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-graining 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 $m$th 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
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 multidimensional 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 nonzero 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 method 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 (\( \mathcal{A} \)) and bold uppercase (A) letters. To refer to a varying index, we use a colon as in Matlab notation, e.g., \( \mathcal{A}(i,:) \) denotes the \( i \)th row of \( \mathcal{A} \).

A tensor with \( M \) dimensions is called an \( M \)-mode tensor and mode \( m \) refers to the \( m \)th 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 \( i \)th mode-1 slice, \( \mathcal{X}(::j,:) \) is the \( j \)th mode-2 slice, and \( \mathcal{X}(:::,k) \) is the \( k \)th 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 $X$ is written in short as $X \approx [\lambda, A, B, C]$.

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

Algorithm 1. CPD-ALS($X$)

1: Randomly initialize factor matrices $A$, $B$, and $C$
2: while not converged do
3: $A \leftarrow X_{(1)}(C \otimes B)(C^T C + B^T B)^{-1}$
4: Normalize columns of $A$ into $\lambda$
5: $B \leftarrow X_{(2)}(C \otimes A)(C^T C + A^T A)^{-1}$
6: Normalize columns of $B$ into $\lambda$
7: $C \leftarrow X_{(3)}(A \otimes B)(A^T A + B^T B)^{-1}$
8: Normalize columns of $C$ into $\lambda$
9: return $[\lambda, A, B, C]$

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 $X_k$ denote the local nonzeros assigned to processor $p_k$. Let $A_k$, $B_k$, and $C_k$ 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 $i$th row of factor matrix $A$ can be rewritten as $A_i^{(i,:)} = \sum_{(i,j,k) \neq 0} X(i,j,k)(B(j,:) \otimes C(k,:))$. In these lines, $A_{ik}$, $B_{ik}$, and $C_{ik}$ denote intermediate results of the corresponding local factor matrices. $A_{ik}^{(i,:)}$ which appears as an input in a local MTTKRP, refers to $A_k$ augmented with the nonlocal rows that are needed by the local MTTKRP. $A_{ik}^{(i,:)}$ which appears as an output in a local MTTKRP, refers to $A_k$ augmented with the nonlocal rows for which the local MTTKRP produces partial results. The same notation applies to $B_{ik}$ and $C_{ik}$ 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 $A_{ik}$, $B_{ik}$, and $C_{ik}$. 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 $A_i^{(i,:)}$ assigned to a sharing processor $p_k$. For the reduce operation, all processors except $p_k$ in the connectivity set of $A_i^{(i,:)}$ send their partial results for nonlocal $A_{ik}^{(i,:)}$ to $p_k$. For the expand operation, $p_k$ sends its local normalized $A_{ik}^{(i,:)}$ to all processors in the connectivity set of $A_i^{(i,:)}$. 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 2$R$ times the sum of the “connectivity – 1’ values of the shared factor-matrix rows along that mode.

With the above-mentioned parallelization scheme, factor-matrix-transpose and factor-matrix multiplication operations (i.e., $A^T A$, $B^T B$, and $C^T C$) 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, communication products as well as pseudoinverse operations associated with these results $R \times R$ matrices (e.g., $(C^T C + B^T B)^{-1}$) 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 $nnz(X)$). 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.

Algorithm 2. Parallel CPD-ALS ($X$)

1: Randomly initialize factor matrices $A$, $B$, and $C$
2: while not converged do
3: $A_{ik}^{(i,:)} \leftarrow \sum_{(i,j,k) \neq 0} X(i,j,k)(B_{ik}(j,:) \otimes C(k,:))$
4: Sparse REDUCE on shared $A$-matrix rows
5: $A_k \leftarrow A_k(C^T C + B^T B)^{-1}$ and norm. cols. of $A$ into $\lambda$
6: ALL-REDUCE to compute $A^T A$
7: Sparse EXPAND on shared $A$-matrix rows
8: $B_{ik}^{(i,:)} \leftarrow \sum_{(i,j,k) \neq 0} X(i,j,k)(A_k^{(i,:) \otimes B_{ik}^{(j,:))}}$
9: Sparse REDUCE on shared $B$-matrix rows
10: $B_k \leftarrow B_k(C^T C + A^T A)^{-1}$ and norm. cols. of $B$ into $\lambda$
11: ALL-REDUCE to compute $B^T B$
12: Sparse EXPAND on shared $B$-matrix rows
13: $C_{ik}^{(k,:)} \leftarrow \sum_{(i,j,k) \neq 0} X(i,j,k)(A_{ik}^{(i,:) \otimes B_{ik}^{(j,:))}}$
14: Sparse REDUCE on shared $C$-matrix rows
15: $C_k \leftarrow C_k(B^T B + A^T A)^{-1}$ and norm. cols. of $C$ into $\lambda$
16: ALL-REDUCE to compute $C^T C$
17: Sparse EXPAND on shared $C$-matrix rows
18: return $[\lambda, A, B, C]$
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 $\text{Pins}(n)$. The set of nets that connect vertex $v$ is denoted by $\text{Nets}(v)$. 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 $\Pi$, the weight of part $\mathcal{V}_k$ is defined as $W(\mathcal{V}_k) = \sum_{v \in \mathcal{V}_k} w(v)$. It is said to satisfy the partitioning constraint if $W(\mathcal{V}_k) \leq W_{\text{avg}}(1+\epsilon)$ for each part $\mathcal{V}_k$ in $\Pi$, for a given maximum allowed imbalance ratio $\epsilon$. Here $W_{\text{avg}} = W_{\text{tot}}/P$, where $W_{\text{tot}} = \sum_{n \in \mathcal{N}} W(n)$.

In a given partition $\Pi$, 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 $n$, $\text{con}(n)$, denotes the number of parts connected by $n$. $n$ is said to be cut if it connects more than one part, i.e., $\text{con}(n) > 1$, and uncut otherwise. Then the connectivity cutsize is defined as $\text{cutsiz}(\Pi) = \sum_{n \in \mathcal{N}} \text{con}(n) - 1)(c(n)$ 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 $i$th horizontal slice of $\mathcal{X}^H$ or the $j$th lateral slice of $\mathcal{X}^L$ or the $k$th 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 $\mathcal{H}_{\text{CGH}}^1(\mathcal{X}^H) = (\mathcal{V}^H, \mathcal{N}^H \cup \mathcal{N}^L \cup \mathcal{N}^F)$ contains one vertex $v^H_i$ for each horizontal slice $\mathcal{X}^H(i, : : )$. It contains nets $n_i^H$, $n_j^H$, and $n_k^H$ for each horizontal slice $\mathcal{X}^H(i, : : )$, lateral slice $\mathcal{X}^L( : : j, : : )$, and frontal slice $\mathcal{X}^F( : : k, : : )$, respectively. Net $n_i^H$ connects vertex $v^H_i$ if intersection of $\mathcal{X}^H(i, : : )$ and $\mathcal{X}^H( : : j, : : )$ is a nonzero fiber. Similarly, net $n_j^H$ connects vertex $v^H_i$ if intersection of $\mathcal{X}^H(i, : : )$ and $\mathcal{X}^H( : : k, : : )$ is a nonzero fiber. That is,

$$\text{Pins}(n_i^H) = \{v^H_i\},$$

$$\text{Pins}(n_j^H) = \{v^H_i : \exists \mathcal{X}^H(i, j, k) \in \mathcal{X}^H(i, : : ) \cap \mathcal{X}^H( : : j, : : )\},$$

$$\text{Pins}(n_k^H) = \{v^H_i : \exists \mathcal{X}^H(i, j, k) \in \mathcal{X}^H(i, : : ) \cap \mathcal{X}^H( : : k, : : )\}.$$
pin-wise amalgamation of nets $n^H_i(X^H)$, $n^L_i(X^L)$, and $n^F_i(X^F)$ into $n^H_i(X)$, where $2R(\text{con}(n^H_i(X)) - 1)$ will correctly encode the volume of communication due to factor-matrix row $A(i,\cdot)$.

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 using 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 $X(i, j, \cdot)$ 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 $X^H, X^L, X^F$, 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 $X(i, j, k)$ we look for the sparsest slice among the three slices $X(i, \cdot, \cdot), X(\cdot, j, \cdot)$, and $X(\cdot, \cdot, k)$. For example, if $\text{nnz}(X(i, \cdot, \cdot))$ is smaller than both $\text{nnz}(X(\cdot, j, \cdot))$ and $\text{nnz}(X(\cdot, \cdot, k))$, then we assign $X(i, j, k)$ to the $i$th slice of $X^H$.

The special case, which occurs when a slice contains a single nonzero, is handled as follows: For example, if the $i$th horizontal slice of $X$ contains a single nonzero $X(i, j, k)$, then the splitting heuristic assigns $X(i, j, k)$ to the $i$th slice of $X^H$. However, this will create a single-pin net $n^H_i$ in both $\mathcal{H}_{CG}(X^H)$ and $\mathcal{H}_{MG}(X)$. Since a single-pin net cannot be cut, we assign nonzero $X(i, j, k)$ to either $X^H(i, \cdot, \cdot)$ or $X^F(i, \cdot, \cdot)$ according to number of nonzeros in $X(i, \cdot, \cdot)$ and $X(\cdot, \cdot, k)$. 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 $4 \times 3 \times 2$ tensor $X$ with 10 nonzeros. The second part of the figure displays the component tensors $X^H, X^L, X^F$ 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 $\mathcal{H}_{CG}(X^H), \mathcal{H}_{CG}(X^L), \mathcal{H}_{CG}(X^F)$. Finally, the rightmost part displays the medium-grain composite hypergraph $\mathcal{H}_{MG}(X)$. As seen in the figure, the heuristic splits eight nonzeros of the dense frontal slice $X(\cdot, \cdot, 1)$ to sparse slices $X^H(2, \cdot, \cdot), X^H(3, \cdot, \cdot), X^L(4, \cdot, \cdot), X^L(1, \cdot, \cdot), X^L(2, \cdot, \cdot)$, and $X^L(3, \cdot, \cdot)$ so that the component-frontal slice $X^F(\cdot, \cdot, 1)$ becomes empty.

#### Algorithm 3. SPLIT-TENSOR($X$)

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

### 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. $n_f(X)$ denotes the sum of the numbers of nonzero fibers along all modes and $\delta(X)$ denotes the number of slices that contain a single nonzero. \(\leq I + J + K - \delta(X)\) 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

---

**Table 1: Hypergraph Sizes**

<table>
<thead>
<tr>
<th>Model</th>
<th>$n_f(X)$</th>
<th>$\delta(X)$</th>
<th>(\leq I + J + K - \delta(X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MG</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

*Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on November 25, 2020 at 10:01:48 UTC from IEEE Xplore. Restrictions apply.*
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 $v_1$ exemplifies the first case, whereas vertex $v_H$ and net $n_H$ 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 $n_H$ and $n_L$ exemplify such nets.

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 \( \mathcal{H}_{MG}(X) \) 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\( \times \)2 tensor at the \( \ell \)th level are denoted by \( X_1^{\ell}, \ldots, X_{2^\ell}^{\ell} \) from left to right for \( 0 \leq \ell \leq \log_2 P \).

A vertex bipartition \( \Pi_2 = \{V_L, V_R\} \) of the medium-grain hypergraph \( \mathcal{H}_{MG}(X'_L) \) of the \( \ell \)th level tensor \( X'_L \) is utilized to form two new vertex-induced subtensors \( X^{2\ell+1}_{2k-1} \) and \( X^{2\ell+1}_{2k} \) both in level \( \ell + 1 \). Here, \( V_L \) and \( V_R \) respectively refer to the left and right parts of the vertex bipartition of \( \mathcal{H}_{MG}(X'_L) \). Because of the nature of the splitting heuristic applied to \( X'_L \), any vertex \( v'^L \) in \( V_L \) incurs the assignment of the respective mode-\( m \) (sub)slice of \( X'_L \) as a whole mode-\( m \) slice to left subtensor \( X_L = X^{2\ell+1}_{2k-1} \) for any mode-\( m \). The right subtensor \( X_R = X^{2\ell+1}_{2k} \) 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 \( \mathcal{H}_{MG}(X^{2\ell+1}_{2k-1}) \) and \( \mathcal{H}_{MG}(X^{2\ell+1}_{2k}) \), 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 \((v'^H, n'^L)\) in \( \mathcal{H}_{MG}(X) \) that represents the same slice is replaced by a node \( v'^H \) in \( \mathcal{G}_{MG}(X) \) with the adjacency list defined as follows:

\[
dj(v'^H) = \{v'^L : v'^L \in Pins(n'^L)\} \cup \{v'^L : v'^L \in Pins(n'^H)\} \cup \{v'^L : v'^L \in Nets(n'^L)\} \cup \{v'^L : v'^L \in Nets(v'^H)\}.
\]

The two sets in the upper line come from the vertices connected by \( n'^H \), whereas the two sets in the lower line come from the nets that connect \( v'^H \).

Fig. 2 displays medium-grain tripartite graph \( \mathcal{G}_{MG}(X) \) obtained from \( \mathcal{H}_{MG}(X) \) displayed in Fig. 1. For example, \( v'^H_1 \) and \( n'^L_1 \) in Fig. 1 is replaced by node \( v'^H_1 \) with adjacency list \( Adj(v'^H_1) = \{v'^L_1, v'^L_2, u'^L_1, u'^L_2\} \). \( Adj(u'^L_1) \) contains: \( u'^L_1 \) and \( u'^L_2 \) because of \( n_1^L \) connecting \( u'^L_1 \) and \( u'^L_2 \); \( u'^L_3 \) because of \( n_3^H \) connecting \( u'^L_1 \) and \( u'^L_2 \).

The topology of the tripartite graph resembles that of the tripartite graph model proposed in [8] for mode-independent slice reordering 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
this way, by applying the splitting heuristic at each RB step, we try to maximize its improvement.

The partitioning objective of minimizing the cutsizes in \( H_{MG}(A) \) and in \( H_{MG}(A^t) \) 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 \( A(i,:) \) 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 \( A(i,:) \) to \( p_r \). The tripartite graph model overestimates the communication volume as if processor \( p_q \) will send multiple partial results for \( A(i,:) \) 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 \( A(i,:) \) 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 Cart\(_{HP} \), FG\(_{HP} \) and MG\(_{HP} \), respectively. For Cart\(_{HP} \), we used the same parameters described in [12]. For MG\(_{HP} \) and FG\(_{HP} \), we use PaToH [22] with \( \text{SPEED} \) 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 MG\(_{TGP} \). 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 g++ version 8.3.0 using \( \text{O3} \) 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-Infini-band (Connect-X4).

5.2 Dataset

The dataset given in Table 2 contains 10 sparse tensors which arise from real-world applications. The density of each tensor \( \lambda \) is computed as \( \text{nnz}(\lambda)/(I \times J \times K \times L) \). 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: 1998DARPA 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, MG\(_{HP} \) and MG\(_{TGP} \) 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>
<thead>
<tr>
<th>tensor</th>
<th>size of dimensions</th>
<th>nnz density</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998DARPA</td>
<td>23.7M 13.5K 17.9K</td>
<td>28.4M 4.92e-09</td>
</tr>
<tr>
<td>delicious</td>
<td>17.2M 532.9K 2.5M</td>
<td>140.1M 4.27e-15</td>
</tr>
<tr>
<td>enron</td>
<td>244.2K 5.7K 6.0K 1.1K</td>
<td>54.2M 5.46e-09</td>
</tr>
<tr>
<td>facebook</td>
<td>42.4K 1.5K 39.9K</td>
<td>738.0K 2.89e-07</td>
</tr>
<tr>
<td>flickr</td>
<td>28.1M 13.5K 17.9K 28.4M</td>
<td>4.92e-09</td>
</tr>
<tr>
<td>freebase_music</td>
<td>22.6M 16.6 23.3M</td>
<td>99.5M 1.14e-09</td>
</tr>
<tr>
<td>lbnl-network</td>
<td>868.1K 4.2K 4.2K</td>
<td>112.9M 1.07e-09</td>
</tr>
<tr>
<td>movies-amazon</td>
<td>226.5K 4.4K 87.8K</td>
<td>15.0M 1.72e-07</td>
</tr>
<tr>
<td>nell-1</td>
<td>25.4M 2.1M 2.9M</td>
<td>143.6M 9.05e-13</td>
</tr>
</tbody>
</table>

TABLE 2 Properties of the Test Tensors
the total communication volume compared to the naive implementation for MG_H and 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 percent increase in load balance, 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 MG_H against FG_H on P=512 processor with R=64. The left part of the table displays the actual values for FG_H, whereas the right part displays the values of MG_H normalized with respect to those of FG_H. The last row shows the geometric averages.

As seen in Table 3, MG_H and FG_H display comparable performance in computational load balance (“max nnz”). Note that FG_H has a large number of vertices with unit weights thus enabling good load balance. Thus, the comparable performance of MG_H and FG_H in load balancing shows the success of the splitting heuristic in reducing the variance on vertex weights.

MG_H achieves considerably better performance than FG_H 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. MG_H achieves this improvement with about two times less preprocessing overhead.

Table 4 compares the performance of MG_H and MG_TGP against the state-of-the-art Cart_H 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 MG_H and MG_TGP normalized with respect to those of Cart_H.

As seen in Table 4, MG_H achieves considerably better computational load balance than Cart_H. On average, MG_H reduces the maximum computational load by 13 percent compared to Cart_H. This experimental finding is because of the following reasons: As discussed above, MG_H performs as good as FG_H in load balancing. However, in Cart_H, 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. MG_H achieves significantly better performance in both bandwidth metrics as expected. However, Cart_H 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 MG_H against Cart_H 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, MG_TGP and MG_H display comparable performance in load balance, whereas 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, MG_TGP still performs better than Cart_H 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 Cart_H, MG_H, and MG_TGP as well as FG_H 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 MG_H means that MG_H 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, MG_H is the clear winner, where FG_H, MG_TGP display close performance
for the 40 and 30 percent of the dataset respectively, and Cart displays significantly worse performance. As seen in Fig. 3b, in terms of partitioning time, MG\textsubscript{TGP} is the clear winner, where F\textsubscript{GHP} performs significantly worse. These experimental findings conform to the expectations.

Table 5 displays the strong scaling results for Cart\textsubscript{HP}, MG\textsubscript{HP}, and MG\textsubscript{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 Cart\textsubscript{HP} in all instances, whereas MG\textsubscript{HP} scales better than MG\textsubscript{TGP} in six out of eight instances.

We provide Table 5 to compare the relative parallel run-time performance of both MG models against those of Cart\textsubscript{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 Cart\textsubscript{HP} decreases as $R$ decreases, as expected. For example, 22 percent better performance of MG\textsubscript{HP} for $R = 64$ reduces to 20, 17, and 15 percent for $R = 32, 16$, and $8$, respectively. For $R = 64$, while MG\textsubscript{HP} achieves better runtime performance than Cart\textsubscript{HP} for all of the 10 instances, for $R = 8$, Cart\textsubscript{HP} achieves better runtime performance.
for three of the 10 instances. A similar relative parallel run-time performance variation for \( \text{MG TGP} \) against those of \( \text{Cart HP} \) can be observed in the table.

### 5.4 Preprocessing Overhead

Tables 6 and 7 display the partitioning times of \( \text{Cart HP} \), \( \text{MG HP} \), and \( \text{MG TGP} \) 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 \( \text{Cart HP} \) is 63 percent of the factorization time, whereas partitioning times of \( \text{MG HP} \) and \( \text{MG TGP} \) 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 \( \text{MG TGP} \) 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 6**
Sequential 512-Way Partitioning Overhead in Terms of Sequential CPD-ALS Factorization Time for \( R = 64 \)

<table>
<thead>
<tr>
<th>tensor</th>
<th>iteration time (s)</th>
<th># of iters.</th>
<th>( \text{Cart HP} )</th>
<th>( \text{MG HP} )</th>
<th>( \text{MG TGP} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998DARPA</td>
<td>77.443</td>
<td>11</td>
<td>0.17</td>
<td>0.34</td>
<td>0.11</td>
</tr>
<tr>
<td>brightkite</td>
<td>2.790</td>
<td>37</td>
<td>0.21</td>
<td>0.39</td>
<td>0.07</td>
</tr>
<tr>
<td>delicious</td>
<td>115.719</td>
<td>40</td>
<td>0.06</td>
<td>0.37</td>
<td>0.18</td>
</tr>
<tr>
<td>enron</td>
<td>12.005</td>
<td>71</td>
<td>0.73</td>
<td>0.75</td>
<td>0.06</td>
</tr>
<tr>
<td>facebook</td>
<td>0.349</td>
<td>33</td>
<td>0.74</td>
<td>1.03</td>
<td>0.23</td>
</tr>
<tr>
<td>flickr</td>
<td>181.528</td>
<td>8</td>
<td>1.08</td>
<td>1.08</td>
<td>0.22</td>
</tr>
<tr>
<td>freebase music</td>
<td>151.535</td>
<td>10</td>
<td>3.46</td>
<td>1.52</td>
<td>0.25</td>
</tr>
<tr>
<td>lbnl-network</td>
<td>2.297</td>
<td>169</td>
<td>1.94</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>movies-amazon</td>
<td>3.412</td>
<td>98</td>
<td>0.19</td>
<td>0.51</td>
<td>0.14</td>
</tr>
<tr>
<td>nell-1</td>
<td>233.521</td>
<td>44</td>
<td>0.73</td>
<td>0.73</td>
<td>0.06</td>
</tr>
<tr>
<td><strong>geo mean</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.78</td>
<td>0.86</td>
<td>0.80</td>
<td>0.90</td>
<td>0.85</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>( R )</th>
<th># of iters.</th>
<th>( \text{Cart HP} )</th>
<th>( \text{MG HP} )</th>
<th>( \text{MG TGP} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>30</td>
<td>0.63</td>
<td>0.54</td>
<td>0.12</td>
</tr>
<tr>
<td>32</td>
<td>21</td>
<td>2.05</td>
<td>1.77</td>
<td>0.40</td>
</tr>
<tr>
<td>16</td>
<td>14</td>
<td>6.18</td>
<td>5.34</td>
<td>1.21</td>
</tr>
</tbody>
</table>

\( \text{HP} \) and \( \text{TGP} \) values are given in terms of parallel CPD-ALS times normalized with respect to those of \( \text{Cart HP} \).

---

Fig. 4. Strong scaling curves for CPD-ALS obtained by \( \text{Cart HP} \), \( \text{MG HP} \), and \( \text{MG TGP} \) up to \( P = 1024 \) processors and \( R = 64 \).
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 MG$_{HP}$ and MG$_{TGP}$ models against Cart$_{HP}$ and FG$_{HP}$, all of which utilize intelligent partitioning methods. Fig. 5b compares MG$_{HP}$ and MG$_{TGP}$ against Cart$_{RP}$ [11], which utilizes random cartesian partitioning and hence involves negligible preprocessing overhead. For 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 Cart$_{RP}$ [11], for which no partitioning overhead was considered, while the performance gap between the proposed methods and 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 model 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 (TÜBİTAK) 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


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.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.