Communication Lower Bounds for Matricized Tensor Times Khatri-Rao Product

Grey Ballard and Kathryn Rouse  
Department of Computer Science  
Wake Forest University  
Winston Salem NC, USA  
Email: {ballard,rousek}@wfu.edu

Nicholas Knight  
Courant Institute of Mathematical Sciences  
New York University  
New York NY, USA  
Email: nknight@nyu.edu

Abstract—The matricized-tensor times Khatri-Rao product (MTTKRP) computation is the typical bottleneck in algorithms for computing a CP decomposition of a tensor. In order to develop high performance sequential and parallel algorithms, we establish communication lower bounds that identify how much data movement is required for this computation in the case of dense tensors. We also present sequential and parallel algorithms that attain the lower bounds and are therefore communication optimal. In particular, we show that the structure of the computation allows for less communication than the straightforward approach of casting the computation as a matrix multiplication operation.

I. INTRODUCTION

Tensor decompositions are a powerful tool in the analysis of multidimensional datasets arising from a wide variety of applications. Two of the most popular decompositions, known as CP and Tucker, are generalizations of the matrix singular value decomposition (or principle component analysis) and form low-rank approximations of tensor data. They are used heavily in the scientific computing, signal processing, and machine learning communities [1]–[3], and the formulations and fundamental algorithms for computing these decompositions are well established.

However, their growing popularity, along with the continued increase in the size of datasets across applications, has increased demand for high-performance parallel algorithms and implementations. To deliver efficient solutions for tensor problems, high performance computing can leverage the wealth of knowledge and experience with dense and sparse matrix computations, which are closely related to the computational kernels within tensor decomposition algorithms. In particular, obtaining high performance requires minimizing the cost of data movement among processors and within the memory hierarchy, as the costs of communication are an increasing bottleneck on today’s architectures.

The goal of this work is to focus on the communication costs of the bottleneck computation within algorithms that compute the CP decomposition. The CP decomposition, as we discuss in Section II, approximates a tensor as a sum of rank-one tensors, typically represented as a set of factor matrices, much like a low-rank approximation of a matrix. Nearly all optimization schemes for computing a CP decomposition spend most of their time in a computation known as matricized-tensor times Khatri-Rao product (MTTKRP), and in this work we focus on MTTKRP in the case of dense tensors. Our results are based on a sequential two-level memory model and a distributed-memory parallel model.

The main contributions of this paper are to

• establish sequential and parallel communication lower bounds for dense MTTKRP (Section IV);
• present communication-optimal sequential and parallel dense MTTKRP algorithms (Section V);
• expose the opportunities within tensor computations to achieve better locality than is available within matrix computations (Section VI).

We discuss related work in Section III and conclude the paper in Section VII.

II. PRELIMINARIES

A. CP Decomposition

The CANDECOMP/PARAFAC or canonical polyadic (CP) decomposition is the approximation of a tensor by a sum of rank-one tensors. Given an N-way tensor $\mathbf{X}$ of dimensions $I_1 \times \cdots \times I_N$, a rank-$R$ CP decomposition, represented by $N$ factor matrices $\{ \mathbf{A}^{(k)} \}_{k \in [N]}$, is given by

$$\mathbf{X} \approx \sum_{r \in [R]} \mathbf{a}_1^{(1)} \circ \cdots \circ \mathbf{a}_N^{(N)},$$

where $\mathbf{a}_r^{(k)}$ is the $r$-th column of matrix $\mathbf{A}^{(k)}$, or equivalently,

$$\mathbf{X}(i) \approx \sum_{r \in [R]} \mathbf{A}^{(1)}(i_1, r) \cdots \mathbf{A}^{(N)}(i_N, r), \quad (1)$$

where $i = (i_1, \ldots, i_N)$.

Computing a CP decomposition involves solving a nonlinear optimization problem to minimize the approximation error, typically measured in the $\ell_2$-norm. The most common optimization algorithms either use an alternating least squares (ALS) approach or a gradient-based algorithm. The ALS algorithm alternates among the factor matrices, improving one factor matrix at a time. When all but one factor matrix are fixed, optimizing the variable factor matrix is a linear optimization problem that can solved in closed form via the normal equations. In a gradient-based algorithm, the gradients with respect to all factor matrices are computed and used to determine the variable updates. In both cases, setting up the
normal equations and computing the gradient are bottlenecked by a particular computation that involves the tensor and all but one of the factor matrices. This computation is known as MTTKRP.

B. MTTKRP

MTTKRP inputs an $N$-way tensor $X$, $N \geq 2$, of dimensions $I_1 \times \cdots \times I_N$, a fixed mode $n \in [N]$, and an $(N-1)$-tuple of matrices $\{A^{(k)}\}_{k \in [N] \setminus \{n\}}$ each of dimensions $I_k \times R$. MTTKRP outputs a single matrix $B^{(n)}$, of dimensions $I_n \times R$. (For a fixed $n$, the matrix $A^{(n)}$ and the superscript on $B^{(n)}$ are irrelevant.) Throughout the discussion, the underlying set of values is any nonempty set closed under two binary operations, denoted by addition and multiplication, say, the real numbers.

**Definition 2.1:** An MTTKRP algorithm maps

$$\left( X, \{A^{(k)}\}_{k \in [N] \setminus \{n\}} \right) \mapsto B^{(n)},$$

where for each $(i_n, r) \in [I_n] \times [R]$,

$$B^{(n)}(i_n, r) = \sum_{i} X(i) \prod_{k \in [N] \setminus \{n\}} A^{(k)}(i, r), \quad (2)$$

where summation is over all $i$ with $n$-th entry $i_n$ in the set $\mathcal{I} = [I_1] \times \cdots \times [I_N] \times [R]$.

The products are evaluated atomically, as $N$-ary multiplies.

The atomicity of the $N$-ary multiplies precludes reusing factors across products. Moreover, the generality of the arithmetic model precludes a number of other practical optimizations. For example, assuming existence of a zero element, many operations could be avoided if $X$ were sparse. Or, assuming distributivity, the operation count decreases when factoring products through the sums. Or, assuming the ring axioms hold, Strassen’s algorithm could be used in place of the classical matrix multiplication algorithm. Our ongoing work addresses these optimizations, which change the algorithmic structure in ways that aren’t captured by the present lower bound proof approach.

C. Computation Models

a) Sequential Model: Our model sequential machine includes a single processor, connected to two storage devices called fast and slow memory. Fast memory can hold up to $M$ values at once, while slow memory has unbounded capacity. The processor performs (binary) adds and $N$-ary multiplies on values in fast memory and communicates values between the two memories. Communication consists of loads and stores, instructions that read individual values from slow memory and write them to fast memory, or vice versa. This model is known as the two-level sequential memory model \cite{5} or the I/O complexity model \cite{4}.

b) Parallel Model: Our parallel model includes $P$ processors, each connected to its own local memory and to all other processors via a network. Local memory holds up to $M$ values, so overall the machine holds at most $PM$ values. As in the sequential case, each processor can operate on values in its local memory, while communication now consists of sends and receives, instructions that read individual values from local memory and write them to the network, or vice versa. We assume each processor can send or receive only one value at a time, but two disjoint pairs of processors can communicate simultaneously. This model is known as the MPI model \cite{6}, or $\alpha$-$\beta$-$\gamma$ model \cite{4}. In this work, we focus on the amount of data communicated (bandwidth cost) and ignore the number of messages communicated (latency cost).

III. RELATED WORK

A. Communication Lower Bounds

The pioneering work of Hong and Kung \cite{5} introduced a framework for communication analysis in the sequential model. Using the red-blue pebble game, Hong and Kung derived lower bounds on the number of words that must be communicated when performing a class of algorithms including conventional matrix multiplication. Irony et al. \cite{7} extended Hong and Kung’s results for matrix multiplication to the parallel case using a segmentation argument that we will follow. Ballard et al. \cite{4} extended communication lower bounds from matrix multiplication algorithms to algorithms for any linear algebra computations that can be written as three nested loop (3NL) computations. Smith and van de Geijn \cite{8} tightened the constants in the lower bounds given by Irony et al. and Ballard et al. by changing the operations to scalar fused multiply-adds, optimizing the segment length, and exploiting a bound on the sum (rather than the max) of the data accessed from each array. Additionally, memory-independent bounds were given by Ballard et al. \cite{9} to determine the ranges where perfect strong scaling can be achieved. Demmel et al. \cite{10} considered how memory-independent bounds must change to remain tight for rectangular matrix multiplication with one, two, or three large dimensions. Finally, Christ et al. \cite{11} extended the generality of 3NL computations to prove lower bounds for more arbitrary loop nests: their results apply to our definition of MTTKRP.

B. Algorithms for MTTKRP

The most straightforward sequential algorithm for MTTKRP, when the tensor is dense, involves permuting the tensor to achieve a column- or row-major matricization, forming the Khatri-Rao product explicitly, and then multiplying these two matrices. Note that this approach violates the assumption in Definition 2.1 that the $N$-ary multiplies are performed atomically. An alternative approach avoids the explicit permutation of the tensor and performs the MTTKRP in two steps, the first involving a matrix-matrix multiplication and the second involving a sequence of matrix-vector multiplications. This approach also violates the atomicity assumption. The two-step approach is particularly advantageous when the MTTKRP is to be performed in each mode, like in the CP-ALS or other gradient-based algorithms, as intermediate quantities can be re-used across modes.
In the case of distributed-memory parallel algorithms for MTTKRP, there have been many efforts to improve performance for sparse tensors [12], [14]–[16] in the context of the CP-ALS algorithm. In particular, Smith and Karypis [16] describe a “medium-grained” parallelization scheme that is designed for sparse tensors but can be applied to dense tensors. Indeed, Liavas et al. [17] apply the preceding approach to dense 3-way tensors in computing CP decompositions with non-negativity constraints. Aggour and Yenner [18] also parallel MTTKRP for dense tensors, using a scheme that parallelizes over only the largest dimension of a 3-way tensor.

IV. LOWER BOUNDS

A. Preliminary Lemmas

In this section we state four lemmas that will be useful in our main results. Lemma 4.1 is an inequality that generalizes the Loomis-Whitney inequality [19], which has been used in proving communication lower bounds for matrix computations [4], [7]. Lemma 4.2 provides the solution to a particular linear program that appears in our lower bound proofs. Lemmas 4.3 and 4.4 give solutions to nonlinear optimization problems that appear in later proofs.

The following result concerning Hölder-Brascamp-Lieb-type multilinear inequalities appears in a more general form in [20, Proposition 7.1]; a simpler proof for our special case is given in [11, Theorem 6.6].

**Lemma 4.1:** Consider any positive integers $d$ and $m$ and any $m$ projections $\phi_j : \mathbb{Z}^d \rightarrow \mathbb{Z}^{d_j}$ ($d_j \leq d$), each of which extracts $d_j$ coordinates $S_j \subseteq [d]$ and forgets the $d-d_j$ others. Define

$$\mathcal{P} = \{ s \in [0,1]^m : \Delta \cdot s \geq 1 \},$$

where the $d \times m$ matrix $\Delta$ has entries

$$\Delta_{i,j} = \begin{cases} 1 & i \in S_j \\ 0 & i \notin S_j \end{cases}.$$

If $s \in \mathcal{P}$, then for all $E \subseteq \mathbb{Z}^d$,

$$|E| \leq \prod_{j \in [m]} |\phi_j(E)|^{s_j}.$$

**Lemma 4.2:** The solution of the linear program

$$\min 1^T s \quad \text{subject to} \quad \Delta \cdot s \geq 1 \quad \text{and} \quad s \geq 0,$$

where

$$\Delta = \begin{pmatrix} I_N \times I_N & I_N \times 1 \\ 1_N \times N & 0 \end{pmatrix},$$

is $s^\ast = (1/N, \ldots, 1/N, 1-1/N)^T$ with $1^T s^\ast = 2-1/N$.

**Proof:** The dual linear program is

$$\max 1^T t \quad \text{subject to} \quad \Delta^T \cdot t \leq 1 \quad \text{and} \quad t \geq 0.$$ 

Note that $t^\ast = s^\ast$ is feasible, and $1^T t^\ast = 1^T s^\ast$, so $s^\ast$ is a solution of the primal by linear duality.

**Lemma 4.3:** Given $s > 0$, the optimization problem

$$\max_{x \geq 0} \prod_{i \in [m]} x_i^{s_i} \quad \text{subject to} \quad \sum_{i \in [m]} x_i \leq c$$

yields the maximum value

$$c^{\sum_{i \in [m]} s_i} \prod_{j \in [m]} \left( \frac{s_j}{\sum_{i \in [m]} s_i} \right)^{s_j}.$$

**Proof:** Without loss of generality, we may tighten our condition on the sum to be equality. If $\sum_{i \in [m]} x_i < c$, we can increase one of the $x_i$ which would increase the product because all $x_i \geq 0$. Therefore the maximum product is achieved with equality in the constraint on the sum.

We use Lagrange multipliers to find the maximum in terms of the exponents given by $s$. Our Lagrangian is

$$\mathcal{L}(x_1, \ldots, x_m, \lambda) = x_1^{s_1} \cdots x_m^{s_m} - \lambda(x_1 + \cdots + x_m - c),$$

which has partial derivatives

$$\frac{\partial \mathcal{L}}{\partial x_j} = s_j x_j^{s_j-1} \prod_{i \neq j} x_i^{s_i} - \lambda,$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = c - \sum_j x_j.$$

Setting $\frac{\partial \mathcal{L}}{\partial x_j} = 0$ for each $j$, we have for all $j \neq i$,

$$s_j x_j^{s_j-1} \prod_{k \neq j} x_k^{s_k} = s_i x_i^{s_i-1} \prod_{k \neq i} x_k^{s_k},$$

or $x_j = \frac{s_j}{s_i} x_i$. Setting $\frac{\partial \mathcal{L}}{\partial \lambda} = 0$, we have

$$c = \sum_{i=1}^{m} x_i = \sum_{i \in [m]} \frac{s_i}{s_j} x_j,$$

or $x_j = \frac{cs_j}{\sum_{i \in [m]} s_i}$ for each $j$, which implies

$$\max_{j \in [m]} \prod_{j \in [m]} x_j^{s_j} = \left( \frac{c}{\sum_{i \in [m]} s_i} \right)^{s_j} = c^{\sum_{i \in [m]} s_i} \prod_{j \in [m]} \left( \frac{s_j}{\sum_{i \in [m]} s_i} \right)^{s_j}.$$

**Lemma 4.4:** For any $s \geq 0$, the optimization problem

$$\min_{x \geq 0} \sum_{i \in [m]} x_i \quad \text{subject to} \quad \prod_{i \in [m]} x_i^{s_i} \geq c$$

yields the minimum value

$$\left( \frac{c}{\prod_{i \in [m]} s_i} \right)^{1/\sum_{i \in [m]} s_i}.$$

**Proof:** As in the proof of Lemma 4.3, we note that our constraint can be changed to an equality constraint. If $\prod_{i \in [m]} x_i > c$, then we may decrease at least one $x_i$ and still have the constraint hold. This would decrease the sum, therefore the minimum sum must occur when there is equality in the constraint.

As before, we use Lagrange multipliers to solve the optimization problem. Our Lagrangian is

$$\mathcal{L}(x_1, \ldots, x_m, \lambda) = \sum_{i \in [m]} x_i - \lambda \left( \prod_{i \in [m]} x_i^{s_i} - c \right).$$
and has partial derivatives

$$\frac{\partial L}{\partial x_i} = 1 - \lambda s_i x_i^{s_i-1} \prod_{j \neq i} x_j^{s_j},$$

$$\frac{\partial L}{\partial \lambda} = c - \prod_{i \in [m]} x_i^{s_i}.$$ 

Setting the partial derivatives with respect to $x_i$ to zero we can again derive that for all pairs $j \neq i$, $x_j = x_i^{s_i}$. Additionally when $\frac{\partial L}{\partial \lambda} = 0$,

$$c = \prod_{i \in [m]} x_i^{s_i} = \left( \frac{x_j}{s_j} \right)^{\sum_{i \in [m]} s_i} \prod_{i \in [m]} s_i^{s_i},$$

or $x_j = s_j \left( \frac{c}{\prod_{i \in [m]} s_i} \right)^{1/\sum_{i \in [m]} s_i}$ for each $j$. So

$$\min_{x \geq 0} \sum_{j \in [m]} x_j = \left( \frac{c}{\prod_{i \in [m]} s_i} \right)^{1/\sum_{i \in [m]} s_i} \sum_{i \in [m]} s_i.$$ 



### B. Memory-Dependent Lower Bounds

We first prove Theorem 4.1, a lower bound for the sequential model that depends on the fast memory size $M$. The proof uses the structure of previous matrix computation lower bound proofs [4, 7]. However, to address MTTKRP, it uses a Hölder-Brascamp-Lieb-type inequality (Lemma 4.1) as has been done for more general computations [11]. It also borrows another technique involving Lemma 4.3 that has been used to tighten the constant of the matrix multiplication bound [8], though the technique improves our bound by more than a constant. Theorem 4.1 implies Corollary 4.1, a similar memory-independent bound for the parallel model, where $M$ corresponds to the size of the local memory. We also state an immediate lower bound result for the sequential case (Fact 4.1) based on the size of the input and output data.

**Theorem 4.1:** Any sequential MTTKRP algorithm involves at least

$$\frac{1}{3^{2-1/N}} \frac{NIR}{M^{1-1/N}} - M \quad (4)$$

loads and stores.

**Proof:** We break the stream of instructions that implement a MTTKRP algorithm into complete segments each of which contains exactly $M$ loads and stores, except the last segment which may contain less than $M$ loads and stores (incomplete). We will determine an upper bound on the number of elements of all arrays $X$, $B^{(n)}$, or $A^{(k)}$ that can be accessed during a segment, then use Lemma 4.1 to bound the number of loop iterations that can be evaluated during a segment. We use this upper bound to generate a lower bound for the number of complete segments, from which we generate the lower bound on the communication for any MTTKRP algorithm.

We begin by considering elements of $B^{(n)}$, the factor matrix that is being computed. We consider an element of $B^{(n)}$ live during the segment if it accumulates the result of one or more $N$-ary multiplies during that segment. Any element of $B^{(n)}$ that is live during the segment must either remain in fast memory at the end of the segment or have been stored into slow memory by the end of the segment. At the end of the segment there can be at most $M$ live elements of $B^{(n)}$ that remain in fast memory. Let $S$ be the number of live elements of $B^{(n)}$ that were stored during the segment. Now, consider input elements of $X$, $A^{(k)}$ that are used as arguments for one or more $N$-ary multiplies during the segment. These elements must have been in fast memory at the start of the segment or loaded into fast memory during the segment. The total number of input elements that are in fast memory at the start of segment is at most $M$, and the total number of input elements that can be loaded during the segment is $M - S$. Thus the total number elements from all arrays that an algorithm can access during the segment is at most $3M$.

If $F$ is the subset of the iteration space $I = [I_1] \times \cdots \times [I_N] \times [R]$ evaluated during the segment, then $\phi_j(F)$ corresponds to the set of entries of the $j$-th array that are accessed during the segment. Thus,

$$\sum_{j \in [m]} |\phi_j(F)| \leq 3M.$$ 

See Figure 1 for an example set $F$ and its projections.

To use Lemma 4.1 we first define the linear constraint matrix $\Delta$. For MTTKRP algorithms, the number of projections/arrays is $m = N+1$, corresponding to $N-1$ input factor matrices, one output factor matrix, and the input tensor. The depth of the nested loops is $d = N+1$, corresponding to one loop for each mode of the tensor and one loop over the rank of the...
factor matrices. The first $N$ projections (rows) correspond to the input and output factor matrices, and the last projection corresponds to the input tensor. The first $N$ indices (columns) are $i_1, \ldots, i_N$, and the last index is $r$. So we have

$$\Delta = \begin{pmatrix} I_{N\times N} & 1_{N\times 1} \\ 1_{1\times N} & 0 \end{pmatrix}. $$

By Lemma 4.1 for any $s \in \mathcal{P}$,

$$|F| \leq \prod_{j \in [m]} |\phi_j(F)|^{s_j}.$$ 

Substituting $|\phi_j(F)|$ for $x_j$ and $3M$ as the constant $c$ in the constraint of Lemma 4.3, we see that for any $s \in \mathcal{P},$

$$\prod_{j \in [m]} |\phi_j(F)|^{s_j} \leq (3M)^{\sum_j s_j} \prod_{j \in [m]} \left( \frac{s_j}{\sum_i s_i} \right)^{s_j}.$$ 

In order to obtain the tightest lower bound possible, we wish to choose the $s \in \mathcal{P}$ that minimizes the left hand side of the preceding inequality. Short of that, we can choose to minimize only the first factor $(3M)^{\sum_j s_j}$, which corresponds to solving the linear program Equation 3. By Lemma 4.2, the exponent is minimized by $2 - 1/N$ with $s^* = (1/N, \ldots, 1/N, 1-1/N)^T$. Note that

$$\prod_{j \in [m]} \left( \frac{s_j}{\sum_i s_i} \right)^{s_j} = \left( \frac{1 - 1/N}{2 - 1/N} \right)^{1-1/N} \prod_{j \in [N]} \left( \frac{1}{2 - 1/N} \right)^{1/N} \geq \left( \frac{1}{2 - 1/N} \right)^{2 - 1/N} \left( \frac{1 - 1/N}{2 - 1/N} \right)^{1-1/N} \prod_{j \in [N]} \left( 1/N \right)^{1/N} \leq 1/N.$$ 

Thus $|F| \leq (3M)^{2 - 1/N}/N$ gives an upper bound on the number of $N$-ary multiplies that can be performed in a segment with exactly $M$ loads and stores.

Because $|I| = IR$ there are at least

$$IR \left( \frac{3N}{M} \right)^{2 - 1/N}$$

complete segments. Each segment loads and stores $M$ words, thus there are at least

$$M \cdot \left[ \frac{NIR}{(3M)^{2 - 1/N}} \right]$$

loads and stores.

**Corollary 4.1:** Any parallel MTTKRP algorithm involves at least

$$\frac{1}{3^2 - 1/N} \frac{NIR}{PM^{1 - 1/N}} - M$$

sends and receives.

**Proof:** Since some processor must be associated with at least $|I|/P = IR/P$ loop iterations, we can apply Theorem 4.1 to the computation performed by that processor.

The following additional lower bound for the sequential case is based on the observation that to perform the MTTKRP, the algorithm must access all of the input and output data. Note that the fast memory could be full of useful data at the beginning and end of the computation.

**Fact 4.1:** Any sequential MTTKRP algorithm must perform at least

$$I + \sum_{k \in [N]} I_k R - 2M \quad (5)$$

loads and stores.

**C. Memory-Independent Lower Bounds**

In this section, we prove bounds that do not depend on the fast or local memory size $M$. These bounds focus on the parallel case. The structures of the proofs follow previous work \cite{9,10}, but again we combine a technique used in the context of matrix multiplication \cite{8} (involving Lemma 4.3) to tighten the bounds. Theorems 4.2 and 4.3 establish separate lower bounds under the same assumptions on the parallelization and data distribution. We prove both because either can be the tightest lower bound, depending on relative sizes of the parameters. To show how the bounds simplify and compare for a particular case, we consider tensors with all dimensions the same ($I_k = 1^{1/N}$ for all $k$) and state Corollary 4.2.

**Theorem 4.2:** In any parallel MTTKRP algorithm where each processor initially and finally owns at most $\delta \sum_k I_k R/P$ factor matrix entries and at most $\gamma I/P$ tensor entries, $\gamma, \delta \geq 1$, some processor performs at least

$$2 \left( \frac{NIR}{P} \right)^{\frac{N}{N-1}} - \gamma \frac{I}{P} - \delta \sum_{k \in [N]} \frac{I_k R}{P} \quad (6)$$

sends and receives.

**Proof:** We follow the argument given by Ballard et al. \cite[Lemma 2.3]{9}. Some processor $p$ must evaluate at least $|I|/P = IR/P$ loop iterations. Let $F$ be the set of loop iterations associated with the $N$-ary multiplies performed by that processor. Then using $|\phi_j(F)|$ as before we have that the number of sends and receives performed by that processor must be at least

$$\sum_{j \in [N+1]} |\phi_j(F)| - I/P - \delta \sum_{k \in [N]} I_k R/P,$$

where the first sum is the size of the data the processor must access to evaluate its loop iterations and the negative terms correspond to the useful data that may be in its local memory at the start and end of the computation. From Lemma 4.1 we can bound the size of $F$ in terms of the sizes of the projections:

$$|F| \leq \prod_{j \in [N+1]} |\phi_j(F)|^{s_j}$$

for any $s$ in $\mathcal{P}$. Using $s^* = (1/N, \ldots, 1/N, 1-1/N)^T$ as before, and substituting $|\phi_j(F)|$ for $x_j$ and $IR/P$ as the constant $c$, Lemma 4.4 gives

$$\sum_{j \in [N+1]} |\phi_j(F)| \geq \left( \frac{IR/P}{\prod_{j \in [N+1]} s_j^{s_j}} \right)^{2^{-1/N}} (2 - 1/N) \geq 2 \left( \frac{NIR}{P} \right)^{\frac{N}{N-1}}.$$
Theorem 4.3: In any parallel MTTKRP algorithm where each processor initially and finally owns at most $\delta \sum k k_R/P$ factor matrix entries and at most $\gamma I/P$ tensor entries, $\gamma, \delta \geq 1$, some processor performs at least

$$\min \left( \frac{\sqrt{3\gamma}}{\sqrt{\gamma}} NR \left( \frac{I}{P} \right)^{1/N} - \delta \sum_{j \in [N]} I_j R \left( \frac{\gamma I}{2P} \right) \right)$$

subject to this constraint on the product when

$$\left( \frac{2}{3\gamma} \right)^{-1} \frac{IR^N}{P} \geq \left( \frac{2}{3\gamma} \right)^{-1/2} \frac{I}{P}$$

The two bounds apply in separate cases, but because the first bound dominates when $NR$ is larger than the threshold $(I/P)^{-1/N}$ and the second bound dominates when $NR$ is smaller than the threshold, we can write the overall bound as a sum of the two bounds, as stated. ■

Proof: Under these assumptions, both Theorems 4.2 and 4.3 apply. Given that $I_k = I^{1/N}$, we can simplify the bound from Theorem 4.2 to

$$\Omega \left( \left( \frac{NR}{P} \right)^{\frac{N}{N^2}} - \frac{I}{P} - \frac{NIR^{1/N}R}{P} \right),$$

and we can simplify the bound from Theorem 4.3 to

$$\Omega \left( \min \left( \left( \frac{NR}{P} \right)^{\frac{1}{N}}, \frac{I}{P} \right) \right),$$

assuming $P > 1$.

We now consider two cases. Suppose $NR \geq (I/P)^{-1/N}$. This implies that $(NR/P)^{\frac{N}{N^2}}$ dominates $I/P$, which implies that Equation (9) dominates Equation (10) and simplifies to

$$\Omega \left( \left( \frac{NIR}{P} \right)^{\frac{N}{N^2}} - \frac{NIR^{1/N}R}{P} \right).$$

Again, $NR \geq (I/P)^{-1/N}$ implies that

$$\left( \frac{NIR}{P} \right)^{\frac{N}{N^2}} \geq NR \left( \frac{I}{P} \right)^{1/N} \geq \frac{NIR^{1/N}R}{P},$$

and the bound

$$\Omega \left( \left( \frac{NIR}{P} \right)^{\frac{N}{N^2}} \right)$$

applies.

Suppose $NR \leq (I/P)^{-1/N}$. This implies that Equation (9) degenerates to a negative bound and Equation (10) simplifies to

$$\Omega \left( \left( \frac{IR}{P} \right)^{1/N} \right).$$

The two bounds apply in separate cases, but because the first bound dominates when $NR$ is larger than the threshold $(I/P)^{-1/N}$ and the second bound dominates when $NR$ is smaller than the threshold, we can write the overall bound as a sum of the two bounds, as stated. ■

V. ALGORITHMS

A. Sequential Unblocked Algorithm

Algorithm 1 illustrates a sequential MTTKRP algorithm. It makes no assumption on the fast memory size besides $M \geq N$ (necessary for $N$-ary multiplies). The communication cost of Algorithm 1 is

$$W \leq I + IR(N + 1);$$

the two terms bound the numbers of tensor entry loads and factor-matrix entry loads/stores, respectively.

This counting neglects the possibility that inputs/outputs begin/end in fast memory. For example, if $I + (I_1 + \cdots + I_N)R \leq M$, then $W = 0$ is attained.
Algorithm 1 Sequential Unblocked Algorithm

1: function $B^{(n)} = \text{SEQ-MTTKR}(X, \{A^{(k)}\}_{k \in [N] \setminus \{n\}}, n)$
2: for $i_1 \leftarrow 1$ to $I_1$ do
3:  
4:     for $i_N \leftarrow 1$ to $I_N$ do
5:         load $X(i_1, \ldots, i_N)$
6:         for $r \leftarrow 1$ to $R$ do
7:             load $A^{(k)}(i_k, r)$ ($k \in [N] \setminus \{n\}$)
8:             load $B^{(n)}(i_n, r)$
9:             $B^{(n)}(i_n, r) \leftarrow B^{(n)}(i_n, r) + X(i_1, \ldots, i_N) \cdot \prod_{k \in [N] \setminus \{n\}} A^{(k)}(i_k, r)$
10:     store $B^{(n)}(i_n, r)$
11: end for
12: end for
13: 
14: end for
15: end function

Algorithm 2 Sequential Blocked Algorithm

1: function $B^{(n)} = \text{SEQ-BLOCKED-MTTKR}(X, \{A^{(k)}\}, n, b)$
2: for $j_1 \leftarrow 1$ to $I_1$ step $b$ do
3:  
4:     for $j_N \leftarrow 1$ to $I_N$ step $b$ do
5:         $J_k \leftarrow \min(I_k, j_k + b - 1)$ ($k \in [N]$)
6:         load block $X(j_1:J_1, \ldots, j_N:J_N)$
7:         for $r \leftarrow 1$ to $R$ do
8:             load vectors $A^{(k)}(j_k:J_k, r)$ ($k \in [N] \setminus \{n\}$)
9:             load vector $B^{(n)}(j_n:J_n, r)$
10:         for $i_1 \leftarrow j_1$ to $j_1$ do
11:             for $i_N \leftarrow j_N$ to $J_N$ do
12:                 $B^{(n)}(i_n, r) \leftarrow B^{(n)}(i_n, r) + X(i_1, \ldots, i_N) \cdot \prod_{k \in [N] \setminus \{n\}} A^{(k)}(i_k, r)$
13:             end for
14:         end for
15:     end for
16: store vector $B^{(n)}(J_n:J_n, r)$
17: end for
18: end for
19: end function

B. Sequential Blocked Algorithm

Algorithm 2 illustrates another sequential MTTKR algorithm. The iterations are performed in a different order, which potentially exposes more data reuse.

We control the blocking with the block size $b$. The code is correct for any positive integer $b$ satisfying

$$b^N + Nb \leq M,$$

whence the communication cost is bounded above by

$$I + \left[ \frac{I_1}{b} \right] \cdots \left[ \frac{I_N}{b} \right] \cdot R(N + 1)b.$$

In Section VI-A within the proof of Theorem 6.1 we will weaken and simplify Equation (12) for easier comparison with the lower bounds Equations 3 and 5. We will assume additionally that the fast memory size $M$ is sufficiently large with respect to the tensor order $N$, but not too large with respect to the tensor dimensions $I_1, \ldots, I_N$. Under these assumptions, picking the block size $b$ to be approximately $M^{1/N}$ gives an upper bound of the form

$$O \left( I + \frac{NR}{M^{1 - 1/N}} \right).$$

To see how Equation (13) might be obtained from Equation (12), substitute $b = (M/2)^{1/N}$, supposing $b$ is a positive integer that satisfies Equation (11) and divides $I_1, \ldots, I_N$.

C. Parallel Stationary Tensor Algorithm

We present two parallel algorithms, Algorithms 3 and 4, with the first of which is a special case of the second. Here in Section V-C we present the special case of Algorithm 3 in detail because its notation is simpler and we expect it to apply more frequently in typical applications, where $NR$ is small relative to $I/P$. The general algorithm, Algorithm 4, is presented in Section V-D.

1) Data Distribution: For an $N$-way tensor, we organize processors into an $N$-way logical processor grid. We factor $P = P_1P_2 \cdots P_N$ and identify each processor by an $N$-tuple

$$P = (p_1, \ldots, p_N) \in [P_1] \times \cdots \times [P_N].$$

We partition each tensor dimension $k \in [N]$ into $P_k$ parts,

$$[I_k] = \left\{ S^{(1)}_{p_1} \right\}_{p_1 \in [P_1]}, \ldots, [I_N] = \left\{ S^{(N)}_{p_N} \right\}_{p_N \in [P_N]}.$$

Each processor $p$ stores, initially (before execution),

- the subtensor $X_p = X(S^{(1)}_{p_1}, \ldots, S^{(N)}_{p_N})$,
- for each $k \in [N] \setminus \{n\}$, a part $A^{(k)}_p$ in a partition of

$$A^{(k)}_p = A^{(k)}(S^{(k)}_{p_k} ::).$$
We use the term each processor gathers all the input factor matrix data that must communicate to reduce values that correspond to the same output matrix entries. The data distributions are organized using an N-ary multiplies involving the local tensor. Let us clarify a notational detail: while A(k)p, B(n)p are matrices, the (sub)sets of matrix entries A(k)p, B(n)p need not be (sub)matrices.

2) Algorithm: The pseudocode is given in Algorithm 3. We use the term stationary (tensor) to describe this algorithm because the input tensor is never communicated. Instead, each processor gathers all the input factor matrix data that participates in N-ary multiplies involving the local tensor entries. Then, the local computation is itself an MTTKRP. To compute the output of the global MTTKRP, processors again must communicate to reduce values that correspond to the same output matrix entries. The data distributions are organized using an N-way processor grid so that the gathers and reduction are performed across processor hyperslices using collective communication operations All-Gather and Reduce-Scatter.

Algorithm 3 Parallel Stationary MTTKRP Algorithm

1: function B_p(n) = PAR-STAT-MTTKRP(X_p, \{A_p(k)\}, n)
2: p = (p_1, \ldots, p_N) is my processor id
3: for each k ∈ [N]\{n\} do
4: \ A_p(k) = All-Gather(A_p(k), (\ldots, p_k, \ldots))
5: end for
6: C_p(n) = Local-MTTKRP(X_p, \{A_p(k)\}, n)
7: B_p(n) = Reduce-Scatter(C_p(n), (\ldots, p_n, \ldots))
8: end function

across processors p^\prime with p^\prime_k = p_k;
during execution,
• the submatrices A_p(k), k ∈ [N]\{n\} and
• a matrix C_p(n) the same size as (and used in the summation of) B_p(n); and,
terminal (after execution),
• a part B_p(n) in a partition of
\[ B_p(n) = B(n)(S_p(n), :), \]
across processors p^\prime with p^\prime_n = p_n.
In words, each mode’s factor matrix is distributed block-rowwise across the processor hyperslices of that mode, and each block row block is then partitioned arbitrarily across the processors in its hyperslice. During execution, these block rows are replicated within hyperslices.

Let us clarify a notational detail: while A_p(k), B_p(n) are matrices, the (sub)sets of matrix entries A_p(k), B_p(n) need not be (sub)matrices.

3) Analysis: We analyze the communication cost first. Communication occurs only in the All-Gather and Reduce-Scatter collectives in Lines 4 and 7. Each processor p is involved in N−1 All-Gathers (Line 4 k ∈ [N]\{n\}) and one Reduce-Scatter (Line 7). Over all processors, Line 4 (k ∈ [N]\{n\}) specifies P_k simultaneous All-Gathers, and Line 7 specifies P_n simultaneous Reduce-Satters, one for each hyperslice of the processor grid normal to the k-th dimension.

Towards an upper bound, we suppose the collectives are performed in a blocking manner. For any N−1 All-Gathers in Line 4 (k ∈ [N]\{n\}) and any Reduce-Scatter in Line 7 there exists a processor p involved in all. This justifies our upper bound approach, to examine any N collectives each with maximal communication cost among the P_k performed simultaneously (we need not specify the common processor p).

Since we do not quantify latency cost in this work, we will use the simpler bucket algorithms. A bucket All-Gather or Reduce-Scatter algorithm with q processors proceeds in q−1 steps, at each of which each processor passes left an array of size at most w. That is, w is the largest local array size before (All-Gather) or after (Reduce-Scatter) the collective. The communication cost is at most (q−1)w, which is (bandwidth-) optimal for perfectly balanced data distributions [21]. For Reduce-Scatter, there is also an arithmetic cost of at most (q−1)w operations (here, additions).

In the present cases, we have q = P/P_k for Line 4 (k ∈ [N]\{n\}) and Line 7 (k = n). The local vector size w depends on the data distributions specified in Section V.C.1

\[ w \leq \begin{cases} \max_p \text{nnz}(A_p(k)) & k \in [N]\{n\} \\ \max_p \text{nnz}(B_p(n)) & k = n. \end{cases} \]

The overall communication cost is thus bounded above,

\[ \sum_{k \in [N]} \left( \frac{P}{P_k} - 1 \right) \begin{cases} \max_p \text{nnz}(A_p(k)) & k \neq n \\ \max_p \text{nnz}(B_p(n)) & k = n. \end{cases} \] (14)

The arithmetic cost is bounded above in terms of the costliest local MTTKRP (Line 6) and the costliest Reduce-
Scatter (Line 7): the number of operations is at most
\[ NR \max_{p} \left( \prod_{k \in [N]} |S_{p_k}^{(k)}| \right) + \left( \frac{P_{n}}{P_{n}} - 1 \right) \max_{p} \left( \text{unz}(B_{p}^{(n)}) \right). \]

The per-processor storage cost is bounded above,
\[ \max_{p} \left( \prod_{k \in [N]} |S_{p_k}^{(k)}| + \sum_{k \in [N]} |S_{p_k}^{(k)}| R \right). \]  

Assuming we can choose a processor grid such that \( P_{k} = I_{k}/(I/P) \) and divides \( I_{k} \) evenly, we choose the data distribution such that \( |S_{p_k}^{(k)}| = I_{k}/P_{k} \) for \( k \in [N] \), which simplifies these upper bounds: the communication cost bound Equation (14) is
\[ O \left( NR \left( \frac{I}{P} \right)^{1/N} \right), \]
the arithmetic cost bound Equation (15) is
\[ O \left( \frac{NR}{P} \right), \]
and the (per-processor) storage cost bound Equation (16) is
\[ O \left( \frac{I}{P} + NR \left( \frac{I}{P} \right)^{1/N} \right). \]

We weaken these assumptions on the processor grid and make them more explicit in the proof of Theorem 6.2.

We note that to save some arithmetic, the algorithm could break the atomicity of the \( N \)-ary multiplies without changing the communication costs of the algorithm: each processor could precompute the explicit local Khatri-Rao product and perform a local matrix multiplication, reducing the first term in Equation (15) to
\[ R \max_{p} \left( \left( \prod_{k \in [N]} |S_{p_k}^{(k)}| \right) \left( 2 + \frac{1}{|S_{p_k}^{(n)}|} \right) \right), \]  

which is \( O(IR/P) \) with a load-balanced tensor distribution.

D. Parallel General Algorithm

This section studies Algorithm 4, a generalization of the stationary tensor algorithm, Algorithm 3, described in Section \( \text{\textsc{V}} \). Algorithm 4 parallelizes over all \( N+1 \) dimensions of the iteration space: the \( N \) tensor dimensions, bounded by \( I_{1}, \ldots, I_{N} \), and the matrix column dimension, bounded by \( R \). In contrast, recall that Algorithm 3 parallelizes over just the \( N \) tensor dimensions. Roughly speaking, Algorithm 4 is more communication efficient than Algorithm 3 when \( NR \) is large relative to \( I/P \).

1) Data Distribution: For an \( N \)-way tensor, we organize processors into an \( (N+1) \)-way logical processor grid. We factor \( P = P_{0} P_{1} P_{2} \cdots P_{N} \) and identify each processor by an \( (N+1) \)-tuple
\[ p = (p_{0}, p_{1}, \ldots, p_{N}) \in [P_{0}] \times [P_{1}] \times \cdots \times [P_{N}], \]

Algorithm 4 Parallel General MTTKRP Algorithm

1: function \( B_{p}^{(n)} = \text{PAR-GEN-MTTKRP}(X_{p}, \{A_{p}^{(k)}\}, n) \)
2: \( p = (p_{0}, p_{1}, \ldots, p_{N}) \) is my processor id
3: \( X_{p_{1}, \ldots, p_{N}} = \text{All-Gather}(X_{p_{1}, \ldots, p_{N}}) \)
4: for each \( k \in [N] \setminus \{n\} \) do
5: \( A_{p_{k}, p_{0}} = \text{All-Gather}(A_{p_{k}, p_{0}} \cdot (p_{0}, \ldots, p_{k}, \ldots, n)) \)
6: end for
7: \( C_{p_{0}, p_{0}} = \text{Local-MTTKRP}(X_{p_{1}, \ldots, p_{N}}, \{A_{p_{k}, p_{0}}\}, n) \)
8: \( B_{p}^{(n)} = \text{Reduce-Scatter}(C_{p_{0}, p_{0}}, (p_{0}, \ldots, p_{k}, \ldots, n)) \)
9: end function

As before, we partition each tensor dimension \( k \in [N] \) into \( P_{k} \) parts,
\[ [I_{k}] = \left\{ s_{p_{k}}^{(1)} \left| p_{k} \in [P_{k}] \right. \right\}, \ldots, [I_{N}] = \left\{ s_{p_{N}}^{(N)} \right\}_{p_{N} \in [P_{N}]}. \]

Additionally we now partition the matrix column dimension into \( P_{0} \) parts,
\[ [R] = \left\{ T_{p_{0}} \right\}_{p_{0} \in [P_{0}]}. \]

Each processor \( p \) stores, initially (before execution),
- a part \( X_{p} \) in a partition of
  \[ X_{p_{1}, \ldots, p_{N}} = \left\{ s_{p_{1}}^{(1)} \right\}_{p_{1} \in [P_{1}]}, \ldots, \left\{ s_{p_{N}}^{(N)} \right\}_{p_{N} \in [P_{N}]}. \]
- and during execution,
  - the subtensor \( X_{p_{1}, \ldots, p_{N}} \),
  - the submatrices \( A_{p_{k}, p_{0}} \), \( k \in [N] \setminus \{n\} \), and \( B_{p_{0}, p_{0}}^{(n)} \),
- and a matrix \( C_{p_{0}, p_{0}} \), the same size as (and used in the summation of) \( B_{p_{0}, p_{0}}^{(n)} \), and,

2) Algorithm: As mentioned at the beginning of Section \( \text{\textsc{V}} \), the general algorithm Algorithm 4 parallelizes over all \( N+1 \) dimensions of the iteration space: unlike the stationary algorithm Algorithm 3, entries of the tensor \( X_{p} \) are now communicated among processors. One can think of Algorithm 4 as logically dividing the output factor matrix \( B_{p}^{(n)} \) into \( P_{0} \) block-columns, each assigned to a separate subset of \( P/P_{0} \) processors, and running Algorithm 3 on each subset of processors.

The structure of Algorithm 4 is very similar to Algorithm 3: each processor gathers the necessary input data, performs local computation, and then participates in a Scatter-Reduce
Equation (18) and the storage cost bound Equation (20) are bounds can be simplified. The communication cost bound has changed due to distributing matrix columns. Comparing with Equation (16), we notice that the second term due to the new All-Gather (Line 3), as well as the modified communication cost.

Comparing with Equation (15), we notice that the first term and the second term has changed for the same reason as the blocking in the matrix column dimension.

The per-processor storage cost is bounded above by

\[
\max_P \left( \prod_{k \in [N]} |S^{(k)}_{pk}| + \sum_{k \in [N]} |S^{(k)}_{pk}| \cdot |T_{pk}| \right). \tag{20}
\]

Comparing with Equation (16), we notice that the second term has changed due to distributing matrix columns.

Assuming we can choose a processor grid such that \(P_0 \approx (NR)^{N/(2N-1)}/((JP)^{(N-1)/(2N-1)})\) and \(P_k \approx I_k/(IP_0/P)^{1/N}\) for \(k \in [N]\), and that we can choose the data distribution such that \(|S^{(k)}_{pk}| = I_k/P_k, \ |T_{pk}| = R/P_0,\) \(nnz(A^{(k)}_p) = I_k/P,\) \(nnz(B^{(n)}_p) = I_kR/P,\) and \(nnz(B^{(n)}_p) = I_kR/P\) (assuming everything divides evenly), these upper bounds can be simplified. The communication cost bound Equation (18) and the storage cost bound Equation (20) are

\[
O \left( NR \left( \frac{I}{P} \right)^{1/N} + \left( \frac{NIR}{P} \right)^{\frac{N}{N-1}} \right),
\]

and the arithmetic cost bound Equation (19) is

\[
O \left( \frac{NIR}{P} \right).
\]

The trick for reducing arithmetic discussed in Section V-C3 breaking atomicity of the \(N\)-ary multiplies, applies here as well. The result is reducing the first term in the upper bound Equation (19) to

\[
\max_P \left( \left| \prod_{k \in [N]} |S^{(k)}_{pk}| \right| \left( 2 + \frac{1}{|S^{(n)}_{p0}|} \right) \right),
\]

which is \(O(IR/P)\) assuming a load-balanced distribution.

VI. DISCUSSION

A. Sequential Case

We would like to compare the upper bound,

\[
W_{ub} = I + (N + 1) \left( \prod_{k \in [N]} \frac{I_k}{b} \right) bR, \tag{21}
\]

due to blocking in the matrix column dimension, and the second term has changed for the same reason as the communication cost.

The arithmetic cost is bounded above by

\[
N \max_P \left( \left| T_{p0} \right| \prod_{k \in [N]} |S^{(k)}_{pk}| \right) + \left( \frac{P}{P_0P_n} - 1 \right) \max_P nnz(B^{(n)}_p). \tag{19}
\]

Comparing with Equation (15), we notice that the first term has changed due to blocking in the matrix column dimension, and the second term has changed for the same reason as the communication cost.

We now show that under certain assumptions on \(M\), for example assuming that the tensor is too large to fit in fast memory, the upper bound and lower bounds differ by no more than a constant.

Theorem 6.1: Suppose \(M\) is sufficiently larger than the number of dimensions \(N\) and that each dimension \(I_k\) is sufficiently larger than \(M^{1/N}\). Then Algorithm 2 is communication optimal to within a constant factor.

Proof: Suppose there exist positive constants \(\alpha, \beta, \gamma, \delta, \epsilon\) such that

\[
M \geq \left( \frac{N^{1/N}}{\alpha} \right)^{\frac{N}{N-1}} \alpha < 1 \tag{25}
\]

\[
M \geq \left( \frac{1}{\alpha^{1/N} - \beta^{1/(N-1)}} \right)^N \beta < \alpha^{1-1/N} \tag{26}
\]

\[
M \leq \left( \frac{N\gamma}{\alpha^{1/N} - 1} \sum_{k \in [N]} I_k \right)^N \gamma > 1 + \frac{1}{N} \tag{27}
\]

\[
M \leq \frac{1}{2} \left( (1 - \delta)I + \sum_{k \in [N]} I_kR \right) \delta < 1 + \frac{\sum_{k \in [N]} I_kR}{IR} \tag{28}
\]

\[
M \leq \left( \frac{1}{3^{2-1/N} - \epsilon} \right) NIR \epsilon < \frac{1}{3^{2-1/N}}. \tag{29}
\]

For Algorithm 2, we choose block size

\[
b = \left( \frac{\beta M^{1/N}}{M^{1/N}} \right) \tag{30}
\]

It follows from Equation (25) that \(b\) satisfies Equation (22). It follows from Equation (26) that \(b \geq 1\) and, moreover,

\[
b^{N-1} \geq \beta M^{1-1/N}. \tag{31}
\]
It follows from Equation (27) that 
\[ \prod_{k \in [N]} \left[ \frac{I_k}{b} \right] \leq \gamma \frac{I}{b^N} \frac{N}{N + 1}. \]
Since \( \beta < 1 \), it then follows that 
\[ W_{ab} \leq \frac{\gamma}{\beta} \left( I + \frac{NIR}{M^{1-1/N}} \right). \]
It follows from Equation (28) that 
\[ W_{102} \geq \delta I. \]
It follows from Equation (29) that 
\[ W_{b1} \geq \epsilon \frac{NIR}{M^{1-1/N}}. \]
Since these are positive lower bounds, 
\[ \max(W_{b1}, W_{102}) \geq \min(\delta, \epsilon) \frac{I + \frac{NIR}{M^{1-1/N}}}{2} > 0, \]
which matches the upper bound to within a constant factor.

To illustrate the hypotheses Equations (25) to (29) of Theorem 6.1, take, for example, the constants \( \beta = 1 - \alpha = 1/100 \), \( \gamma = 100 \), and \( \delta = \epsilon = 1/10 \), which satisfy the right-hand inequalities for all fast memory sizes \( M \) and problem parameters \( N, I_1, \ldots, I_N, R \). Clearly there are infinitely many choices of \( M \) and the problem parameters that satisfy the left-hand inequalities. For example, supposing \( N \leq 10 \) and \( I_1 = I_2 = \cdots = I_N \), the left-hand inequalities require that the fast memory size \( M \) is bounded by \( 10^4 \) (due to Equations (25) and (26), and above by the minimum of \( I/1000 \) (due to Equations (27) and (28)) and \( \sqrt{NIR} \) (due to Equation (29))). We claim that this example includes parameters that are representative of real-world machines and problems of practical interest. Of course, since we have placed a constant upper bound on \( N \), this example does not illustrate (asymptotic) behavior with respect to \( N \).

We also compare the communication cost of Algorithm 3 with MTTKRP via matrix multiplication approach. We assume a communication-optional matrix multiplication is used, achieving \( O(I + IR/M^{1/2}) \) communication cost and performing \( 2IR \) operations. Here, the cost of explicitly forming the Khatri-Rao product matrix is a lower order term, assuming \( R < I_k \) for all \( k \in [N] \). Assuming \( N = O(M^{1/2-1/N}) \), the communication cost of Algorithm 3 never exceeds that of MTTKRP via matrix multiplication.

If the communication cost is dominated by accessing the tensor elements (i.e., \( R = O(M^{1/2}) \)), then both approaches perform the same amount of communication and Algorithm 3 performs a factor of \( N/2 \) more computation. If the communication cost is dominated by repeatedly accessing the factor matrix elements (i.e., \( NR = \Omega(M^{1-1/N}) \)), then Algorithm 3 is more efficient, requiring a factor of \( O(M^{1/2-1/N}/N) \) less communication.

In practice, we expect \( N \) to be very small relative to \( M \), so the assumption \( N = O(M^{1/2-1/N}) \) is mild. However, we also expect \( R \) to be small relative to \( M \), and in that case, the dominant communication cost of reading tensor elements from memory is shared by both approaches. In this case, the matrix multiplication approach benefits from fewer operations, and in practice it can also exploit highly tuned software for matrix multiplication.

B. Parallel Case

Recall from Sections V-C and V-D that we presented two parallel algorithms, Algorithms 3 and 4, the former being the special case of the latter with \( P_0 = 1 \).

The communication upper bound for Algorithm 4
\[ (P_0 - 1) \cdot \max_k \text{nnz}(X_p) \]
\[ + \sum_{k \in [N]} \left( \frac{P}{P_0 P_k} - 1 \right) \cdot \text{nnz}(A_p^{(k)}) \]
\[ = \sum_{k \in [N]} \left( \frac{P}{P_0 P_k} - 1 \right) \cdot \text{nnz}(B_p^{(n)}) \]
\[ = \sum_{k \in [N]} \left( \frac{P}{P_0 P_k} - 1 \right) \cdot \text{nnz}(B_p^{(n)}) \]
\[ = \sum_{k \in [N]} \left( \frac{P}{P_0 P_k} - 1 \right) \cdot \text{nnz}(B_p^{(n)}) \]

is valid for any factorization \( P = P_0 P_1 \cdots P_N \) and data distribution specified in Section V-D1 (Recall that \( X_p, A_p^{(k)} \) \( (k \in [N] \setminus \{n\}) \), and \( B_p^{(n)} \) denote the distributed subsets of tensor and factor matrix entries.) We wish to compare this upper bound with the lower bound from Theorem 4.2
\[ 2 \left( \frac{NIR}{P} \right)^{\frac{2}{N} - 1} - \gamma I_P - \delta \sum_{k \in [N]} \frac{I_k R}{P}, \]
and the lower bound from Theorem 4.3
\[ \min \left\{ \left( \frac{NIR}{P} \right)^{\frac{1}{N}} - \gamma I_P \right\} \]
\[ \leq \sum_{k \in [N]} \frac{I_k R}{P}, \]

Theorem 6.2: Suppose the number of processors \( P \) is sufficiently large and factorable, and suppose that the tensor dimensions and rank \( R \) are sufficiently large with respect to \( P \). Then Algorithm 4 is communication optimal to within a constant factor.

Proof: To instantiate \( W_{\text{par}}^{(ab)} \), we must specify a processor grid (i.e., a factorization of \( P \) into a product \( P_0 P_1 \cdots P_N \) of positive integers) as well as the distributions of the tensor and factor matrices. For any processor grid, recalling the notation of Section V-D1, we can define a data distribution where, for each processor \( p \),
\[ \text{nnz}(X_p) \leq 1 \prod_{k \in [N]} \frac{I_k / P_k}{P_0}, \]
\[ \text{nnz}(A_p^{(k)}) \leq \left[ \frac{I_k / P_k}{R/P_0} \right]/(P_0/P_0), \]
\[ \text{nnz}(B_p^{(n)}) \leq \left[ \frac{I_n / P_n}{R/P_0} \right]/(P_0/P_0). \]

To instantiate \( W_{\text{par}}^{(ab)} \), we must assume that that no processor owns more than \( \gamma I/P \) tensor entries or \( \delta \sum_k I_k R/P \) factor matrix entries, for some constants \( \gamma, \delta \geq 1 \). For any \( \gamma, \delta > 1 \), we can manipulate the upper bounds in Equation 33 to derive relations on the machine and problem parameters such that these balance constraints hold. In particular, we suppose there exist constants \( \alpha, \beta > 1 \) such that \( \gamma > \alpha, \beta > 1 \).
\[ \delta > \alpha^{1/N} \beta, \text{ and, for all } k \in [N], \]
\[ P_k \leq (\alpha^{1/N} - 1)I_k, \quad P \leq (\gamma - \alpha)I, \]
\[ P_0 \leq (\beta - 1)R, \quad P \leq (\gamma - \alpha^{1/N} \beta)I_kR. \]

(34)

These hypotheses also yield a simpler upper bound,
\[ W_{\text{ub}}^\text{par} \leq \gamma (P_0 - 1) \frac{I}{P} + \delta \sum_{k \in [N]} I_kR \frac{P}{P}. \]

(35)

We now consider two cases, when \( NR \leq (I/P)^{1-1/N} \) and when \( NR > (I/P)^{1-1/N} \). In each case, under additional hypotheses, Equation (35) attains one of the two lower bounds Equations (6) and (7).

In the first case, \( NR \leq (I/P)^{1-1/N} \), we suppose there exist a constant \( \epsilon > 0 \) such that \( P \) factors as \( P_0P_1 \cdots P_N \) with \( P_0 = 1 \) and, for all \( k \in [N] \), \( I_k/P_k \leq (\epsilon/\delta)(I/P)^{1/N} \).

Additionally, we suppose there exists a constant \( \eta \), \( 0 < \eta < \sqrt{2/(3\gamma)} \) such that

\[ P \geq \left( \frac{\delta}{\sqrt{2/(3\gamma) - \eta NI/N^2}} \right)^{N^2}. \]

The first hypothesis simplifies the upper bound Equation (35) to \( W_{\text{ub}}^\text{par} \leq \epsilon \cdot NR(I/P)^{1/N} \), while the second hypothesis simplifies the lower bound Equation (7) to \( W_{\text{lb}}^\text{hr} \geq \eta \cdot NR(I/P)^{1/N} \).

In the second case, \( (NR)^N > (I/P)^{N-1} \), we suppose there exist constants \( \mu, \nu > 0 \) such that \( P \) factors as \( P_0P_1 \cdots P_N \),

\[ \frac{\delta}{\nu} \left( \frac{(NR)^{N-1}}{(I/P)^{N-1}} \right)^{\frac{1}{N^2}} I_k \]
\[ P_k \leq P_0 \leq \mu \gamma \left( \frac{(NR)^{N}}{(I/P)^{N-1}} \right)^{\frac{1}{N^2}}, \]

for each \( k \in [N] \). Additionally, we suppose there exists a constant \( \tau, 0 < \tau < 2 - \gamma \), such that

\[ P \geq \left( \frac{\delta}{\sqrt{2/(\gamma + \tau)} \sum I_k} \right)^{\frac{2N-1}{N^2}} R. \]

The first hypothesis simplifies the upper bound Equation (35) to \( W_{\text{ub}}^\text{par} \leq (\mu + \nu) \cdot (NR/P)^{N/(2N-1)} \), while the second hypothesis simplifies the lower bound Equation (6) to \( W_{\text{lb}}^\text{hr} \geq \tau \cdot (NR/P)^{N/(2N-1)} \). In each of the two cases, the gap is a constant factor.

To illustrate the hypotheses of Theorem 6.2, we set \( \gamma = \delta = 1.75, \alpha^{1/N} = 1.05, \) and \( \beta = 1.5 \) and assume \( 3 \leq N \leq 10 \), for example, and the assumptions in Equation (34) for the upper bound simplification to apply become \( P_k \leq 0.05I_k, \) \( P \leq 0.7I, P_0 \leq 0.5R, \) and \( P \leq 0.175I_kR \). With \( \eta = \tau = 0.1 \) and assuming \( I_k = I^{1/N} \) for all \( k \), the assumptions necessary for the lower bound simplifications to apply become \( P \geq 7 \) and \( P \geq 465NR/I^{1-1/N} \), respectively.

We also compare Algorithm 4 with the MTTKRP via matrix multiplication approach. For comparison, we use the theoretical costs of communication-optimal parallel matrix multiplication algorithms [10]. We assume the Khatri-Rao product matrix is constructed explicitly without communication and in the distribution required to achieve the optimal communication costs of the matrix multiplication. For simplicity, we consider the case that \( I_k = I^{1/N} \) for all \( k \in [N] \). As in the case of our parallel algorithm, the optimal choice of matrix multiplication algorithm depends on the relative size of \( P \), yielding many cases for comparison.

We consider only the extreme cases, “small \( P \)” and “large \( P \)”, though we expect our algorithm to yield benefits in all cases. For parallel multiplication of matrices of dimensions \( I^{1/N} \times I^{N-1} \) and \( I^{N-1} \times R \), if \( P \leq I^{1-1/N} \), then the communication cost is \( I^{1/N} R \), and if \( P \geq I^2/R^2 \), then the communication cost is \( (IR/P)^{2/3} \), assuming enough memory is available [10]. For comparison, if \( P \leq I/(NR)^{N/(N-1)} \), then Algorithm 4 (which reduces to Algorithm 3 in this case) is optimal with communication cost \( NR(I/P)^{1/N} \); if \( P \geq I/(NR)^{N/(N-1)} \), then Algorithm 4 is optimal with communication cost \( (NR/P)^{N/(2(N-1))} \).

Thus, we define the small \( P \) case by \( P \leq \min \{ I^{1-1/N}, I/(NR)^{N/(N-1)} \} \) and the large \( P \) case by \( P \geq \max \{ I^2/R^2, I/(NR)^{N/(N-1)} \} \). In the small \( P \) case, our algorithm performs a factor of \( O(P^{1/N}/N) \) less communication than MTTKRP via matrix multiplication. In the large \( P \) case, our algorithm performs a factor of \( O((IR/P)^{(N-2)/(6N-3)}N^{N/(2(N-1))}) \) less communication. Again, this comparison ignores the communication cost required to form the explicit Khatri-Rao product assuming only one copy of the input matrices are initially distributed across processors.

Figure 4 provides a concrete comparison for a particular
case, where \( I_1 = I_2 = I_3 = R = 2^{15} \) and the number of processors ranges from \( 2^0 \) up to \( 2^{10} \). We see that our proposed algorithms perform less communication than matrix multiplication throughout the range of processors, and that Algorithm 3 and Algorithm 4 diverge only when \( P \geq 2^{27} \). When there are \( 2^{17} = 131.072 \) processors, Algorithm 3 and Algorithm 4 perform approximately \( 25 \times \) less communication than the matrix multiplication approach. This illustrates the benefits of exploiting the multi-way structure of the computation and the observation that Algorithm 3 is sufficient for most practical problems. We note that the kink in the matrix multiplication curve is due to a switch from a 1D parallel algorithm (“1 large dimension” case) to a 2D parallel algorithm (“2 large dimension” case) and that these communication costs are optimal for matrix multiplication, up to constant factors. We also note that for \( P > 2^{30} \), which is the number of elements in each factor matrix, the All-Gather and Reduce-Scatter collectives require more efficient algorithms than the ones described in Section VII.

In summary, the main disadvantage of the matrix multiplication approach is that the Khatri-Rao product is treated as a general matrix despite the fact that its structure means that it depends on fewer parameters and therefore can be communicated more efficiently (in fewer words) across processors.

VII. CONCLUSION

Because efficient algorithms and high performance implementations exist for matrix computations, it is reasonable to recast tensor computations as matrix computations. However, the lower bounds proved in this work demonstrate an opportunity to avoid communication by exploiting the structure of the tensor computation itself. In particular, we have shown how to extend a lower bound approach for generic programs [11] for a particular tensor computation known as MTTKRP, which is the bottleneck for algorithms that compute CP decompositions. By demonstrating (optimal) algorithms that attain these lower bounds, we have identified a design space for implementations that we expect to achieve high performance in practice.

In many applications, the rank \( R \) is small relative to the tensor dimensions. When \( R \) is also small relative to the fast memory size \( M \), as discussed in Section VIIA, we expect only limited practical benefits of the sequential algorithm (Algorithm 2). However, we believe the parallel algorithms will be very competitive in practice. The simpler algorithm (Algorithm 3) may be the most useful, particularly when \( R \) is small. However, the general algorithm (Algorithm 4) will likely perform better for large numbers of processors, even when \( R \) is small. The parallel data distributions are also natural ones for tensors, generalizing distributions already used for other parallel tensor computations.

While this work focuses on a single MTTKRP computation (corresponding to a single mode), the computation nearly always occurs in the context of an optimization algorithm that requires repeatedly computing MTTKRP for each mode of the tensor. In this context, it is beneficial to optimize across multiple MTTKRP computations, because they share both data and intermediate computations. Thus, optimizing over multiple MTTKRP computations can save both communication and computation.

Our communication lower-bound approach extends to algorithms for multiple MTTKRP computations. Extensions are possible for other related computational kernels, such as those within algorithms for computing Tucker and other decompositions.

ACKNOWLEDGMENT

This work has been funded in part by the Laboratory-Directed Research & Development (LDRD) program at Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA-0003525.

REFERENCES

[1] T. G. Kolda and B. W. Bader, “Tensor decompositions and applications,” SIAM Review, vol. 51, no. 3, pp. 455–500, September 2009. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/07070111X
[2] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, “Tensor decompositions for learning latent variable models,” Journal of Machine Learning Research, vol. 15, pp. 2773–2832, 2014. [Online]. Available: http://jmlr.org/papers/v15/anandkumar14b.html
[3] N. D. Sidiropoulos, L. D. Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, “Tensor decomposition for signal processing and machine learning,” IEEE Transactions on Signal Processing, vol. 65, no. 13, pp. 3551–3582, July 2017.
[4] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz, “Communication lower bounds and optimal algorithms for numerical linear algebra.” Acta Numerica, vol. 23, pp. 1–155, May 2014. [Online]. Available: http://journals.cambridge.org/article_S0962492914000038
[5] J. W. Hong and H. T. Kung, “I/O complexity: The red-blue pebble game,” in Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, ser. STOC ’81. ACM, 1981, pp. 326–333.
[6] R. Thakur, R. Rabenseifner, and W. Gropp, “Optimization of collective communication operations in MPICH,” International Journal of High Performance Computing Applications, vol. 19, no. 1, pp. 49–66, 2005. [Online]. Available: http://hpc.sagepub.com/content/19/1/49.abstract
[7] D. Irony, S. Toledo, and A. Tiskin, “Communication lower bounds for distributed-memory matrix multiplication,” J. Parallel Distrib. Comput., vol. 64, no. 9, pp. 1017–1026, 2004.
[8] T. M. Smith and R. A. van de Geijn, “Pushing the bounds for matrix-matrix multiplication,” arXiv, Tech. Rep. 1702.02017, 2017. [Online]. Available: http://arxiv.org/abs/1702.02017
[9] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, “Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds,” in Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures, ser. SPAA ’12. New York, NY, USA: ACM, June 2012, pp. 77–79. [Online]. Available: http://doi.acm.org/10.1145/2312005.2312021
[10] J. Demmel, D. Eilam, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger, “Communication-optimal parallel recursive rectangular matrix multiplication,” in Proceedings of the 27th IEEE International Symposium on Parallel and Distributed Processing, ser. IPDPS ’13, 2013, pp. 261–272. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2013.80
[11] M. Christ, J. Demmel, N. Knight, T. Scanlon, and K. Yelick, “Communication lower bounds and optimal algorithms for programs that reference arrays - part 1,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-61, May 2013. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-61.html

[12] B. W. Bader and T. G. Kolda, “Efficient MATLAB computations with sparse and factored tensors,” SIAM Journal on Scientific Computing, vol. 30, no. 1, pp. 205–231, December 2007.

[13] A.-H. Phan, P. Tichavsky, and A. Cichocki, “Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations,” Signal Processing, IEEE Transactions on, vol. 61, no. 19, pp. 4834–4846, Oct 2013.

[14] J. H. Choi and S. V. N. Vishwanathan, “DFaCTo: Distributed factorization of tensors,” in Proceedings of the 27th International Conference on Neural Information Processing Systems, ser. NIPS ’14. Cambridge, MA, USA: MIT Press, 2014, pp. 1296–1304. [Online]. Available: http://dl.acm.org/citation.cfm?id=2968826.2968971

[15] O. Kaya and B. Uçar, “Scalable sparse tensor decompositions in distributed memory systems,” in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ser. SC ’15. New York, NY, USA: ACM, 2015, pp. 77:1–77:11. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807624

[16] S. Smith and G. Karypis, “A medium-grained algorithm for distributed sparse tensor factorization,” in IEEE 30th International Parallel and Distributed Processing Symposium, May 2016, pp. 902–911.

[17] A. P. Liavas, G. Kostoulas, G. Louakis, K. Huang, and N. D. Sidiropoulos, “Nesterov-based parallel algorithm for large-scale nonnegative tensor factorization,” in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), March 2017, pp. 5895–5899.

[18] K. S. Aggour and B. Yener, “A parallel PARAFAC implementation & scalability testing for large-scale dense tensor decomposition,” Rensselaer Polytechnic Institute, Tech. Rep. 16-02, 2016. [Online]. Available: https://www.cs.rpi.edu/research/pdf/16-02.pdf

[19] L. H. Loomis and H. Whitney, “An inequality related to the isoperimetric inequality,” Bulletin of the AMS, vol. 55, pp. 961–962, 1949.

[20] J. Bennett, A. Carbery, M. Christ, and T. Tao, “Finite bounds for Hölder-Brascamp-Lieb multilinear inequalities,” Mathematical Research Letters, vol. 17, no. 4, pp. 647–666, 2010.

[21] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, “Collective communication: theory, practice, and experience,” Concurrency and Computation: Practice and Experience, vol. 19, no. 13, pp. 1749–1783, 2007. [Online]. Available: http://dx.doi.org/10.1002/cpe.1206

[22] W. Austin, G. Ballard, and T. G. Kolda, “Parallel tensor compression for large-scale scientific data,” in Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium, May 2016, pp. 912–922. [Online]. Available: https://www.computer.org/csdl/proceedings/ipdps/2016/2140002140a912-abs.html

[23] G. Ballard, A. Drusinsky, N. Knight, and O. Schwartz, “Hypergraph partitioning for sparse matrix-matrix multiplication,” ACM Transactions on Parallel Computing, vol. 3, no. 3, pp. 18:1–18:34, Dec. 2016. [Online]. Available: http://doi.acm.org/10.1145/3015144