SIAM J. MATRIX ANAL. APPL.
Vol. 45, No. 1, pp. 450–477

COMMUNICATION LOWER BOUNDS AND OPTIMAL ALGORITHMS FOR MULTIPLE TENSOR-TIMES-MATRIX COMPUTATION*

HUSSAM AL DAAS†, GREY BALLARD‡, LAURA GRIGORI§, SURAJ KUMAR¶, AND KATHRYN ROUSE‖

Abstract. Multiple tensor-times-matrix (Multi-TTM) is a key computation in algorithms for computing and operating with the Tucker tensor decomposition, which is frequently used in multidimensional data analysis. We establish communication lower bounds that determine how much data movement is required (under mild conditions) to perform the Multi-TTM computation in parallel. The crux of the proof relies on analytically solving a constrained, nonlinear optimization problem. We also present a parallel algorithm to perform this computation that organizes the processors into a logical grid with twice as many modes as the input tensor. We show that, with correct choices of grid dimensions, the communication cost of the algorithm attains the lower bounds and is therefore communication optimal. Finally, we show that our algorithm can significantly reduce communication compared to the straightforward approach of expressing the computation as a sequence of tensor-times-matrix operations when the input and output tensors vary greatly in size.

Key words. communication lower bounds, Multi-TTM, tensor computations, parallel algorithms, HBL-inequalities

MSC codes. 15A69, 68Q17, 68Q25, 68W10, 68W15, 68W40

DOI. 10.1137/22M1510443

1. Introduction.

The Tucker tensor decomposition is a low-rank representation or approximation that enables significant compression of multidimensional data. The Tucker format consists of a core tensor, which is much smaller than the original data tensor, along with a factor matrix for each mode, or dimension, of the data. Computations involving Tucker-format tensors, such as tensor inner products, often require far fewer operations than with their full-format, dense representations. As a result, the Tucker decomposition is often used as a dimensionality reduction technique before other types of analysis are done, including computing a CP decomposition [8], for example.

*Received by the editors July 20, 2022; accepted for publication (in revised form) August 4, 2023; published electronically February 6, 2024.
https://doi.org/10.1137/22M1510443
Funding: This work is supported by the National Science Foundation under grant CCF-1942892 and OAC-2106920. This material is based upon work supported by the US Department of Energy, Office of Science, Advanced Scientific Computing Research program under award DE-SC-0023296. This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement 810367).

†Science and Technology Facilities Council, Rutherford Appleton Laboratory, Didcot, Oxfordshire, OX11 0QX, UK (hussam.al-daas@stfc.ac.uk).
‡Computer Science, Wake Forest University, Winston-Salem, NC 27106 USA (ballard@wfu.edu).
§Ecole Polytechnique Fédérale de Lausanne PFL Institute of Mathematics, 1015 Lausanne and Paul Scherrer Institute, Laboratory for Simulation and Modelling, 5232 PSI Villigen, Switzerland; Significant part of the work performed while at Sorbonne Université, Inria, CNRS, Université de Paris, Laboratoire Jacques-Louis Lions, Paris, France (laura.grigori@epfl.ch).
¶Inria Lyon Centre, France; Significant part of the work performed while at Sorbonne Université, Inria, CNRS, Université de Paris, Laboratoire Jacques-Louis Lions, Paris, France (suraj.kumar@inria.fr).
‖Inmar Intelligence, Winston-Salem, NC 27101 USA (kathryn.rouse@inmar.com).

A 3-way Tucker-format tensor can be expressed using the tensor notation $\mathcal{T} = \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \mathbf{A}^{(3)}$, where $\mathcal{G}$ is the 3-way core tensor, $\mathbf{A}^{(n)}$ is a tall-skinny factor matrix corresponding to mode n, and $\times_n$ denotes the tensor-times-matrix (TTM) operation in the nth mode [18]. Here, $\mathcal{T}$ is the full-format representation of the tensor that can be constructed explicitly by performing multiple TTM operations. We call this collective operation the multiple TTM (Multi-TTM) computation, which is the focus of this work.

Multi-TTM is a fundamental computation in the context of Tucker-format tensors. When the Tucker decomposition is used as a data compression tool, Multi-TTM is exactly the decompression operation, which is necessary when the full format is required for visualization [19], for example. In the case of full decompression, the input tensor is small and the output tensor is large. One of the quasi-optimal algorithms for computing the Tucker decomposition is the truncated higher-order SVD algorithm [29, 20], in which each factor matrix is computed as the leading left singular vectors of a matrix unfolding of the tensor. In this algorithm, the smaller core tensor is computed via Multi-TTM involving the larger data tensor and the computed factor matrices. When the computational costs of the matrix SVDs are reduced using randomization via sketching, Multi-TTM becomes the overwhelming bottleneck computation [24, 27]. Recent work [23] has proposed a randomized algorithm for computing Tucker decompositions in which the sketches themselves are performed using Multi-TTM with a very large ratio of input to output tensor size.

Since the overall size of multidimensional data grows quickly, there have been many recent efforts to parallelize the computation of the Tucker decomposition and the operations on Tucker-format tensors [2, 9, 22, 11, 4]. There has also been recent progress in establishing lower bounds on the communication costs of parallel algorithms for tensor computations, including the matricized-tensor times Khatri–Rao product (MTTKRP) [5, 6, 30] and symmetric tensor contractions [26]. However, to our knowledge, no communication lower bounds have been previously established for computations relating to Tucker-format tensors. In this work, we prove communication lower bounds for a class of Multi-TTM algorithms. Additionally, we provide a parallel algorithm that attains the lower bound to within a constant factor and is therefore communication optimal.

To minimize the number of arithmetic operations in a Multi-TTM computation, the TTM operations should be performed in sequence, forming temporary intermediate tensors after each step. A single TTM corresponds to a matrix multiplication along a particular mode of the tensor; therefore, a series of matrix multiplications is performed in the sequence approach to compute the final result. One of the key observations of this work is that when Multi-TTM is performed in parallel, this approach may communicate more data than necessary, even if communication-optimal algorithms are used for each individual TTM. By considering the Multi-TTM computation as a whole, we devise an *atomic*[1] parallel algorithm that communicates less than this TTM-in-Sequence approach when the input and output tensors vary greatly in size, often with negligible increase in computation.

The main contributions of this paper are to

- establish communication lower bounds for the parallel atomic Multi-TTM computation;
- propose a communication optimal parallel algorithm; and

[1]See Definition 3.2.

- show that in many typical scenarios, the straightforward approach based on a sequence of TTM operations communicates more than performing Multi-TTM as a whole.

The rest of the paper is organized as follows. Section 2 describes previous work on communication lower bounds for matrix multiplication and some tensor operations. In section 3, we present our notations and preliminaries for the general Multi-TTM computation. To reduce the complexity of notations, we first focus on 3-dimensional (3D) Multi-TTM computation for which we present communication lower bounds and a communication optimal algorithm in section 4 and section 5, respectively. In section 6, we validate the optimality of the proposed algorithm and show that it significantly reduces communication compared to the TTM-in-Sequence approach with negligible increase in computation in many practical cases. We present our general results in sections 7 and 8 and propose conclusions and perspectives in section 9.

2. Related work. A number of studies have focused on communication lower bounds for matrix multiplication, starting with the work by Hong and Kung [15] to determine the minimum number of input/output operations for sequential matrix multiplication using the red-blue pebble game. Irony, Toledo, and Tiskin [16] extended this work for the parallel case. Demmel et al. [14] studied memory independent communication lower bounds for rectangular matrix multiplication based on aspect ratios of matrices. Recently, Smith et al. [25] and Al Daas et al. [1] have tightened communication lower bounds for matrix multiplication. Ballard et al. [3] extended communication lower bounds of the matrix multiplication for any computations that can be written as 3 nested loops. Christ et al. [12] generalized the method to prove communication lower bounds of 3 nested loop computations for arbitrary loop nesting. We apply their approach to our Multi-TTM definition.

There is limited work on communication lower bounds for tensor operations. Solomonik, Demmel, and Hoefler [26] proposed communication lower bounds for symmetric tensor contraction algorithms. Ballard, Knight, and Rouse [5] proposed communication lower bounds for MTTKRP computation with cubical tensors. This work is extended in [6] to handle varying tensor dimensions. A sequential lower bound for tile-based MTTKRP algorithms is proved by Ziogas et al. [30]. We use some results from [5, 6] to prove communication lower bounds for Multi-TTM.

3. Notations and preliminaries. In this section, we present our notations and basic lemmas for d-dimensional Multi-TTM computation. In sections 4 to 6, we focus on $d = 3$, i.e., $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)^\mathsf{T}} \times_2 \mathbf{A}^{(2)^\mathsf{T}} \times_3 \mathbf{A}^{(3)^\mathsf{T}}$. We present our general results in sections 7 and 8.

We use boldface uppercase Euler script letters to denote tensors ($\mathcal{X}$) and boldface uppercase letters with superscripts to denote matrices ($\mathbf{A}^{(1)}$). We use lowercase letters with subscripts to denote sizes (n_1) and add the prime symbol to them to denote the indices (n_1'). We use one-based indexing throughout and $[d]$ to denote the set $\{1, 2, \cdots, d\}$. To improve the presentation, we denote the product of elements having the same lowercase letter with all subscripts by the lowercase letter only ($n_1, \ldots, n_d$ by n and $r_1, \ldots, r_d$ by r). We denote the product of the i rightmost terms with the capital letter with subscript i, $N_i = \prod_{j=d-i+1}^{d} n_j$, and $R_i = \prod_{j=d-i+1}^{d} r_i$; thus, $n = N_d$, and $n_d = N_1$.

Let $\mathcal{Y} \in \mathbb{R}^{r_1 \times \cdots \times r_d}$ be the d-mode output tensor, $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ be the d-mode input tensor, and $\mathbf{A}^{(k)} \in \mathbb{R}^{n_k \times r_k}$ be the matrix of the kth mode. Then the Multi-TTM computation can be represented as $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)^\mathsf{T}} \cdots \times_d \mathbf{A}^{(d)^\mathsf{T}}$. Without loss

of generality and to simplify notation, we consider that the input tensor $\mathcal{X}$ is larger than the output tensor $\mathcal{Y}$, or $n \geq r$. This corresponds to computing the core tensor of a Tucker decomposition given computed factor matrices, for example. However, the opposite relationship where the output tensor is larger (e.g., $\mathcal{X} = \mathcal{Y} \times_1 \mathbf{A}^{(1)} \cdots \times_d \mathbf{A}^{(d)}$) is also an important use case, corresponding to forming an explicit representation of a (sub)tensor of a Tucker-format tensor. Our results extend straightforwardly to this case. We consider d-dimensional input and output tensors and therefore assume $n_i \geq 2$ and $r_i \geq 2$ for $1 \leq i \leq d$. We also assume without loss of generality that the tensor modes are ordered in such a way that $n_1 r_1 \leq n_2 r_2 \leq \cdots \leq n_d r_d$.

DEFINITION 3.1. *Let $\mathcal{X}$ be an $n_1 \times \cdots \times n_d$ tensor, $\mathcal{Y}$ be an $r_1 \times \cdots \times r_d$ tensor, and $\mathbf{A}^{(j)}$ be an $n_j \times r_j$ matrix for $j \in [d]$. Multi-TTM computes*

$$\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)^\mathsf{T}} \cdots \times_d \mathbf{A}^{(d)^\mathsf{T}},$$

where for each $(r'_1, \ldots, r'_d) \in [r_1] \times \cdots \times [r_d]$,

$$\mathcal{Y}(r'_1, \ldots, r'_d) = \sum_{\{n'_k \in [n_k]\}_{k \in [d]}} \mathcal{X}(n'_1, \ldots, n'_d) \prod_{j \in [d]} \mathbf{A}^{(j)}(n'_j, r'_j). \tag{3.1}$$

Let us consider an example when $d = 2$. In this scenario, the input and output tensors are in fact matrices $\mathbf{X}, \mathbf{Y}$, and $\mathbf{Y} = \mathbf{A}^{(1)\mathsf{T}} \mathbf{X} \mathbf{A}^{(2)}$. As mentioned earlier, Multi-TTM computation can be performed as a sequence of TTM operations, in this case two matrix multiplications. However, we define the Multi-TTM to perform all the products at once for each term of the summation of (3.1). Our definition comes at greater arithmetic cost, as partial $(d+1)$-ary multiplies are not computed and reused, but we will see that this approach can reduce communication cost. We describe how the extra computation can often be reduced to a negligible cost in subsection 5.1 and compare it to the computation cost of TTM-in-Sequence in subsection 6.2.2.

We can write pseudocode for the Multi-TTM with the following:

```
for n'_1 = 1:n_1, ..., for n'_d = 1:n_d,
  for r'_1 = 1:r_1, ..., for r'_d = 1:r_d,
    Y(r'_1, ..., r'_d) += X(n'_1, ..., n'_d) · A^(1)(n'_1, r'_1) ··· A^(d)(n'_d, r'_d).
```

DEFINITION 3.2. *A* parallel atomic Multi-TTM algorithm computes *each term of the summation of* (3.1) *atomically on a unique processor, but it can distribute the nr terms over processors in any way.*

Here atomic computation of a single $(d+1)$-ary multiplication for a parallel algorithm means that all the multiplications of this operation are performed on only one processor; i.e., all $d+1$ inputs are accessed on that processor in order to compute the single output value. This assumption is necessary for our communication lower bounds. Processors can reorganize their local atomic operations to reduce computational costs without changing the communication or violating parallel atomicity. However it is reasonable for an algorithm to break this assumption in order to improve arithmetic costs by reusing partial results across processors, and we compare against such algorithms in section 6.

3.1. Parallel computation model. We consider that the computation is distributed across P processors. Each processor has its own local memory and is connected to all other processors via a fully connected network. Every processor can

operate on data in its local memory and must communicate to access data of other processors. Hence, communication refers to send and receive operations that transfer data from local memory to the network and vice versa. Communication cost mainly depends on two factors: the amount of data communicated (bandwidth cost) and the number of messages (latency cost). Latency cost is dominated by bandwidth cost for computations involving large messages, so we focus on bandwidth cost in this work and refer it as communication cost throughout the text. We assume the links of the network are bidirectional and that the communication cost is independent of the number of pairs of processors that are communicating. Each processor can send and receive at most one message at the same time. In our model, the communication cost of an algorithm refers to the cost along the critical path.

3.2. Existing results. Our work relies on two fundamental results. The first, a geometric result on lattices, allows us to relate the volume of computation to the amount of data accessed by determining the maximum data reuse. The result is a specialization of the Hölder–Brascamp–Lieb inequalities [7]. This result has previously been used to derive lower bounds for tensor computations [5, 6, 12, 17] in a similar way to the use of the Loomis–Whitney inequality [21] in derivations of communication lower bounds for several linear algebra computations [3]. The result is proved in [12], but we use the statement from [5, Lemma 4.1]. Here $\mathbf{1}$ represents a vector of all ones and $\geq$ relation between vectors applies elementwise.

LEMMA 3.3. *Consider any positive integers ℓ and m and any m projections $\phi_j : \mathbb{Z}^\ell \to \mathbb{Z}^{\ell_j}$ ($\ell_j \leq \ell$), each of which extracts ℓ_j coordinates $S_j \subseteq [\ell]$ and forgets the $\ell - \ell_j$ others. Define $\mathcal{C} = \left\{\mathbf{s} = [s_1 \ \cdots \ s_m]^\mathsf{T} : 0 \leq s_i \leq 1 \text{ for } i = 1, 2, \ldots, m \text{ and } \boldsymbol{\Delta} \cdot \mathbf{s} \geq \mathbf{1}\right\}$, where the $\ell \times m$ matrix $\boldsymbol{\Delta}$ has entries $\boldsymbol{\Delta}_{i,j} = 1$ if $i \in S_j$ and $\boldsymbol{\Delta}_{i,j} = 0$ otherwise. If $[s_1 \ \cdots \ s_m]^\mathsf{T} \in \mathcal{C}$, then $\forall F \subseteq \mathbb{Z}^\ell$,*

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

The second result, a general constrained optimization problem, allows us to cast the communication cost of an algorithm as the objective function in an optimization problem where the constraints are imposed by properties of the computation within the algorithm. A version of the result is proved in [6, Lemma 5.1] and used to derive the general communication lower bound for MTTKRP.

THEOREM 3.4. *Consider the constrained optimization problem:*

$$\min \sum_{j \in [d]} x_j$$

such that

$$\frac{nr}{P} \leq \prod_{j \in [d]} x_j \quad \text{and} \quad 0 \leq x_j \leq k_j \quad \forall \quad 1 \leq j \leq d$$

for some positive constants $k_1 \leq k_2 \leq \cdots \leq k_d$ with $\prod_{j \in [d]} k_j = nr$. Then the minimum value of the objective function is

$$I\,(K_I/P)^{1/I} + \sum_{j \in [d-I]} k_j,$$

where we use the notation $K_I = \prod_{j=d-I+1}^{d} k_j$ *and* $1 \le I \le d$ *is defined such that* $L_I \le P < L_{I+1}$.

$$\textit{Here} \quad L_j = \frac{K_j}{(k_{d-j+1})^j} \quad \textit{for} \quad 1 \le j \le d \quad \textit{and} \quad L_{d+1} = \infty.$$

The minimum is achieved at the point $\mathbf{x}^*$ *defined by* ${x_j}^* = k_j$ *for* $1 \le j \le d - I$, ${x_\ell}^* = (K_I/P)^{1/I}$ *for* $d - I < \ell \le d$.

While Theorem 3.4 can be straightforwardly derived from the previous work, we provide an alternate proof in an extended version [13]. We represent it in this form to be directly applicable to all the constrained optimization problems in this paper. The constraints $nr/P \le \prod_{j\in[d]} x_j$ and $\prod_{j\in[d]} k_j = nr$ are derived from the Multi-TTM computation. The equality constraint on $\prod_{j\in[d]} k_j$ implies that there is always a feasible solution to the optimization problem for $P \ge 1$. We calculate the ranges of P for each I in Corollaries 4.1, 4.2, and 7.1.

4. Lower bounds for 3D Multi-TTM. We obtain the lower bound results for 3D tensors in this section, presented as Theorem 4.3. The lower bound is independent of the size of the local memory of each processor, similar to previous results for matrix multiplication [1, 14] and MTTKRP [5, 6], and it varies with respect to the number of processors P relative to the matrix and tensor dimensions of the problem.

The proof focuses on a processor that performs $1/P$th of the computation and owns at most $1/P$th of the data. It reduces the problem of finding a lower bound on the amount of data the processor must communicate to solve a constrained optimization problem: we seek to minimize the number of elements of the matrices and tensors that the processor must access or partially compute in order to execute its computation subject to structure constraints of Multi-TTM. The most important constraint derives from Lemma 3.3, which relates a subset of the computation within a Multi-TTM algorithm to the data it requires. The other constraints provide upper bounds on the data required from each array. The upper bounds are necessary to establish the tightest lower bounds in the cases where P is small. We show that the optimization problem can be separated into two independent problems, one for the matrix data and one for the tensor data. Corollaries 4.1 and 4.2 state the two constrained optimization problems, along with their analytic solutions, both of which follow from Theorem 3.4. That is, setting $d = 3$, $k_1 = n_1 r_1$, $k_2 = n_2 r_2$, and $k_3 = n_3 r_3$ in Theorem 3.4, we obtain Corollary 4.1. Similarly, setting $d = 2$ with $k_1 = r$ and $k_2 = n$, we obtain Corollary 4.2. We recall here that $r = r_1 r_2 r_3$ and $n = n_1 n_2 n_3$.

COROLLARY 4.1. *Consider the following optimization problem:*

$$\min_{x,y,z} x + y + z$$

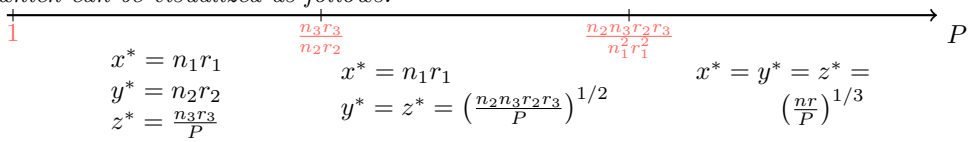
such that

$$\begin{aligned} \frac{nr}{P} &\le xyz, \\ 0 \le \ x \ &\le n_1 r_1, \\ 0 \le \ y \ &\le n_2 r_2, \\ 0 \le \ z \ &\le n_3 r_3, \end{aligned}$$

where $n_1 r_1 \le n_2 r_2 \le n_3 r_3$, *and* $n_1, n_2, n_3, r_1, r_2, r_3, P \ge 1$. *The optimal solution* (x^*, y^*, z^*) *depends on the relative values of the constraints, yielding three cases:*

1. *if* $P < \frac{n_3 r_3}{n_2 r_2}$, *then* $x^* = n_1 r_1$, $y^* = n_2 r_2$, $z^* = \frac{n_3 r_3}{P}$;
2. *if* $\frac{n_3 r_3}{n_2 r_2} \leq P < \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}$, *then* $x^* = n_1 r_1$, $y^* = z^* = \left(\frac{n_2 n_3 r_2 r_3}{P}\right)^{\frac{1}{2}}$;
3. *if* $\frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2} \leq P$, *then* $x^* = y^* = z^* = \left(\frac{nr}{P}\right)^{\frac{1}{3}}$,

which can be visualized as follows.



COROLLARY 4.2. *Consider the following optimization problem:*

$$\min_{u,v} u + v$$

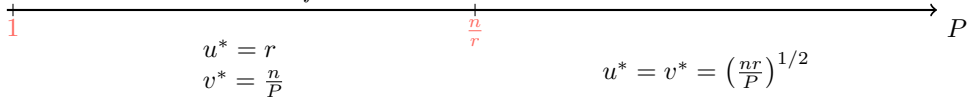
such that

$$\begin{aligned} \frac{nr}{P} &\leq uv, \\ 0 &\leq u \leq r, \\ 0 &\leq v \leq n, \end{aligned}$$

where $n \geq r$, *and* $n, r, P \geq 1$. *The optimal solution* (u^*, v^*) *depends on the relative values of the constraints, yielding two cases:*

1. *if* $P < \frac{n}{r}$, *then* $u^* = r$, $v^* = \frac{n}{P}$;
2. *if* $\frac{n}{r} \leq P$, *then* $u^* = v^* = \left(\frac{nr}{P}\right)^{\frac{1}{2}}$,

which can be visualized as follows.



4.1. Communication lower bounds for Multi-TTM. We now state the lower bounds for 3D Multi-TTM. After this, we also present a corollary for cubical tensors.

THEOREM 4.3. *Any computationally load-balanced atomic Multi-TTM algorithm that starts and ends with one copy of the data distributed across processors involving 3D tensors with dimensions* n_1, n_2, n_3 *and* r_1, r_2, r_3 *performs at least* $A + B - (\frac{n}{P} + \frac{r}{P} + \sum_{j=1}^{3} \frac{n_j r_j}{P})$ *sends or receives where*

$$A = \begin{cases} n_1 r_1 + n_2 r_2 + \frac{n_3 r_3}{P} & \text{if } P < \frac{n_3 r_3}{n_2 r_2}, \\ n_1 r_1 + 2\left(\frac{n_2 n_3 r_2 r_3}{P}\right)^{\frac{1}{2}} & \text{if } \frac{n_3 r_3}{n_2 r_2} \leq P < \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}, \\ 3\left(\frac{nr}{P}\right)^{\frac{1}{3}} & \text{if } \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2} \leq P, \end{cases}$$

$$B = \begin{cases} r + \frac{n}{P} & \text{if } P < \frac{n}{r}, \\ 2\left(\frac{nr}{P}\right)^{\frac{1}{2}} & \text{if } \frac{n}{r} \leq P. \end{cases}$$

Proof. Let F be the set of loop indices associated with the 4-ary multiplication performed by a processor. As we assumed the algorithm is computationally load balanced, $|F| = nr/P$. We define $\phi_{\mathcal{X}}(F)$, $\phi_{\mathcal{Y}}(F)$, and $\phi_j(F)$ to be the projections of F onto the indices of the arrays $\mathcal{X}, \mathcal{Y}$, and $\mathbf{A}^{(j)}$ for $1 \leq j \leq 3$ which correspond to the elements of the array that must be accessed or partially computed by the processor.

We use Lemma 3.3 to obtain a lower bound on the number of array elements that must be accessed or partially computed by the processor. The computation involves 5 arrays (2 tensors and 3 matrices) with 6 loop indices (see the atomic Multi-TTM definition in section 3); hence, the 6×5 matrix corresponding to the projections above is given by

$$\boldsymbol{\Delta} = \begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{1}_3 & \mathbf{0}_3 \\ \mathbf{I}_{3\times 3} & \mathbf{0}_3 & \mathbf{1}_3 \end{bmatrix}.$$

Here $\mathbf{1}_3$ and $\mathbf{0}_3$ represent the 3D vectors of all ones and zeros, respectively, and $\mathbf{I}_{3\times 3}$ represents the 3×3 identity matrix. We recall from Lemma 3.3 that $\boldsymbol{\Delta}_{i,j} = 1$ if loop index i is used to access array j and $\boldsymbol{\Delta}_{i,j} = 0$ otherwise. The first three columns of $\boldsymbol{\Delta}$ correspond to matrices and the remaining two columns correspond to tensors. In this case, we have

$$\mathcal{C} = \left\{ \mathbf{s} = [s_1 \ \cdots \ s_5]^{\mathsf{T}} : 0 \le s_i \le 1 \text{ for } i = 1, 2, \ldots, 5 \text{ and } \boldsymbol{\Delta} \cdot \mathbf{s} \ge \mathbf{1} \right\}.$$

Recall that $\mathbf{1}$ represents a vector of all ones. Here $\boldsymbol{\Delta}$ is not full rank, therefore, we consider all vectors $\mathbf{v} \in \mathcal{C}$ such that $\boldsymbol{\Delta} \cdot \mathbf{v} = \mathbf{1}$. Such a vector $\mathbf{v}$ is of the form $[a\ a\ a\ 1\text{-}a\ 1\text{-}a]$, where $0 \le a \le 1$. Therefore, we obtain

$$\frac{nr}{P} \le \left(\prod_{j \in [3]} |\phi_j(F)| \right)^a \left(|\phi_{\mathbf{X}}(F)||\phi_{\mathbf{Y}}(F)| \right)^{1\text{-}a} \ \forall \ 0 \le a \le 1.$$

The above constraint is equivalent to $\frac{nr}{P} \le \prod_{j \in [3]} |\phi_j(F)|$ and $\frac{nr}{P} \le |\phi_{\mathbf{X}}(F)||\phi_{\mathbf{Y}}(F)|$. To see this equivalence, note that the forward direction is implied by setting $a = 0$ and $a = 1$. For the opposite direction, taking the first of the two constraints to the power a and the second to the power $1 - a$ then multiplying the two terms yields the original.

Clearly a projection onto an array cannot be larger than the array itself, thus $|\phi_{\mathbf{X}}(F)| \le n$, $|\phi_{\mathbf{Y}}(F)| \le r$, and $|\phi_j(F)| \le n_j r_j$ for $1 \le j \le 3$.

As the constraints related to projections of matrices and tensors are disjoint, we solve them separately and then sum the results to get a lower bound on the set of elements that must be accessed or partially computed by the processor. We obtain a lower bound on A, the number of relevant elements of the matrices by using Corollary 4.1, and a lower bound on B, the number of relevant elements of the tensors by using Corollary 4.2. By summing both, we get the positive terms of the lower bound.

To bound the sends or receives, we consider how much data the processor could have had at the beginning or at the end of the computation. Assuming there is exactly one copy of the data at the beginning and at the end of the computation, there must exist a processor which owns at most $1/P$ of the elements of the arrays at the beginning or at the end of the computation. By employing the previous analysis, this processor must access or partially compute $A + B$ elements of the arrays, but can only own $\frac{n}{P} + \frac{r}{P} + \sum_{j \in [3]} \frac{n_j r_j}{P}$ elements of the arrays. Thus it must perform the specified amount of sends or receives. ∎

We denote the lower bound of Theorem 4.3 by LB and use it extensively in subsection 5.2 while analyzing the communication cost of our parallel algorithm.

We also state the result for 3D Multi-TTM computation with cubical tensors, which is a direct application of Theorem 4.3 with $n_1 = n_2 = n_3 = n^{\frac{1}{3}}$ and $r_1 = r_2 = r_3 = r^{\frac{1}{3}}$.

COROLLARY 4.4. *Any computationally load-balanced atomic Multi-TTM algorithm that starts and ends with one copy of the data distributed across processors involving 3D cubical tensors with dimensions* $n^{\frac{1}{3}} \times n^{\frac{1}{3}} \times n^{\frac{1}{3}}$ *and* $r^{\frac{1}{3}} \times r^{\frac{1}{3}} \times r^{\frac{1}{3}}$ *(with* $n \geq r$*) performs at least*

$$3\left(\frac{nr}{P}\right)^{\frac{1}{3}} + r - \frac{3(nr)^{\frac{1}{3}} + r}{P}$$

sends or receives when $P < \frac{n}{r}$ *and at least*

$$3\left(\frac{nr}{P}\right)^{\frac{1}{3}} + 2\left(\frac{nr}{P}\right)^{\frac{1}{2}} - \frac{n + 3(nr)^{\frac{1}{3}} + r}{P}$$

sends or receives when $P \geq \frac{n}{r}$.

In particular, we note that the lower bound for cubical atomic Multi-TTM algorithms is smaller than that of a TTM-in-Sequence approach for many typical scenarios in the case $P < n/r$, as we discuss further in section 6.

5. Parallel algorithm for 3D Multi-TTM. We organize P processors into a 6-dimensional $p_1 \times p_2 \times p_3 \times q_1 \times q_2 \times q_3$ logical processor grid. We arrange the grid dimensions such that p_1, p_2, p_3, q_1, q_2, q_3 evenly distribute n_1, n_2, n_3, r_1, r_2, r_3, respectively. A processor coordinate is represented as $(p_1', p_2', p_3', q_1', q_2', q_3')$, where $1 \leq p_k' \leq p_k$, $1 \leq q_k' \leq q_k$ for $k = 1, 2, 3$. To be consistent with our notation, we denote $p_1 p_2 p_3$ and $q_1 q_2 q_3$ by p and q.

$\mathcal{X}_{p_1' p_2' p_3'}$ denotes the subtensor of $\mathcal{X}$ owned by processors $(p_1', p_2', p_3', *, *, *)$. Similarly, $\mathcal{Y}_{q_1' q_2' q_3'}$ denotes the subtensor of $\mathcal{Y}$ owned by processors $(*, *, *, q_1', q_2', q_3')$. $\mathbf{A}^{(1)}_{p_1' q_1'}$, $\mathbf{A}^{(2)}_{p_2' q_2'}$ and $\mathbf{A}^{(3)}_{p_3' q_3'}$ denote submatrices of $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, and $\mathbf{A}^{(3)}$ owned by processors $(p_1', *, *, q_1', *, *)$, $(*, p_2', *, *, q_2', *)$, and $(*, *, p_3', *, *, q_3')$, respectively.

We impose that there is one copy of data in the system at the start and end of the computation, and every array is distributed evenly among the sets of processors whose coordinates are different for the corresponding dimensions of the variable. For example, $\mathcal{X}_{111} = \mathcal{X}(1 : \frac{n_1}{p_1}, 1 : \frac{n_2}{p_2}, 1 : \frac{n_3}{p_3})$ is owned by processors $(1, 1, 1, *, *, *)$. Similarly, $\mathbf{A}^{(1)}_{12} = \mathbf{A}^{(1)}(1 : \frac{n_1}{p_1}, \frac{r_1}{q_1} + 1 : 2\frac{r_1}{q_1})$ is owned by processors $(1, *, *, 2, *, *)$. We assume that data inside these sets of processors is also evenly distributed. For example, in the beginning, processor $(1, 1, 1, 2, 1, 3)$ owns $\frac{1}{P}$th portion of each input variable: $\frac{p}{P}$th portion of $\mathcal{X}_{111}$, $\frac{p_1 q_1}{P}$th portion of $\mathbf{A}^{(1)}_{12}$, $\frac{p_2 q_2}{P}$th portion of $\mathbf{A}^{(2)}_{11}$, and $\frac{p_3 q_3}{P}$th portion of $\mathbf{A}^{(3)}_{13}$. Figure 1 illustrates examples of our data distribution model for two of the arrays.

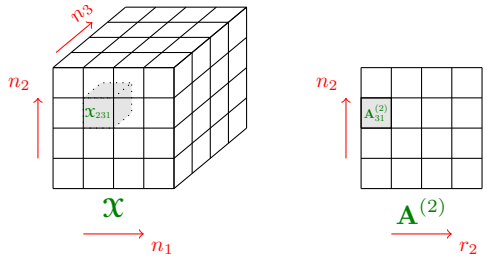


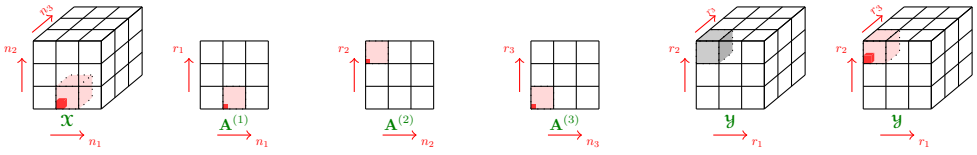
FIG. 1. *Subtensor* $\mathcal{X}_{231}$ *is distributed evenly among processors* $(2, 3, 1, *, *, *)$. *Similarly, submatrix* $\mathbf{A}^{(2)}_{31}$ *is distributed evenly among processors* $(*, 3, *, *, 1, *)$.

Algorithm 5.1. Parallel atomic 3D Multi-TTM.

Require: $\mathcal{X}$, $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, $\mathbf{A}^{(3)}$, $p_1 \times p_2 \times p_3 \times q_1 \times q_2 \times q_3$ logical processor grid

Ensure: $\mathcal{Y}$ such that $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)\mathsf{T}} \times_2 \mathbf{A}^{(2)\mathsf{T}} \times_3 \mathbf{A}^{(3)\mathsf{T}}$

1: $(p'_1, p'_2, p'_3, q'_1, q'_2, q'_3)$ is my processor id
2: //All-gather input tensor $\mathcal{X}$
3: $\mathcal{X}_{p'_1 p'_2 p'_3}$ = All-Gather($\mathcal{X}$, $(p'_1, p'_2, p'_3, *, *, *)$)
4: //All-gather input matrices
5: $\mathbf{A}^{(1)}_{p'_1 q'_1}$ = All-Gather($\mathbf{A}^{(1)}$, $(p'_1, *, *, q'_1, *, *)$)
6: $\mathbf{A}^{(2)}_{p'_2 q'_2}$ = All-Gather($\mathbf{A}^{(2)}$, $(*, p'_2, *, *, q'_2, *)$)
7: $\mathbf{A}^{(3)}_{p'_3 q'_3}$ = All-Gather($\mathbf{A}^{(3)}$, $(*, *, p'_3, *, *, q'_3)$)
8: //Local computations in a temporary tensor $\mathcal{T}$
9: $\mathcal{T}$ = Local-Multi-TTM($\mathcal{X}_{p'_1 p'_2 p'_3}$, $\mathbf{A}^{(1)}_{p'_1 q'_1}$, $\mathbf{A}^{(2)}_{p'_2 q'_2}$, $\mathbf{A}^{(3)}_{p'_3 q'_3}$)
10: //Reduce-scatter the output tensor in $\mathcal{Y}_{q'_1 q'_2 q'_3}$
11: Reduce-Scatter($\mathcal{Y}_{q'_1 q'_2 q'_3}$, $\mathcal{T}$, $(*, *, *, q'_1, q'_2, q'_3)$)



(a) Perform All-Gather on processors $(2, 1, 1, *, *, *)$ to obtain $\mathcal{X}_{211}$.
(b) Perform All-Gather on processors $(2, *, *, 1, *, *)$ to obtain $\mathbf{A}^{(1)}_{21}$.
(c) Perform All-Gather on processors $(*, 1, *, *, 3, *)$ to obtain $\mathbf{A}^{(2)}_{13}$.
(d) Perform All-Gather on processors $(*, *, 1, *, *, 1)$ to obtain $\mathbf{A}^{(3)}_{11}$.
(e) Perform local Multi-TTM to compute partial $\mathcal{Y}_{131}$.
(f) Perform Reduce-Scatter on processors $(*, *, *, 1, 3, 1)$ to compute/distribute $\mathcal{Y}_{131}$.

FIG. 2. *Steps of Algorithm* 5.1 *for processor* $(2, 1, 1, 1, 3, 1)$, *where* $p_1 = p_2 = p_3 = q_1 = q_2 = q_3 = 3$. *Highlighted areas correspond to the data blocks on which the processor is operating. The dark red highlighting represents the input/output data initially/finally owned by the processor, and the light red highlighting corresponds to received/sent data from/to other processors in All-Gather/Reduce-Scatter collectives to compute* $\mathcal{Y}_{131}$.

Algorithm 5.1 presents a parallel algorithm to compute 3D Multi-TTM. When it completes, $\mathcal{Y}_{q'_1 q'_2 q'_3}$ is distributed evenly among processors $(*, *, *, q'_1, q'_2, q'_3)$. Figure 2 shows the steps of the algorithm for a single processor in a $3 \times 3 \times 3 \times 3 \times 3 \times 3$ grid.

5.1. Cost analysis. Now we analyze computation and communication costs of the algorithm. The dimension of the local tensor $\mathcal{X}_{p'_1 p'_2 p'_3}$ is $\frac{n_1}{p_1} \times \frac{n_2}{p_2} \times \frac{n_3}{p_3}$, the dimension of the local matrix $\mathbf{A}^{(k)}_{p'_k q'_k}$ is $\frac{n_i}{p_i} \times \frac{r_i}{q_i}$ for $i = 1, 2, 3$, and the dimension of the temporary tensor $\mathcal{T}$ is $\frac{r_1}{q_1} \times \frac{r_2}{q_2} \times \frac{r_3}{q_3}$. For simplicity of analysis, we assume that the numerator is divisible by the denominator for each cost expression.

The local Multi-TTM computation in line 9 can be performed as a sequence of TTM operations to mininimize the number of arithmetic operations. Assuming the TTM operations are performed in their order, first with $\mathbf{A}^{(1)}$, then with $\mathbf{A}^{(2)}$, and in the end with $\mathbf{A}^{(3)}$, then each processor performs $2\left(\frac{n_1 n_2 n_3 r_1}{p_1 p_2 p_3 q_1} + \frac{n_2 n_3 r_1 r_2}{p_2 p_3 q_1 q_2} + \frac{n_3 r_1 r_2 r_3}{p_3 q_1 q_2 q_3}\right)$ operations to perform the local computation.

Communication occurs only in the All-Gather and Reduce-Scatter collectives in lines 3, 5, 6, 7, and 11. Each processor is involved in one All-Gather involving the input tensor, three All-Gathers involving input matrices, and one Reduce-Scatter involving the output tensor. Therefore, the communication cost of the algorithm along the critical path is the sum of communication costs of these five collectives. Lines 3, 5, 6, and 7 specify p, p_1q_1, p_2q_2, and p_3q_3 All-Gathers over disjoint sets of $\frac{P}{p}$, $\frac{P}{p_1q_1}$, $\frac{P}{p_2q_2}$, and $\frac{P}{p_3q_3}$ processors, respectively. Similarly, line 11 specifies q Reduce-Scatters over disjoint sets of $\frac{P}{q}$ processors.

For simplicity of discussion, we consider that the number of processors involved in the collectives is a power of 2. We also assume that communication-optimal collective algorithms are used. The optimal latency and bandwidth costs of both collectives on Q processors are $\log_2(Q)$ and $(1-\frac{1}{Q})w$, respectively, where w denotes the words of data in each processor after All-Gather or before Reduce-Scatter collectives. Each processor also performs $(1-\frac{1}{Q})w$ computations for the Reduce-Scatter collective. We point the reader to [28, 10] for more details on efficient algorithms for collectives.

Hence the bandwidth costs of lines 3, 5, 6, 7 in Algorithm 5.1 are $(1-\frac{p}{P})\frac{n}{p}$, $(1-\frac{p_1q_1}{P})\frac{n_1r_1}{p_1q_1}$, $(1-\frac{p_2q_2}{P})\frac{n_2r_2}{p_2q_2}$, $(1-\frac{p_3q_3}{P})\frac{n_3r_3}{p_3q_3}$, respectively, to accomplish All-Gather operations, and the bandwidth cost of performing the Reduce-Scatter operation in line 11 is $(1-\frac{q}{P})\frac{r}{q}$. Thus the overall bandwidth cost of Algorithm 5.1 is

$$\frac{n}{p}+\frac{n_1r_1}{p_1q_1}+\frac{n_2r_2}{p_2q_2}+\frac{n_3r_3}{p_3q_3}+\frac{r}{q}-\left(\frac{n+n_1r_1+n_2r_2+n_3r_3+r}{P}\right). \tag{5.1}$$

The latency costs of lines 3, 5, 6, 7, 11 are $\log_2(\frac{P}{p})$, $\log_2(\frac{P}{p_1q_1})$, $\log_2(\frac{P}{p_2q_2})$, $\log_2(\frac{P}{p_3q_3})$, $\log_2(\frac{P}{q})$, respectively. Thus the overall latency cost of Algorithm 5.1 is $\log_2(\frac{P}{p})+\log_2(\frac{P}{p_1q_1})+\log_2(\frac{P}{p_2q_2})+\log_2(\frac{P}{p_3q_3})+\log_2(\frac{P}{q})=\log_2(\frac{P^5}{p^2q^2})=3\log_2(P)$.

Due to the Reduce-Scatter operation, each processor also performs $(1-\frac{q}{P})\frac{r}{q}$ computations, which is dominated by the computations of line 9 (as $n_3 \geq p_3$).

5.2. Selection of p_i and q_i in Algorithm 5.1. We must select the processor dimensions carefully such that Algorithm 5.1 is communication optimal.

We attempt to select the processor dimensions p_i and q_i in such a way that the terms in the communication cost match the optimal solutions of Corollaries 4.1 and 4.2. In other words, we want to select p_i and q_i such that $\frac{n_1r_1}{p_1q_1}=x^*$, $\frac{n_2r_2}{p_2q_2}=y^*$, and $\frac{n_3r_3}{p_3q_3}=z^*$ from Corollary 4.1, and $\frac{n}{p}=v^*$, $\frac{r}{q}=u^*$ from Corollary 4.2.

We need to fix two or three processor grid dimensions for each equation, and each processor grid dimension appears in two equations. In general, we are able to set the processor grid dimensions in a way that is consistent with these equations. However, they are subject to additional constraints that are not imposed by the optimization problem. Specifically, we have $1 \leq p_i \leq n_i$ and $1 \leq q_i \leq r_i$ for $1 \leq i \leq 3$. The lower bounds are imposed because processor grid dimensions must be at least 1. The upper bounds are imposed to ensure that each processor performs its fair share of the computations. We assume that $P \leq nr$, so that every processor has at least one 4-ary multiplication term to compute. For simplicity, we assume that the final grid dimensions are integers and perfectly divide the corresponding input and output dimensions. However, we also discuss how to handle noninteger grid dimensions for a specific set of inputs in subsection 6.3.1.

In order to define processor grid dimensions, we begin by determining a set of values that match the lower bound terms and denote these by $\hat{p}_i, \hat{q}_i$ with their products denoted $\hat{p}$ and $\hat{q}$. Then, we will consider how to adapt $\hat{p}_i$ and $\hat{q}_i$ so that the additional

constraints are met. During the adaption, we maintain the tensor communication costs, modify the matrix communication costs, and then bound the additional costs in terms of communication lower bounds of tensors.

As $\mathcal{X}$ and $\mathcal{Y}$ are 3D tensors, we have $n_i, r_i \geq 2 \; \forall \; 1 \leq i \leq 3$. For better readability, we use the notation $O = \frac{\sum_{j\in[3]} n_j r_j + r + n}{P}$, the amount of data owned by a single processor at the beginning and end of the algorithm.

THEOREM 5.1. *There exist p_i, q_i with $1 \leq p_i \leq n_i, 1 \leq q_i \leq r_i$ for $i = 1, 2, 3$ such that Algorithm* 5.1 *is communication optimal to within a constant factor.*

Proof. We break our analysis into 2 scenarios which are further broken down into cases. In each case, we obtain $\hat{p_i}$ and $\hat{q_i}$ such that the terms in the communication cost match the corresponding lower bound terms and also satisfy at least one of the two constraints: $\forall i, 1 \leq \hat{p_i} \leq n_i, 1 \leq \hat{q_i}$ or $\forall i, 1 \leq \hat{q_i} \leq r_i, 1 \leq \hat{p_i}$. We handle all cases from both scenarios together in the end, and adapt these values to get p_i and q_i which respect both lower and upper bounds.

• *Scenario* I ($P < \frac{n}{r}$): This scenario corresponds to the first case of the tensor term in LB. Thus, we set $\hat{p_i}, \hat{q_i}$ in such a way that the tensor terms in the communication cost match the tensor terms of LB:

$$\hat{p} = P, \hat{q} = 1. \tag{5.2}$$

This implies $\hat{q_i} = 1$ for $1 \leq i \leq 3$. We break this scenario into 3 cases, each corresponding to a case in the matrix term of LB.

Case 1. $P < \frac{n_3 r_3}{n_2 r_2}$: Setting the matrix communication costs to the matrix terms in the corresponding case of LB yields

$$\frac{n_1 r_1}{\hat{p_1}\hat{q_1}} = n_1 r_1, \; \frac{n_2 r_2}{\hat{p_2}\hat{q_2}} = n_2 r_2, \; \frac{n_3 r_3}{\hat{p_3}\hat{q_3}} = \frac{n_3 r_3}{P}. \tag{5.3}$$

Thus, we set $\hat{p_1} = \hat{p_2} = \hat{q_1} = \hat{q_2} = \hat{q_3} = 1$ and $\hat{p_3} = P$ to satisfy (5.2) and (5.3).

Case 2. $\frac{n_3 r_3}{n_2 r_2} \leq P < \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}$: Setting the matrix communication costs to the matrix terms in the corresponding case of LB yields

$$\frac{n_1 r_1}{\hat{p_1}\hat{q_1}} = n_1 r_1, \; \frac{n_2 r_2}{\hat{p_2}\hat{q_2}} = \frac{n_3 r_3}{\hat{p_3}\hat{q_3}} = \Big(\frac{n_2 n_3 r_2 r_3}{P}\Big)^{1/2}. \tag{5.4}$$

We set $\hat{p_1} = \hat{q_1} = \hat{q_2} = \hat{q_3} = 1$, $\hat{p_2} = n_2 r_2 (\frac{P}{n_2 n_3 r_2 r_3})^{\frac{1}{2}}$, and $\hat{p_3} = n_3 r_3 (\frac{P}{n_2 n_3 r_2 r_3})^{\frac{1}{2}}$ to satisfy (5.2) and (5.4). $\frac{n_3 r_3}{n_2 r_2} \leq P$ implies $\hat{p_2} \geq 1$ and $\hat{p_3} \geq 1$.

Case 3. $\frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2} \leq P$: Setting the matrix communication costs to match the matrix terms in the corresponding case of LB yields

$$\frac{n_1 r_1}{\hat{p_1}\hat{q_1}} = \frac{n_2 r_2}{\hat{p_2}\hat{q_2}} = \frac{n_3 r_3}{\hat{p_3}\hat{q_3}} = \Big(\frac{nr}{P}\Big)^{1/3}. \tag{5.5}$$

Thus we set $\hat{q_1} = \hat{q_2} = \hat{q_3} = 1$, $\hat{p_1} = n_1 r_1 \big(\frac{P}{nr}\big)^{\frac{1}{3}}$, $\hat{p_2} = n_2 r_2 \big(\frac{P}{nr}\big)^{\frac{1}{3}}$, and $\hat{p_3} = n_3 r_3 \big(\frac{P}{nr}\big)^{\frac{1}{3}}$ to satisfy (5.2) and (5.5). $\frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2} \leq P$ implies $\hat{p_i} \geq 1$ for $1 \leq i \leq 3$.

Note that in all the cases of this scenario we have $1 \leq \hat{q_i} = 1 < r_i, 1 \leq \hat{p_i}$ for $1 \leq i \leq 3$, but we cannot ensure $\hat{p_i} \leq n_i$ for each i. We will adapt processor grid dimensions for both scenarios in the end as they require the same steps.

• *Scenario* II ($\frac{n}{r} \leq P$): This scenario corresponds to the second case of the tensor term in LB. Thus, we set $\hat{p_i}, \hat{q_i}$ in such a way that

$$\frac{n}{\hat{p}} = \frac{r}{\hat{q}} = \Big(\frac{nr}{P}\Big)^{1/2}. \tag{5.6}$$

Again, we break this scenario into 3 cases, each corresponding to a case in the matrix term of LB.

Case 1. $P < \frac{n_3 r_3}{n_2 r_2}$: Setting the matrix communication costs to the matrix terms in the corresponding case of LB yields

$$\frac{n_1 r_1}{\hat{p_1}\hat{q_1}} = n_1 r_1,\ \frac{n_2 r_2}{\hat{p_2}\hat{q_2}} = n_2 r_2,\ \frac{n_3 r_3}{\hat{p_3}\hat{q_3}} = \frac{n_3 r_3}{P}. \tag{5.7}$$

Thus we set $\hat{p_1} = \hat{q_1} = \hat{p_2} = \hat{q_2} = 1$, $\hat{p_3} = n\left(\frac{P}{nr}\right)^{1/2}$, and $\hat{q_3} = r\left(\frac{P}{nr}\right)^{1/2}$ to satisfy (5.6) and (5.7). As $\frac{n}{r} \leq P \leq nr$ and $r \leq n$, we have $1 \leq \hat{p_3} \leq n$ and $1 \leq \hat{q_3} \leq r$, but cannot ensure $\hat{p_3} \leq n_3$ or $\hat{q_3} \leq r_3$. However, $\hat{p_3}\hat{q_3} = P < \frac{n_3 r_3}{n_2 r_2}$ implies that at least one is satisfied. Therefore, we have $\forall i, 1 \leq \hat{p_i} \leq n_i, 1 \leq \hat{q_i}$, and/or $\forall i, 1 \leq \hat{p_i}, 1 \leq \hat{q_i} \leq r_i$.

Case 2. $\frac{n_3 r_3}{n_2 r_2} \leq P < \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}$: Setting the matrix communication costs to the matrix terms in the corresponding case of LB yields

$$\frac{n_1 r_1}{\hat{p_1}\hat{q_1}} = n_1 r_1, \frac{n_2 r_2}{\hat{p_2}\hat{q_2}} = \frac{n_3 r_3}{\hat{p_3}\hat{q_3}} = \left(\frac{n_2 n_3 r_2 r_3}{P}\right)^{1/2}. \tag{5.8}$$

We set $\hat{p_1} = \hat{q_1} = 1$. Equations (5.6) and (5.8) do not uniquely determine $\hat{p_2}, \hat{p_3}, \hat{q_2}$, and $\hat{q_3}$. The following is one possible solution: $\hat{p_2} = n_2(\frac{n_1 P}{n_2 n_3 r})^{1/4}$, $\hat{p_3} = n_3(\frac{n_1 P}{n_2 n_3 r})^{1/4}$, $\hat{q_2} = r_2(\frac{r_1 P}{n r_2 r_3})^{1/4}$, and $\hat{q_3} = r_3(\frac{r_1 P}{n r_2 r_3})^{1/4}$. Note that $P < \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}$ implies that $\hat{p_2} < n_2$, $\hat{p_3} < n_3$, $\hat{q_2} < r_2$, and $\hat{q_3} < r_3$. We are not able to ensure $\hat{p_2}, \hat{p_3}, \hat{q_2}, \hat{q_3}$ are all at least 1 in this case. We will handle both Case 2 and Case 3 together as they require the same analysis.

Case 3. $\frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2} \leq P$: Setting the matrix communication costs to the matrix terms in the corresponding case of LB yields

$$\frac{n_1 r_1}{\hat{p_1}\hat{q_1}} = \frac{n_2 r_2}{\hat{p_2}\hat{q_2}} = \frac{n_3 r_3}{\hat{p_3}\hat{q_3}} = \left(\frac{nr}{P}\right)^{\frac{1}{3}}. \tag{5.9}$$

Similar to Case 2, (5.6) and (5.9) do not uniquely determine $\hat{p_i}, \hat{q_i}$ for $1 \leq i \leq 3$. We choose a cubical distribution, namely, $\frac{n_1}{p_1} = \frac{n_2}{p_2} = \frac{n_3}{p_3} = \frac{r_1}{q_1} = \frac{r_2}{q_2} = \frac{r_3}{q_3}$ and obtain the following solution: $\hat{p_i} = n_i\left(\frac{P}{nr}\right)^{1/6}$, $\hat{q_i} = r_i\left(\frac{P}{nr}\right)^{1/6}$ for $1 \leq i \leq 3$. As $P \leq nr$ we have $\hat{p_i} \leq n_i$ and $\hat{q_i} \leq r_i$ for $1 \leq i \leq 3$. Again we are not able to ensure $\hat{p_i}$ and $\hat{q_i}$ are all at least 1 for $1 \leq i \leq 3$.

Now we handle Case 2 and Case 3 of Scenario II here. We have $\hat{p_i} \leq n_i$ and $\hat{q_i} \leq r_i$ for $1 \leq i \leq 3$ in both cases. The communication cost for the obtained set of values matches the lower bound, and each term in the lower bound is at least 1; therefore, $1 \leq \frac{n_i r_i}{\hat{p_i}\hat{q_i}} \leq n_i r_i$ for $1 \leq i \leq 3$, $1 \leq \frac{n}{\hat{p}} \leq n$, and $1 \leq \frac{r}{\hat{q}} \leq r$. This implies that $1 \leq \hat{p_i}\hat{q_i} \leq n_i r_i$ for $1 \leq i \leq 3$, $1 \leq \hat{p} \leq n$, and $1 \leq \hat{q} \leq r$. For $1 \leq i \leq 3$, at most, one of $\hat{p_i}$ and $\hat{q_i}$ can be smaller than one. In such a case, we multiply the largest by the smallest (say $\widetilde{p_i} = \hat{p_i} \cdot \hat{q_i}$) and set the smallest to one ($\widetilde{q_i} = 1$) so that their product remains the same ($\widetilde{p_i} \cdot \widetilde{q_i} = \hat{p_i} \cdot \hat{q_i}$). After doing that, the products $\widetilde{p}$ and $\widetilde{q}$ might change. Let $f = \widetilde{q}/\hat{q}$ be the rate of change, and suppose $f > 1$. As $\hat{q} = \widetilde{q}/f \geq 1$, we can factor $f = f_1 \cdot f_2 \cdot f_3$ with $f_i \geq 1$ so that $\hat{q_i} := \widetilde{q_i}/f_i \geq 1$ and $\hat{p_i} := \widetilde{p_i} f_i \geq 1$. We can obtain the factors f_i by the following iterative procedure:

1. for $i = 1 : 3$
2. if $\widetilde{q_i} \geq f$ then $f_i = f$, $f = 1$, $(\hat{p_i}, \hat{q_i}) := (\widetilde{p_i} f_i, \widetilde{q_i}/f_i)$
3. else $f_i = \widetilde{q_i}$, $f = f/f_i$, $(\hat{p_i}, \hat{q_i}) := (\widetilde{p_i} f_i, 1)$

It is straightforward to verify that at the end of this process we have $1 \le \hat{q}_i \le r_i$, and $1 \le \hat{p}_i$. If $f < 1$, the process is applied by exchanging the p's and the q's so that we end up with the inequalities $1 \le \hat{p}_i \le n_i$, and $1 \le \hat{q}_i$.

Now we consider all the cases of both scenarios. It remains to adapt $\hat{p}_i$ and $\hat{q}_i$ such that $\hat{p}_i \le n_i$ and $\hat{q}_i \le r_i$. We can note that due to our particular selections of p_i and q_i in each case, $\nexists i, j \in [3]$ such that $\hat{p}_i > n_i$ and $\hat{q}_j > r_j$. We will use this fact while assessing the additional communication cost. We now obtain $p_1, p_2, p_3, q_1, q_2, q_3$ from $\hat{p}_i, \hat{q}_i$ such that both lower and upper bounds are respected, and $p_1 p_2 p_3 = \hat{p}$ and $q_1 q_2 q_3 = \hat{q}$. The intuition is to maintain the tensor communication terms in the lower bound.

Initially we set $p_i = \hat{p}_i, q_i = \hat{q}_i$ for $1 \le i \le 3$. If $1 \le \hat{q}_i \le r_i$, $1 \le \hat{p}_i$ for $1 \le i \le 3$, and $\hat{p}_l > n_l$ for some index l. We represent the other two indices with j and k. As $\hat{p} \le n$, therefore $\hat{p}_j \le n_j$ and/or $\hat{p}_k \le n_k$. Without loss of generality, we assume that $\hat{p}_k \le n_k$. Now we first update p_l, and then p_j, and in the end, p_k with the following expressions: $p_l := n_l$, $p_j := \min\{n_j, \frac{\hat{p}}{p_k p_l}\}$, $p_k := \frac{\hat{p}}{p_l p_j}$. We note that the product is unchanged by these updates as $p_k p_l p_j = \hat{p}$. The same update can be done to q_i's if $1 \le \hat{p}_i \le n_i$, $1 \le \hat{q}_i$ for $1 \le i \le 3$, and $\hat{q}_l > r_l$ for some l.

Now we assess how much additional communication is required for the matrices. If $\nexists i \in [3]$ such that $\hat{p}_i > n_i$ or $\hat{q}_i > r_i$, then $\sum_{i\in[3]} \frac{n_i r_i}{p_i q_i} = \sum_{i\in[3]} \frac{n_i r_i}{\hat{p}_i \hat{q}_i}$. We can note that due to our particular selections of $\hat{p}_i$ and $\hat{q}_i$, $\nexists i, j \in [3]$ such that $\hat{p}_i > n_i$ and $\hat{q}_j > r_j$. Suppose $\exists i \in [3]$ such that $\hat{p}_i > n_i$, then $\hat{p} > 2$ and

$$
\begin{aligned}
\sum_{i\in[3]} \frac{n_i r_i}{p_i q_i} &\le \sum_{i\in[3]} \max\left\{\frac{n_i r_i}{\hat{p}_i \hat{q}_i}, \frac{r_i}{\hat{q}_i}\right\} && q_i = \hat{q}_i, \text{ and } p_i \ge \hat{p}_i \text{ or } p_i = n_i \\
&= \sum_{i\in[3]} \left(\frac{n_i r_i}{\hat{p}_i \hat{q}_i} + \frac{r_i}{\hat{q}_i} - \min\left\{\frac{n_i r_i}{\hat{p}_i \hat{q}_i}, \frac{r_i}{\hat{q}_i}\right\}\right) && \max\{a,b\} = a + b - \min\{a,b\} \\
&< \sum_{i\in[3]} \left(\frac{n_i r_i}{\hat{p}_i \hat{q}_i} + \frac{r_i}{\hat{q}_i}\right) - 2 && \hat{p}_i \hat{q}_i \le n_i r_i \text{ and } \hat{q}_i \le r_i \\
&\le \sum_{i\in[3]} \frac{n_i r_i}{\hat{p}_i \hat{q}_i} + \frac{r}{\hat{q}} && \forall a_i \ge 1, a_1 + a_2 + a_3\text{-}2 \le a_1 a_2 a_3 \\
&< \sum_{i\in[3]} \frac{n_i r_i}{\hat{p}_i \hat{q}_i} + 2\left(\frac{r}{\hat{q}} - \frac{r}{\hat{p}\hat{q}}\right) \\
&= \sum_{i\in[3]} \frac{n_i r_i}{\hat{p}_i \hat{q}_i} + 2\left(\frac{r}{\hat{q}} - \frac{r}{P}\right).
\end{aligned}
$$

We have used $\forall a_i \ge 1, a_1 + a_2 + a_3 - 2 \le a_1 a_2 a_3$ in the fourth line above. It can be proved in the following way: $\forall a_i \ge 1, a_1 a_2 a_3 = (1 + a_1 - 1)(1 + a_2 - 1)(1 + a_3 - 1) \ge 1 + (a_1 - 1) + (a_2 - 1) + (a_3 - 1) = a_1 + a_2 + a_3 - 2$.

Similarly, if $\hat{q}_i > r_i$ for some i then $\sum_{i\in[3]} \frac{n_i r_i}{p_i q_i}$ is bounded by $\sum_{i\in[3]} \max\{\frac{n_i r_i}{\hat{p}_i \hat{q}_i}, \frac{n_i}{\hat{p}_i}\}$ and we can obtain $\sum_{i\in[3]} \frac{n_i r_i}{p_i q_i} < \sum_{i\in[3]} \frac{n_i r_i}{\hat{p}_i \hat{q}_i} + 2\left(\frac{n}{\hat{p}} - \frac{n}{P}\right)$.

Therefore, in all situations, $\sum_{i\in[3]} \frac{n_i r_i}{p_i q_i} + \frac{r}{q} + \frac{n}{p} - O \le 3(\sum_{i\in[3]} \frac{n_i r_i}{\hat{p}_i \hat{q}_i} + \frac{r}{\hat{q}} + \frac{n}{\hat{p}} - O) = 3LB$. ∎

6. Simulated evaluation. In this section, we verify our theoretical claims on particular sets of 3D tensor dimensions with a simulated evaluation. We use (5.1) to calculate the communication cost of Algorithm 5.1. In subsection 6.1, we demonstrate that the communication cost of Algorithm 5.1 matches the lower bound of Theorem 4.3, and we provide intuition for relationships among the communication costs of the individual tensors and matrices. In subsection 6.2, we compare the approach of

Algorithm 5.1 for evaluating Multi-TTM with a TTM-in-Sequence approach, demonstrating realistic scenarios when Algorithm 5.1 communicates significantly less data and performs a small amount of extra computation.

Throughout this section, we restrict to cases where all tensor dimensions and numbers of processors are powers of 2. We vary the number of processors P from 2 to $P_{\max} = \min\{n_1 r_1, n_2 r_2, n_3 r_3, n, r\}$, which ensures that each processor owns some data of every tensor and matrix. Some of these P values may be practically unrealistic for a particular experiment. However, we also consider them in simulations to understand general characteristics of the cost functions of both approaches.

We note that Tucker decomposition has been employed on datasets that range into the terabytes [2, 4, 11], justifying consideration of values of n on the order of 2^{40}. Likewise, Multi-TTM has been performed in the context of randomized sketching in cases where the output tensor dimensions are smaller than the ranks of the Tucker approximation [23], justifying consideration of values of n/r on the order of 2^{32}. We expect available parallelism and tensor dataset sizes to continue to grow, so we model the behavior of the various Multi-TTM approaches beyond the limits of the current state of the art.

The costs of Algorithm 5.1 depend on the processor grid, and in these experiments we perform an exhaustive search for the best configuration. We describe in subsection 6.3.1 how to adapt the processor grid selection scheme described in subsection 5.2 to obtain integer-valued processor grid dimensions, and we show that we can obtain nearly optimal configurations without exhaustive search.

6.1. Verifying optimality of Algorithm 5.1. Theorem 5.1 states that Algorithm 5.1 attains the communication lower bound to within a constant factor, and in this section we verify the result in a variety of scenarios. Recall from Theorem 4.3 that the lower bound is $A + B - O$, where

$$A = \begin{cases} n_1 r_1 + n_2 r_2 + \frac{n_3 r_3}{P} & \text{if } P < \frac{n_3 r_3}{n_2 r_2}, \\ n_1 r_1 + 2\left(\frac{n_2 n_3 r_2 r_3}{P}\right)^{\frac{1}{2}} & \text{if } \frac{n_3 r_3}{n_2 r_2} \le P < \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}, \\ 3\left(\frac{nr}{P}\right)^{\frac{1}{3}} & \text{if } \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2} \le P, \end{cases}$$

$$B = \begin{cases} r + \frac{n}{P} & \text{if } P < \frac{n}{r}, \\ 2\left(\frac{nr}{P}\right)^{\frac{1}{2}} & \text{if } \frac{n}{r} \le P. \end{cases}$$

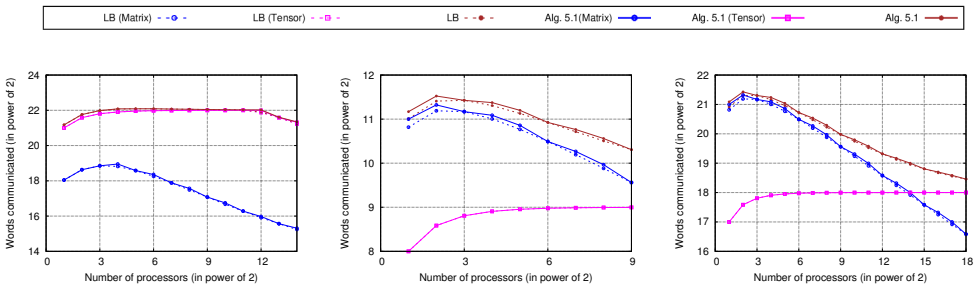
$$O = \frac{n_1 r_1 + n_2 r_2 + n_3 r_3 + r + n}{P}.$$

Here, A corresponds to the matrix entries accessed, B corresponds to the tensor entries accessed or partially computed, and O corresponds to the data owned by a single processor. The costs of Algorithm 5.1 are given by (5.1), which we rewrite here as

$$\frac{n_1 r_1}{p_1 q_1} + \frac{n_2 r_2}{p_2 q_2} + \frac{n_3 r_3}{p_3 q_3} + \frac{n}{p} + \frac{r}{q} - O,$$

where $\{p_i\}$ and $\{q_i\}$ specify the processor grid dimensions. The first three terms correspond to matrix entries and the middle two terms correspond to tensor entries.

Figure 3 shows both components, matrix and tensor communication costs, for three distinct input sizes as we vary the number of processors. In these plots, we show both algorithmic costs (upper bounds) and lower bounds, but they are indistinguishable because the largest differences in overall costs we observe are 9% for



(a) $\{n_1, n_2, n_3, r_1, r_2, r_3\} = \{2^9, 2^{10}, 2^{14}, 2^5, 2^9, 2^8\}$.

(b) $n_1 = n_2 = n_3 = 2^8$, and $r_1 = r_2 = r_3 = 2^3$.

(c) $n_1 = n_2 = n_3 = 2^{15}$, and $r_1 = r_2 = r_3 = 2^6$.

FIG. 3. *Matrix and tensor communication costs in LB and Algorithm* 5.1 *for different configurations. The sum of LB(Matrix) and LB(Tensor) equals to the lower bound LB, and the sum of Algorithm* 5.1 *(matrix) and Algorithm* 5.1 *(tensor) equals to the upper bound (Algorithm* 5.1*). Lower bounds are almost indistinguishable from the corresponding upper bounds.*

Figure 3(a) at $P = 2^{12}$, 12% for Figure 3(b) at $P = 2$, and 13% for Figure 3(c) at $P = 2$, verifying Theorem 4.3 for these scenarios.

In Figure 3(a), the input and output tensors have varying dimensions: the input tensor is $2^9 \times 2^{10} \times 2^{14}$ and the output is $2^5 \times 2^9 \times 2^8$. We choose these dimensions so that all five cases of the values of A and B are represented. For these inputs, the tensor communication cost dominates the matrix communication for all values of P considered. When $P < 2^3$, the first cases for A and B apply, and the algorithm selects a processor grid such that $p_3 = P$, implying that only one tensor and two matrices are communicated. In this case, both expressions simplify to $(r + n_1 r_1 + n_2 r_2)(1 - 1/P)$, which is why we see initial increase as P increases at the left end of the plot. For $2^3 \leq P < 2^{11}$, the second case for A and the first case for B apply, and the algorithm selects a processor grid with $p_2 > 1$ and $p_3 > 1$. Here, the matrix communication begins to decrease, but it is dominated by the tensor communication, which is maintained at $r(1 - 1/P)$. For $2^{11} \leq P$, the second case for B applies, and we see that tensor communication decreases as P increases (proportional to $P^{-1/2}$ as we see from the lower bound). In this regime, the algorithm is selecting grids with both $p > 1$ and $q > 1$ and communicating both tensors. Another transition occurs at $P = 2^{13}$, switching from the second to third case of A, but this change in matrix cost has a negligible effect.

Figure 3(b) demonstrates a scenario where the matrix costs dominate the tensor costs: the input tensor is cubical with dimension 2^8 and the output tensor is cubical with dimension 2^3. Here we scale P only up to 2^9, the number of entries in the output tensor. Because the tensors are cubical, the lower bounds simplify as in Corollary 4.4, and the algorithm chooses processor grids that are as cubical as possible. For all values of P in this experiment, the third case of A and the first case of B apply, and the algorithm selects $p_1 \approx p_2 \approx p_3$ and $q = 1$. We see that the overall cost is deceasing proportional to $P^{-1/3}$ until the tensor communication cost starts to contribute more significantly.

Figure 3(c) considers cubical tensors with larger dimensions to show a more general pattern. For tensor dimensions $n_i = 2^{15}$ and $r_i = 2^6$, we observe a transition point where tensor communication overtakes matrix communication. Similar to the case of Figure 3(b), matrix costs dominate for small P and scale like $P^{-1/3}$. However, for $P \geq 2^{14}$, the tensor costs dominate the matrix costs and therefore communication costs scale less efficiently. We emphasize that for all three of these experiments, the algorithmic costs match the lower bounds nearly exactly for all values of P.

6.2. Comparing Algorithm 5.1 with TTM-in-Sequence. As mentioned previously, a Multi-TTM computation may be performed as a sequence of TTM operations. In this TTM-in-Sequence approach, a single matrix is multiplied with the tensor and an intermediate tensor is computed and stored. For each remaining matrix, single-matrix TTMs are performed in sequence until the final result is computed. This approach can reduce the number of arithmetic operations compared to direct evaluation of atomic expression given in Definition 3.1. The computational cost depends (often significantly) on the order of the TTMs performed. The TTM-in-Sequence approach is parallelized in the TuckerMPI library [4]. We note that Theorem 4.3 does not apply to this parallelization, as it violates the parallel atomicity assumption.

In this section, we provide a comparison between Algorithm 5.1 and the TTM-in-Sequence approach to show that our approach can significantly reduce communication in important scenarios without performing too much extra computation. In particular, we observe the greatest benefit of Algorithm 5.1 when r is very small relative to n (or vice versa) and P is small relative to the ratio of n and r. These scenarios occur in the context of computing and using Tucker decompositions for highly compressible tensors that exhibit small multilinear ranks.

The computational cost of TuckerMPI's algorithm with cubical tensors is the same for all possible orderings of the TTMs. In our comparison, we consider that the TTMs are performed in increasing mode order. While no single communication lower bound exists for parallel TTM-in-Sequence algorithms, we show in subsection 6.3.2 that TuckerMPI's algorithm attains nearly the same cost as tight matrix multiplication lower bounds [1] applied to each TTM it chooses to perform. Thus, no other parallelization of the TTM-in-Sequence approach can reduce communication without breaking the assumptions of the matrix multiplication lower bounds (e.g., using fast matrix multiplication).

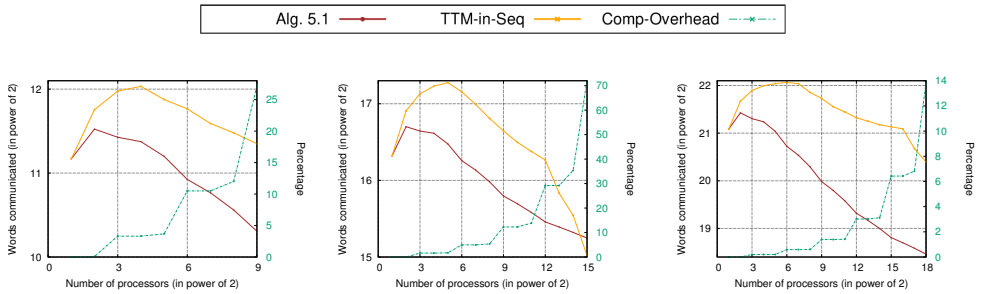
The TuckerMPI parallelization uses a 3D logical processor grid with dimensions $\tilde{p_1} \times \tilde{p_2} \times \tilde{p_3}$. When the TTMs are performed in increasing mode order, the overall communication cost of their algorithm is

$$\frac{r_1 n_2 n_3}{\tilde{p_2}\tilde{p_3}} + \frac{n_1 r_1}{\tilde{p_1}} + \frac{r_1 r_2 n_3}{\tilde{p_1}\tilde{p_3}} + \frac{n_2 r_2}{\tilde{p_2}} + \frac{r_1 r_2 r_3}{\tilde{p_1}\tilde{p_2}} + \frac{n_3 r_3}{\tilde{p_3}} - \frac{r_1 n_2 n_3 + r_1 r_2 n_3 + r_1 r_2 r_3 + n_1 r_1 + n_2 r_2 + n_3 r_3}{P}, \tag{6.1}$$

as specified in [4, section 6.3], though we include the cost of communicating the matrices (their analysis assumes the matrices are already redundantly distributed). We use exhaustive search to determine the processor grid that minimizes the cost of (6.1) in our comparisons.

6.2.1. Communication cost. To compare communications costs, we perform four experiments involving cubical tensors. The first three simulated evaluations consider strong scaling and are presented in Figure 4. Two of these experiments use the same tensor dimensions as the two cubical examples in Figure 3. The first experiment involves an input tensor of dimension $n_i = 2^8$ and output dimension $r_i = 2^3$ (Figure 4(a)), the second has dimensions $n_i = 2^{11}$ and $r_i = 2^5$ (Figure 4(b)), and the third has the largest dimensions, $n_i = 2^{15}$ and $r_i = 2^6$ (Figure 4(c)).

Figure 4(a) shows that Algorithm 5.1 performs less communication than TTM-in-Sequence for $P \leq 2^9 < n/r$. The largest communication reduction occurs at $P = 2^9$ and is approximately $2\times$. In the second experiment, we see cases where TTM-in-Sequence performs less communication than Algorithm 5.1 and in fact beats the lower bound of Theorem 4.3 (which is possible because it breaks the atomicity assumption).



(a) $n_i = 2^8$, $r_i = 2^3$. (b) $n_i = 2^{11}$, $r_i = 2^5$. (c) $n_i = 2^{15}$, $r_i = 2^6$.

FIG. 4. *Communication cost comparison of Algorithm* 5.1 *and TTM-in-Sequence* [4]. *Comp-Overhead shows the percentage of computational overhead of Algorithm* 5.1 *with respect to the TTM-in-Sequence approach.*

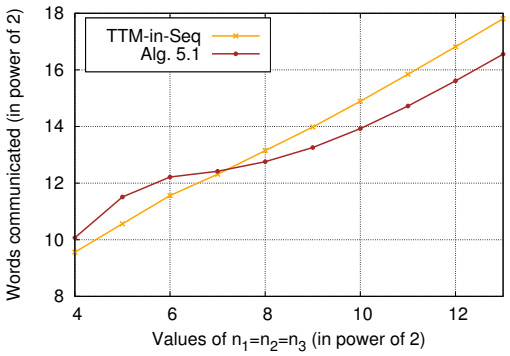


FIG. 5. *Comparison of Algorithm* 5.1 *and the TTM-in-Sequence approach for fixed* $r_1 = r_2 = r_3 = 2^4$ *and* $P = 2^6$.

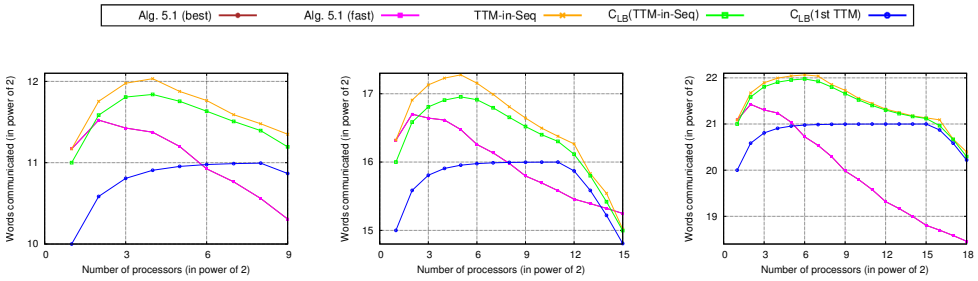
Algorithm 5.1 is more communication efficient for $P \leq 2^{14}$, achieving a speedup of approximately $2\times$, but communicates more for larger P. In the third experiment with larger tensors, Figure 4(c) demonstrates similar qualitative behavior to the first, with Algorithm 5.1 outperforming TTM-in-Sequence and a maximum communication reduction of approximately $5\times$ at $P = 2^{16}$.

In the fourth experiment, with results shown in Figure 5, we fix the output tensor dimension $r_i = 2^4$ and the number of processors $P = 2^6$ and vary the input tensor dimension n_i. We observe that for $2^4 \leq n_i < 2^8$, the TTM-in-Sequence approach communicates less data than Algorithm 5.1. For $n_i \geq 2^8$, Algorithm 5.1 communicates less data, and the factor of improvement is maintained at approximately $2\times$ as n_i scales up.

6.2.2. Computation cost. Assuming TuckerMPI uses increasing mode order, the parallel computational cost is

$$2 \cdot \frac{r_1 n_1 n_2 n_3 + r_1 r_2 n_2 n_3 + r_1 r_2 r_3 n_3}{P} = 2\left(\frac{r^{1/3} n}{P} + \frac{r^{2/3} n^{2/3}}{P} + \frac{r n^{1/3}}{P}\right),$$

where the right-hand side is simplified under the assumption of cubical tensors. In these experiments where $n \gg r$, Algorithm 5.1 selects a processor grid such that $q = 1$ and $p_1 \approx p_2 \approx p_3$. In this case the computation cost given in subsection 5.1 simplifies to



(a) $n_i = 2^8$, $r_i = 2^3$. (b) $n_i = 2^{11}$, $r_i = 2^5$. (c) $n_i = 2^{15}$, $r_i = 2^6$.

FIG. 6. *Communication cost comparison of Algorithm* 5.1 *using best processor grid against fast method and of the TTM-in-Sequence approach implemented by TuckerMPI against the lower bounds. Algorithm* 5.1 *(fast) and Algorithm* 5.1 *(best) are the same for all the configurations.*

$$2\left(\frac{r^{1/3}n}{P} + \frac{r^{2/3}n^{2/3}}{P^{2/3}} + \frac{rn^{1/3}}{P^{1/3}}\right).$$

Note that this cost is much smaller than $4nr/P$, the cost of evaluating (3.1) directly with computational load balance, and it is achieved by performing local computation using a TTM-in-Sequence approach.

While the starting terms of the two computational cost expressions match, we observe greater computational cost from Algorithm 5.1 in the second and third terms. These terms are lower order when $P \ll n/r$, in which case the extra computational cost of Algorithm 5.1 is often negligible. This is also validated by Figure 4 for the first three experiments. When $P = n/r$, the ratio of computational costs of Algorithm 5.1 and the TTM-in-Sequence approach is no more than $3\times$.

In the first three experiments, when our approach reduces communication, the extra computational costs were at most 27%, 35%, and 13%, respectively. The extra computation required for the greatest reductions in communication in those experiments were 27%, 5%, and 6%. For the fourth experiment, the extra computation is approximately 23% at $n_i = 2^8$, where Algorithm 5.1 provides communication reduction and decreases as n_i increases.

In all these experiments, we see that when Algorithm 5.1 provides a reduction in communication costs, the extra computational costs often remain negligible.

6.3. Details for evaluation of our algorithm. Here we provide more details for the simulated evaluation of our algorithm and its comparison to the TTM-in-Sequence approach. The analysis of the communication optimality of Algorithm 5.1 did not consider integrality constraints on the processor grid dimensions. The simulated evaluation in the previous subsection considered all possible processor grid configurations using exhaustive search; we explain in subsection 6.3.1 a more efficient process for determining an optimal grid when P is a power of two. In the previous subsection, we also compare Algorithm 5.1 against an implementation of the TTM-in-Sequence approach as implemented by TuckerMPI [4]. We argue in subsection 6.3.2 that this implementation is nearly communication optimal given the computation that it performs, validating our comparison against it. Figure 6 presents results relevant to both subsections 6.3.1 and 6.3.2.

6.3.1. Obtaining integral processor grids for Algorithm 5.1. In order to determine the communication cost of Algorithm 5.1, one must determine the processor grid. Obtaining p_i and q_i from the procedure in section 5 may yield noninteger values. The following procedure allows us to convert these to integers under our assumption that all parameters are powers of 2. Recall that we consider $P = pq$ with $p = p_1p_2p_3$ and $q = q_1q_2q_3$.

If $\lfloor \log_2(p) + 0.5 \rfloor = \lfloor \log_2(p) \rfloor$, then we set $p = 2^{\lfloor \log_2(p) \rfloor}$; otherwise we set $p = 2^{\lceil \log_2(p) \rceil}$, distributing the modification evenly between p_1, p_2, and p_3. Now, we keep $p = p_1p_2p_3$ constant and convert each p_i to an integer. We set $p_1 = 2^{\lfloor \log_2(p_1)+0.5 \rfloor}$, distributing the changes evenly among p_2 and p_3. To see that our new value of p_1 must still be smaller than n_1, we note that our original p_1 was less than n_1 which is a power of 2 by our assumption. If we increased p in our first step, then distributing the modifications evenly between p_1, p_2, and p_3 increased them by at most $2^{1/6}$. Thus $p_1 \leq n_1$ will imply that $\lfloor \log_2(p_1 \cdot 2^{1/6}) + 0.5 \rfloor \leq \log_2(n_1)$. Note that this most recent modification to p_1 changes p_2 and p_3. Then, we set $p_2 = 2^{\lfloor \log_2(p_2)+0.5 \rfloor}$ and adapt p_3 accordingly. A similar argument to what is used for p_1 will show that p_2 and p_3 are also not larger than their corresponding dimensions. Having completed our work on the processor dimensions associated with the first tensor, we set $q = \frac{P}{p}$, distributing the changes evenly among the q_i, then force each q_i to be an integer following the same procedure as for the p_i.

We denote the communication cost of Algorithm 5.1 for the grid determined using this method by Algorithm 5.1 (fast) and the communication cost using exhaustive search by Algorithm 5.1 (best). We note that this procedure can increase the total number of accessed elements of any variable at most 4 times, but we see in Figure 6 that the communication costs of both procedures are exactly the same for the examples we consider. These problems match those presented in Figure 4.

6.3.2. TTM-in-Sequence Lower Bounds. Here we discuss communication lower bounds for the TTM-in-Sequence approach with cubical tensors. There has not been any proven bound for this approach other than individual bounds for each TTM (a single matrix multiply) computation, assuming the sequence of TTMs has been specified. The sum of individual bounds provides a communication lower bound for this approach. We obtain the tightest (and obtainable) lower bound for each TTM from [1], which depends on the relative matrix dimensions and number of processors, and represent the sum by C_{LB}*(TTM-in-Seq).* We also note that C_{LB}*(TTM-in-Seq)* may not be always attainable as data distributions for two successive TTMs may be noncompatible and require extra communication. When the input tensor dimensions are much larger than the output tensor dimensions, most of the computation and communication occur in the first TTM, so we also consider the communication lower bound of only that matrix multiplication, which also provides a valid lower bound for the entire TTM-in-Sequence computation. Recall that we obtain the algorithmic cost of TTM-in-Sequence by exhaustively searching for the best processor grid configuration given the communication costs specified by (6.1). Figure 6 shows a comparison of *TTM-in-Seq* and C_{LB}*(TTM-in-Seq)* for the tensor dimensions presented in Figure 4. We can see that the communication costs of *TTM-in-Seq* are very close to C_{LB}*(TTM-in-Seq),* the largest differences are 14% for Figure 6(a) at $P = 2^4$, 25% for Figure 6(b) at $P = 2^5$, and 9% for Figure 6(c) at $P = 2^{16}$. Comparing C_{LB}*(1st TTM)* and C_{LB}*(TTM-in-Seq),* we see that for these examples at least half the communication of the entire TTM-in-Sequence is required by the first TTM, and it is completely dominated by the first TTM when P is large.

7. Lower Bounds of general Multi-TTM. We present our lower bound results for d-dimensional tensors in this section. Similar to the 3D lower bound proof, we consider a single processor that performs $1/P$th of the computation and owns at most $1/P$th of the data. We again seek to minimize the number of elements of the matrices and tensors that the processor must access or partially compute in order to execute its computation subject to the constraints of the structure of Multi-TTM by solving two independent problems, one for the matrix data and one for the tensor data.

7.1. General constrained optimization problems. Here we present a generalization of Corollary 4.1 for d dimensions. As before, this corollary is a direct result of Theorem 3.4. Recall the notation $N_i = \prod_{j=d-i+1}^{d} n_j$ and $R_i = \prod_{j=d-i+1}^{d} r_i$.

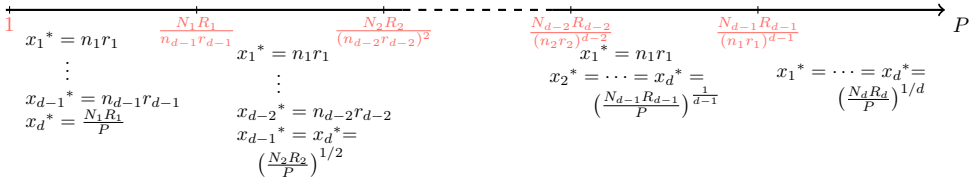
COROLLARY 7.1. *Consider the following optimization problem:*

$$\min_{\mathbf{x}} \sum_{i\in[d]} x_i$$

such that

$$\frac{nr}{P} \le \prod_{i\in[d]} x_i \quad \text{and} \quad 0 \le x_i \le n_i r_i \quad \forall \quad 1 \le i \le d,$$

where $n_i, r_i, P \ge 1$ *and* $n_i r_i \le n_{i+1} r_{i+1}$. *The optimal solution* $\mathbf{x} = [x_1{}^* \ \cdots \ x_d{}^*]$ *depends on the values of constants, yielding* d *cases.*



- *If* $P < \frac{N_1R_1}{n_{d-1}r_{d-1}}$, *then*

$$x_j{}^* = n_j r_j \quad \text{for} \quad 1 \le j \le d-1 \quad \text{and} \quad x_d{}^* = \frac{N_1R_1}{P}.$$

- *If* $\frac{N_{i-1}R_{i-1}}{(n_{d+1-i}r_{d+1-i})^{i-1}} \le P < \frac{N_iR_i}{(n_{d-i}r_{d-i})^i}$ *for some* $i = 2, \ldots, d-1$, *then*

$$x_j{}^* = n_j r_j \quad \text{for} \quad 1 \le j \le d-i \quad \text{and} \quad x_{d+1-i}{}^* = \cdots = x_d{}^* = (N_iR_i/P)^{1/i}.$$

- *If* $\frac{N_{d-1}R_{d-1}}{(n_1r_1)^{d-1}} \le P$, *then*

$$x_1{}^* = \cdots = x_d{}^* = (N_dR_d/P)^{1/d}.$$

7.2. Communication Lower Bounds. We now present the lower bounds for the general Multi-TTM computation. We prove this by applying Corollaries 4.2 and 7.1 and extending the arguments of Theorem 4.3 in a straightforward way (though with more complicated notation).

THEOREM 7.2. *Any computationally load-balanced atomic Multi-TTM algorithm that starts and ends with one copy of the data distributed across processors and involves* d*-dimensional tensors with dimensions* $n_1, n_2, \ldots, n_d$ *and* $r_1, r_2, \ldots, r_d$ *performs at least* $A + B - (\frac{n}{P} + \frac{r}{P} + \sum_{j=1}^{d} \frac{n_j r_j}{P})$ *sends or receives, where*

$$A = \begin{cases} \sum_{j=1}^{d\text{-}1} n_j r_j + \frac{N_1 R_1}{P} & \text{if } P < \frac{N_1 R_1}{n_{d\text{-}1} r_{d\text{-}1}}, \\ \sum_{j=1}^{(d\text{-}i)} n_j r_j + i \left(\frac{N_i R_i}{P}\right)^{\frac{1}{i}} & \text{if } \frac{N_{i\text{-}1} R_{i\text{-}1}}{(n_{d+1\text{-}i} r_{d+1\text{-}i})^{i\text{-}1}} \le P < \frac{N_i R_i}{(n_{d\text{-}i} r_{d\text{-}i})^i}, \\ & \text{for some } 2 \le i \le d-1, \\ d \left(\frac{N_d R_d}{P}\right)^{\frac{1}{d}} & \text{if } \frac{N_{d\text{-}1} R_{d\text{-}1}}{(n_1 r_1)^{d\text{-}1}} \le P, \end{cases}$$

$$B = \begin{cases} r + \frac{n}{P} & \text{if } P < \frac{n}{r}, \\ 2 \left(\frac{nr}{P}\right)^{\frac{1}{2}} & \text{if } \frac{n}{r} \le P. \end{cases}$$

Proof. Let F be the set of loop indices associated with the $(d+1)$-ary multiplications performed by a processor. As we assumed the algorithm is computationally load balanced, $|F| = nr/P$. We define $\phi_{\mathcal{X}}(F)$, $\phi_{\mathcal{Y}}(F)$, and $\phi_j(F)$ to be the projections of F onto the indices of the arrays $\mathcal{X}, \mathcal{Y}$, and $\mathbf{A}^{(j)}$ for $1 \le j \le d$ which correspond to the elements of the arrays that must be accessed or partially computed by the processor.

We use Lemma 3.3 to obtain a lower bound on the number of array elements that must be accessed or partially computed by the processor. The matrix corresponding to the projections above is given by

$$\mathbf{\Delta} = \begin{bmatrix} \mathbf{I}_{d\times d} & \mathbf{1}_d & \mathbf{0}_d \\ \mathbf{I}_{d\times d} & \mathbf{0}_d & \mathbf{1}_d \end{bmatrix}.$$

Here $\mathbf{1}_d$ and $\mathbf{0}_d$ denote the d-dimensional vectors of all ones and zeros, respectively, and $\mathbf{I}_{d\times d}$ denotes the $d \times d$ identity matrix. As before, we define

$$\mathcal{C} = \left\{ \mathbf{s} = [s_1 \ \cdots \ s_{d+2}]^{\mathsf{T}} : 0 \le s_i \le 1 \text{ for } i = 1, 2, \ldots, d+2 \text{ and } \mathbf{\Delta} \cdot \mathbf{s} \ge \mathbf{1} \right\}.$$

We recall that $\mathbf{1}$ represents a vector of all ones. As in the proof of Theorem 4.3, $\mathbf{\Delta}$ is not full rank, so we again consider each vector $\mathbf{v} \in \mathcal{C}$ such that $\mathbf{\Delta} \cdot \mathbf{v} = \mathbf{1}$. Such a vector $\mathbf{v}$ is of the form $\begin{bmatrix} a & \cdots & a & 1-a & 1-a \end{bmatrix}$, where $0 \le a \le 1$. Thus, we obtain

$$\frac{nr}{P} \le \left(\prod_{j \in [d]} |\phi_j(F)| \right)^a \left(|\phi_{\mathcal{X}}(F)| |\phi_{\mathcal{Y}}(F)| \right)^{1\text{-}a}.$$

Similar to the 3D case, the above constraint is equivalent to $\frac{nr}{P} \le \prod_{j\in[d]} |\phi_j(F)|$ and $\frac{nr}{P} \le |\phi_{\mathcal{X}}(F)||\phi_{\mathcal{Y}}(F)|$.

Clearly a projection onto an array cannot be larger than the array itself, thus $|\phi_{\mathcal{X}}(F)| \le n$, $|\phi_{\mathcal{Y}}(F)| \le r$, and $|\phi_j(F)| \le n_j r_j$ for $1 \le j \le d$.

As the constraints related to the projections of matrices and tensors are disjoint, we solve them separately and then sum the results to get a lower bound on the number of elements that must be accessed or partially computed by the processor. We obtain a lower bound on A, the number of relevant elements of the matrices by using Corollary 7.1, and a lower bound on B, the number of relevant elements of the tensors by using Corollary 4.2. By summing both, we get the positive terms of the lower bound.

To bound the sends or receives, we consider how much data the processor could have had at the beginning or at the end of the computation. Assuming there is exactly one copy of the data at the beginning and at the end of the computation, there must exist a processor which owns at most $1/P$ of the elements of the arrays at the beginning or at the end of the computation. By employing the previous analysis, this processor must access or partially compute $A + B$ elements of the arrays, but can only own $\frac{n}{P} + \frac{r}{P} + \sum_{j\in[d]} \frac{n_j r_j}{P}$ elements of the arrays. Thus it must perform the specified amount of sends or receives. ∎

Algorithm 8.1. Parallel atomic d-dimensional Multi-TTM.

Require: $\mathcal{X}$, $\mathbf{A}^{(1)}$, ..., $\mathbf{A}^{(d)}$, $p_1 \times \cdots \times p_d \times q_1 \times \cdots \times q_d$ logical processor grid
Ensure: $\mathcal{Y}$ such that $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)\mathsf{T}} \cdots \times_d \mathbf{A}^{(d)\mathsf{T}}$
1: $(p'_1, \ldots, p'_d, q'_1, \ldots, q'_d)$ is my processor id
2: //All-gather input tensor $\mathcal{X}$
3: $\mathcal{X}_{p'_1 \cdots p'_d}$ = All-Gather($\mathcal{X}$, $(p'_1, \ldots, p'_d, *, \ldots, *)$)
4: //All-gather all input matrices
5: **for** $i = 1, \ldots, d$ **do**
6: $\quad \mathbf{A}^{(i)}_{p'_i q'_i}$ = All-Gather($\mathbf{A}^{(i)}$, $(*, \ldots, *, p'_i, *, \ldots, *, q'_i, *)$)
7: **end for**
8: //Perform local computations in a temporary tensor $\mathcal{T}$
9: $\mathcal{T}$ = Local-Multi-TTM($\mathcal{X}_{p'_1 \cdots p'_d}$, $\mathbf{A}^{(1)}_{p'_1 q'_1}$, ..., $\mathbf{A}^{(d)}_{p'_d q'_d}$)
10: //Reduce-scatter the output tensor in $\mathcal{Y}_{q'_1 \cdots q'_d}$
11: Reduce-Scatter($\mathcal{Y}_{q'_1 \cdots q'_d}$, $\mathcal{T}$, $(*, \ldots, *, q'_1, \ldots, q'_d)$)

8. Parallel algorithm for general Multi-TTM. We present a parallel algorithm to compute d-dimensional Multi-TTM in Algorithm 8.1, which is analogous to Algorithm 5.1. We organize P processors into a $2d$-dimensional logical processor grid with dimensions $p_1 \times \cdots \times p_d \times q_1 \times \cdots \times q_d$. As before, we consider that $\forall i \in [d]$, p_i and q_i evenly divide n_i and r_i, respectively. A processor coordinate is represented as $(p'_1, \ldots, p'_d, q'_1, \ldots, q'_d)$, where $\forall i \in [d]$, $1 \le p'_i \le p_i$ and $1 \le q'_i \le q_i$.

Here we discuss our data distribution model for Algorithm 8.1, which is similar to that of Algorithm 5.1. $\mathcal{X}_{p'_1 \cdots p'_d}$ and $\mathcal{Y}_{q'_1 \cdots q'_d}$ denote the subtensors of $\mathcal{X}$ and $\mathcal{Y}$ owned by processors $(p'_1, \ldots, p'_d, *, \ldots, *)$ and $(*, \ldots, *, q'_1, \ldots, q'_d)$, respectively. $\mathbf{A}^{(i)}_{p'_i q'_i}$ denotes the submatrix of $\mathbf{A}^{(i)}$ owned by processors $(*, \ldots, *, p'_i, *, \ldots, *, q'_i, *, \ldots, *)$. We impose that there is one copy of data in the system at the beginning and the end of the computation, and each subarray is distributed evenly among the set of processors which own the data.

When Algorithm 8.1 completes, $\mathcal{Y}_{q'_1 \cdots q'_d}$ is distributed evenly among processors $(*, \ldots, *, q'_1, \ldots, q'_d)$. We recall that $\prod_{i=1}^{d} p_i$ and $\prod_{i=1}^{d} q_i$ are denoted by p and q, respectively.

8.1. Cost analysis. Now we analyze computation and communication costs of the algorithm. As before, the local Multi-TTM computation in line 9 can be performed as a sequence of TTM operations to mininimize the number of arithmetic operations. Assuming the TTM operations are performed in their order, first with $\mathbf{A}^{(1)}$, then with $\mathbf{A}^{(2)}$, and so on until the last is performed with $\mathbf{A}^{(d)}$, then each processor performs $\sum_{k=1}^{d} (2 \prod_{i=1}^{k} \frac{r_i}{q_i} \prod_{j=k}^{d} \frac{n_j}{p_j})$ operations. In line 11, each processor also performs $(1 - \frac{q}{P}) \frac{r}{q}$ computations due to the Reduce-Scatter operation.

Communication occurs only in All-Gather and Reduce-Scatter collectives in lines 3, 6, and 11. Line 3 specifies p All-Gathers over disjoint sets of $\frac{P}{p}$ processors, line 6 specifies $p_i q_i$ All-Gathers over disjoint sets of $\frac{P}{p_i q_i}$ processors in the ith loop iteration, and line 11 specifies q Reduce-Scatters over disjoint sets of $\frac{P}{q}$ processors. Each processor is involved in one All-Gather involving the input tensor, d All-Gathers involving input matrices, and one Reduce-Scatter involving the output tensor.

As before, we assume bandwidth and latency optimal algorithms are used for the All-Gather and Reduce-Scatter collectives. Hence the bandwidth costs of the All-Gather operations are $(1-\frac{p}{P})\frac{n}{p}$ for line 3 and $\sum_{i=1}^{d}(1-\frac{p_iq_i}{P})\frac{n_ir_i}{p_iq_i}$ for the d iterations of line 6. The bandwidth cost of the Reduce-Scatter operation in line 11 is $(1-\frac{q}{P})\frac{r}{q}$. Hence the overall bandwidth cost of Algorithm 8.1 along the critical path is $\frac{n}{p}+\frac{r}{q}+\sum_{i=1}^{d}\frac{n_ir_i}{p_iq_i}-(\frac{n+r+\sum_{i=1}^{d}n_ir_i}{P})$. The latency costs are $\log_2(\frac{P}{p})$ and $\log_2(\frac{P}{q})$ for lines 3 and 11, respectively, and $\sum_{i=1}^{d}\log_2(\frac{P}{p_iq_i})$ for the d iterations of line 6. Thus the overall latency cost of Algorithm 8.1 along the critical path is $\log_2(\frac{P}{p})+\sum_{i=1}^{d}\log_2(\frac{P}{p_iq_i})+\log_2(\frac{P}{q})=d\log_2(P)$.

The existence of p_i, q_i with $1 \le p_i \le n_i, 1 \le q_i \le r_i$ for $i = 1,\dots,d$ such that Algorithm 8.1 is communication optimal to within a constant factor is shown by an extension of the arguments of Theorem 5.1 [13].

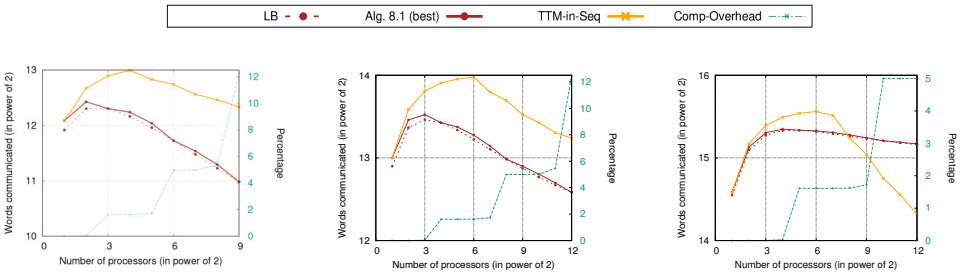
8.2. Simulated evaluation. Similar to section 6, we compare communication costs of our algorithm and a TTM-in-Sequence approach implemented in the TuckerMPI library. We again restrict to cases where all dimensions are powers of 2, and vary the number of processors P from 2 to $P_{\max}$ in multiples of 2, where $P_{\max}=\min\{n_1r_1,\dots,n_dr_d,n,r\}$.

Like section 6, we consider n and n/r on the order of 2^{40} and 2^{32}, respectively. We look at all possible processor grid dimensions and represent the minimum communication costs of our algorithm and TuckerMPI algorithm by Algorithm 8.1 *(best)* and *TTM-in-Seq,* respectively. The TTM-in-Sequence approach described in [4] organizes P in a d-dimensional $\tilde{p_1}\times\cdots\times\tilde{p_d}$ logical processor grid. Assuming TTMs are performed in increasing mode order, the overall communication cost of this algorithm is

$$\begin{aligned}&\frac{r_1n_2\cdots n_d}{\frac{P}{\tilde{p_1}}}+\frac{r_1r_2n_3\cdots n_d}{\frac{P}{\tilde{p_2}}}+\cdots+\frac{r_1r_2\cdots r_d}{\frac{P}{\tilde{p_d}}}\\&\quad-\frac{r_1n_2\cdots n_d+r_1r_2n_3\cdots n_d+\cdots+r_1r_2\cdots r_d}{P}\\&\quad+\frac{n_1r_1}{\tilde{p_1}}+\cdots+\frac{n_dr_d}{\tilde{p_d}}-\frac{n_1r_1+\cdots+n_dr_d}{P}.\end{aligned}$$

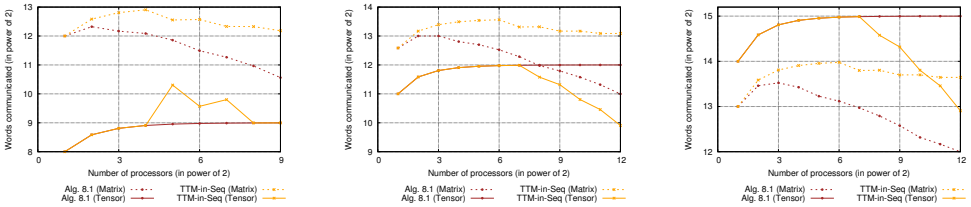
The first two lines correspond to tensor communication and the third line corresponds to matrix communication. As mentioned earlier, the TTM-in-Sequence approach forms a tensor after each TTM computation. Each term of the first line corresponds to the number of entries of such a tensor partially computed by a processor in TuckerMPI.

We again look at cases where the input tensors are large and the output tensors are small. Figure 7 shows comparison of Algorithm 8.1 *(best)* and *TTM-in-Seq* with our communication lower bounds for 3/4/5-dimensional Multi-TTM computations. For $P=2$, both approaches perform the same amount of communication. After that, the total number of accessed elements in both approaches decreases; however, the rate of owned elements decreases at the faster rate. Hence we see slight increase in both curves. This behavior continues roughly till $2^{n_i-r_i}$ processors for *TTM-in-Seq* curve. In this region, the TTM-in-Sequence approach selects $\tilde{p_1}=\cdots=\tilde{p_{d-1}}=1$ and $\tilde{p_d}=P$, and our algorithm selects $p_1\approx\cdots\approx p_d$ and $q_1=\cdots=q_d=1$. These processor grid dimensions result in the same tensor communication cost for both approaches. However our approach reduces matrix communication cost roughly $(1-\frac{1}{d})P^{\frac{1}{d}}$ times; hence it is better than the TTM-in-Sequence approach. Figure 8 shows the distribution of matrix and tensor communication costs in both approaches. In general, our approach significantly minimizes the matrix communication costs in all the plots



(a) $n_1 = n_2 = n_3 = 2^9, r_1 = r_2 = r_3 = 2^3$ (b) $n_1 = n_2 = n_3 = n_4 = 2^9, r_1 = r_2 = r_3 = r_4 = 2^3$ (c) $n_1 = \cdots = n_5 = 2^9, r_1 = \cdots = r_5 = 2^3$

FIG. 7. *Communication cost comparison of Algorithm* 8.1 *and the TTM-in-Sequence approach implemented by the TuckerMPI library. Note that LB is a communication lower bound for atomic Multi-TTM algorithms, not for the TTM-in-Sequence approach. Communication cost of our approach (Algorithm* 8.1 *(best)) is very close to LB.*



(a) $n_1 = n_2 = n_3 = 2^9, r_1 = r_2 = r_3 = 2^3$ (b) $n_1 = n_2 = n_3 = n_4 = 2^9, r_1 = r_2 = r_3 = r_4 = 2^3$ (c) $n_1 = \cdots = n_5 = 2^9, r_1 = \cdots = r_5 = 2^3$

FIG. 8. *Matrix and tensor communication costs in Algorithm* 8.1 *and the TTM-in-Sequence approach.*

and is better when the number of the entries in the output tensor is less than that of the matrices. When the communication cost is dominated by the output tensor, our approach is outperformed by the TTM-in-Sequence approach, which is the case in Figure 7(c).

The parallel computational cost of TuckerMPI with cubical tensors is

$$2\left(\frac{r^{\frac{1}{d}}n}{P} + \frac{r^{\frac{2}{d}}n^{\frac{d-1}{d}}}{P} + \cdots + \frac{rn^{\frac{1}{d}}}{P}\right).$$

When $n \gg r$, Algorithm 8.1 selects a processor grid such that $q = 1$ and $p_1 \approx p_2 \approx \cdots \approx p_d$. In this case the computation cost given in subsection 8.1 simplifies to

$$2\left(\frac{r^{\frac{1}{d}}n}{P} + \frac{r^{\frac{2}{d}}n^{\frac{d-1}{d}}}{P^{\frac{d-1}{d}}} + \cdots + \frac{rn^{\frac{1}{d}}}{P^{\frac{1}{d}}}\right).$$

While the starting terms of the two computational cost expressions match, we observe greater computational cost from Algorithm 8.1 in the remaining terms. These terms are lower order when $P \ll n/r$, in which case the extra computational cost of Algorithm 8.1 is often negligible. We plot computational overheads of our algorithm, with *Comp-Overhead* label, in Figure 7. We can note that the overheads are less than 13% for the considered experiments.

Our results are consistent with what we observe for 3D Multi-TTM computations in section 6. When the input tensor is much larger than the output tensor and the

number of entries in the output tensor is less than that of the matrices, our algorithm significantly reduces communication compared to the TTM-in-Sequence approach. As in the 3D case, when $P \ll n/r$, the extra computation is often negligible when the TTM-in-Sequence approach is used locally to reduce computation.

9. Conclusions. In this work, we establish communication lower bounds for the parallel Multi-TTM computation and present an optimal parallel algorithm that organizes the processors in a $2d$-dimensional grid for d-dimensional tensors. By judiciously selecting the processor grid dimensions, we prove that our algorithm attains the lower bounds to within a constant factor. To verify the theoretical analysis, we simulate Multi-TTM computations using a variety of values for the number of processors, P, the dimension, d, and sizes, n_i and r_i; we compute the communication costs of our algorithm corresponding to each simulation; and we compute the optimal communication cost provided by the theoretical lower bound. These simulations show that the communication costs of the proposed algorithm are close to optimal. When one of the tensors is much larger than the other tensor, which is typical in compression algorithms based on the Tucker decomposition, our algorithm significantly reduces communication costs over the conventional approach of performing the computation as a sequence of TTM operations.

Motivated by the simulated communication cost comparisons, our next goal is to implement the parallel atomic algorithm and verify the performance improvement in practice. Further, because neither the atomic or TTM-in-sequence approach is always superior in terms of communication, we wish to explore hybrid algorithms to account for significant dimension reduction in some modes but modest reduction in others. Given the computation and communication capabilities of a parallel platform, it would also be interesting to study the computation-communication tradeoff for these two approaches and how to minimize the overall execution time in practice. Finally, this work considers that each processor has enough memory. A natural extension is to study communication lower bounds for Multi-TTM computations with limited memory sizes.

REFERENCES

[1] H. Al Daas, G. Ballard, L. Grigori, S. Kumar, and K. Rouse, *Tight memory-independent parallel matrix multiplication communication lower bounds*, in Proceedings of the 34th Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '22, ACM, New York, NY, 2022, pp. 445–448, https://doi.org/10.1145/3490148.3538552.

[2] 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, IEEE, 2016, pp. 912–922, https://doi.org/10.1109/IPDPS.2016.67.

[3] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz, *Communication lower bounds and optimal algorithms for numerical linear algebra*, Acta Numer., 23 (2014), pp. 1–155, https://doi.org/10.1017/S0962492914000038.

[4] G. Ballard, A. Klinvex, and T. G. Kolda, *TuckerMPI: A parallel C++/MPI software package for large-scale data compression via the Tucker tensor decomposition*, ACM Trans. Math. Software, 46 (2020), pp. 1–31, https://doi.org/10.1145/3378445.

[5] G. Ballard, N. Knight, and K. Rouse, *Communication lower bounds for matricized tensor times Khatri–Rao product*, in 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2018, pp. 557–567, https://doi.org/10.1109/IPDPS.2018.00065.

[6] G. Ballard and K. Rouse, *General memory-independent lower bound for MTTKRP*, in Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing (PP), SIAM, 2020, pp. 1–11, https://doi.org/10.1137/1.9781611976137.1.

[7] J. Bennett, A. Carbery, M. Christ, and T. Tao, *Finite bounds for Hölder–Brascamp–Lieb multilinear inequalities*, Math. Res. Lett., 17 (2010), pp. 647–666, https://doi.org/10.4310/MRL.2010.v17.n4.a6.

[8] R. Bro and C. A. Andersson, *Improving the speed of multiway algorithms: Part* II*: Compression*, Chemometrics Intell. Lab. Syst., 42 (1998), pp. 105–113, https://doi.org/10.1016/S0169-7439(98)00011-2.

[9] V. T. Chakaravarthy, J. W. Choi, D. J. Joseph, X. Liu, P. Murali, Y. Sabharwal, and D. Sreedhar, *On optimizing distributed Tucker decomposition for dense tensors*, in 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2017, pp. 1038–1047, https://doi.org/10.1109/IPDPS.2017.86.

[10] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, *Collective communication: Theory, practice, and experience*, Concurrency Comput. Pract. Experience, 19 (2007), pp. 1749–1783, https://doi.org/10.1002/cpe.1206.

[11] J. Choi, X. Liu, and V. Chakaravarthy, *High-performance dense Tucker decomposition on GPU clusters*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC '18, IEEE, Piscataway, NJ, 2018, pp. 1–11, http://dl.acm.org/citation.cfm?id=3291656.3291712.

[12] M. Christ, J. Demmel, N. Knight, T. Scanlon, and K. A. Yelick, *Communication Lower Bounds and Optimal Algorithms for Programs that Reference Arrays - Part* 1, Technical report UCB/EECS-2013-61, EECS Department, University of California, Berkeley, 2013, http://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-61.html.

[13] H. A. Daas, G. Ballard, L. Grigori, S. Kumar, and K. Rouse, *Communication Lower Bounds and Optimal Algorithms for Multiple Tensor-Times-Matrix Computation*, 2022, https://arxiv.org/abs/2207.10437.

[14] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger, *Communication-optimal parallel recursive rectangular matrix multiplication*, in 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, IEEE, 2013, pp. 261–272, https://doi.org/10.1109/IPDPS.2013.80.

[15] 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, STOC '81, Association for Computing Machinery, New York, NY, 1981, pp. 326–333, https://doi.org/10.1145/800076.802486.

[16] D. Irony, S. Toledo, and A. Tiskin, *Communication lower bounds for distributed-memory matrix multiplication*, J. Parallel Distrib. Comput., 64 (2004), pp. 1017–1026, https://doi.org/10.1016/j.jpdc.2004.03.021.

[17] N. Knight, *Communication-Optimal Loop Nests*, Ph.D. thesis, University of California, Berkeley, 2015, http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-185.html.

[18] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500, https://doi.org/10.1137/07070111X.

[19] H. Kolla, K. Aditya, and J. H. Chen, *Higher order tensors for DNS data analysis and compression*, in Data Analysis for Direct Numerical Simulations of Turbulent Combustion, H. Pitsch and A. Attili, eds., Springer, Cham, Switzerland, 2020, pp. 109–134, https://doi.org/10.1007/978-3-030-44718-2_6.

[20] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278, https://doi.org/10.1137/S0895479896305696.

[21] L. H. Loomis and H. Whitney, *An inequality related to the isoperimetric inequality*, Bull. Amer. Math. Soc. (N.S.), 55 (1949), pp. 961–962.

[22] L. Ma and E. Solomonik, *Accelerating alternating least squares for tensor decomposition by pairwise perturbation*, Numer. Linear Algebra Appl., 29 (2022), pp. 1–33, https://doi.org/10.1002/nla.2431.

[23] R. Minster, Z. Li, and G. Ballard, *Parallel Randomized Tucker Decomposition Algorithms*, 2022, https://arxiv.org/abs/2211.13028.

[24] R. Minster, A. K. Saibaba, and M. E. Kilmer, *Randomized algorithms for low-rank tensor decompositions in the Tucker format*, SIAM J. Math. Data Sci., 2 (2020), pp. 189–215, https://doi.org/10.1137/19M1261043.

[25] T. M. Smith, B. Lowery, J. Langou, and R. A. van de Geijn, *A Tight I/O Lower Bound for Matrix Multiplication*, 2019, https://arxiv.org/abs/1702.02017.

[26] E. Solomonik, J. Demmel, and T. Hoefler, *Communication lower bounds of bilinear algorithms for symmetric tensor contractions*, SIAM J. Sci. Comput., 43 (2021), pp. A3328–A3356, https://doi.org/10.1137/20M1338599.

[27] Y. Sun, Y. Guo, C. Luo, J. Tropp, and M. Udell, *Low-rank Tucker approximation of a tensor from streaming data*, SIAM J. Math. Data Sci., 2 (2020), pp. 1123–1150, https://doi.org/10.1137/19M1257718.

[28] R. Thakur, R. Rabenseifner, and W. Gropp, *Optimization of collective communication operations in MPICH*, Int. J. High Performance Comput. Appl., 19 (2005), pp. 49–66, https://doi.org/10.1177/1094342005051521.

[29] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311, https://doi.org/10.1007/BF02289464.

[30] A. N. Ziogas, G. Kwasniewski, T. Ben-Nun, T. Schneider, and T. Hoefler, *Deinsum: Practically I/O optimal multi-linear algebra*, in SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2022, pp. 1–15, https://doi.org/10.1109/SC41404.2022.00030.