2018 IEEE International Parallel and Distributed Processing Symposium

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,rousekm}@wfu.edu

Nicholas Knight
Department of Mathematics
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. One of the most popular decompositions, known as CANDECOMP/PARAFAC or CP, is a generalization of the matrix singular value decomposition (or principle component analysis) and forms a low-rank approximation of tensor data. Such decompositions 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 Sec. 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

- derive communication lower bounds for MTTKRP in sequential and parallel models: see Sec. IV;
- propose communication-optimal sequential and parallel MTTKRP algorithms: see Sec. V;
- demonstrate that the conventional MTTKRP approach based on matrix multiplication communicates more than performing MTTKRP as a multiway tensor contraction: see Sec. VI.

We discuss related work in Sec. III and conclude in Sec. 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 $\mathcal{X}$ 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

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

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

Computing a CP decomposition involves solving a nonlinear optimization problem to minimize the approximation error, typically measured in the ℓ_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*.

1530-2075/18/$31.00 ©2018 IEEE
DOI 10.1109/IPDPS.2018.00065

B. MTTKRP

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

Definition 2.1: An *MTTKRP algorithm* maps

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

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

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

where summation is over all $\mathbf{i} \in [I_1] \times \cdots \times [I_N]$ with n-th entry i_n, and we require that the $N-1$ multiplies for each $(\mathbf{i}, r)$ are evaluated atomically as an *N-ary multiply*.

Atomic evaluation of an N-ary multiply means that all N inputs are present in memory when the single output value is computed. This assumption is necessary for the proofs of our communication lower bounds. However, it is natural for an algorithm to break the assumption in order to reduce arithmetic, as partial products are shared across multiple N-ary multiplies. For example, a popular choice is to precompute the (explicit) Khatri-Rao product and then apply the matricized tensor in a single matrix multiplication (see Sec. III-B). As we discuss in Sec. VI, this approach does reduce arithmetic but usually increases the communication cost. Our ongoing work includes extending our communication cost analysis to address such atomicity-violating optimizations.

If the inputs or the operations satisfy additional properties, equivalent formulations of the right-hand side of Eq. (2) may yield more efficient algorithms, which are excluded from Def. 2.1. As a practical example, if some value '0' is an identity element for addition and an absorbing element for multiplication, then we can avoid arithmetic by, e.g., skipping the summation indices $\mathbf{i}$ such that $\mathcal{X}(\mathbf{i}) = 0$. Our lower bounds readily extend to address such 'sparse' algorithms by simply replacing the number of tensor entries I with the number that are nonzero; however, our algorithms may not attain these new, smaller bounds.

Lastly, note that the extreme case $N = 2$ is just matrix multiplication, e.g., $\mathbf{B}^{(1)} = \mathcal{X} \cdot \mathbf{A}^{(2)}$.

C. Computation Models

a) Sequential Model: Our sequential model 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 [4] or the I/O complexity model [5].

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 [6], or α-β-γ model [4]. In this work, we focus on the amount of data communicated (bandwidth cost) and ignore the number of messages communicated (latency cost). As our proofs do not exploit the model's restriction of half duplex communication, our parallel lower bounds also apply to the BSP model of computation [7]. Additionally all our parallel algorithms are valid BSP algorithms. As we ignore latency cost, and all communication is performed in collectives which have the same cost in the BSP model, our parallel upper bounds also apply to BSP computations.

III. Related Work

A. Communication Lower Bounds

The pioneering work of Hong and Kung [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.* [8] extended Hong and Kung's results for matrix multiplication to the parallel case using a segmentation argument that we will follow. Ballard *et al.* [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 [9] 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.* [10] to determine the ranges where perfect strong scaling can be achieved. Demmel *et al.* [11] 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.* [12] extended the generality of 3NL computations to prove lower bounds for more arbitrary loop nests: their approach applies 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 [13]. Note that this approach violates the assumption in Def. 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 [14], [15]. 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 [13], [16]–[18] in the context of the CP-ALS algorithm. In particular, Smith and Karypis [18] describe a "medium-grained" parallelization scheme that is designed for sparse tensors but can be applied to dense tensors. Indeed, Liavas *et al.* [19] apply the preceding approach to dense 3-way tensors in computing CP decompositions with non-negativity constraints, using 1D, 2D, and 3D processor grids depending on the tensor dimensions. Aggour and Yenner [20] also parallelize MTTKRP for dense tensors, using a scheme that parallelizes over only the largest dimension (using only 1D processor grids) of a 3-way tensor.

IV. Lower Bounds

In Sec. IV we derive lower bounds on the amount of data communicated by sequential and parallel MTTKRP algorithms. The key, formalized in Lem. 4.1 (Sec. IV-A), is that in any MTTKRP algorithm the size of any subset of operations is bounded in terms of the numbers of associated operands. These upper bounds on data reuse yield lower bounds on communication, in both sequential and parallel models, via a counting argument: see Thm. 4.1 and Cor. 4.1 (Sec. IV-B). These lower bounds are called *memory-dependent*, since they explicitly depend on the fast/local memory size M. A different counting argument and stronger hypotheses yield *memory-independent* lower bounds — see Thms. 4.2 and 4.3 (Sec. IV-C).

Our arguments build on previous communication lower bounds for matrix computations [4], [8], which used the Loomis-Whitney inequality. These lower bounds do not directly apply to MTTKRP computations, as formalized in Def. 2.1. However, a generalization to a larger class of nested-loop programs [12], [21], leveraging the more general class of Hölder-Brascamp-Lieb inequalities [22], do apply to MTTKRP. These previous results assumed the number of nested loops is a fixed constant, whereas in MTTKRP this number varies with the order N of the tensor. A key contribution of the present work is that we extend the previous analyses to allow the number of nested loops to vary.

A. Preliminary Lemmas

In this section we state four lemmas that will be useful in our main results. Lem. 4.1, a generalization of the Loomis-Whitney inequality [23], will be used to derive an upper bound on possible data reuse within an MTTKRP computation. Lem. 4.2 provides the solution to a particular linear program that appears in our lower bound proofs. Lems. 4.3 and 4.4 give solutions to nonlinear optimization problems that appear in later proofs.

The following result appears in a more general form in [22, Proposition 7.1]; a simpler proof for our special case is given in [12, Theorem 6.6].

Lemma 4.1: Consider any positive integers d and m and any m projections $\phi_j \colon \mathbb{Z}^d \to \mathbb{Z}^{d_j}$ ($d_j \leq d$), each of which extracts d_j coordinates $S_j \subseteq [d]$ and forgets the $d - d_j$ others. Define $\mathcal{P} = \left\{\mathbf{s} \in [0,1]^m : \mathbf{\Delta} \cdot \mathbf{s} \geq \mathbf{1}\right\}$, where the $d \times m$ matrix $\mathbf{\Delta}$ has entries $\mathbf{\Delta}(i,j) = 1$ if $i \in S_j$ and $\mathbf{\Delta}(i,j) = 0$ otherwise. If $\mathbf{s} \in \mathcal{P}$, 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

$$\min \mathbf{1}^T \mathbf{s} \quad \text{subject to} \quad \mathbf{\Delta} \cdot \mathbf{s} \geq \mathbf{1} \text{ and } \mathbf{s} \geq 0, \tag{3}$$

where

$$\mathbf{\Delta} = \begin{pmatrix} \mathbf{I}_{N \times N} & \mathbf{1}_{N \times 1} \\ \mathbf{1}_{1 \times N} & 0 \end{pmatrix},$$

is $\mathbf{s}^* = (1/N, \ldots, 1/N, 1 - 1/N)^T$ with $\mathbf{1}^T \mathbf{s}^* = 2 - 1/N$.

Proof: The dual linear program is

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

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

The following two lemmas can be proved using the method of Lagrange multipliers [24].

Lemma 4.3: Given $\mathbf{s} > \mathbf{0}$, the optimization problem

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

yields the maximum value $c^{\sum_i s_i} \prod_{j \in [m]} \left(\frac{s_j}{\sum_i s_i}\right)^{s_j}$.

Lemma 4.4: For any $\mathbf{s} \geq \mathbf{0}$, the optimization problem

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

yields the minimum value $\left(\frac{c}{\prod_i s_i^{s_i}}\right)^{1/\sum_i s_i} \sum_{i \in [m]} s_i$.

B. Memory-Dependent Lower Bounds

We first prove Thm. 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], [8]. However, to address MTTKRP, it uses a Hölder-Brascamp-Lieb-type inquality (Lem. 4.1) as has been done for more general computations [12]. It also borrows another technique involving Lem. 4.3 that has been used

to tighten the constant of the matrix multiplication bound [9], though the technique improves our bound by more than a constant. Thm. 4.1 implies Cor. 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

$$\frac{1}{3^{2-1/N}} \frac{NIR}{M^{1-1/N}} - M \tag{4}$$

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 $\mathcal{X}$, $\mathbf{B}^{(n)}$, or $\mathbf{A}^{(k)}$ that can be accessed during a segment, then use Lem. 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 $\mathbf{B}^{(n)}$, the factor matrix that is being computed. We consider an element of $\mathbf{B}^{(n)}$ *live* during the segment if it accumulates the result of one or more N-ary multiplies during that segment. Any element of $\mathbf{B}^{(n)}$ 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 $\mathbf{B}^{(n)}$ that remain in fast memory. Let S be the number of live elements of $\mathbf{B}^{(n)}$ that were stored during the segment. Now, consider input elements of $\mathcal{X}$ and $\mathbf{A}^{(k)}$ 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 the 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 $\mathcal{I} = [I_1] \times \cdots \times [I_N] \times [R]$ 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 Fig. 1 for an example set F and its projections.

To use Lem. 4.1 we first define the linear constraint matrix Δ. 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

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

By Lem. 4.1, for any $\mathbf{s} \in \mathcal{P}$, $|F| \leq \prod_{j \in [m]} |\phi_j(F)|^{s_j}$.

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

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

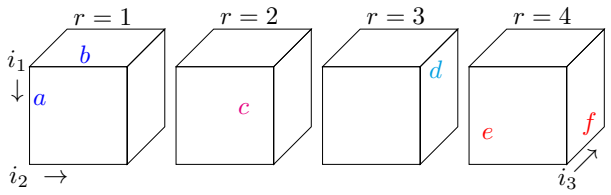
In order to obtain the tightest lower bound possible, we wish to choose the $\mathbf{s} \in \mathcal{P}$ that minimizes the left hand side of the preceding inequality. Short of that, we can choose to minimize only the first factor $(3M)^{\sum_j s_j}$, which corresponds to solving the linear program Eq.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

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

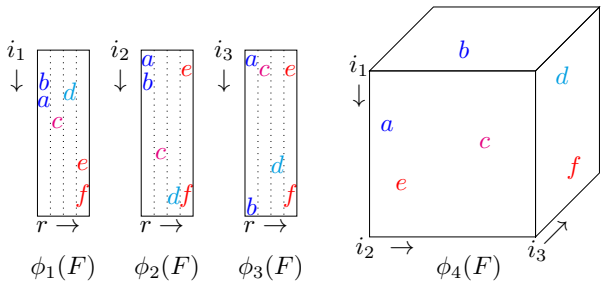
$$= \left(\frac{1}{2 - 1/N} \right)^{2-1/N} (1 - 1/N)^{1-1/N} \prod_{j \in [N]} (1/N)^{1/N} \leq 1/N.$$

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 $|\mathcal{I}| = IR$ there are at least $\lfloor IR/((3M)^{2-1/N}/N) \rfloor$ complete segments. Each segment loads/stores M words, thus



(a) Example subset F of 4-way iteration space. The subset F consists of the six coordinates a (5,1,1,1), b (3,3,15,1), c (7,10,2,2), d (4,14,11,3), e (11,2,2,4), and f (14,14,14,4), which are color coded by their last index.



(b) Projections of F onto data arrays (2-way factor matrices and 3-way tensor). For example, the set $\phi_2(F)$ consists of the six coordinates a (1,1), b (3,1), c (10,2), d (14,3), e (2,4), and f (14,4).

Fig. 1. Example subset of computation and the data required to perform it, for $N = 3$, $I_1 = I_2 = I_3 = 15$, and $R = 4$. Fig. 1a shows the iteration space and specifies six coordinates in the subset, where the coordinates correspond to N-ary multiplies. Fig. 1b show the elements of the arrays that are involved in the computation, which are determined by projections of the coordinates.

there are at least $M \cdot \left\lfloor NIR/(3M)^{2-1/N} \right\rfloor$ 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 $|\mathcal{I}|/P = IR/P$ loop iterations, we can apply Thm. 4.1 to the computation performed by that processor. ■

Both sequential and parallel lower bounds indicate a tradeoff between the memory size M and communication. We demonstrate in Sec. V-A a sequential algorithm that attains the lower bound of Thm. 4.1 within a constant factor, navigating the tradeoff by appropriately choosing a block-size parameter. It remains open whether there exists a parallel algorithm that navigates the tradeoff in Cor. 4.1 in a similar manner.

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 $I + \sum_{k\in[N]} I_k R - 2M$ 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 [10], [11], but again we combine a technique used in the context of matrix multiplication [9] (involving Lem. 4.3) to tighten the bounds. Thms. 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 ($I_k = I^{1/N}$ for all k) and state Cor. 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

$$2\left(\frac{NIR}{P}\right)^{\frac{N}{2N-1}} - \gamma\frac{I}{P} - \delta \sum_{k\in[N]} \frac{I_k R}{P} \tag{5}$$

sends and receives.

Proof: We follow the argument given by Ballard *et al.* [10, Lemma 2.3]. Some processor p must evaluate at least $|\mathcal{I}|/P = IR/P$ 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 $\sum_{j\in[N+1]} |\phi_j(F)| - \gamma I/P - \delta \sum_{k\in[N]} I_k R/P$, 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 Lem. 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 $\mathbf{s}$ in $\mathcal{P}$. Using $\mathbf{s}^* = (1/N, \ldots, 1/N, 1-1/N)$ as before, and substituting $|\phi_j(F)|$ for x_j and IR/P as the constant c, Lem. 4.4 gives

$$\sum_{j\in[N+1]} |\phi_j(F)| \geq (2-1/N)\left(\frac{IR/P}{\prod s_j^{s_j}}\right)^{\frac{N}{2N-1}} \geq 2\left(\frac{NIR}{P}\right)^{\frac{N}{2N-1}}.$$

■

Theorem 4.3: 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

$$\min\left(\sqrt{\frac{2}{3\gamma}} NR\left(\frac{I}{P}\right)^{1/N} - \delta \sum_{j\in[N]} \frac{I_j R}{P},\ \frac{\gamma I}{2P}\right) \tag{6}$$

sends and receives.

Proof: We follow the argument given by Demmel *et al.* [11, Section II.B.2]. As before, F is the set of loop iterations evaluated by a processor that computes at least IR/P N-ary multiplies. By Lem. 4.1 with $\mathbf{s}^* = (1/N, \ldots, 1/N, 1-1/N)^T$, we have

$$\frac{IR}{P} \leq |\phi_{N+1}(F)|^{\frac{N-1}{N}} \prod_{j\in[N]} |\phi_j(F)|^{1/N}. \tag{7}$$

We consider two cases based on $|\phi_{N+1}(F)|$, the number of tensor entries accessed by the processor. Suppose that $|\phi_{N+1}(F)| \geq \frac{3\gamma I}{2P}$. By our assumption of load balanced data distribution, the processor must read at least $\frac{\gamma I}{2P}$ elements of $\mathcal{X}$ to perform its computations. Now consider the case when $|\phi_{N+1}(F)| < \frac{3\gamma I}{2P}$. Replacing $|\phi_{N+1}(F)|$ with $\frac{3\gamma I}{2P}$ in the right hand side of Eq. (7) and rearranging, we have $\prod_{j\in[N]} |\phi_j(F)| \geq (2/(3\gamma))^{N-1} \frac{IR^N}{P}$. By Lem. 4.4, we know that $\sum_{j\in[N]} |\phi_j(F)|$ is minimized subject to this constraint on the product when $|\phi_j(F)| = (2/(3\gamma))^{\frac{N-1}{N}} (I/P)^{1/N} R$. Given that the factor matrices are load balanced up to a factor of δ, we see that some processor performs at least $\sum_{j\in[N]} |\phi_j(F)| - \delta \sum_{j\in[N]} \frac{I_j R}{P} \geq N\left(\frac{2}{3\gamma}\right)^{\frac{N-1}{N}} \left(\frac{I}{P}\right)^{1/N} R - \delta \sum_{j\in[N]} \frac{I_j R}{P}$ sends and receives.

Because the number of tensor entries the processor must access may be bigger or smaller than $\frac{3\gamma I}{2P}$, the lower bound is the minimum of the two cases. ■

Corollary 4.2: Any parallel MTTKRP algorithm involving a tensor with $I_k = I^{1/N}$ for all k and that starts with one copy of the inputs evenly distributed across processors and ends with one copy of the output evenly distributed across processors involves at least

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

sends and receives.

Proof: The proof involves simplifying and combining the results of Thms. 4.2 and 4.3 under the additional assumptions, see [24] for details. ■

Algorithm 1 Sequential Blocked Algorithm

function $\mathbf{B}^{(n)} = $ SEQ-BLOCKED-MTTKRP($\mathcal{X}$, $\{\mathbf{A}^{(k)}\}$, n, b)
  for $j_1 \leftarrow 1$ **to** I_1 **step** b **do**
    $\ddots$
      for $j_N \leftarrow 1$ **to** I_N **step** b **do**
        $J_k \leftarrow \min(I_k, j_k + b - 1)$ $(k \in [N])$
        load block $\mathcal{X}(j_1{:}J_1, \ldots, j_N{:}J_N)$
        for $r \leftarrow 1$ **to** R **do**
          load vectors $\mathbf{A}^{(k)}(j_k{:}J_k, r)$ $(k \in [N] \setminus \{n\})$
          load vector $\mathbf{B}^{(n)}(j_n{:}J_n, r)$
          for $i_1 \leftarrow j_1$ **to** J_1 **do**
            $\ddots$
              for $i_N \leftarrow j_N$ **to** J_N **do**
                $\mathbf{B}^{(n)}(i_n, r) \leftarrow \mathbf{B}^{(n)}(i_n, r) + \mathcal{X}(i_1, \ldots, i_N) \cdot \prod_{k \in [N] \setminus \{n\}} \mathbf{A}^{(k)}(i_k, r)$
              end for
            $\ddots$
          end for
          store vector $\mathbf{B}^{(n)}(j_n{:}J_n, r)$
        end for
      end for
    $\ddots$
  end for
end function

V. ALGORITHMS

A. Sequential Blocked Algorithm

Alg. 1 and Fig. 2 illustrate a sequential blocked MTTKRP algorithm. We control the blocking with the block size b. The code is correct for any positive integer b satisfying

$$b^N + Nb \leq M, \tag{8}$$

whence the communication cost is bounded above by

$$I + \left\lceil \frac{I_1}{b} \right\rceil \cdots \left\lceil \frac{I_N}{b} \right\rceil \cdot R(N+1)b. \tag{9}$$

In Sec. VI-A, within the proof of Thm. 6.1, we will weaken and simplify Eq. (9) for easier comparison with the lower bounds of Thm. 4.1 and Fact 4.1. 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 $b \approx M^{1/N}$ gives an upper bound of the form

$$O\left(I + \frac{NIR}{M^{1-1/N}}\right). \tag{10}$$

To see how Eq. (10) might be obtained from Eq. (9), substitute $b = (M/2)^{1/N}$, supposing b is a positive integer that satisfies Eq. (8) and divides $I_1, \ldots, I_N$.

B. Parallel Stationary Tensor Algorithm

We present two parallel algorithms, Algs. 2 and 3, the first of which is a special case of the second. Here in Sec. V-B we present the special case of Alg. 2 separately because its notation is simpler and we expect it to apply more frequently in typical applications, where NR is small relative to I/P. See also Fig. 3 for an illustration of Alg. 2. The general algorithm, Alg. 3, is presented in Sec. V-C. We note that Alg. 2 is essentially the same as the medium-grained algorithm applied to dense tensors [18], [19], though the communication pattern simplifies in the dense case.

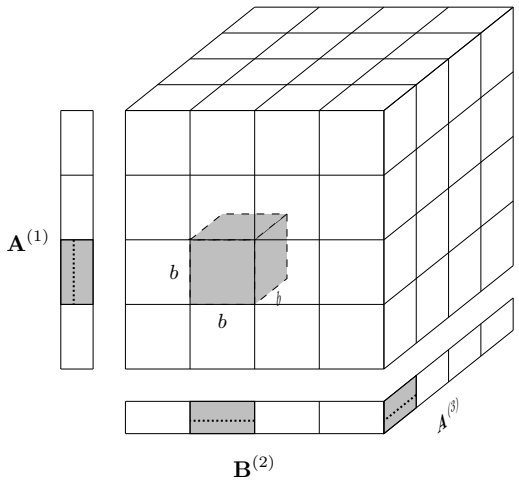


Fig. 2. Sequential Blocked Algorithm for $N = 3$ and $n = 2$: subtensor $\mathcal{X}(j_1{:}J_1, j_2{:}J_2, j_3{:}J_3)$ is highlighted, and subcolumns $\mathbf{A}^{(1)}(j_1{:}J_1, r)$, $\mathbf{B}^{(2)}(j_2{:}J_2, r)$, $\mathbf{A}^{(3)}(j_3{:}J_3, r)$ are shown with dotted lines.

1) Data Distribution: For an N-way tensor, we organize processors into an N-way logical processor grid. We factor $P = P_1 P_2 \cdots P_N$ and identify each processor by an N-tuple $\mathbf{p} = (p_1, \ldots, p_N) \in [P_1] \times \cdots \times [P_N]$. We partition each tensor dimension $k \in [N]$ into P_k parts: $[I_k] = \{S_{p_k}^{(k)}\}_{p_k \in [P_k]}$. Each processor $\mathbf{p}$ initially stores the subtensor $\mathcal{X}_{\mathbf{p}} = \mathcal{X}(S_{p_1}^{(1)}, \ldots, S_{p_N}^{(N)})$, and, for each $k \in [N] \setminus \{n\}$, a part $\mathbf{A}_{\mathbf{p}}^{(k)}$ in a partition of $\mathbf{A}_{p_k}^{(k)} = \mathbf{A}^{(k)}(S_{p_k}^{(k)}, :)$, across processors $\mathbf{p}'$ with $p'_k = p_k$. During execution, processor $\mathbf{p}$ also stores the submatrices $\mathbf{A}_{p_k}^{(k)}$, $k \in [N] \backslash \{n\}$ and a matrix $\mathbf{C}_{p_n}$ the same size as (and used in the summation of) $\mathbf{B}_{p_n}^{(n)}$. After execution, the processor stores a part $\mathbf{B}_{\mathbf{p}}^{(n)}$ in a partition of $\mathbf{B}_{p_n}^{(n)} = \mathbf{B}^{(n)}(S_{p_n}^{(n)}, :)$, across processors $\mathbf{p}'$ with $p'_n = p_n$. In words, each mode's factor matrix is distributed block-rowwise across the processor hyperslices of that mode, and each row block is then partitioned arbitrarily across the processors in its hyperslice. During execution, these block rows are replicated within hyperslices.

2) Algorithm: The pseudocode is given in Alg. 2, and Fig. 3 provides an illustration of the major steps. 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. 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 communication is performed

Algorithm 2 Parallel Stationary Tensor MTTKRP Algorithm

1: **function** $\mathbf{B}^{(n)}_{\mathbf{p}} = $ PAR-STAT-MTTKRP($\mathcal{X}_{\mathbf{p}}, \{\mathbf{A}^{(k)}_{\mathbf{p}}\}, n$)
2: $\mathbf{p} = (p_1, \ldots, p_N)$ is my processor id
3: **for each** $k \in [N] \setminus \{n\}$ **do**
4: $\mathbf{A}^{(k)}_{p_k} = $ All-Gather($\mathbf{A}^{(k)}_{\mathbf{p}}, (:, \ldots, :, p_k, :, \ldots, :)$)
5: **end for**
6: $\mathbf{C}_{p_n} = $ Local-MTTKRP($\mathcal{X}_{\mathbf{p}}, \{\mathbf{A}^{(k)}_{p_k}\}, n$)
7: $\mathbf{B}^{(n)}_{\mathbf{p}} = $ Reduce-Scatter($\mathbf{C}_{p_n}, (:, \ldots, :, p_n, :, \ldots, :)$)
8: **end function**

Algorithm 3 Parallel General MTTKRP Algorithm

1: **function** $\mathbf{B}^{(n)}_{\mathbf{p}} = $ PAR-GEN-MTTKRP($\mathcal{X}_{\mathbf{p}}, \{\mathbf{A}^{(k)}_{\mathbf{p}}\}, n$)
2: $\mathbf{p} = (p_0, p_1, \ldots, p_N)$ is my processor id
3: $\mathcal{X}_{p_1,\ldots,p_N} = $ All-Gather($\mathcal{X}_{\mathbf{p}}$,$(:, p_1, \ldots, p_N)$)
4: **for each** $k \in [N] \setminus \{n\}$ **do**
5: $\mathbf{A}^{(k)}_{p_k,p_0} = $ All-Gather($\mathbf{A}^{(k)}_{\mathbf{p}}$,$(p_0, :, \ldots, :, p_k, :, \ldots, :)$)
6: **end for**
7: $\mathbf{C}_{p_n,p_0} = $ Local-MTTKRP($\mathcal{X}_{p_1,\ldots,p_N}$, $\{\mathbf{A}^{(k)}_{p_k,p_0}\}$, n)
8: $\mathbf{B}^{(n)}_{\mathbf{p}} = $ Reduce-Scatter($\mathbf{C}_{p_n,p_0}$,$(p_0, :, \ldots, :, p_n, :, \ldots, :)$)
9: **end function**

across processor hyperslices using collective communication operations All-Gather and Reduce-Scatter.

3) Analysis: We defer detailed analysis of Alg. 2 to that of the more general Alg. 3 in Sec. V-C3. Setting $P_0 = 1$ in Alg. 3 yields Alg. 2. Analysis of Alg. 2 has already appeared for cubical tensors [18] and 3-way tensors [19].

Assuming we can choose a processor grid such that $P_k \approx I_k/(I/P)^{1/N}$ and divides I_k evenly, we choose the data distribution such that $|S^{(k)}_{p_k}| = I_k/P_k$ for $k \in [N]$, which simplifies these upper bounds. The communication cost bound is $O\left(NR(I/P)^{1/N}\right)$, the arithmetic cost bound is $O(NIR/P)$ (which can be reduced by a factor of $O(N)$), and the (per-processor) storage cost bound is $O\left(I/P + NR(I/P)^{1/N}\right)$.

The temporary storage (the second term in the bound) could be reduced by a factor of at most R by using another layer of blocking over the columns of the matrices. While this would not affect the amount of communication, it would increase the number of communication collectives by the same factor.

C. Parallel General Algorithm

This section studies Alg. 3, a generalization of the stationary tensor algorithm, Alg. 2, described in Sec. V-B. Alg. 3 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 Alg. 2 parallelizes over just the N tensor dimensions. Roughly speaking, Alg. 3 is more efficient than Alg. 2 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 $\mathbf{p} = (p_0, p_1, \ldots, p_N) \in [P_0] \times [P_1] \times \cdots \times [P_N]$. As before, we partition each tensor dimension $k \in [N]$ into P_k parts, $[I_k] = \{S^{(k)}_{p_k}\}_{p_k \in [P_k]}$. Additionally we now partition the matrix column dimension into P_0 parts, $[R] = \{T_{p_0}\}_{p_0 \in [P_0]}$. Each processor $\mathbf{p}$ initially (before execution) stores a part $\mathcal{X}_{\mathbf{p}}$ in a partition of $\mathcal{X}_{p_1,\ldots,p_N} = \mathcal{X}(S^{(1)}_{p_1}, \ldots, S^{(N)}_{p_N})$, across processors $\mathbf{p}'$ with $p'_k = p_k$ ($k \in [N]$), and for each $k \in [N] \setminus \{n\}$, a part $\mathbf{A}^{(k)}_{\mathbf{p}}$ in a partition of $\mathbf{A}^{(k)}_{p_k,p_0} = \mathbf{A}^{(k)}(S^{(k)}_{p_k}, T_{p_0})$, across processors $\mathbf{p}'$ with $p'_0 = p_0$ and $p'_k = p_k$. During execution it also stores the subtensor $\mathcal{X}_{p_1,\ldots,p_N}$, the submatrices $\mathbf{A}^{(k)}_{p_k,p_0}$, $k \in [N] \setminus \{n\}$, and $\mathbf{B}^{(n)}_{p_n,p_0}$, and a matrix $\mathbf{C}_{p_n,p_0}$ the same size as (and used in the summation of) $\mathbf{B}^{(n)}_{p_n,p_0}$. After execution, the processor stores a part $\mathbf{B}^{(n)}_{\mathbf{p}}$ in a partition of $\mathbf{B}^{(n)}_{p_n,p_0} = \mathbf{B}^{(n)}(S^{(n)}_{p_n}, T_{p_0})$, across processors $\mathbf{p}'$ with $p'_0 = p_0$ and $p'_n = p_n$. Let us clarify a notational detail: while $\mathcal{X}_{p_1,\ldots,p_N}$, $\mathbf{A}^{(k)}_{p_k}$, $\mathbf{B}^{(n)}_{p_n}$ are tensors/matrices, the (sub)sets of tensor/matrix entries $\mathcal{X}_{\mathbf{p}}$, $\mathbf{A}^{(k)}_{\mathbf{p}}$, $\mathbf{B}^{(n)}_{\mathbf{p}}$ need not be (sub)tensors/matrices.

2) Algorithm: As mentioned in Sec. V-B, the more general Alg. 3 parallelizes over all $N+1$ dimensions of the iteration space: unlike the stationary algorithm Alg. 2, entries of the tensor $\mathcal{X}$ are now communicated among processors. One can think of Alg. 3 as logically dividing the output factor matrix $\mathbf{B}^{(n)}$ into P_0 block-columns, each assigned to a separate subset of P/P_0 processors, and running Alg. 2 on each subset of processors, where each subset owns a copy of the tensor.

3) Analysis: We analyze the communication cost first. Communication occurs only in the All-Gather and Reduce-Scatter collectives in Lines 3, 5 and 8. Each processor $\mathbf{p}$ is involved in one All-Gather involving the tensor (Line 3), $N-1$ All-Gathers involving factor matrices (Line 5, $k \in [N] \setminus \{n\}$) and one Reduce-Scatter (Line 8). Over all processors, Line 3 specifies P/P_0 simultaneous All-Gathers, Line 5 ($k \in [N] \setminus \{n\}$) specifies P_k simultaneous All-Gathers, and Line 8 specifies P_n simultaneous Reduce-Scatters.

In this analysis, we assume *bucket algorithms* are used for the collectives. 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 [25]. For Reduce-Scatter, there is also an arithmetic cost of at most $(q-1)w$ operations.

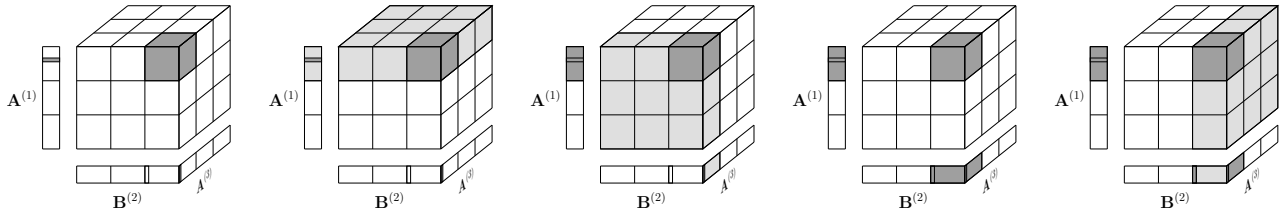
In these cases, we have $q_0 = P_0$ for Line 3 and $q_k = P/(P_0 P_k)$ for Line 5 ($k \in [N] \setminus \{n\}$) and Line 8 ($k = n$). The largest local sizes depend on the data distributions specified in Sec. V-C1: $w_0 = \max_{\mathbf{p}} \operatorname{nnz}(\mathcal{X}_{\mathbf{p}})$ and

$$w_k = \begin{cases} \max_{\mathbf{p}} \operatorname{nnz}(\mathbf{A}^{(k)}_{\mathbf{p}}) & k \in [N] \setminus \{n\} \\ \max_{\mathbf{p}} \operatorname{nnz}(\mathbf{B}^{(n)}_{\mathbf{p}}) & k = n. \end{cases}$$

The overall communication cost is then bounded above by

$$(P_0 - 1) \cdot \max_{\mathbf{p}} \operatorname{nnz}(\mathcal{X}_{\mathbf{p}}) + \sum_{k \in [N]} \left(\frac{P}{P_0 P_k} - 1 \right) \cdot \begin{cases} \max_{\mathbf{p}} \operatorname{nnz}(\mathbf{A}^{(k)}_{\mathbf{p}}) & k \neq n \\ \max_{\mathbf{p}} \operatorname{nnz}(\mathbf{B}^{(n)}_{\mathbf{p}}) & k = n. \end{cases} \quad (11)$$

The arithmetic cost is bounded above in terms of the



(a) Start with one subtensor and subset of rows of each input matrix.

(b) All-Gather rows from input matrix $\mathbf{A}^{(1)}$ (Line 4).

(c) All-Gather rows from input matrix $\mathbf{A}^{(3)}$ (Line 4).

(d) Compute local contribution to rows of output matrix $\mathbf{B}^{(2)}$ (Line 6).

(e) Reduce-Scatter to compute/distribute rows of output matrix $\mathbf{B}^{(2)}$ (Line 7).

Fig. 3. Parallel Stationary Tensor Algorithm data distribution, communication, and computation across steps for $N = 3$ and $n = 2$. Highlighted areas correspond to processor $(1, 3, 1)$ and its subcommunicators.

costliest local MTTKRP (Line 7) and the costliest Reduce-Scatter (Line 8): the number of operations is at most

$$N \max_{\mathbf{p}} \left(|T_{p_0}| \prod_{k \in [N]} |S^{(k)}_{p_k}| \right) + \left(\frac{P}{P_0 P_n} - 1 \right) \max_{\mathbf{p}} \operatorname{nnz}(\mathbf{B}^{(n)}_{\mathbf{p}}). \quad (12)$$

The per-processor storage cost is bounded above by

$$\max_{\mathbf{p}} \left(\prod_{k \in [N]} |S^{(k)}_{p_k}| + \sum_{k \in [N]} |S^{(k)}_{p_k}| \cdot |T_{p_0}| \right). \quad (13)$$

Let us simplify these upper bounds, assuming that we can choose a processor grid with dimensions

$$P_0 \approx \frac{(NR)^{\frac{N}{2N-1}}}{(I/P)^{\frac{N-1}{2N-1}}} \text{ and } P_k \approx \frac{I_k}{(IP_0/P)^{\frac{1}{N}}} \quad (k \in [N]),$$

and that we can choose the data distribution such that

$$|S^{(k)}_{p_k}| = I_k/P_k, \quad |T_{p_0}| = R/P_0, \quad \operatorname{nnz}(\mathcal{X}_{\mathbf{p}}) = I/P,$$
$$\operatorname{nnz}(\mathbf{A}^{(k)}_{\mathbf{p}}) = I_k R/P, \quad \text{and} \quad \operatorname{nnz}(\mathbf{B}^{(n)}_{\mathbf{p}}) = I_n R/P,$$

where everything divides evenly. The communication cost bound Eq. (11) and the storage cost bound Eq. (13) are $O\left(NR\,(I/P)^{1/N} + (NIR/P)^{N/(2N-1)}\right)$, and the arithmetic cost bound Eq. (12) is $O(NIR/P)$. We weaken these assumptions on the processor grid and make them more explicit in the proof of Thm. 6.2.

We note that to save arithmetic, the algorithm could break the atomicity of the N-ary multiplies without changing the communication costs of the algorithm. Each processor can precompute the explicit local Khatri-Rao product and perform a local matrix multiplication, reducing the arithmetic cost bound from $O(NIR/P)$ to $O(IR/P)$ (these costs assume a reasonably load-balanced distribution).

VI. Discussion

A. Sequential Case

We would like to compare the upper bound,

$$W^{\text{seq}}_{\text{ub}} = I + (N+1) \left(\prod_{k \in [N]} \left\lceil \frac{I_k}{b} \right\rceil \right) bR,$$

valid for any $b \in \{1, 2, \ldots\}$ satisfying $M \geq b^N + Nb$, with the lower bounds $W^{\text{seq}}_{\text{lb1}} = \frac{NIR}{3(3M)^{1-1/N}} - M$ (Thm. 4.1) and $W^{\text{seq}}_{\text{lb2}} = I + \sum_{k \in [N]} I_k R - 2M$ (Fact 4.1).

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 Alg. 1 is communication optimal to within a constant factor.

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

$$M \geq \left(\frac{N\alpha^{1/N}}{1-\alpha} \right)^{\frac{N}{N-1}} \qquad \alpha < 1 \quad (14)$$

$$M \geq \left(\frac{1}{\alpha^{1/N} - \beta^{1/(N-1)}} \right)^N \qquad \beta < \alpha^{1-1/N} \quad (15)$$

$$M \leq \left(\frac{\left(\frac{N}{N+1}\gamma\right)^{1/N} - 1}{\alpha^{1/N}} \min_{k \in [N]} I_k \right)^N \qquad \gamma > 1 + \frac{1}{N} \quad (16)$$

$$M \leq \frac{1}{2} \left((1-\delta) I + \sum_{k \in [N]} I_k R \right) \qquad \delta < 1 + \sum_{k \in [N]} \frac{I_k}{I} R \quad (17)$$

$$M \leq \left(\left(\frac{1}{3^{2-1/N}} - \epsilon \right) NIR \right)^{\frac{N}{2N-1}} \qquad \epsilon < \frac{1}{3^{2-1/N}}. \quad (18)$$

For Alg. 1, we choose block size $b = \lfloor (\alpha M)^{1/N} \rfloor$. It then follows from Eqs. (14) to (16) that $W^{\text{seq}}_{\text{ub}} \leq \frac{\gamma}{\beta} \left(I + \frac{NIR}{M^{1-1/N}} \right)$. It follows from Eqs. (17) and (18) that $\max(W^{\text{seq}}_{\text{lb1}}, W^{\text{seq}}_{\text{lb2}}) \geq \frac{\min(\delta,\epsilon)}{2} \left(I + \frac{NIR}{M^{1-1/N}} \right)$, which matches the upper bound to within a constant factor. For a more detailed argument, see [24]. ∎

To illustrate the hypotheses Eqs. (14) to (18) of Thm. 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 below by 10^4 (due to Eqs. (14) and (15)), and above by the minimum of $I/1000$

(due to Eqs. (16) and (17)) and $\sqrt{NIR}/10$ (due to Eq. (18)).

We also compare the communication cost of Alg. 1, $O(I + NIR/M^{1-1/N})$, with the MTTKRP via matrix multiplication approach. We assume a communication-optimal 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 Alg. 1 never exceeds that of MTTKRP via matrix multiplication.

If the communication cost is dominated by accessing the tensor (i.e., $R = O(M^{1/2})$), then the approaches perform the same amount of communication and Alg. 1 performs a factor of $N/2$ more computation. If the communication cost is dominated by repeatedly accessing the factor matrices (i.e., $NR = \Omega(M^{1-1/N})$), then Alg. 1 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 it can also use existing software for matrix multiplication.

B. Parallel Case

The communication upper bound for Alg. 3, $W^{\text{par}}_{\text{ub}}$, is valid for any factorization $P = P_0 P_1 \cdots P_N$ and data distribution specified in Sec. V-C1. We wish to compare this upper bound with the lower bound from Thm. 4.2, $W^{\text{par}}_{\text{lb1}}$, and the lower bound from Thm. 4.3, $W^{\text{par}}_{\text{lb2}}$.

Theorem 6.2: Suppose the number of processors P is sufficiently large and factorable, and suppose that the tensor dimensions and rank are sufficiently large with respect to P. Then Alg. 3 is communication optimal to within a constant factor.

Proof: To instantiate $W^{\text{par}}_{\text{ub}}$, 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 Sec. V-C1, we can define a data distribution where, for each processor $\mathbf{p}$,

$$\begin{aligned} \text{nnz}(\mathcal{X}_{\mathbf{p}}) &\leq \Big\lceil \prod_k \lceil I_k/P_k \rceil / P_0 \Big\rceil, \\ \text{nnz}(\mathbf{A}^{(k)}_{\mathbf{p}}) &\leq \big\lceil \lceil I_k/P_k \rceil \lceil R/P_0 \rceil / (P/(P_k P_0)) \big\rceil, \\ \text{nnz}(\mathbf{B}^{(n)}_{\mathbf{p}}) &\leq \big\lceil \lceil I_n/P_n \rceil \lceil R/P_0 \rceil / (P/(P_n P_0)) \big\rceil. \end{aligned} \tag{19}$$

To instantiate $W^{\text{par}}_{\text{lb1}}, W^{\text{par}}_{\text{lb2}}$, 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 Eq. (19) 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 $\gamma > \alpha$, $\delta > \alpha^{1/N}\beta$, and, for all $k \in [N]$,

$$\begin{aligned} P_k &\leq (\alpha^{1/N} - 1) I_k, \quad & P &\leq (\gamma - \alpha) I, \\ P_0 &\leq (\beta - 1) R, & P &\leq (\delta - \alpha^{1/N}\beta) I_k R. \end{aligned} \tag{20}$$

These hypotheses also yield a simpler upper bound,

$$W^{\text{par}}_{\text{ub}} \leq \gamma (P_0 - 1) \frac{I}{P} + \delta \sum_{k \in [N]} \frac{I_k R}{P_k P_0}. \tag{21}$$

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, Eq. (21) attains one of the two lower bounds Eqs. (5) and (6).

In the first case, $NR \leq (I/P)^{1-1/N}$, we suppose there exists a constant $\epsilon > 0$ such that P factors as $P_0 P_1 \cdots P_N$ 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 η, $0 < \eta < \sqrt{2/(3\gamma)}$ such that

$$P \geq \left(\frac{\delta}{\sqrt{2/(3\gamma)} - \eta} \frac{\sum I_k}{N I^{1/N}} \right)^{\frac{N}{N-1}}.$$

The first hypothesis simplifies the upper bound Eq. (21) to $W^{\text{par}}_{\text{ub}} \leq \epsilon \cdot NR(I/P)^{1/N}$, while the second hypothesis simplifies the lower bound Eq. (6) to $W^{\text{par}}_{\text{lb2}} \geq \eta \cdot NR(I/P)^{1/N}$.

In the second case, $(NR)^N > (I/P)^{N-1}$, we suppose there exist constants $\mu, \nu > 0$ such that P factors as $P_0 P_1 \cdots P_N$,

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

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

$$P \geq \frac{\left(\frac{\delta}{2-(\gamma+\tau)} \sum I_k \right)^{\frac{2N-1}{N-1}} R}{(NI)^{\frac{N}{N-1}}}.$$

The first hypothesis simplifies the upper bound Eq. (21) to $W^{\text{par}}_{\text{ub}} \leq (\mu + \nu) \cdot (NIR/P)^{N/(2N-1)}$, while the second hypothesis simplifies the lower bound Eq. (5) to $W^{\text{par}}_{\text{lb1}} \geq \tau \cdot (NIR/P)^{N/(2N-1)}$. In each of the two cases, the gap is a constant factor. ∎

To illustrate the hypotheses of Thm. 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 Eq. (20) for the upper bound simplification to apply become $P_k \leq 0.05 I_k$, $P \leq 0.7I$, $P_0 \leq 0.5R$, and $P \leq 0.175 I_k R$. 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 $P \geq 465 NR/I^{1-1/N}$, respectively.

We note that in the first case of the proof, when $NR \leq (I/P)^{1-1/N}$, the medium-grained algorithm applied to dense tensors [18], [19] achieves the lower bound. In the second case, Alg. 3, which generalizes Alg. 2 and previous work, is necessary to attain the lower bound.

We also compare Alg. 3 with the MTTKRP via matrix mul-

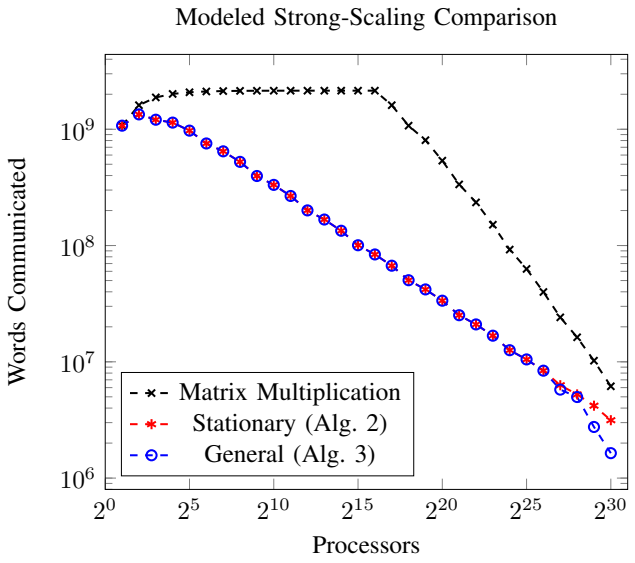


Fig. 4. Model of strong-scaling communication performance comparing Alg. 2, Alg. 3, and MTTKRP via matrix multiplication for a 3-way cubical tensor where $I = 2^{45}$ and $R = 2^{15}$. The matrix multiplication costs are computed using the CARMA algorithm [11], but they do not include the communication costs of forming the Khatri-Rao product.

tiplication approach. For comparison, we use the theoretical costs of communication-optimal parallel matrix multiplication algorithms [11]. 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 $k \in [N]$. 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 $P \geq I/R^2$, then the communication cost is $(IR/P)^{2/3}$, assuming enough memory is available [11]. For comparison, if $P \leq I/(NR)^{N/(N-1)}$, then Alg. 3 (which reduces to Alg. 2 in this case) is optimal with communication cost $NR(I/P)^{1/N}$; if $P \geq I/(NR)^{N/(N-1)}$, then Alg. 3 is optimal with communication cost $(NIR/P)^{N/(2N-1)}$.

Motivated thus, let us associate the small P case with $P \leq \min\left(I^{1-1/N}, I/(NR)^{N/(N-1)}\right)$ and the large P case with $P \geq \max\left(I/R^2, I/(NR)^{N/(N-1)}\right)$. 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 $O((IR/P)^{(N-2)/(6N-3)}/N^{N/(2N-1)})$ less communication.

Fig. 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^{30}. We see that our proposed algorithms perform less communication than matrix multiplication throughout the range of processors, and that Alg. 2 and Alg. 3 diverge only when $P \geq 2^{27}$. When there are 2^{17}=131,072 processors, Alg. 2 and Alg. 3 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 Alg. 2 is sufficient for most practical problems. We note that the change in slope in the matrix multiplication curve is due to a switch from a 1D parallel algorithm to a 2D parallel algorithm and that these communication costs are optimal for matrix multiplication, up to constant factors [11]. 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 Sec. V.

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 [12] 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 Sec. VI-A, we expect only limited practical benefits of the sequential algorithm (Alg. 1). However, we believe the parallel algorithms will be very competitive in practice. The simpler algorithm (Alg. 2) may be the most useful, particularly when R and P are small. Indeed, good performance has been demonstrated for sparse tensors [18], and good scaling has been reported for dense tensors [19]. However, the general algorithm (Alg. 3) 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 used for other tensor computations [26].

While this work focuses on a single MTTKRP computation (corresponding to a single mode), the computation often occurs in the context of an optimization algorithm that repeatedly computes an MTTKRP for each mode of the tensor. In this context, one can optimize across multiple MTTKRP computations, because they share both data and intermediate computations [14], and save both communication and computation.

Our communication lower-bound approach extends to algorithms for multiple MTTKRPs. Extensions are possible for other related computational kernels, such as those within algorithms for computing Tucker and other decompositions. Another natural extension is MTTKRPs involving sparse tensors: in this case, the communication requirements depend on the nonzero structure and can be expressed in terms of a hypergraph partitioning problem [17], [27].

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. This work was partially funded by the Laboratory-Directed Research & Development (LDRD) program at Oak Ridge National Laboratory. This material is also based upon work supported by the NSF Grant No. ACI-1642385.

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. [Online]. Available: http://doi.org/10.1109/TSP.2017.2690524

[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 *STOC '81*. ACM, 1981, pp. 326–333. [Online]. Available: http://doi.acm.org/10.1145/800076.802486

[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] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990. [Online]. Available: https://dl.acm.org/citation.cfm?id=79181

[8] D. Irony, S. Toledo, and A. Tiskin, "Communication lower bounds for distributed-memory matrix multiplication," *Journal of Parallel and Distributed Computing*, vol. 64, no. 9, pp. 1017–1026, 2004. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2004.03.021

[9] T. M. Smith and R. A. van de Geijn, "Pushing the bounds for matrix-matrix multiplication," arXiv, Tech. Rep., 2017. [Online]. Available: http://arxiv.org/abs/1702.02017

[10] 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 *SPAA '12*. New York, NY, USA: ACM, June 2012, pp. 77–79. [Online]. Available: http://doi.acm.org/10.1145/2312005.2312021

[11] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger, "Communication-optimal parallel recursive rectangular matrix multiplication," in *IPDPS '13*, ser. IPDPS '13, 2013, pp. 261–272. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2013.80

[12] M. Christ, J. Demmel, N. Knight, T. Scanlon, and K. Yelick, "Communication lower bounds and optimal algorithms for programs that reference arrays - part 1," UC Berkeley, Tech. Rep. UCB/EECS-2013-61, May 2013. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-61.html

[13] 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. [Online]. Available: http://doi.org/10.1137/060676489

[14] A.-H. Phan, P. Tichavsky, and A. Cichocki, "Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations," *IEEE Transactions on Signal Processing*, vol. 61, no. 19, pp. 4834–4846, Oct 2013. [Online]. Available: http://doi.acm.org/10.1109/TSP.2013.2269903

[15] K. Hayashi, G. Ballard, J. Jiang, and M. J. Tobia, "Shared memory parallelization of MTTKRP for dense tensors," arXiv, Tech. Rep., 2017. [Online]. Available: http://arxiv.org/abs/1708.08976

[16] J. H. Choi and S. V. N. Vishwanathan, "DFacTo: Distributed factorization of tensors," in *NIPS '14*. Cambridge, MA, USA: MIT Press, 2014, pp. 1296–1304. [Online]. Available: http://dl.acm.org/citation.cfm?id=2968826.2968971

[17] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *SC '15*. New York, NY, USA: ACM, 2015, pp. 77:1–77:11. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807624

[18] S. Smith and G. Karypis, "A medium-grained algorithm for distributed sparse tensor factorization," in *IPDPS '16*, May 2016, pp. 902–911. [Online]. Available: http://doi.org/10.1109/IPDPS.2016.113

[19] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos, "Nesterov-based alternating optimization for nonnegative tensor factorization: Algorithm and parallel implementation," *IEEE Transactions on Signal Processing*, Nov 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8119874/

[20] 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: http://www.cs.rpi.edu/research/pdf/16-02.pdf

[21] N. Knight, "Communication-optimal loop nests," 2015, PhD dissertation, Dept. of Electrical Engineering and Computer Science, UC-Berkeley. (UCB/EECS-2015-185). [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-185.pdf

[22] 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. [Online]. Available: http://doi.org/10.1007/s00039-007-0619-6

[23] L. H. Loomis and H. Whitney, "An inequality related to the isoperimetric inequality," *Bulletin of the AMS*, vol. 55, pp. 961–962, 1949. [Online]. Available: http://doi.org/10.1090/S0002-9904-1949-09320-5

[24] G. Ballard, N. Knight, and K. Rouse, "Communication lower bounds for matricized tensor times Khatri-Rao product," arXiv, Tech. Rep., 2017. [Online]. Available: http://arxiv.org/pdf/1708.07401.pdf

[25] 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

[26] W. Austin, G. Ballard, and T. G. Kolda, "Parallel tensor compression for large-scale scientific data," in *IPDPS '16*, May 2016, pp. 912–922. [Online]. Available: http://www.computer.org/csdl/proceedings/ipdps/2016/2140/00/2140a912-abs.html

[27] G. Ballard, A. Druinsky, 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