{q,r,s}$ receives the partial results for its local $\hat{\mathbf{A}}$-matrix rows ($\hat{\mathbf{A}}_{q,r,s}$) from the processors in the same layer and sums them to finalize $\hat{\mathbf{A}}_{q,r,s}$. We refer to this communication step as the fold step. Then, $p_{q,r,s}$ multiplies $\hat{\mathbf{A}}_{q,r,s}$ with $(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{-1}$ and obtains $\mathbf{A}_{q,r,s}$. $\mathbf{A}$ is finalized by normalizing its columns using an all-to-all reduction on local norms. Then, $\mathbf{A}^T\mathbf{A}$ is obtained by another all-to-all reduction on locally computed $\mathbf{A}^T\mathbf{A}$ matrices.

Finally, each processor $p_{q,r,s}$ sends the updated rows in $\mathbf{A}_{q,r,s}$ to the processors that need these rows in the following two phases where $\mathbf{B}$ and $\mathbf{C}$ are computed. These processors are the ones that $p_{q,r,s}$ receives partial results from in the fold step. In a dual manner, $p_{q,r,s}$ receives the updated $\mathbf{A}$-matrix rows that it needs in the following two phases from their owner processors. These processors are the ones that $p_{q,r,s}$ sends partial results to in the fold step. We refer to this communication step as the expand step. At the end of each iteration, a residual is computed to test the convergence.

The communications in the fold and expand steps are confined to the processor layers. In the first, second and third phases, $p_{q,r,s}$ communicates with at most $R \times S - 1$, $Q \times S - 1$ and $Q \times R - 1$ processors residing in layers $p_{q,:,:}$, $p_{:,r,:}$ and $p_{:,:,s}$, respectively.

2.4 Hypergraph Partitioning Problem

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 connects a subset of vertices, which is denoted by $Pins(n)$. Each vertex v is assigned a weight of $w(v)$, whereas each net is assigned a cost of $c(n)$. $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is a K-way partition of $\mathcal{H}$ if parts are mutually disjoint and exhaustive. The cutsize of a given Π is defined as $\sum_{n \in \mathcal{N}} (\lambda(n) - 1)c(n)$, where $\lambda(n)$ denotes the number of parts connected by n.

The hypergraph partitioning (HP) problem is defined as finding a K-way partition Π of a given hypergraph $\mathcal{H}$ with the objective of minimizing the cutsize and the constraint of maintaining balance on the weights of the parts. In the case of multi-constraint hypergraph partitioning with C constraints, the cth constraint for $c = 1, 2, \ldots, C$ is formulated as $W_c(\mathcal{V}_k) \leq W_c^{tot}(1+\epsilon)/K$. Here, $W_c(\mathcal{V}_k)$ and W_c^{tot} denote the sums of the cth weights of the vertices in $\mathcal{V}_k$ and $\mathcal{V}$, respectively, whereas ϵ denotes a maximum allowable imbalance ratio.

3 Related Work

There are several distributed-memory CPD-ALS parallelization approaches for sparse tensors, varying on how they define and distribute atomic tasks. DFacTo [21] obtains a coarse-grain partition of the tensor by performing an independent one-dimensional block partitioning along each mode and is reported to be significantly faster than two earlier alternatives, Tensor Toolbox [22] and GigaTensor [23], when compared in a sequential setting. However, DFacTo is not memory scalable since it needs to store the matricized tensor along each mode as well as all factor matrices at each processor.

Kaya and Uçar [24] propose HP models that exploit the sparsity pattern of the tensor to minimize the total communication volumes of coarse- and fine-grain tensor partitionings. The coarse-grain HP model does not lead to a significant reduction in the total communication volume compared to block partitioning. This is due to the inherent limitation of coarse-grain partitioning, where each processor may need all factor-matrix rows in the non-partitioned modes. The fine-grain HP model overcomes this problem by distributing the tensor nonzeros individually, obtaining a multi-dimensional partition. The major drawback of the fine-grain model is the overhead of partitioning a large hypergraph containing vertices at least as many as the number of tensor nonzeros. The fine-grain HP model also suffers from inducing high number of messages, which is a consequence of disturbing the slice coherences.

To overcome these performance bottlenecks of coarse- and fine-grain models, Smith and Karypis [14] propose a successful medium-grain model which is based on multi-dimensional cartesian tensor partitioning. This cartesian tensor partitioning is also used by Austin et al. [25] for parallel Tucker decomposition.

4 Optimizing Medium-Grain CPD-ALS

Here, we first describe the communication volume requirement of a given cartesian partition of a three-mode tensor.

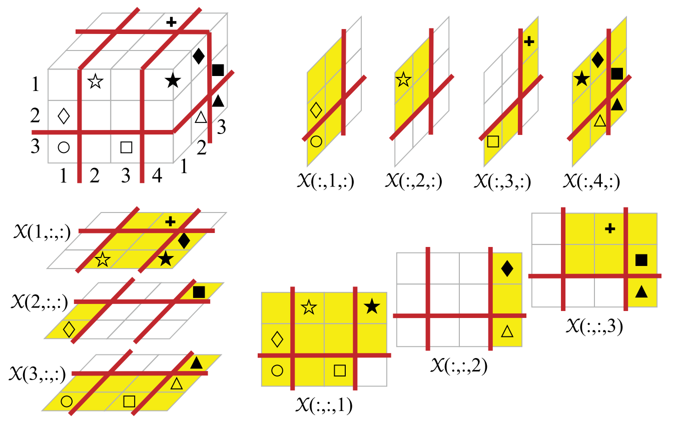


Fig. 2. A 3D cartesian partition of a $3 \times 4 \times 3$ tensor for a $2 \times 3 \times 2$ virtual processor mesh.

Then, we propose an HP model, referred to as CartHP, for obtaining a 3D cartesian partition of the tensor with minimum communication volume. Finally, we briefly discuss the extension of the proposed model to more than three modes.

4.1 Communication Volume Requirement

A given cartesian partition of the tensor divides each slice/fiber into subslices/subfibers, each of which is owned by a different processor. We denote any (sub)tensor γ owned by a set α of processor(s) by γ_α. For instance, $\mathcal{X}(i,:,:)_{q,r,s}$ and $\mathcal{X}(i,j,:)_{q,r,s}$ respectively denote the subslice of $\mathcal{X}(i,:,:)$ and the subfiber of $\mathcal{X}(i,j,:)$ which are owned by processor $p_{q,r,s}$. Similarly, $\mathcal{X}(i,:,:)_{:,r,:}$ denotes the subslice of $\mathcal{X}(i,:,:)$ owned by processor layer $p_{:,r,:}$. To differentiate the subslices owned by a single processor from those owned by multiple processors, we refer to the former ones as *unshared* subslices. A (sub) slice/(sub)fiber containing at least one nonzero element is called a *nonzero* (sub)slice/(sub)fiber. Fig. 2 displays a cartesian partition of a $3 \times 4 \times 3$ tensor for a $2 \times 3 \times 2$ virtual processor mesh and the respective divisions of slices into subslices induced by this partition. In this figure, each tensor nonzero is denoted by a different symbol and each nonzero subslice is highlighted. For example, slice $\mathcal{X}(1,:,:)$ contains 4 nonzero elements and 3 nonzero unshared subslices.

Let Z_i^A, Z_j^B and Z_k^C respectively denote the sets of nonzero unshared subslices of $\mathcal{X}(i,:,:)$, $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$. For the example given in Fig. 2, $Z_1^A = \{\mathcal{X}(1,:,:)_{1,2,1}, \mathcal{X}(1,:,:)_{1,2,2}, \mathcal{X}(1,:,:)_{1,3,1}\}$. We assume that each slice contains at least one nonzero, hence, these sets are nonempty. In the first phase of medium-grain CPD-ALS, only the processors that own a subslice in Z_i^A produce partial results for $\hat{\mathbf{A}}(i,:)$. Similarly in the second and third phases, only the processors that own a subslice in Z_j^B and Z_k^C produce partial results for $\hat{\mathbf{B}}(j,:)$ and $\hat{\mathbf{C}}(k,:)$, respectively.

We assume that each factor-matrix row is assigned to a processor which owns a nonzero subslice in the corresponding slice. We refer to this assumption as the *consistency condition* for the correctness of our hypergraph model to be proposed in Section 4.2. Let $\hat{\mathbf{A}}(i,:)$ be assigned to a processor, say p, that owns a nonzero subslice in Z_i^A. Each of the other processors that own a nonzero subslice in Z_i^A sends a partial result for $\hat{\mathbf{A}}(i,:)$ to p in the fold step. Then, the communication volume regarding the fold operation on $\hat{\mathbf{A}}(i,:)$ amounts to $(|Z_i^A|-1)F$. In a dual manner, p sends the updated row $\mathbf{A}(i,:)$ to these processors in the expand step, incurring a communication of volume $(|Z_i^A|-1)F$ again. Since the same volume of communication is incurred regarding the expand operation on $\mathbf{A}(i,:)$ and the fold operation on $\hat{\mathbf{A}}(i,:)$, we only consider the one regarding $\hat{\mathbf{A}}(i,:)$ and formulate it as

$$vol_i^A = (|Z_i^A| - 1)F. \tag{2}$$

Then, the total volume in the first phase is the sum of the volumes regarding the rows of $\hat{\mathbf{A}}$, that is

$$vol^A = \sum_{i=1}^{I} vol_i^A = \left(\sum_{i=1}^{I}(|Z_i^A| - 1)\right)F.$$

With similar discussions for the second and third phases, we obtain $vol^B = (\sum_{j=1}^{J}(|Z_j^B| - 1))F$ and $vol^C = (\sum_{k=1}^{K}(|Z_k^C| - 1))F$. Then, $vol^A + vol^B + vol^C$ gives the overall total volume per iteration.

In Fig. 2, the volume of communication regarding $\hat{\mathbf{A}}(1,:)$ is $vol_1^A = (|Z_1^A| - 1)F = 2F$. The total volume in the first phase is $vol^A = (2+1+3)F = 6F$ and the overall total volume is $(6+5+7)F = 18F$.

4.2 CartHP: Proposed HP Model

For a given tensor $\mathcal{X}$ and a $Q \times R \times S$ virtual mesh of processors, CartHP contains partitioning phases $\phi1$, $\phi2$ and $\phi3$, in which hypergraphs $\mathcal{H}^A$, $\mathcal{H}^B$ and $\mathcal{H}^C$ are constructed with vertex sets representing the horizontal, lateral and frontal slices of $\mathcal{X}$, respectively. In $\phi1$, CartHP obtains a Q-way partition of $\mathcal{H}^A$ and uses this partition to reorder the horizontal slices to form Q horizontal chunks. These horizontal chunks divide each lateral and frontal slice into Q subslices along mode 1. Similarly in $\phi2$, CartHP obtains an R-way partition of $\mathcal{H}^B$ and uses this partition to reorder the lateral slices to form R lateral chunks. These lateral chunks divide each horizontal slice into R subslices along mode 2 and each frontal subslice into R subsubslices along mode 2. Note that each frontal slice has $Q \times R$ subsubslices at the end of $\phi2$. Finally in $\phi3$, CartHP obtains an S-way partition of $\mathcal{H}^C$ and uses this partition to reorder the frontal slices to form S frontal chunks. These frontal chunks divide each horizontal and lateral subslice into S subsubslices along mode 3. Note that each horizontal and lateral slice have $R \times S$ and $Q \times S$ subsubslices at the end of $\phi3$, respectively. Fig. 3 illustrates a tensor which is partitioned by CartHP for a $3 \times 4 \times 2$ virtual processor mesh and three sample slices along different modes.

Algorithm 2. CartHP

```
Require: tensor $\mathcal{X}$, 3D processor mesh size $Q \times R \times S$, imbalance
    ratios $\epsilon_1, \epsilon_2, \epsilon_3$
1: $\phi1(\mathcal{X}, Q, \epsilon_1)$ obtains $Q$ horizontal chunks
2: $\phi2(\mathcal{X}, R, \epsilon_2)$ obtains $R$ lateral chunks
3: $\phi3(\mathcal{X}, S, \epsilon_3)$ obtains $S$ frontal chunks
4: for each subtensor $\mathcal{X}_{q,r,s}$ do
5:     Assign $\mathcal{X}_{q,r,s}$ to processor $p_{q,r,s}$
```

Algorithm 2 displays the basic layout of CartHP. Here, we abuse the notation for simplicity and use the same symbol $\mathcal{X}$ for the original tensor (line 1) and the reordered tensors (lines 2-3). Consequently, each subtensor $\mathcal{X}_{q,r,s}$ (line 4)

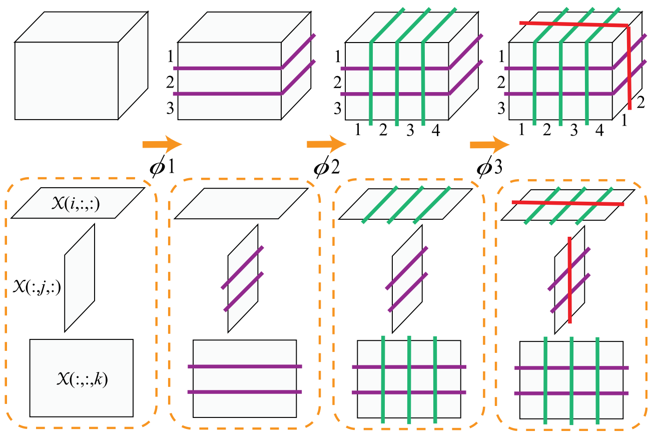


Fig. 3. Slice chunks obtained in phases $\phi 1$, $\phi 2$ and $\phi 3$ and (sub)subslices of $\mathcal{X}(i,:,:)$, $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$ divided by these chunks.

is the intersection of the respective chunks of the reordered tensor.

Algorithm 3 displays phase $\phi 1$, in which we construct (lines 1-13) and partition (line 14) $\mathcal{H}^A = (\mathcal{V}^A, \mathcal{N}^B \cup \mathcal{N}^C)$ to obtain Q horizontal chunks (lines 15-17). In $\mathcal{H}^A$, $\mathcal{V}^A = \{v_1^A, \ldots, v_I^A\}$ contains a vertex v_i^A for each horizontal slice $\mathcal{X}(i,:,:)$. $\mathcal{N}^B$ contains a net n_j^B for each nonzero lateral slice $\mathcal{X}(:,j,:)$, whereas $\mathcal{N}^C$ contains a net n_k^C for each nonzero frontal slice $\mathcal{X}(:,:,k)$. Since all slices are assumed to have at least one nonzero element, $\mathcal{N}^B = \{n_1^B, \ldots, n_J^B\}$ and $\mathcal{N}^C = \{n_1^C, \ldots, n_K^C\}$. Net n_j^B connects vertex v_i^A if the intersection of $\mathcal{X}(i,:,:)$ and $\mathcal{X}(:,j,:)$ contains at least one nonzero (lines 7-8). Similarly, n_k^C connects v_i^A if the intersection of $\mathcal{X}(i,:,:)$ and $\mathcal{X}(:,:,k)$ has at least one nonzero (lines 11-12). Each vertex v_i^A is assigned a single weight $w(v_i^A) = nnz(\mathcal{X}(i,:,:))$ (lines 2-3). Here, $nnz(\cdot)$ denotes the number of nonzeros of the given (sub)tensor. Then, a Q-way partition Π^A of $\mathcal{H}^A$ is obtained (line 14).

Algorithm 3. $\phi 1(\mathcal{X}, Q, \epsilon_1)$

```
1: $\mathcal{V}^A \leftarrow \{v_1^A, \ldots, v_I^A\}$
2: for each horizontal slice $\mathcal{X}(i,:,:)$ do
3:     $w(v_i^A) \leftarrow nnz(\mathcal{X}(i,:,:))$
4: $\mathcal{N}^B \leftarrow \mathcal{N}^C \leftarrow \emptyset$
5: for each lateral slice $\mathcal{X}(:,j,:)$ do
6:     $\mathcal{N}^B \leftarrow \mathcal{N}^B \cup \{n_j^B\}$ with $Pins(n_j^B) = \emptyset$
7:     for each nonzero fiber $\mathcal{X}(i,j,:)$ do
8:         $Pins(n_j^B) \leftarrow Pins(n_j^B) \cup \{v_i^A\}$
9: for each frontal slice $\mathcal{X}(:,:,k)$ do
10:    $\mathcal{N}^C \leftarrow \mathcal{N}^C \cup \{n_k^C\}$ with $Pins(n_k^C) = \emptyset$
11:    for each nonzero fiber $\mathcal{X}(i,:,k)$ do
12:        $Pins(n_k^C) \leftarrow Pins(n_k^C) \cup \{v_i^A\}$
13: $\mathcal{H}^A \leftarrow (\mathcal{V}^A, \mathcal{N}^B \cup \mathcal{N}^C)$
14: $\Pi^A = \{\mathcal{V}_1^A, \ldots, \mathcal{V}_Q^A\} \leftarrow$HP$(\mathcal{H}^A, Q, \epsilon_1)$
15: for $q \leftarrow 1$ to $Q$ do
16:    for each $v_i^A \in \mathcal{V}_q^A$ do
17:        Assign slice $\mathcal{X}(i,:,:)$ to chunk $\mathcal{X}_{q,:,:}$
```

Algorithm 4 displays phase $\phi 2$, in which we construct (lines 1-13) and partition (line 14) $\mathcal{H}^B = (\mathcal{V}^B, \mathcal{N}^A \cup \mathcal{N}^C)$ to obtain R lateral chunks (lines 15-17). In $\mathcal{H}^B$, $\mathcal{V}^B = \{v_1^B, \ldots, v_J^B\}$ contains a vertex v_j^B for each lateral slice $\mathcal{X}(:,j,:)$. $\mathcal{N}^A$ contains a net n_i^A for each nonzero horizontal slice $\mathcal{X}(i,:,:)$, that is, $\mathcal{N}^A = \{n_i^A, \ldots, n_I^A\}$. Net n_i^A connects vertex v_j^B if the intersection of $\mathcal{X}(:,j,:)$ and $X(i,:,:)$ contains at least one nonzero (lines 7-8). The nets in $\mathcal{N}^A$ are similar to those in $\phi 1$ since horizontal slices are not yet divided into subslices. Frontal slices, on the other hand, have been divided into Q subslices along mode 1 by the horizontal chunks formed in $\phi 1$. Instead of a single net, each frontal slice $\mathcal{X}(:,:,k)$ is represented by a number of nets as many as the number of its nonzero subslices. $\mathcal{N}^C$ contains a net $n_{k(q)}^C$ for each nonzero subslice $\mathcal{X}(:,:,k)_{q,:,:}$ (lines 9-10). We only include nets for nonzero subslices as the zero subslices do not incur any increase in the number of nonzero unshared subslices. Net $n_{k(q)}^C$ connects vertex v_j^B if the intersection of $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)_{q,:,:}$ contains at least one nonzero (lines 11-12). Since each slice $\mathcal{X}(:,j,:)$ contains Q subslices, each vertex v_j^B is assigned Q weights $w_q(v_j^B) = nnz(\mathcal{X}(:,j,:)_{q,:,:})$ for $q = 1, \ldots, Q$ (lines 2-3). Then, an R-way partition Π^B of $\mathcal{H}^B$ is obtained by multi-constraint HP (MC-HP) (line 14).

Algorithm 4. $\phi 2(\mathcal{X}, R, \epsilon_2)$

```
1: $\mathcal{V}^B \leftarrow \{v_1^B, \ldots, v_J^B\}$
2: for each lateral subslice $\mathcal{X}(:,j,:)_{q,:,:}$ do
3:     $w_q(v_j^B) \leftarrow nnz(\mathcal{X}(:,j,:)_{q,:,:})$
4: $\mathcal{N}^A \leftarrow \mathcal{N}^C \leftarrow \emptyset$
5: for each horizontal slice $\mathcal{X}(i,:,:)$ do
6:     $\mathcal{N}^A \leftarrow \mathcal{N}^A \cup \{n_i^A\}$ with $Pins(n_i^A) = \emptyset$
7:     for each nonzero fiber $\mathcal{X}(i,j,:)$ do
8:         $Pins(n_i^A) \leftarrow Pins(n_i^A) \cup \{v_j^B\}$
9: for each nonzero frontal subslice $\mathcal{X}(:,:,k)_{q,:,:}$ do
10:    $\mathcal{N}^C \leftarrow \mathcal{N}^C \cup \{n_{k(q)}^C\}$ with $Pins(n_{k(q)}^C) = \emptyset$
11:    for each nonzero subfiber $\mathcal{X}(:,j,k)_{q,:,:}$ do
12:        $Pins(n_{k(q)}^C) \leftarrow Pins(n_{k(q)}^C) \cup \{v_j^B\}$
13: $\mathcal{H}^B \leftarrow (\mathcal{V}^B, \mathcal{N}^A \cup \mathcal{N}^C)$
14: $\Pi^B = \{\mathcal{V}_1^B, \ldots, \mathcal{V}_R^B\} \leftarrow$ MC-HP$(\mathcal{H}^B, R, \epsilon_2)$
15: for $r \leftarrow 1$ to $R$ do
16:    for each $v_j^B \in \mathcal{V}_r^B$ do
17:        Assign slice $\mathcal{X}(:,j,:)$ to chunk $\mathcal{X}_{:,r,:}$
```

Algorithm 5. $\phi 3(\mathcal{X}, S, \epsilon_3)$

```
1: $\mathcal{V}^C \leftarrow \{v_1^C, \ldots, v_K^C\}$
2: for each frontal subslice $\mathcal{X}(:,:,k)_{q,r,:}$ do
3:     $w_{q,r}(v_k^C) \leftarrow nnz(\mathcal{X}(:,:,k)_{q,r,:})$
4: $\mathcal{N}^A \leftarrow \mathcal{N}^B \leftarrow \emptyset$
5: for eachnonzerohorizontalsubslice$\mathcal{X}(i,:,:)_{:,r,:}$ do
6:     $\mathcal{N}^A \leftarrow \mathcal{N}^A \cup \{n_{i(r)}^A\}$ with $Pins(n_{i(r)}^A) = \emptyset$
7:     for each nonzero subfiber $\mathcal{X}(i,:,k)_{:,r,:}$ do
8:         $Pins(n_{i(r)}^A) \leftarrow Pins(n_{i(r)}^A) \cup \{v_k^C\}$
9: for each nonzero lateral subslice $\mathcal{X}(:,j,:)_{q,:,:}$ do
10:    $\mathcal{N}^B \leftarrow \mathcal{N}^B \cup \{n_{j(q)}^C\}$ with $Pins(n_{j(q)}^C) = \emptyset$
11:    for each nonzero subfiber $\mathcal{X}(:,j,k)_{q,:,:}$ do
12:        $Pins(n_{j(q)}^B) \leftarrow Pins(n_{j(q)}^B) \cup \{v_k^C\}$
13: $\mathcal{H}^C \leftarrow (\mathcal{V}^C, \mathcal{N}^A \cup \mathcal{N}^B)$
14: $\Pi^C = \{\mathcal{V}_1^C, \ldots, \mathcal{V}_S^C\} \leftarrow$ MC-HP$(\mathcal{H}^C, S, \epsilon_3)$
15: for $s \leftarrow 1$ to $S$ do
16:    for each $v_k^C \in \mathcal{V}_s^C$ do
17:        Assign slice $\mathcal{X}(:,:,k)$ to chunk $\mathcal{X}_{:,:,s}$
```

Algorithm 5 displays phase $\phi 3$, in which we construct (lines 1-13) and partition (line 14) $\mathcal{H}^C = (\mathcal{V}^C, \mathcal{N}^A \cup \mathcal{N}^B)$ to obtain S frontal chunks (lines 15-17). In $\mathcal{H}^C$, $\mathcal{V}^C =$

$\{v_1^C, \ldots, v_K^C\}$ contains a vertex v_k^C for each frontal slice $\mathcal{X}(:,:,k)$. As in $\phi 2$, each divided slice is represented by a number of nets as many as the number of its nonzero subslices. Note that each horizontal slice has been divided into R subslices along mode 2 in $\phi 2$, whereas each lateral slice has been divided into Q subslices along mode 1 in $\phi 1$. $\mathcal{N}^A$ contains a net $n_{i(r)}^A$ for each nonzero subslice $\mathcal{X}(i,:,:)_{:,r,:}$, whereas $\mathcal{N}^B$ contains a net $n_{j(q)}^B$ for each nonzero subslice $\mathcal{X}(:,j,:)_{q,:,:}$ (lines 5-6 and 9-10). Net $n_{i(r)}^A$ connects vertex v_k^C if the intersection of $\mathcal{X}(:,:,k)$ and $\mathcal{X}(i,:,:)_{:,r,:}$ contains at least one nonzero (lines 7-8). Similarly, $n_{j(q)}^B$ connects v_k^C if the intersection of $\mathcal{X}(:,:,k)$ and $\mathcal{X}(:,j,:)_{q,:,:}$ contains at least one nonzero (lines 11-12). Since each slice $\mathcal{X}(:,:,k)$ contains $Q \times R$ subsubslices, each vertex v_k^C is assigned $Q \times R$ weights $w_{q,r}(v_k^C) = nnz(\mathcal{X}(:,:,k)_{q,r,:})$ for $q=1,\ldots,Q$ and $r=1,\ldots,R$ (lines 2-3). Then, an S-way partition Π^C of $\mathcal{H}^C$ is obtained by MC-HP (line 14).

All nets in the hypergraphs constructed in our model are assigned a cost of F. That is, $c(n) = F$ for each net n in $\mathcal{H}^A$, $\mathcal{H}^B$ and $\mathcal{H}^C$.

In partitioning $\mathcal{H}^A$, $\mathcal{H}^B$ and $\mathcal{H}^C$, the maximum allowed imbalance ratios are set to ϵ_1, ϵ_2 and ϵ_3, respectively. It can be shown that at the end of three partitoning phases, the number of nonzeros assigned to a processor is bounded above by

$$(1/P)nnz(\mathcal{X})(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3). \tag{3}$$

The derivation of Equation (3) is given in Appendix B, available in the online supplemental material.

Fig. 4 illustrates an example for CartHP applied on a $4 \times 4 \times 3$ tensor $\mathcal{X}$ for a $2 \times 2 \times 2$ virtual mesh of processors. The vertices that represent horizontal, lateral and frontal slices are colored with purple, green and red, respectively. The same color encoding also applies to the nets in each phase. In ϕm, the tensor is displayed in terms of mode-m slices. For each hypergraph, the array of weights associated to each vertex/part is displayed next to the corresponding vertex/part. For example, consider v_3^B in $\phi 2$. Vertex v_3^B is connected by nets n_2^A and n_4^A due to nonzero fibers $\mathcal{X}(2,3,:)$ and $\mathcal{X}(4,3,:)$, respectively, and by nets $n_{1(2)}^C$, $n_{2(1)}^C$ and $n_{3(2)}^C$ due to nonzero subfibers $\mathcal{X}(:,3,1)_{2,:,:}$, $\mathcal{X}(:,3,2)_{1,:,:}$ and $\mathcal{X}(:,3,3)_{2,:,:}$, respectively. Since $nnz(\mathcal{X}(:,3,:)_{1,:,:})=1$ and $nnz(\mathcal{X}(:,3,:)_{2,:,:})=2$, $w_1(v_3^B)=1$ and $w_2(v_3^B)=2$. Since $v_3^B \in \mathcal{V}_1^B$ in Π^B, slice $\mathcal{X}(:,3,:)$ is reordered in chunk $\mathcal{X}_{:,1,:}$.

4.3 Correctness of CartHP

In this section, we show the correctness of the proposed CartHP model in minimizing the total communication volume of medium-grain CPD-ALS.

Suppose that we have a cartesian partition of $\mathcal{X}$ obtained by CartHP and consider a horizontal slice $\mathcal{X}(i,:,:)$. Note that $\mathcal{X}(i,:,:)$ is not divided into any subslices in $\phi 1$. In $\phi 2$, $\mathcal{X}(i,:,:)$ is divided into R subslices $\mathcal{X}(i,:,:)_{:,r,:}$ for $r=1,\ldots,R$ along mode 2. Let $Z^B(i,:,:)$ denote the set of mode-2 indices of the nonzero subslices among these R subslices, i.e.,

$$Z^B(i,:,:) = \{r \mid \mathcal{X}(i,:,:)_{:,r,:} \text{ is a nonzero subslice}\}.$$

Note that $Z^B(i,:,:) \subseteq \{1,\ldots,R\}$. For example in Fig. 4, $Z^B(1,:,:) = \{1,2\}$ and $Z^B(2,:,:) = \{1\}$. In $\phi 3$, each subslice $\mathcal{X}(i,:,:)_{:,r,:}$ is divided into S subsubslices $\mathcal{X}(i,:,:)_{:,r,s}$ for $s=1,\ldots,S$ along mode 3. Let $Z^C(i,:,:)_{:,r,:}$ denote the set of mode-3 indices of the nonzero subslices among these S subsubslices, that is

$$Z^C(i,:,:)_{:,r,:} = \{s \mid \mathcal{X}(i,:,:)_{:,r,s} \text{ is a nonzero subslice}\}.$$

Note that $Z^C(i,:,:)_{:,r,:} \subseteq \{1,\ldots,S\}$. For example in Fig. 4, $Z^C(1,:,:)_{:,1,:} = \{1\}$ and $Z^C(1,:,:)_{:,2,:} = \{2\}$.

$\mathcal{X}(i,:,:)$ is represented by a single net n_i^A in $\phi 2$ and by at most R nets $n_{i(r)}^A$ in $\phi 3$, but not represented by any nets in $\phi 1$. Let $cs_i^A(\phi)$ denote the total cutsize incurred by the nets representing $\mathcal{X}(i,:,:)$ in a phase ϕ. Since $\lambda(n_i^A)$ in $\phi 2$ amounts to the number of $\mathcal{X}(i,:,:)$'s nonzero subslices along mode 2, which is $|Z^B(i,:,:)|$, the cutsize incurred by n_i^A in $\phi 2$ is

$$cs_i^A(\phi 2) = (\lambda(n_i^A)-1)c(n_i^A) = (|Z^B(i,:,:)|-1)F.$$

$\lambda(n_{i(r)}^A)$ in $\phi 3$ amounts to the number of $\mathcal{X}(i,:,:)_{:,r,:}$'s nonzero unshared subslices, which is $|Z^C(i,:,:)_{:,r,:}|$. Then, the total cutsize incurred by the nets representing $\mathcal{X}(i,:,:)$ in $\phi 3$ is

$$\begin{aligned} cs_i^A(\phi^3) &= \sum\nolimits_{r\in Z^B(i,:,:)} (\lambda(n_{i(r)}^A)-1)c(n_{i(r)}^A) \\ &= \sum\nolimits_{r\in Z^B(i,:,:)} (|Z^C(i,:,:)_{:,r,:}|-1)F \\ &= \Big(\sum\nolimits_{r\in Z^B(i,:,:)} |Z^C(i,:,:)_{:,r,:}| - |Z^B(i,:,:)|\Big)F. \end{aligned}$$

Let cs_i^A denote the total cutsize incurred by the nets representing $\mathcal{X}(i,:,:)$ in all phases. Since $cs_i^A = cs_i^A(\phi 2) + cs_i^A(\phi 3)$ and the term $|Z^B(i,:,:)|F$ is cancelled out in this summation, we obtain

$$cs_i^A = \left(\sum_{r\in Z^B(i,:,:)} |Z^C(i,:,:)_{:,r,:}| - 1\right)F.$$

Note that the sum of the number of nonzero subsubslices in $Z^C(i,:,:)_{:,r,:}$ for all $r \in Z^B(i,:,:)$ gives the total number of unshared subslices in $\mathcal{X}(i,:,:)$. Then

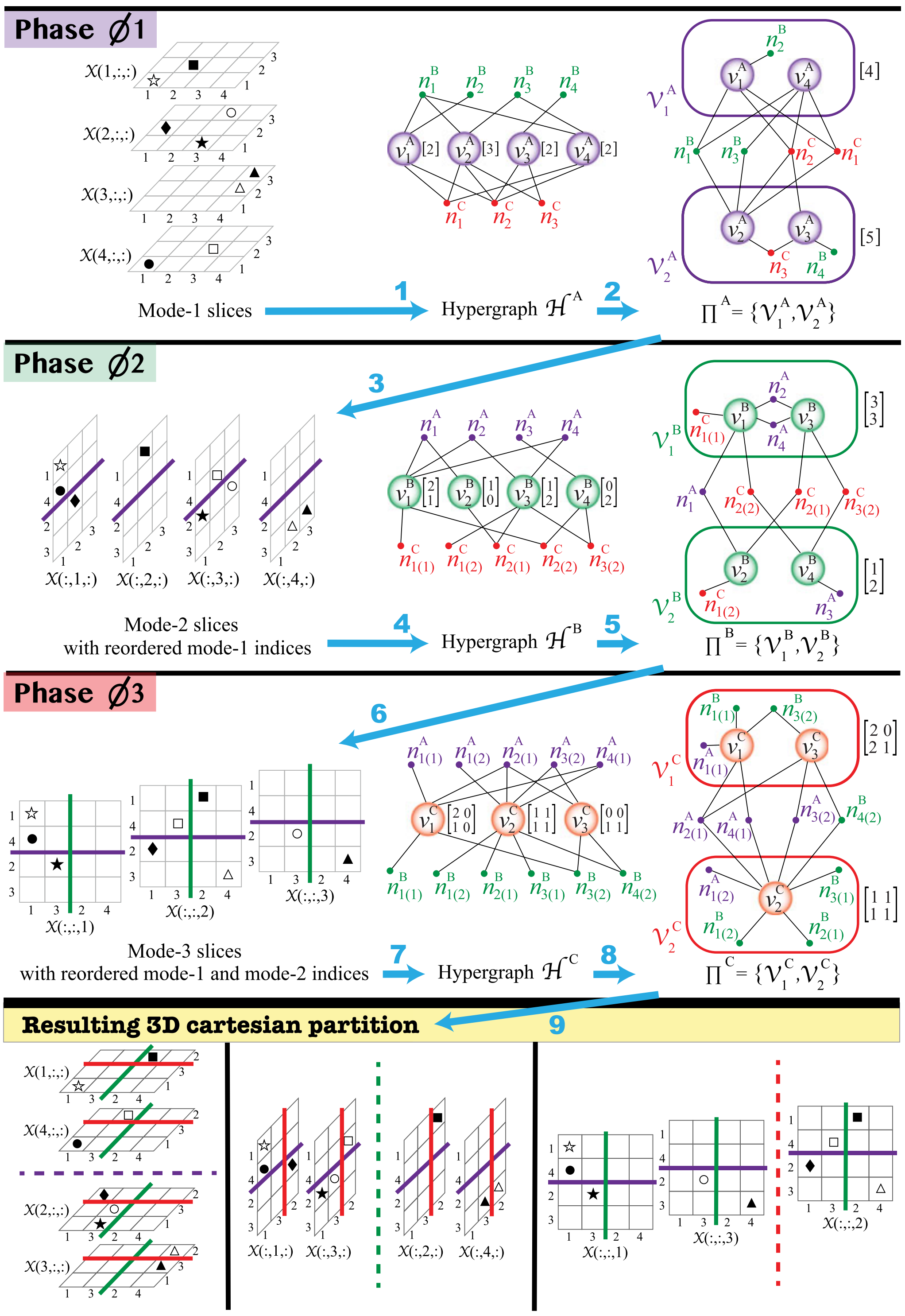
$$cs_i^A = (|Z_i^A|-1)F. \tag{4}$$

By Equations (2) and (4), we obtain

$$cs_i^A = vol_i^A.$$

These findings apply to mode-2 and mode-3 slices as follows: $cs_j^B = cs_j^B(\phi 1) + cs_j^B(\phi 3)$ and $cs_k^C = cs_k^C(\phi 1) + cs_k^C(\phi 2)$, where cs_j^B and cs_k^C denote total cutsizes incurred by the nets representing $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$ in all phases, respectively. Then, $cs_j^B = (|Z_j^B|-1)F = vol_j^B$ and $cs_k^C = (|Z_k^C|-1)F = vol_k^C$. That is, the total cutsizes incurred by the nets representing $\mathcal{X}(i,:,:)$, $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$ are equal to the communication volumes regarding factor-matrix rows $\hat{\mathbf{A}}(i,:)$, $\hat{\mathbf{B}}(j,:)$ and $\hat{\mathbf{C}}(k,:)$, respectively. Since the overall cutsize of CartHP is equal to the sum of the cutsizes of the nets representing individual slices in all phases, minimizing the overall cutsize corresponds to minimizing the total communication volume.

In Fig. 4, consider slice $\mathcal{X}(:,:,2)$ and the nets that represent this slice. In $\phi 1$, the cutsize incurred by n_2^C is $cs_2^C(\phi 1) = (2-1)F = F$. In $\phi 2$, the total cutsize incurred by



Phase ϕ1
Mode-1 slices
Hypergraph $\mathcal{H}^A$
$\Pi^A = \{\mathcal{V}_1^A, \mathcal{V}_2^A\}$
Phase ϕ2
Mode-2 slices with reordered mode-1 indices
Hypergraph $\mathcal{H}^B$
$\Pi^B = \{\mathcal{V}_1^B, \mathcal{V}_2^B\}$
Phase ϕ3
Mode-3 slices with reordered mode-1 and mode-2 indices
Hypergraph $\mathcal{H}^C$
$\Pi^C = \{\mathcal{V}_1^C, \mathcal{V}_2^C\}$
Resulting 3D cartesian partition

Fig. 4. CartHP on a $4\times4\times3$ tensor $\mathcal{X}$ for a $2\times2\times2$ virtual mesh of processors. $\phi1$: Horizontal slices of the original tensor, hypergraph $\mathcal{H}^A$ and a 2-way partition Π^A of $\mathcal{H}^A$. $\phi2$: Lateral slices of the tensor with reordered mode-1 indices, hypergraph $\mathcal{H}^B$ and a 2-way partition Π^B of $\mathcal{H}^B$. $\phi3$: Frontal slices of the tensor with reordered mode-1 and mode-2 indices, hypergraph $\mathcal{H}^C$ and a 2-way partition Π^C of $\mathcal{H}^C$. Bottom: Slices of the final tensor reordered along all modes.

nets $n^C_{2(1)}$ and $n^C_{2(2)}$ is $cs^C_2(\phi2)=(2-1)F+(2-1)F=2F$. Then, the total cutsize of $cs^C_2=3F$ incurred by the nets representing $\mathcal{X}(:,:,2)$ is equal to the communication volume regarding $\hat{\mathbf{C}}(2,:)$, which is given by $vol^C_2=(|Z^C_2|-1)F=3F$. Similarly, $cs^C_1=vol^C_1=F$ and $cs^C_3=vol^C_3=F$. Then, the total cutsize incurred by the nets representing the frontal slices is $5F$, which is equal to the total communication volume in the third phase of medium-grain CPD-ALS , i.e., $vol^C=5F$. With similar discussions for the first and second phases, the total cutsize of $12F$ in CartHP is equal to the total communication volume.

4.4 1D Factor Matrix Partitioning

Recall that the correctness of CartHP in encapsulating total communication volume depends on the consistency condition. In order to satisfy this condition, we assign each factor-matrix row to one of the processors that own a nonzero subslice in the corresponding slice.

The rows of a factor matrix are partitioned among processors, independently for each factor matrix. Note that the communications regarding each row chunk (e.g., $\mathbf{A}_q$) are confined to a distinct processor layer (e.g., $p_{q,:,:}$). Hence, the rows in a chunk are partitioned among the processors in the corresponding layer, independently for each chunk. For partitioning the rows in a chunk, we adopt the best-fit-decreasing heuristic used for solving the P-feasible bin-packing problem [26]. The rows are considered in decreasing order of the number of their nonzero unshared subslices. That is, $\mathbf{A}(i,:)$ is processed earlier than $\mathbf{A}(i',:)$ if $|Z_i^A| \geq |Z_{i'}^A|$. The best-fit criterion corresponds to assigning a row to a processor that currently has the minimum communication volume among the processors that own a nonzero subslice in the corresponding slice. After assigning a row to a processor, the volumes of the respective processors are increased accordingly.

4.5 Mode Processing Order

In our model, we determine the number of chunks along each mode, i.e., Q, R and S values, to be proportional to the tensor dimension in that mode, i.e., I, J and K values, as proposed in [14]. Recall that CartHP introduces the number of chunks along a mode as a multiplicative factor to the number of constraints in each further partitioning phase. For example, Q chunks obtained in $\phi 1$ lead to Q and $Q \times R$ constraints in $\phi 2$ and $\phi 3$, respectively. However, the performance of the multi-constraint partitioning tools is known to degrade with increasing number of constraints [27]. In order to have fewer constraints, the modes with fewer chunks should be processed earlier. For this purpose, CartHP processes the modes in increasing order of the number of chunks.

4.6 Extension to More Than Three Modes

For an M-mode tensor $\mathcal{X}$ and an $P_1 \times \cdots \times P_M$ virtual mesh of processors, CartHP consists of M partitioning phases. In phase ϕm, hypergraph $\mathcal{H}^m = (\mathcal{V}^m, \bigcup_{1 \leq k \leq M, k \neq m} \mathcal{N}^k)$ is constructed and partitioned into P_m parts. In $\mathcal{H}^m$, each mode-m slice is represented by a vertex with $\Pi_{i=1}^m P_{i-1}$ weights (with $P_0 = 1$) in $\mathcal{V}^m$, whereas each nonzero mode-k (sub)slice is represented by a net in $\mathcal{N}^k$ for $k = 1, \ldots, m-1, m+1, \ldots, M$. Net n connects vertex v if the intersection of the (sub)slices represented by v and n contains at least one nonzero. Here, the slices are $M-1$ dimensional, hence the intersection of two slices along different modes is $M-2$ dimensional.

A P_m-way partition of $\mathcal{H}^m$ induces P_m slice chunks along mode m. As a result, each slice along a mode different than mode m is divided into P_m subslices along mode m. In $\mathcal{H}^m$, each nonzero mode-k subslice is represented by a net in $\mathcal{N}^k$ in order to correctly encapsulate the communication volume. Here, these nonzero subslices are the smallest possible subslices divided by the chunks. Similarly, the number of nonzeros in each subslice of a mode-m slice constitutes a different weight to the vertex representing that slice for achieving computational load balance via multi-constraint partitioning.

5 Experiments

We evaluate the performance of the proposed CartHP method against the baseline multi-dimensional cartesian partitioning method [14]. For obtaining balance on the number of tensor nonzeros, this method randomly permutes the slices at each mode before obtaining respective slice chunks. We refer to this baseline method as CartR, with "R" standing for "random". The performance comparison is conducted in terms of partition statistics and parallel CPD-ALS runtimes for 12 tensors on 64, 128, 256, 512 and 1024 processors. Finally, we discuss the amortization of the partitioning overhead introduced by CartHP in terms of CPD-ALS solutions.

5.1 Setting

For partitioning hypergraphs in CartHP (line 14 in Algorithms 3, 4 and 5), we use PaToH [15] (version 3.2) in speed mode with maximum allowable imbalance ratio set to 0.04, i.e., $\epsilon_m = 0.04$. In PaToH, we set the refinement algorithm to FM with tight balance. Since PaToH contains randomized algorithms, we ran CartHP five times for each instance and report the geometric average of the results.

For conducting the parallel CPD-ALS experiments, we implemented the medium-grain CPD-ALS algorithm in C using MPI for interprocess communication. The source code is compiled with Cray C compiler (version 2.5.9) using the optimization level three. For the fold and expand operations on factor-matrix rows, personalized all-to-all collective operations are used. For storing the subtensors in processors, an extension of the compressed row storage (CRS) scheme for tensors [28] is utilized. MTTKRP operation is performed in a fiber-centric manner to reduce the FLOP counts, as described in [28]. For the rest of the computations, efficient CBLAS routines provided by Intel MKL library (version 2017) are used whenever needed. Our parallel implementation is orthogonal to the data partitioning method, hence it can take any medium-grain partition as input. For a fair comparison, we use the same parallel implementation for evaluating the partitions obtained by CartR. In our experiments, we set the number of components in CPD-ALS to 16, i.e., $F = 16$. For each instance, the runtime of one CPD-ALS iteration is reported by taking the average of the total runtime of 1,000 iterations.

We conducted our parallel experiments on a Cray XC40 machine. A node of this machine consists of 24 cores (two 12-core Intel Haswell Xeon processors) with 2.5 GHz clock frequency and 128 GB memory. The nodes are connected with CRAY Aries, which is a high speed network with Dragonfly topology.

5.2 Dataset

In our experiments, we use 12 sparse tensors whose properties are given in Table 1. All of these tensors are obtained from the datasets arising in real-world applications. First nine of them have three modes, whereas the remaining three have four modes. Columns 2–5 and 6 respectively display the dimensions and the number of nonzeros in the respective tensor.

`Facebook` consists of the wall-posting information in the form of owner-poster-date triplets from the Facebook New Orleans networks[29]. `NELL-b` and `NELL-c` consist of the beliefs in the form of entity-relation-entity triplets discovered by the Never Ending Language Learning (NELL) project [30]. `NELL-b` contains the relations that NELL believes to be true, whereas `NELL-c` contains only the candidate beliefs.

TABLE 1
Properties of the Test Tensors

name	I	J	K	L	nnz
Facebook	42.4 K	40.0 K	1.5 K	–	738.1 K
NELL-b	2.4 M	428	344.6 K	–	3.0 M
Brightkite	51.4 K	942	773.0 K	–	2.7 M
Finefoods	67.1 K	11.8 K	82.3 K	–	5.6 M
Gowalla	107.1 K	597	1.3 M	–	6.3 M
MovieAmazon	87.9 K	4.4 K	226.5 K	–	15.0 M
NELL-c	5.1 M	435	716.3 K	–	96.7 M
Netflix	17.8 K	480.2 K	2.2 K	–	100.5 M
Yelp	686.6 K	85.5 K	773.3 K	–	185.6 M
MovieLens	7.8 K	19.5 K	38.6 K	3.4 K	465.6 K
Flickr	319.7 K	28.2 M	1.6 M	730	112.9 M
Delicious	532.9 K	17.3 M	2.5 M	1.4 K	140.1 M

`Brightkite` and `Gowalla` consist of checkin information in the form of user-date-location triplets obtained from location-based social networks [31]. `Finefoods` and `MovieAmazon` consist of user-product-word triplets obtained from food and movie reviews in Amazon, respectively [32]. `Netflix` consists of user-item-time triplets obtained from the ratings in Netflix Prize competition [33]. Similar to `Finefoods`, `Yelp` consists of user-business-word triplets obtained from business reviews in Yelp academic dataset[1]. `MovieLens` consists of user-movie-tag-time quadruplets obtained from free-text taggings in MovieLens 20M dataset [34]. `Flickr` and `Delicious` consist of user-resource-tag-time quadruplets which were first crawled by Görlitz et al. [35] respectively from flickr.com and delicious.com.

5.3 Parallel CPD-ALS Results

Table 2 presents the average results obtained by CartHP normalized with respect to those obtained by CartR. Each row displays the geometric average of the results on 12 tensors for the respective number of processors. The detailed results for each tensor are given in Appendix C, available in the online supplemental material. Column "imb" denotes load imbalance, which we compute as the ratio of the maximum to the average number of nonzeros assigned to a processor. Columns under "number of messages" and "comm volume" denote the number of messages sent and received by a processor regarding the expand and fold steps through all phases and the volume of data communicated along these messages, respectively. Under both, "max" and "avg" denote the maximum and average amount of the corresponding metric over all processors, respectively. Under "parallel runtime", columns "comm" and "total" respectively denote the communication time and total runtime of a single iteration in medium-grain-parallel CPD-ALS.

As seen in Table 2, CartHP drastically reduces average communication volume compared to CartR. Note that the reduction in average communication volume also refers to the reduction in total communication volume. CartHP reduces average (total) volume by 58, 55, 51, 49 and 47 percent for 64, 128, 256, 512 and 1024 processors, respectively. These improvements are expected since CartHP minimizes this metric while CartR only provides a loose upper bound on it. The reduction in average

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

TABLE 2
Average Results Obtained by CartHP Normalized with Respect to Those Obtained by CartR

number of procs		number of messages		comm volume		parallel runtime	
	imb	max	avg	max	avg	comm	total
64	1.01	0.97	0.93	0.61	0.42	0.50	0.82
128	1.01	0.97	0.93	0.60	0.45	0.56	0.78
256	1.05	0.97	0.91	0.60	0.49	0.59	0.74
512	1.05	0.98	0.90	0.53	0.51	0.61	0.72
1024	1.05	0.97	0.85	0.53	0.53	0.61	0.72
average	**1.03**	**0.97**	**0.90**	**0.57**	**0.48**	**0.57**	**0.76**

volume leads to a similar reduction in maximum volume, by 39, 40, 40, 47 and 47 percent for 64, 128, 256, 512 and 1024 processors, respectively. The reduction in average volume also leads to a significant reduction in average (total) number of messages. CartHP reduces average number of messages by 7, 7, 9, 10 and 15 percent for 64, 128, 256, 512 and 1024 processors, respectively. The reduction in average number of messages leads to a slight reduction of 2-3 percent in maximum number of messages.

The drastic reductions in communication cost metrics lead to a drastic reduction in the communication time of CPD-ALS by 50, 44, 41, 39 and 39 percent for 64, 128, 256, 512 and 1024 processors, respectively. Although CartHP causes an increase in load imbalance by at most 5 percent on the average, the reduction obtained in communication time conceals this increase and leads to a significant reduction in total CPD-ALS runtime. CartHP reduces total runtime by 28, 32, 36, 38 and 38 percent for 64, 128, 256, 512 and 1024 processors, respectively.

Table 3 presents the detailed results obtained by CartR and CartHP on 512 processors for each tensor. The values given for maximum and average communication volumes are in terms of words. For each tensor, the best result attained for each metric is given in boldface.

As seen in Table 3, CartHP attains a better result in average communication volume for all tensors and in maximum communication volume for 9 out of 12 tensors. In communication time and total CPD-ALS runtime, it achieves a better result for 11 and 10 tensors, respectively. For the rest of the metrics, CartHP and CartR have comparable performances since each achieves a better result for half of the tensors. The highest reduction rates in total runtime are observed for `Gowalla`, `Flickr` and `Delicious`. This can be explained by the drastic amounts of decrease achieved in both maximum volume and total volume for these tensors. CartHP performs comparable to CartR for `Netflix` since the reduction in the communication time and the increase in the imbalance compensate each other. For `MovieAmazon`, CartHP performs worse than CartR due to the increase in the communication time stemming from the increase in maximum volume despite the decrease in total volume. Note that a similar increase is also observed for `Netflix`, but it does not degrade the communication time much due to a higher decrease in total volume.

Fig. 5 displays the strong scaling curves for all tensors in terms of total CPD-ALS runtime. For 9 out of 12 tensors, CartHP achieves better CPD-ALS scalability compared to

TABLE 3
Partition Statistics and Parallel Runtime Results Obtained by CartR and CartHP for one CPD-ALS Iteration on 512 Processors

	CartR							CartHP						
		number of messages		comm volume		parallel runtime (ms)			number of messages		comm volume		parallel runtime (ms)	
tensor	imb	max	avg	max	avg	comm	total	imb	max	avg	max	avg	comm	total
Facebook	1.32	2,162	1,956	114 K	83 K	2.7	3.4	**1.01**	**2,043**	**1,901**	**67 K**	**58 K**	**1.9**	**2.8**
NELL-b	1.06	1,400	534	158 K	75 K	4.4	7.5	**1.01**	**1,262**	**224**	**38 K**	**11 K**	**2.1**	**4.5**
Brightkite	**1.73**	2,323	2,306	231 K	142 K	5.1	8.8	4.25	**2,300**	**2,155**	**85 K**	**64 K**	**3.3**	**6.0**
Finefoods	1.08	**1,259**	1,225	356 K	257 K	7.4	11.1	**1.05**	1,263	**1,191**	**308 K**	**203 K**	**5.1**	**9.4**
Gowalla	1.08	**2,136**	1,866	687 K	443 K	7.6	13.2	**1.01**	2,182	**1,757**	**186 K**	**133 K**	**4.0**	**7.0**
MovieAmazon	**1.09**	**2,209**	**2,154**	**607 K**	474 K	**8.3**	**13.9**	1.10	2,228	2,209	1.1 M	**423 K**	8.5	16.3
NELL-c	**1.01**	1,941	1,504	2.5 M	1.4 M	34.5	72.6	1.07	**1,845**	**1,254**	**943 K**	**491 K**	**15.4**	**44.5**
Netflix	**1.01**	**2,564**	**2,562**	**594 K**	551 K	9.9	**35.5**	1.14	**2,564**	2,564	729 K	**471 K**	**9.3**	35.7
Yelp	**1.06**	**1,267**	**1,267**	**4.1 M**	3.3 M	62.5	126.7	1.07	1,268	1,268	5.7 M	**2.3 M**	**47.9**	**113.4**
MovieLens	1.30	2,464	2,043	198 K	85 K	2.9	4.3	**1.08**	**2,219**	**1,969**	**77 K**	**65 K**	**2.4**	**3.9**
Flickr	**1.01**	**4,603**	**4,595**	17.7 M	10.6 M	327.0	505.2	1.14	4,608	4,597	**4.0 M**	**3.4 M**	**108.0**	**216.3**
Delicious	1.06	**4,367**	**4,367**	24.0M	11.3 M	398.2	649.7	**1.05**	4,368	4,368	**8.8 M**	**6.1 M**	**171.6**	**355.9**

CartR. This is because CartHP obtains drastic reductions in both maximum and average communication volume metrics for these tensors. CartHP performs comparable to CartR for Netflix and Yelp and slightly worse than CartR for MovieAmazon since CartHP increases maximum volume while decreasing average volume for these tensors on all processor counts. For Facebook and MovieLens, although CartHP performs better than CartR, both methods display poor scalability for these tensor since they are small.

5.4 Partitioning Overhead and Amortization

Table 4 reports the partitioning time of CartHP in seconds as well as the ratio of this partitioning time to the factorization time for each tensor. Here, each factorization involves a number of CPD-ALS iterations required to converge with tolerance 10^{-5} (as computed in [28]), where the number of iterations typically increases with increasing F. Both partitioning and factorization are performed in a sequential setting. As seen in the table, for Netflix, partitioning takes

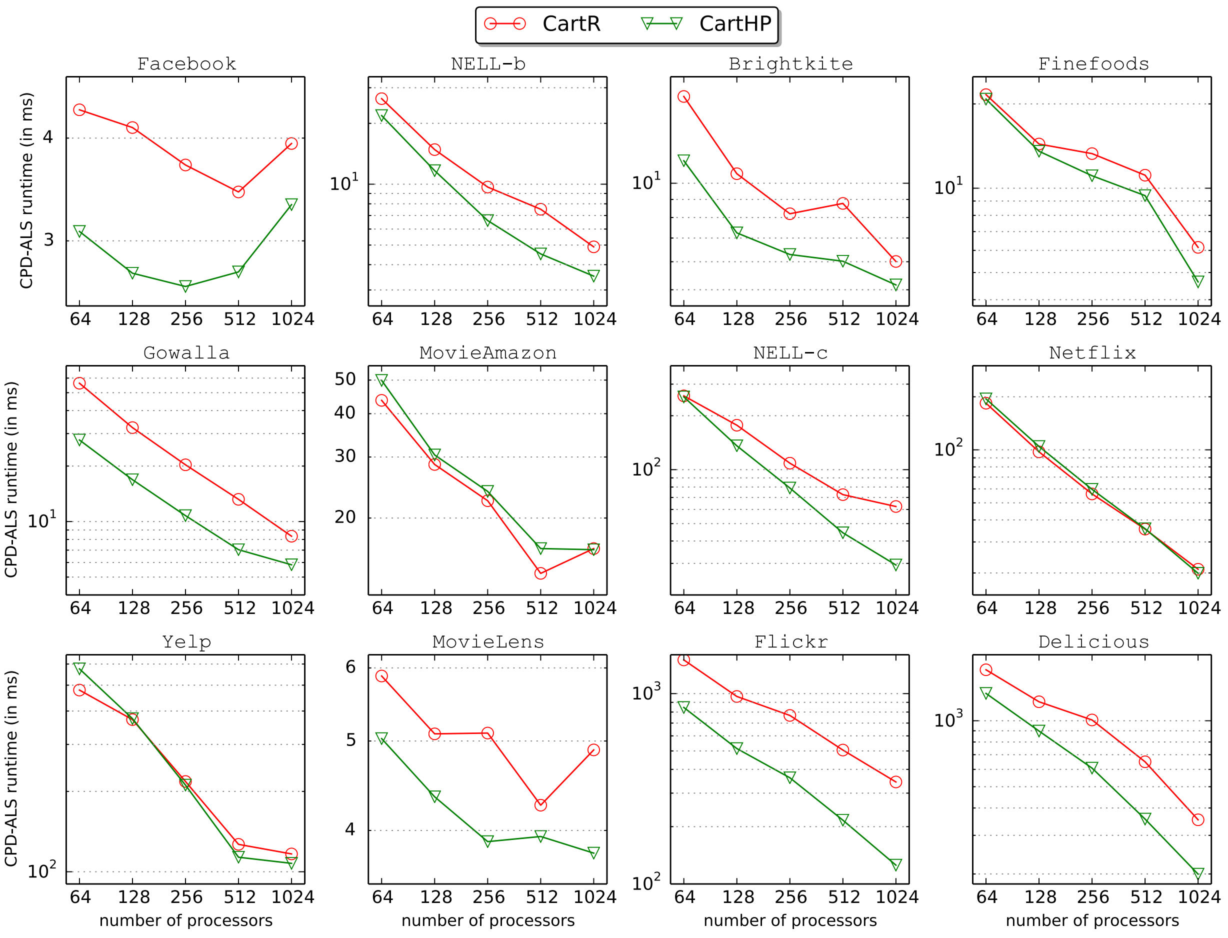


Fig. 5. Strong scaling curves for medium-grain-parallel CPD-ALS obtained by CartR and CartHP.

TABLE 4
Comparison of Partitioning Overhead of CartHP Against Factorization in Terms of Sequential Runtime

tensor	CartHP time (s)	CartHP/factorization $F=16$	CartHP/factorization $F=64$
Facebook	5.8	4.68	0.82
NELL-b	9.8	0.53	0.10
Brightkite	9.2	1.73	0.18
Finefoods	22.3	2.32	0.32
Gowalla	31.5	3.93	0.47
MovieAmazon	35.0	1.17	0.12
NELL-c	62.3	0.50	0.08
Netflix	36.2	0.39	0.08
Yelp	380.6	6.28	1.10
MovieLens	4.6	9.22	1.38
Flickr	569.6	7.97	1.69
Delicious	1693.0	23.10	5.06
average	-	**2.60**	**0.41**

TABLE 5
Average Number of CPD Solutions that Amortize the Sequential Partitioning Time of CartHP

$P=64$	$P=128$	$P=256$	$P=512$	$P=1024$	avg
3.39	3.91	4.92	8.02	14.18	**5.94**

0.39 and 0.08 factorizations for $F=16$ and $F=64$, respectively. On average, it takes 2.60 and 0.41 factorizations for $F=16$ and $F=64$, respectively.

Table 5 displays the average number of CPD solutions that amortize the sequential partitioning time of CartHP for each processor count, i.e., P value. Here, each CPD solution refers to running the parallel CPD-ALS algorithm for computing a factorization for ten different F values [36] starting from three different sets of initial factor matrices [37]. For each F value and initial factor matrix set, a factorization is assumed to require 25 iterations, so, each CPD solution is assumed to involve $10 \times 3 \times 25 = 750$ iterations. As seen in the table, on the average, the partitioning time of CartHP amortizes in only 3.39, 3.91, 4.92, 8.02, and 14.18 CPD solutions for 64, 128, 256, 512, and 1,024 processors, respectively, where the overall average is computed as 5.94 CPD solutions.

6 Conclusion

We investigated the utilization of the sparsity pattern of a given tensor for minimizing the total communication volume in medium-grain CPD-ALS algorithm which adopts multi-dimensional cartesian tensor partitioning. We proposed a novel hypergraph-partitioning model that correctly encapsulates the total communication volume of medium-grain-parallel CPD-ALS. We demonstrated the effectiveness of the proposed model by conducting experiments on 12 tensors for up to 1,024 processors. Our model drastically reduces the communication volume and the communication time of medium-grain-parallel CPD-ALS, hence the total parallel runtime.

Acknowledgments

This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under project EEEAG-116E043. We acknowledge PRACE for awarding us access to resource Hazel Hen (Cray XC40) based in Germany at HLRS.

References

[1] J. D. Carroll and J.-J. Chang, “Analysis of individual differences in multidimensional scaling via an N-way generalization of “Eckart-Young” decomposition,” *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970. [Online]. Available: http://dx.doi.org/10.1007/BF02310791

[2] R. A. Harshman, Foundations of the PARAFAC Procedure: Models and Conditions for An “Explanatory” Multi-Modal Factor Analysis. Los Angeles, USA: Univ. California, 1970.

[3] T. G. Kolda and B. W. Bader, “Tensor decompositions and applications,” *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, 2009. [Online]. Available: http://dx.doi.org/10.1137/07070111X

[4] C. M. Andersen and R. Bro, “Practical aspects of PARAFAC modeling of fluorescence excitation-emission data,” *J. Chemometrics*, vol. 17, no. 4, pp. 200–215, 2003. [Online]. Available: http://dx.doi.org/10.1002/cem.790

[5] N. D. Sidiropoulos, R. Bro, and G. B. Giannakis, “Parallel factor analysis in sensor array processing,” *IEEE Trans. Signal Process.*, vol. 48, no. 8, pp. 2377–2388, Aug. 2000.

[6] A. H. Andersen and W. S. Rayens, “Structure-seeking multilinear methods for the analysis of fMRI data,” *NeuroImage*, vol. 22, no. 2, pp. 728–739, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1053811904001181

[7] E. Martinez-Montes, P. A. Valdes-Sosa, F. Miwakeichi, R. I. Goldman, and M. S. Cohen, “Concurrent EEG/fMRI analysis by multiway partial least squares,” *NeuroImage*, vol. 22, no. 3, pp. 1023–1034, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1053811904001946

[8] A. Shashua and A. Levin, “Linear image coding for regression and classification using the tensor-rank principle,” in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2001, pp. I-42–I-49.

[9] E. Acar, S. A. Çamtepe, M. S. Krishnamoorthy, and B. Yener, *Modeling and Multiway Analysis of Chatroom Tensors*. Berlin, Germany: Springer, 2005, pp. 256–268. [Online]. Available: http://dx.doi.org/10.1007/11427995_21

[10] B. W. Bader, M. W. Berry, and M. Browne, *Discussion Tracking in Enron Email Using PARAFAC*. London, U.K.: Springer, 2008, pp. 147–163. [Online]. Available: http://dx.doi.org/10.1007/978–1-84800-046-98

[11] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, and T. M. Mitchell, “Toward an architecture for never-ending language learning,” in *Proc. 24th AAAI Conf. Artif. Intell.*, 2010, pp. 1306–1313.

[12] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, A. Hanjalic, and N. Oliver, “TFMAP: Optimizing map for top-N context-aware recommendation,” in *Proc. 35th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2012, pp. 155–164. [Online]. Available: http://doi.acm.org/10.1145/2348283.2348308

[13] N. K. M. Faber, R. Bro, and P. K. Hopke, “Recent developments in CANDECOMP/PARAFAC algorithms: A critical review,” *Chemometrics Intell. Laboratory Syst.*, vol. 65, no. 1, pp. 119–137, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169743902000898

[14] S. Smith and G. Karypis, “A medium-grained algorithm for distributed sparse tensor factorization,” in *Proc. 30th IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 902–911.

[15] U. 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.

[16] U. 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, Feb. 2010. [Online]. Available: http://dx.doi.org/10.1137/080737770

[17] B. Uçar and C. Aykanat, “Revisiting hypergraph models for sparse matrix partitioning,” *SIAM Rev.*, vol. 49, no. 4, pp. 595–603, 2007.

[18] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Redwood City, CA, USA: Benjamin/Cummings, 1994.

[19] B. Hendrickson, R. Leland, and S. Plimpton, “An efficient parallel algorithm for matrix-vector multiplication,” *Int. J. High Speed Comput.*, vol. 07, no. 01, pp. 73–88, 1995. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0129053395000051

[20] U. V. Catalyurek and C. Aykanat, "A hypergraph-partitioning approach for coarse-grain decomposition," in *Proc. ACM/IEEE Conf. Supercomput.*, Nov. 2001, pp. 42–42.
[21] J. H. Choi and S. Vishwanathan, "DFacTo: Distributed factorization of tensors," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 1296–1304. [Online]. Available: http://papers.nips.cc/paper/5395-dfacto-distributed-factorization-of-te nsors.pdf
[22] B. W. Bader and T. G. Kolda, "Efficient MATLAB computations with sparse and factored tensors," *SIAM J. Sci. Comput.*, vol. 30, no. 1, pp. 205–231, Dec. 2007.
[23] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "GigaTensor: Scaling tensor analysis up by 100 times - algorithms and discoveries," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 316–324. [Online]. Available: http://doi.acm.org/10.1145/2339530.2339583
[24] 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. 77:1–77:11. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807624
[25] W. Austin, G. Ballard, and T. G. Kolda, "Parallel tensor compression for large-scale scientific data," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2016, pp. 912–922.
[26] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*. Rockville, MD, USA: Computer Science, 1978.
[27] C. Aykanat, B. B. Cambazoglu, and B. Uçar, "Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices," *J. Parallel Distrib. Comput.*, vol. 68, no. 5, pp. 609–625, 2008.
[28] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "SPLATT: Efficient and parallel sparse tensor-matrix multiplication," in *Proc. IEEE Int. Parallel Distrib. Processing Symp.*, May 2015, pp. 61–70.
[29] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in Facebook," in *Proc. 2nd ACM SIGCOMM Workshop Social Netw.*, Aug. 2009, pp. 37–42 .
[30] A. Carlson, J. Betteridge, B. Kisiel, and B. Settles, "Toward an architecture for never-ending language learning," in *Proc. 24th AAAI Conf. Art. Intell.*, 2010, pp. 1306–1313.
[31] 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. [Online]. Available: http://doi.acm.org/10.1145/2020408.2020579
[32] J. J. McAuley and J. Leskovec, "From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 897–908. [Online]. Available: http://doi.acm.org/10.1145/2488388.2488466
[33] J. Bennett, S. Lanning, and N. Netflix, "The netflix prize," in *Proc. KDD Cup Workshop Conjunction KDD*, 2007, pp. 3–6.
[34] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, Dec. 2015. [Online]. Available: http://doi.acm.org/10.1145/2827872
[35] O. Görlitz, S. Sizov, and S. Staab, "PINTS: Peer-to-peer infrastructure for tagging systems," in *Proc. 7th Int. Conf. Peer-to-Peer Syst.*, 2008, pp. 19–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855641.1855660
[36] 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. [Online]. Available: http://doi.acm.org/10.1145/1835449.1835591
[37] R. A. Harshman and M. E. Lundy, "The PARAFAC model for three-way factor analysis and multidimensional scaling, " in, *Research Methods for Multi-Mode Data Analysis*. New York, NY, USA: Praeger, 1984.

Seher Acer received the BS, MS and PhD degrees in computer engineering from Bilkent University, Turkey, where she is currently a postdoctoral researcher. Her research interests include combinatorial scientific computing, graph and hypergraph partitioning for sparse matrix and tensor computations, and parallel computing.

Tugba Torun received the BS degree in mathematics and the MS degree in computer engineering both from Bilkent University. She is currently working toward the PhD degree at Bilkent University. Her research interests include combinatorial scientific computing, graph and hypergraph partitioning, and tensor computations.

Cevdet Aykanat received the BS and MS degrees from Middle East Technical University, Ankara, Turkey, both in electrical engineering, and the PhD degree from Ohio State University, Columbus, in electrical and computer engineering. Since 1989, he has been affiliated with Computer Engineering Department, Bilkent University, Ankara, 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 the *IEEE Transactions of Parallel and Distributed Systems* between 2008 and 2012.