Anatomy of Parallel Computation with Tensors

FLAME Working Note #72
Ph.D. Dissertation Proposal

Martin D. Schatz

Department of Computer Science
and
Institute for Computational Engineering and Sciences
The University of Texas at Austin
Austin, Texas 78712
martin.schatz@utexas.edu

December 17, 2013

Abstract

Recent data models have become more complex leading to the need for multi-dimensional representations to express data in a more meaningful way. Commonly, tensors are used to represent such data. Multi-linear algebra, the math associated with tensors, has become essential for tackling problems in big data and scientific computing. To solve the largest problems of today, libraries designed for supercomputers consisting of thousands of compute nodes connected via a network are utilized. Such compute architectures are referred to as "distributed-memory" architectures. Up to now, the main approach for problems of multi-linear algebra has been based on mapping multi-linear algebra to linear algebra and rely on highly efficient linear algebra libraries to perform the equivalent computation [6, 23]. Unfortunately, there are inherent inefficiencies associated with this approach. In this proposal, we define a domain-specific language for distributed tensor computation. Additionally, through a process akin to constraint propagation, we show how, using the language, algorithms can be systematically derived and required collective communications identified for the tensor contraction operation.

1 Introduction

Branches of scientific computing, particularly computational chemistry, express data as a tensor which can be viewed as a multi-dimensional analog of a matrix. Chemical methods heavily rely on an operation commonly referred to as the tensor contraction, which can be viewed as a generalization of matrix-matrix multiplication. For instance, in the following fragment of the third-order Møller-Plesset (MP3) method [5] found in computational chemistry,

$$\epsilon = \mathcal{T}^{ABIJ}\left(\mathcal{U}^{ABCD}\mathcal{T}^{CDIJ} + \mathcal{V}^{ACIK}\mathcal{T}^{BCJK} + \mathcal{W}^{IJKL}\mathcal{T}^{ABKL}\right), \tag{1}$$

the term

$$\mathcal{V}^{ACIK}\mathcal{T}^{BCJK} \tag{2}$$

is a tensor contraction involving the four-dimensional tensor $\mathcal{V}$ and $\mathcal{T}$ expressed using Einstein notation [11]. In fact, all terms denoted as the product of two variables (indicated as regular multiplication) in (1) represent

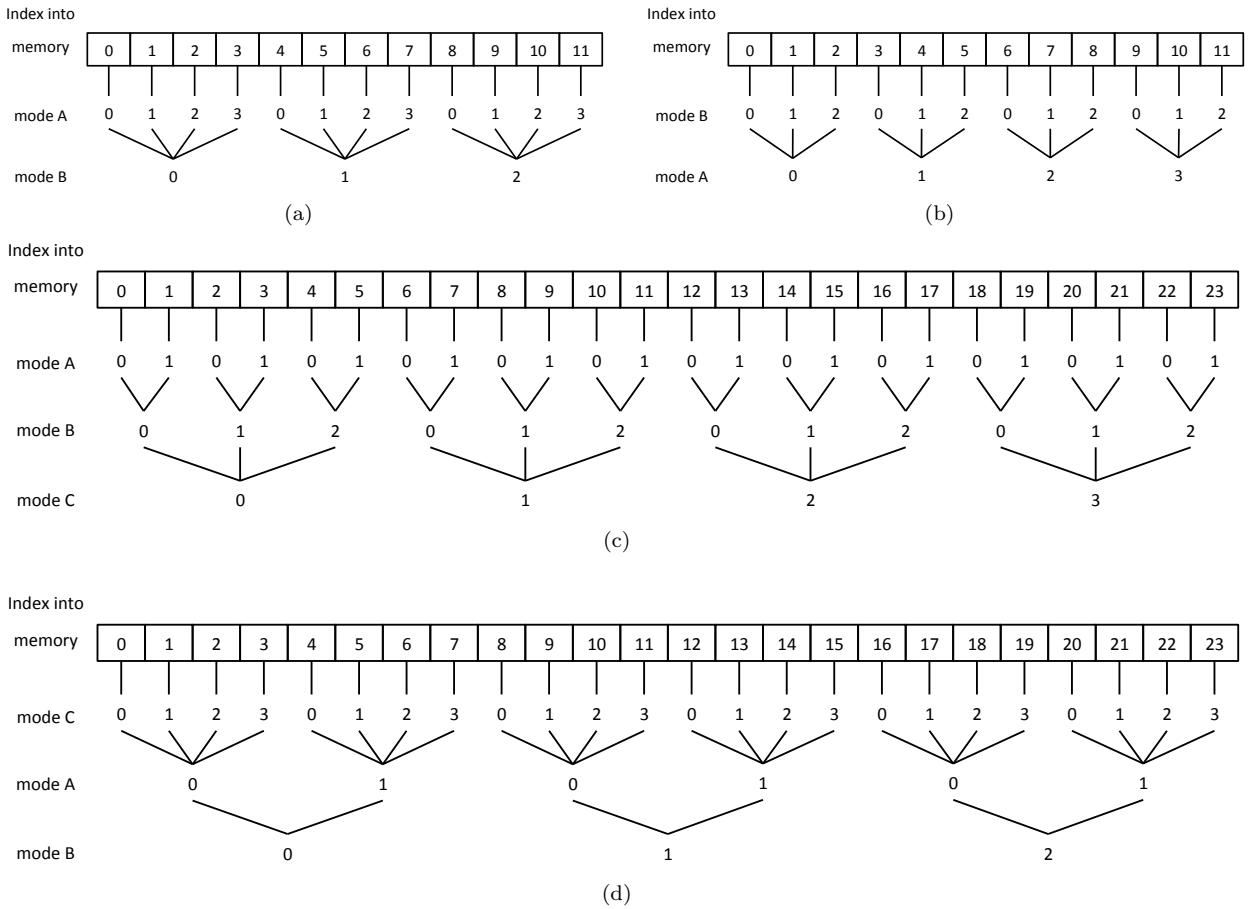

Figure 1: Generalized column-major storage for different objects. The first line in each image corresponds to linear memory, and each subsequent line corresponds to the modes associated with each object. (a) $\mathbf{A}^{AB} \in \mathbb{R}^{L_A \times L_B}$, (b) $\mathbf{A}^{BA} \in \mathbb{R}^{L_B \times L_A}$, (c) $\boldsymbol{\mathcal{A}}^{ABC} \in \mathbb{R}^{L_A \times L_B \times L_C}$ and (d) $\boldsymbol{\mathcal{A}}^{CAB} \in \mathbb{R}^{L_C \times L_A \times L_B}$.

tensor contractions[1]. As reproduced, the fragment is incorrect for utilization within chemistry, however the fragment allows us to discuss many of the challenges associated with computing with tensors.

The number of *modes* associated with each tensor expresses the number of dimensions, or order, of the tensor. The modes of each tensor represents a feature of the application and are labeled accordingly (indicated by superscripts). Assuming a generalization of column-major storage, the order of the modes can also indicate how each tensor is stored in memory. Examples of how modes can be used to indicate the storage scheme utilized is given in Figure 1.

We could write an expression such as

$$\boldsymbol{\mathcal{X}}^{ABJI} \mathrel{+}= \boldsymbol{\mathcal{V}}^{ACIK} \boldsymbol{\mathcal{T}}^{BCJK} \tag{3}$$

to compute the result of (2). Here, $\boldsymbol{\mathcal{X}}^{ABJI}$ merely acts as an output to store the results of the contraction. The order of the modes for $\boldsymbol{\mathcal{X}}^{ABJI}$ was arbitrarily chosen for example purposes in this paper. If

[1]Throughout this proposal, we ignore the distinction between covariant and contravariant vectors. This assumption does not affect the following discussions.

$$\begin{aligned}\boldsymbol{\mathcal{X}}^{ABJI} &\in \mathbb{R}^{L_A \times L_B \times L_J \times L_I},\\ \boldsymbol{\mathcal{V}}^{ACIK} &\in \mathbb{R}^{L_A \times L_C \times L_I \times L_K},\\ \boldsymbol{\mathcal{T}}^{BCJK} &\in \mathbb{R}^{L_B \times L_C \times L_J \times L_K},\end{aligned}$$

then each element of $\boldsymbol{\mathcal{X}}^{ABJI}$ is computed as

$$\chi_{i_0 i_1 i_2 i_3} += \sum_{i_4=0}^{L_C-1} \sum_{i_5=0}^{L_K-1} \nu_{i_0 i_4 i_3 i_5} \tau_{i_1 i_4 i_2 i_5}.$$

where $\chi_{i_0 i_1 i_2 i_3}$ is an element of $\boldsymbol{\mathcal{X}}^{ABJI}$, $\nu_{i_0 i_4 i_3 i_5}$ is an element of $\boldsymbol{\mathcal{V}}^{ACIK}$, and $\tau_{i_1 i_4 i_2 i_5}$ is an element of $\boldsymbol{\mathcal{T}}^{BCJK}$. Given an ordered list of modes used in (3), the subscript i_k denotes an element's index within the kth mode of that list. In this example, the ordered list used is (A, B, J, I, C, K).

Notice that the relative order of modes is not constant between tensors (each tensor can have a different order of modes). It is this freedom in order of modes that makes it difficult to ensure high-performance implementations for tensor contractions. To achieve high-performance for all cases, one would need an algorithm tailored for every permutation of modes, but this is infeasible as the number of permutations is related to the order of the tensors, which can be arbitrarily large (requiring an arbitrarily large number of algorithms to account for the entire space of possible operands).

One approach commonly used to circumvent this issue is to reorder the data via a permutation and reshaping of data, so that a general matrix-matrix multiplication (for which high-performance implementations exist) can be performed to compute the same result. We refer to this as a permute-and-reshape approach. Work by Bader and Kolda refers to the permutation and reshaping of the tensor as a matricization of a tensor [4, 3]. To see how this is performed, consider the equivalent (but with rearranged modes) operation

$$\boldsymbol{\mathcal{X}}^{AIBJ} += \boldsymbol{\mathcal{V}}^{AICK} \boldsymbol{\mathcal{T}}^{CKBJ}. \tag{4}$$

One can view (4) as the matrix-matrix multiplication

$$\mathbf{C}^{MN} = \mathbf{A}^{MP} \mathbf{B}^{PN} \tag{5}$$

if we map each element $\chi_{i_0 i_1 i_2 i_3}$, $\nu_{i_0 i_1 i_2 i_3}$, and $\tau_{i_0 i_1 i_2 i_3}$ to elements of $\mathbf{C}^{MN}$, $\mathbf{A}^{MP}$, and $\mathbf{B}^{PN}$ according to the mappings

$$\begin{aligned}\chi_{i_0 i_1 i_2 i_3} &= \gamma_{(i_0 + i_1 L_A)(i_2 + i_3 L_B)},\\ \nu_{i_0 i_1 i_2 i_3} &= \alpha_{(i_0 + i_1 L_A)(i_2 + i_3 L_C)},\\ \tau_{i_0 i_1 i_2 i_3} &= \beta_{(i_0 + i_1 L_C)(i_2 + i_3 L_B)}.\end{aligned}$$

where $\gamma_{(i_0 + i_1 L_A)(i_2 + i_3 L_B)}$, $\alpha_{(i_0 + i_1 L_A)(i_2 + i_3 L_C)}$, and $\beta_{(i_0 + i_1 L_C)(i_2 + i_3 L_B)}$ are elements of $\mathbf{C}^{MN}$, $\mathbf{A}^{MP}$, and $\mathbf{B}^{PN}$ respectively.

Assuming the same generalized column-major ordering of elements in memory, the result of computing (5) is the same as directly computing (4). As (5) is a matrix-matrix multiplication, at the expense of permuting the data (needed to get (3) into the form of (4)), we can achieve high-performance when computing the operation. The disadvantage of this approach, most pronounced in distributed-memory architecture computing environments, is that these rearrangements of data can amount to a large communication overhead associated with the overall computation even though they contribute a lower-order term.

Currently, many libraries and projects view, in some form, the tensor contraction operation as two permute-and-reshape operations followed by a matrix-matrix multiplication, followed finally by a permute-and-reshape operation. The initial permute-and-reshape operations ensure the correct result is computed in the subsequent matrix-matrix multiplication. The final permute-and-reshape operation is required to ensure the elements in memory appear as if the tensor contraction were directly computed. The *Tensor Contraction Engine* (TCE) [6], a project designed to generate efficient code for a given expression of tensor contractions which essentially generates a series of distributed matrix-matrix multiplication with necessary data redistributions before and after the computation.

Notation	Meaning
$\mathcal{L}$	Set of all possible modes of tensor
$\boldsymbol{a}, \boldsymbol{b}, \ldots$	Set of modes assigned to a tensor
$\boldsymbol{l}_k$	k-th mode assigned to a tensor
i_k	k-th index into a mode
$\overline{\mathbf{r}}_{\boldsymbol{l}_k}$	Range of mode $\boldsymbol{l}_k$, i.e., the list of possible index values of mode $\boldsymbol{l}_k$
$L_{\boldsymbol{l}_k}$	Dimension of mode $\boldsymbol{l}_k$
$\overline{\mathbf{a}}, \overline{\mathbf{b}}, \ldots$	Sequences
$\overline{\alpha}, \overline{\beta}, \ldots$	Elements of sequences
$\mathcal{G}$	Processing grid
m_g	Order of shape of processing grid mesh
$\mathcal{G}_{\langle p_0, \ldots, p_{m_g-1} \rangle}$	The process at location $\langle p_0, \ldots, p_{m_g-1} \rangle$ in $\mathcal{G}$
d_k	dimension of k-th mode of $\mathcal{G}$
$\alpha, \beta, \ldots$	Scalars
$\mathbf{a}, \mathbf{b}, \ldots$	Vectors
$\mathbf{A}, \mathbf{B}, \ldots$	Matrices
$\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \ldots$	Tensors
$D_{\langle p_0, \ldots, p_k \rangle}$	Distribution Partitioning defined by the sequence $\langle p_0, \ldots, p_k \rangle$
$\chi \setminus \psi$	The scalar χ *divides* (is a divisor of) ψ

Figure 2: Glossary of notation used throughout this proposal

When discussing distributed-memory computing, it is also important to consider the communication overhead required when redistributing data among processes. Many commonly utilized communication patterns involving many processes at once, referred to as collective communications, are used to create high-performance implementations for linear algebra operations. It is important to understand what conditions each collective communication should be utilized to increase performance. By having a model which incorporates this information, the task of accommodating new computing architectures is mitigated.

For high-performance distributed-memory dense linear algebra operations, the Elemental library (Elemental) [18] expresses algorithms in terms of a formal notation which allows one to not only systematically derive algorithms for matrix-matrix multiplication, but also express different types of redistributions of data in terms of different collectives. By combining these two aspects, tools can automate the derivation process, and, as a result, identify which collective communications are required for a given efficient algorithm to proceed.

In this paper, we more formally define tensors and the tensor contraction, briefly describe challenges that must be addressed when devising a distributed-memory library to support general tensors, and describe current projects that aim to address the challenges detailed along with their perceived limitations. Finally, we conclude by proposing a notation for distributing data of tensors in distributed-memory environments as well as show how the proposed notation can be used to systematically derive algorithms for the tensor contraction operation.

2 Preliminaries

In this paper, ideas that are likely unfamiliar to the reader will be used. This section provides the necessary background for subsequent discussions. All terminology introduced in this section is summarized in Figure 2 and the glossary at the end of this proposal.

In this section, we define tensors as well as the tensor contraction operation. In addition, we introduce the notion of a sequence, relevant notation, and relevant collective communications.

Roman	Greek
a	α
b	β
c	γ
e	ϵ
t	τ
v	ν
x	χ
y	ψ

Figure 3: List of Roman letters and their Greek counterparts used to indicate scalars throughout this proposal. Depending on the type of object, the Roman letter used may be used with different accenting or casing.

2.1 Notation

We assume all indexing begins at zero.

We refer to the set of all modes used as $\mathcal{L}$. A set of modes is denoted with a bold-italisized lower-case Roman letters ($\boldsymbol{a}$, $\boldsymbol{b}$, ...). The k-th mode of $\boldsymbol{l}$ is denoted $\boldsymbol{l}_k$ (assuming some ordering of modes in $\boldsymbol{l}$). We reserve the lower-case Roman letter i for indexing within modes of a tensor. Subscripts are used to differentiate between different indexing variables (for instance i_0 is different from i_1).

In coming discussions we refer to the processing grid used for computation. We use $\mathcal{G}$ to refer to the processing grid and m_g to refer to the order of the processing mesh (number of modes used to specify the shape of the mesh). We reserve the lower-case Roman letter p for indexing within modes of $\mathcal{G}$ (subscripts used to differentiate between modes of $\mathcal{G}$). If we view $\mathcal{G}$ as an order-m_g mesh, the process at location specified by the coordinate $(p_0, p_1, \ldots, p_{m_g-1})$ is referred to by $\mathcal{G}_{\langle p_0,\ldots,p_{m_g-1}\rangle}$[2]. We reserve the lower-case Roman letter d for the dimension of the modes of $\mathcal{G}$ (subscripts differentiate modes).

An element of an order-m tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{L_{\iota_0} \times L_{\iota_1} \times \cdots \times L_{\iota_{m-1}}}$ is denoted as $\alpha_{i_0 i_1 \cdots i_{m-1}}$ where $0 \leq i_k < L_{\boldsymbol{l}_k}$ for all $0 \leq k < m-1$. In general, we use Greek lower-case letters for scalars (α, β, γ, ...), bold lower-case Roman letters for vectors ($\mathbf{a}$,$\mathbf{b}$,$\mathbf{c}$,...), bold upper case Roman letters for matrices ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, ...), and uppercase scripted letters for tensors ($\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{B}}$, $\boldsymbol{\mathcal{C}}$, ...). For convenience, the mapping from Roman to Greek letter is summarized in Figure 3.

2.2 Sequences

We will rely on the notion of a sequence of integers, an ordered set of integers where duplicates are allowed, for describing the different distributions of tensor data similar to the approach taken in [12].

We use bold lower-case Roman letters with an overline to refer to sequences ($\overline{\mathbf{a}}$, $\overline{\mathbf{b}}$, $\overline{\mathbf{c}}$, ...). Elements of sequences are denoted as bold lower-case Greek letters with an overline ($\overline{\boldsymbol{\alpha}}$, $\overline{\boldsymbol{\beta}}$, $\overline{\boldsymbol{\gamma}}$, ...). Angle brackets, $\langle$ and $\rangle$, are used to explicitly define a sequences.

We illustrate this idea with the following example:

[2]The reason for using sequence notation instead of tuple notation for a process will be explained later.

Example 1. *The sequence* $\overline{\mathbf{x}} = \langle 2, 4, 5, 3, 1\rangle$ *represents the set of elements* $\{1, 2, 3, 4, 5\}$ *ordered such that*

$$\overline{\mathbf{x}} = \langle \overline{\chi}_0, \overline{\chi}_1, \overline{\chi}_2, \overline{\chi}_3, \overline{\chi}_4\rangle = \langle 2, 4, 5, 3, 1\rangle$$

The sequence $\overline{\mathbf{x}} = \langle 2, 2, 5\rangle$ *represents the set of elements* $\{2, 5\}$ *ordered such that*

$$\langle \overline{\chi}_0, \overline{\chi}_1, \overline{\chi}_2\rangle = \langle 2, 2, 5\rangle$$

The sequence $\overline{\mathbf{y}} = \langle\rangle$ *represents the empty sequence.*

We now define operations on sequences along with the associated notation we will use throughout this paper.

Definition 1 (**Cardinality**).

$$|\overline{\mathbf{x}}|$$

refers to the cardinality of the sequence $\overline{\mathbf{x}}$*; i.e., the number of elements comprising the sequence* $\overline{\mathbf{x}}$ *(including duplicates).*

Definition 2 (**Element of**).

$$\overline{\chi} \in \overline{\mathbf{x}}$$

indicates that $\overline{\chi}$ *is an element of the sequence* $\overline{\mathbf{x}}$*.*

Definition 3 (**Concatenation**).

$$\overline{\mathbf{x}} \sqcup \overline{\mathbf{y}}$$

indicates the concatenation of $\overline{\mathbf{y}}$ *to* $\overline{\mathbf{x}}$*; i.e.,*

$$\overline{\mathbf{x}} \sqcup \overline{\mathbf{y}} = \langle \overline{\chi}_0, \ldots, \overline{\chi}_{|\overline{\mathbf{x}}|-1}, \overline{\psi}_0, \ldots, \overline{\psi}_{|\overline{\mathbf{y}}|-1}\rangle$$

Definition 4 (**Subsequence**).

$$\overline{\mathbf{y}} \sqsubseteq \overline{\mathbf{x}}$$

denotes that $\overline{\mathbf{y}}$ *is a subsequence of* $\overline{\mathbf{x}}$*; i.e., all elements in* $\overline{\mathbf{y}}$ *appear in the same relative order in* $\overline{\mathbf{x}}$*.*

We illustrate the idea of a subsequence with the following example

Example 2. *Consider the sequences*

$$\begin{aligned}\overline{\mathbf{x}} &= \langle 2, 5, 3\rangle,\\ \overline{\mathbf{y}} &= \langle 2, 3\rangle,\\ \overline{\mathbf{z}} &= \langle 2, 1\rangle,\\ \overline{\mathbf{w}} &= \langle 3, 2\rangle.\end{aligned}$$

The sequence $\overline{\mathbf{y}}$ *is a subsequence of* $\overline{\mathbf{x}}$ *as all elements of* $\overline{\mathbf{y}}$ *appear in* $\overline{\mathbf{x}}$ *and appear in the same relative order. The sequence* $\overline{\mathbf{z}}$ *is not a subsequence of* $\overline{\mathbf{x}}$ *as the element* $\overline{\zeta}_1 = 1$ *does not appear in* $\overline{\mathbf{x}}$*. The sequence* $\overline{\mathbf{w}}$ *is not a subsequence of* $\overline{\mathbf{x}}$ *as all elements of* $\overline{\mathbf{w}}$ *appear in* $\overline{\mathbf{x}}$ *but in a different relative order (3 appears after 2 in sequence* $\overline{\mathbf{x}}$*, not before as in* $\overline{\mathbf{w}}$*).*

A table summarizing the symbols and meanings for sequence operations has been provided in Figure 4.

Symbol used	Meaning
$\overline{\mathbf{x}}$	Sequence
$\langle\rangle$	Empty sequence
$\lvert\overline{\mathbf{x}}\rvert$	Cardinality
$\overline{\chi} \in \overline{\mathbf{x}}$	Element of
$\overline{\mathbf{x}} \sqcup \overline{\mathbf{y}}$	Concatenation
$\overline{\mathbf{x}} \sqsubseteq \overline{\mathbf{y}}$	Subsequence

Figure 4: List of symbols and their meanings associated with sequences.

2.3 Tensors

A tensor can be viewed as an m-modal array[3] where each *mode* is related to a certain feature of the application labeled accordingly. Each mode is written as a superscript of the tensor. The set of all possible modes is denoted as $\mathcal{L}$. For this proposal, we use uppercase Roman letters as possible modes for tensors understanding that the set of modes is, in general, infinite.

The *order* of a tensor is the number of modes associated with the tensor. The *dimension* of a mode labeled I, denoted L_I, is the length or size of the mode. Under this definition of dimension, *any* mode labeled the same, regardless of tensor, must have the same dimension.

The *range* of a mode labeled I, denoted $\overline{\mathbf{r}}_I$, is the sequence which corresponds to all valid indexing values of the mode (ordered increasingly). This means $\overline{\mathbf{r}}_I$ refers to the sequence $\langle 0, \ldots, L_I - 1\rangle$. A *subrange* is a subsequence of a range. When referring to a particular element of a tensor, we indicate the location of the scalar element using subscripts. Each subscript corresponds to an *index* of the corresponding mode.

We can define an order-m tensor representing the modes $\boldsymbol{l}_0, \boldsymbol{l}_1, \ldots, \boldsymbol{l}_{m-1}$ of the set of modes $\boldsymbol{l}$ as

$$\boldsymbol{\mathcal{A}}^{\boldsymbol{l}_0\boldsymbol{l}_1\cdots\boldsymbol{l}_{m-1}} \in \mathbb{R}^{L_{\boldsymbol{l}_0}\times L_{\boldsymbol{l}_1}\times\cdots\times L_{\boldsymbol{l}_{m-1}}}.$$

The element of $\boldsymbol{\mathcal{A}}^{\boldsymbol{l}_0\boldsymbol{l}_1\cdots\boldsymbol{l}_{m-1}}$ at location $i_0, i_1, \ldots, i_{m-1}$ (index i_k of mode $\boldsymbol{l}_k$) can be referred to as

$$\alpha_{i_0 i_1 \cdots i_{m-1}}.$$

We illustrate the introduced terminology with the following example:

Example 3. *Given an order-3 tensor*

$$\boldsymbol{\mathcal{A}}^{ACB} \in \mathbb{R}^{3\times 4\times 2},$$

$\boldsymbol{\mathcal{A}}^{ACB}$ *has the following associated with each mode:*

Mode	*Range*	*Dimension*
A	$\langle 0, 1, 2\rangle$	*3*
C	$\langle 0, 1, 2, 3\rangle$	*4*
B	$\langle 0, 1\rangle$	*2*

The entry α_{201} *is defined by the following indices:*

Mode	*Index*
A	*2*
C	*0*
B	*1*

[3] An m-modal array is equivalent to an m-dimensional array, however, to avoid confusion we use the former terminology.

2.4 Tensor Contraction

A tensor contraction can be viewed as a generalization of matrix-matrix multiplication. Strictly speaking, the tensor contraction definition with which we are concerned is the contraction defined as the summation over the product of two tensors. It is this definition that views matrix-matrix multiplication as a special case. For example, we can write matrix-matrix multiplication as

$$\mathbf{C}^{IJ} = \mathbf{A}^{IK}\mathbf{B}^{KJ}.$$

This views each matrix as an order-2 tensor where we are performing a contraction over the mode K. The definition of a tensor contraction as a summation over the product of tensors is made explicit by noting that elements of $\mathbf{C}^{IJ}$ are computed as

$$\gamma_{i_0 i_1} = \sum_{i_2=0}^{L_K - 1} \alpha_{i_0 i_2}\beta_{i_2 i_1}.$$

Notice that this corresponds to the definition of matrix-matrix multiplication, and that each output entry of the matrix $\mathbf{C}$ is defined as the summation over the product of entries in $\mathbf{A}$ and $\mathbf{B}$ along the K mode.

Similarly, we can define matrix-vector multiplication in terms of a tensor contraction as

$$\mathbf{c}^{I} = \mathbf{A}^{IK}\mathbf{b}^{K}.$$

In addition to viewing matrix-matrix multiplication as a special case of tensor contractions, all transpose variants are special cases of a tensor contraction as well. Take the matrix-matrix multiplication

$$\mathbf{C}^{T} = \mathbf{A}\mathbf{B}^{T}.$$

This can be written as the tensor contraction

$$\mathbf{C}^{JI} = \mathbf{A}^{IK}\mathbf{B}^{JK}.$$

We leave it as an exercise to the reader to see why the previous two statements are equivalent.

In the previous examples, we only discussed the case where at most two modes were involved with each object involved in the computation. The generalization used to include tensors of arbitrary order, states that each output element is formed by the product of input tensors summed along *all* modes shared by both input tensors. Consider the contraction

$$\mathcal{C}^{\boldsymbol{l}_0\cdots\boldsymbol{l}_{m-1}} = \mathcal{A}^{\boldsymbol{l}_0\cdots\boldsymbol{l}_k\boldsymbol{s}_0\cdots\boldsymbol{s}_{n-1}}\mathcal{B}^{\boldsymbol{l}_{k+1}\cdots\boldsymbol{l}_{m-1}\boldsymbol{s}_0\cdots\boldsymbol{s}_{n-1}},$$

where $\boldsymbol{l} \cap \boldsymbol{s} = \emptyset$. As mentioned earlier, the contraction presently considered has been expressed in Einstein notation [11]. This notation states that all shared modes by inputs are implicitly summed over all indices. Using this notation, the tensor $\mathcal{C}$ is defined element-wise as

$$\gamma_{i_0\cdots i_{m-1}} = \sum_{i_m=0}^{L_{\boldsymbol{s}_0}} \cdots \sum_{i_{m+n-1}=0}^{L_{\boldsymbol{s}_{n-1}}} \beta_{i_0\cdots i_k i_m\cdots i_{m+n-1}}\alpha_{i_{k+1}\cdots i_{m-1}i_m\cdots i_{m+n-1}}.$$

Notice that we are performing summations over all modes contained in $\boldsymbol{s}$ because each mode in this set is shared by inputs $\mathcal{A}$ and $\mathcal{B}$.

Later, it will be useful to include which modes are being summed over in the contraction without using an element-wise definition. We do this by alternatively expressing the contraction in Einstein notation, prefixed with the summation symbol which has, as a subscript, all modes being summed over. This should be viewed only as an alternative notation, not as a difference in operation being performed. For instance,

$$\mathcal{C}^{\boldsymbol{l}_0\boldsymbol{l}_2} = \sum_{\boldsymbol{l}_1,\boldsymbol{l}_3} \mathcal{A}^{\boldsymbol{l}_0\boldsymbol{l}_1\boldsymbol{l}_3}\mathcal{B}^{\boldsymbol{l}_1\boldsymbol{l}_2\boldsymbol{l}_3} = \mathcal{A}^{\boldsymbol{l}_0\boldsymbol{l}_1\boldsymbol{l}_3}\mathcal{B}^{\boldsymbol{l}_1\boldsymbol{l}_2\boldsymbol{l}_3}$$

denotes $\mathcal{C}$ should be computed element-wise as

$$\mathcal{C}_{i_0 i_2} = \sum_{i_1=0}^{L_{\iota_1}-1} \sum_{i_3=0}^{L_{\iota_3}-1} \alpha_{i_0 i_1 i_3} \beta_{i_1 i_2 i_3}.$$

Additionally, we are allowed to permute the order of modes in all operands. If one associates a storage scheme related to the order of modes of each tensor, then a permutation of modes means the data is stored in permuted order. The computations performed are equivalent, it is just the order of modes which has changed.

It is this added flexibility/generalization that makes creating high-performance implementations difficult. However, as shown in Theorem 8, when computing a tensor contraction resulting in an order-m tensor, we can theoretically expect to achieve $O(n^{\frac{m}{2}})$ useful floating point operations per element involved (assuming each mode of dimension n).

2.5 Collective Communications

The language described in this proposal assumes an elemental-cyclic wrapping of data on the computing grid. This is the same view taken by the Elemental [18] library for dense linear algebra. ScaLAPACK [9] assumes a block-cyclic wrapping of data on the computing grid. An elemental-cyclic wrapping can be viewed as a block-cyclic wrapping with unit block-size. One can view the ideas underpinning Elemental as an example of Physically Based Matrix Distributions (PBMD) [10]. PLAPACK [26] is another example of a library using PBMD. Redistributions defined in Elemental and PLAPACK, by design, are implemented with collective communications (network communications involving a group of processes). The notation used by Elemental more formally describes how distributions are related via a particular collective communication relative to the efforts made in PLAPACK. As we argue later, redistributions are also, by design, implemented as collective communications.

A thorough presentation of the collective communications used in this paper is given in Chan et al. [8]. A summary of the collective communications used in this proposal is given in Figure 5 and their lower bounds, with respect to communication, under reasonable assumptions are given in Figure 6. We will primarily be focused in the allgather, permutation (simultaneous point-to-point), all-to-all, and reduce-scatter operations.

3 Tensor Data Distribution

In this section, we propose a notation for describing how to distribute data of an order-m tensor $\mathcal{A}$ on a processing grid, $\mathcal{G}$, viewed as an order-m_g mesh. We then show how redistributions of data between valid distributions can be implemented via collective communications. Before moving on, we recommend becoming familiar with the ideas presented in Schatz et al. [22] regarding the notation used for distributing matrices and vectors in Elemental as well as how this notation is used to systematically derive algorithms for matrix-matrix multiplication. This will give the reader a thorough understanding of the special case of tensor contraction involving only order-2 tensors (matrices). The forthcoming discussions should be viewed as generalizations of the ideas presented therein. To make this document self-contained, the pertinent sections from that paper are summarized in Appendix A.

3.1 Preliminaries

Our goal is to distribute elements of an order-m tensor $\mathcal{A}$ among processes of a processing grid $\mathcal{G}$. Our approach will be to view $\mathcal{G}$ as a logical m_g-modal mesh and then assign unique subranges of modes of $\mathcal{A}$ to each process in the processing grid. By doing this, each process will be assigned the elements of $\mathcal{A}$ defined by the assigned subranges of modes.

The partitioning of the ranges of modes among processes is fundamental to coming discussions, and the following formalism specifies our notation.

Operation	Before				After			
Peer-to-peer	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	x_0	x_1	x_2	x_3	x_1	x_0	x_3	x_2
Broadcast	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	x				x	x	x	x
Reduce(-to-one)	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$\sum_j x^{(j)}$			
Scatter	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	x_0				x_0			
	x_1					x_1		
	x_2						x_2	
	x_3							x_3
Gather	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	x_0				x_0			
		x_1			x_1			
			x_2		x_2			
				x_3	x_3			
Allgather	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	x_0				x_0	x_0	x_0	x_0
		x_1			x_1	x_1	x_1	x_1
			x_2		x_2	x_2	x_2	x_2
				x_3	x_3	x_3	x_3	x_3
Reduce-scatter	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	$x_0^{(0)}$	$x_0^{(1)}$	$x_0^{(2)}$	$x_0^{(3)}$	$\sum_j x_0^{(j)}$			
	$x_1^{(0)}$	$x_1^{(1)}$	$x_1^{(2)}$	$x_1^{(3)}$		$\sum_j x_1^{(j)}$		
	$x_2^{(0)}$	$x_2^{(1)}$	$x_2^{(2)}$	$x_2^{(3)}$			$\sum_j x_2^{(j)}$	
	$x_3^{(0)}$	$x_3^{(1)}$	$x_3^{(2)}$	$x_3^{(3)}$				$\sum_j x_3^{(j)}$
Allreduce	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$\sum_j x^{(j)}$	$\sum_j x^{(j)}$	$\sum_j x^{(j)}$	$\sum_j x^{(j)}$
All-to-all	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	$x_0^{(0)}$	$x_0^{(1)}$	$x_0^{(2)}$	$x_0^{(3)}$	$x_0^{(0)}$	$x_1^{(0)}$	$x_2^{(0)}$	$x_3^{(0)}$
	$x_1^{(0)}$	$x_1^{(1)}$	$x_1^{(2)}$	$x_1^{(3)}$	$x_0^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$
	$x_2^{(0)}$	$x_2^{(1)}$	$x_2^{(2)}$	$x_2^{(3)}$	$x_0^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$
	$x_3^{(0)}$	$x_3^{(1)}$	$x_3^{(2)}$	$x_3^{(3)}$	$x_0^{(3)}$	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$

Figure 5: Collective communications considered in this paper.

Communication	Latency	Bandw.	Computation	Cost used for analysis
Peer-to-peer	α	$n\beta$	–	$\alpha + n\beta$
Broadcast	$\lceil \log_2(p) \rceil \alpha$	$n\beta$	–	$\log_2(p)\alpha + n\beta$
Reduce(-to-one)	$\lceil \log_2(p) \rceil \alpha$	$n\beta$	$\frac{p-1}{p} n\gamma$	$\log_2(p)\alpha + n(\beta + \gamma)$
Scatter	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	–	$\log_2(p)\alpha + \frac{p-1}{p} n\beta$
Gather	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	–	$\log_2(p)\alpha + \frac{p-1}{p} n\beta$
Allgather	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	–	$\log_2(p)\alpha + \frac{p-1}{p} n\beta$
Reduce-scatter	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	$\frac{p-1}{p} n\gamma$	$\log_2(p)\alpha + \frac{p-1}{p} n(\beta + \gamma)$
Allreduce	$\lceil \log_2(p) \rceil \alpha$	$2\frac{p-1}{p} n\beta$	$\frac{p-1}{p} n\gamma$	$2\log_2(p)\alpha + \frac{p-1}{p} n(2\beta + \gamma)$
All-to-all	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	–	$\log_2(p)\alpha + \frac{p-1}{p} n\beta$

Figure 6: Lower bounds with respect to communication for the different components of communication. Conditions for the lower bounds given in [8] and [7]. Here, p corresponds to the number of processes involved in the communication and n corresponds to the amount of data communicated. The last column gives the cost functions that we use in our analyses. For architectures with sufficient connectivity, simple algorithms exist with costs that remain within a small constant factor of all but one of the given formulae. The exception is the all-to-all, for which there are algorithms that achieve the lower bound for the α and β term separately, but it is not clear whether an algorithm that consistently achieves performance within a constant factor of the given cost function exists.

Definition 5 (**Partitioning of** $\mathbb{N}$). *A collection of k sequences $\{\bar{\mathbf{s}}_0, \ldots, \bar{\mathbf{s}}_{k-1}\}$ is said to be a partitioning of the sequence of natural numbers (including zero), $\langle \mathbb{N} \rangle$, if the following conditions are met:*

1. $\forall_{i,j \in \{0,\ldots,k-1\}} \forall_{\ell_0 \in \bar{\mathbf{s}}_i} \forall_{\ell_1 \in \bar{\mathbf{s}}_j} \ell_0 \neq \ell_1$ *(No two sequences share the same element)*
2. $\forall_{\ell \in \{0,\ldots,k-1\}} \overline{\sigma}_{\ell i} \neq \overline{\sigma}_{\ell j}$ *when* $i \neq j$ *(No elements duplicated in any sequence)*
3. $\forall_{i \in \mathbb{N}} i \in \cup_{j=0}^{k-1} \bar{\mathbf{s}}_j$ *(All natural numbers appear in a sequence)*

The basic idea will be to use different partitionings of the natural numbers to describe the distribution of ranges of modes of the distributed tensor. This process is referred to as "distributing" a mode. In this manner, we view an element of a partitioning as specifying a filter to apply to the range of a mode. By applying a range to a filter, we create a subrange which indicates which indices of the mode to be distributed are assigned to each process in our computing grid. We formally describe the notion of a sequence filter.

Definition 6 (**Sequence filter**). *Consider a sequence, $\overline{\mathbf{x}}$, we wish to filter; i.e., form a new sequence comprised of elements of $\overline{\mathbf{x}}$. Given a sequence, $\bar{\mathbf{s}}$, which we view as our filter, then we say $\overline{\mathbf{x}}(\bar{\mathbf{s}})$ is the application of a filter $\bar{\mathbf{s}}$ to the sequence $\overline{\mathbf{x}}$; i.e.*

$$\overline{\mathbf{x}}(\bar{\mathbf{s}}) = \langle \overline{\chi}_{\overline{\sigma}_i} : (0 \leq i < |\bar{\mathbf{s}}|) \wedge (0 \leq \overline{\sigma}_i < |\overline{\mathbf{x}}|) \rangle.$$

In other words, $\overline{\mathbf{x}}(\bar{\mathbf{s}})$ is the sequence composed of elements of $\overline{\mathbf{x}}$ in the order specified by elements of $\bar{\mathbf{s}}$. All elements of $\bar{\mathbf{s}}$ which are out of the range of $\overline{\mathbf{x}}$ are excluded.

We illustrate this idea with a simple example:

$\mathcal{G}_{\langle 0,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,1,\ldots\rangle)$	$\mathcal{G}_{\langle 0,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,1,\ldots\rangle)$	$\mathcal{G}_{\langle 0,2\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,1,\ldots\rangle)$
$\mathcal{G}_{\langle 1,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,1,\ldots\rangle)$	$\mathcal{G}_{\langle 1,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,1,\ldots\rangle)$	$\mathcal{G}_{\langle 1,2\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,1,\ldots\rangle)$

Figure 7: Example showing which subranges of mode I are assigned to a 2×3 processing grid using the distribution partitioning $D_{\langle\rangle}$. The top entry of each box designates the process in $\mathcal{G}$, and the bottom entry corresponds to the subrange assigned.

Example 4. *Let* $\overline{\mathbf{x}} = \langle 3, 6, 12, 9\rangle$ *and* $\overline{\mathbf{s}} = \langle 0, 2, 4, ...\rangle$. *Then*

$$\overline{\mathbf{x}}(\overline{\mathbf{s}}) = \langle 3, 12\rangle.$$

Let $\overline{\mathbf{s}} = \langle 3, 8, 1\rangle$. *Then*

$$\overline{\mathbf{x}}(\overline{\mathbf{s}}) = \langle 9, 6\rangle.$$

By filtering a range, we are creating a subrange of the mode. The elements assigned to a process in $\mathcal{G}$ are determined based on which subranges of modes are assigned to the process.

3.2 Distribution Partitionings

We are now ready to define the partitionings we will consider for assigning subranges of modes to processes of $\mathcal{G}$ when distributing elements of a tensor. We will first give examples of desired partitionings and then provide a definition for the general case.

3.3 Partitionings on an order-2 mesh

Consider a mode whose indices we wish to distribute, I, among processes of $\mathcal{G}$. Throughout this subsection, we view $\mathcal{G}$ as a $d_0 \times d_1$ mesh. We would like to allow for the case where I is replicated on all processes. This corresponds to all processes being assigned the subrange

$$\overline{\mathbf{r}}_I(\langle\mathbb{N}\rangle),$$

of mode I.

This is a useful distribution as it allows for the replication of elements among processes. However, if *all* modes were distributed in this manner, the tensor being distributed would be fully replicated on all processes which, in general, defeats the purpose of utilizing a distributed-memory architecture. Figure 7 provides a visual depiction of this distribution. To address this problem, we describe distributions that constrain the subranges assigned to processes based on the process's location in the grid.

Consider the case where we assign indices of I to processes in a round-robin fashion based solely on the process's location within the column-mode of $\mathcal{G}$. That is, given a process, $\mathcal{G}_{\langle p_0,p_1\rangle}$, we wish to assign it the subrange

$$\overline{\mathbf{r}}_I(\langle j \in \mathbb{N} : j \equiv p_0 \bmod d_0\rangle),$$

of mode I.

Notice that the difference between the previous two distributions is that the original distribution has been constrained based on a process's index of a mode of $\mathcal{G}$. We can similarly define a distribution where I

$\mathcal{G}_{\langle 0,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,2,4,\ldots\rangle)$	$\mathcal{G}_{\langle 0,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,2,4,\ldots\rangle)$	$\mathcal{G}_{\langle 0,2\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,2,4,\ldots\rangle)$
$\mathcal{G}_{\langle 1,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 1,3,5,\ldots\rangle)$	$\mathcal{G}_{\langle 1,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 1,3,5,\ldots\rangle)$	$\mathcal{G}_{\langle 1,2\rangle}$ $\overline{\mathbf{r}}_I(\langle 1,3,5,\ldots\rangle)$

(a)

$\mathcal{G}_{\langle 0,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,3,6,\ldots\rangle)$	$\mathcal{G}_{\langle 0,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 1,4,7,\ldots\rangle)$	$\mathcal{G}_{\langle 0,2\rangle}$ $\overline{\mathbf{r}}_I(\langle 2,5,8,\ldots\rangle)$
$\mathcal{G}_{\langle 1,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,3,6,\ldots\rangle)$	$\mathcal{G}_{\langle 1,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 1,4,7,\ldots\rangle)$	$\mathcal{G}_{\langle 1,2\rangle}$ $\overline{\mathbf{r}}_I(\langle 2,5,8,\ldots\rangle)$

(b)

Figure 8: Example showing which subranges of mode I are assigned to a 2×3 processing grid using the distribution partitioning (a) $D_{\langle 0\rangle}$ and (b) $D_{\langle 1\rangle}$. The top entry of each box designates the process in $\mathcal{G}$, and the bottom entry corresponds to the subrange assigned.

is distributed based on a process's location within the row-mode of $\mathcal{G}$. This distribution results in process $\mathcal{G}_{\langle p_0,p_1\rangle}$ being assigned

$$\overline{\mathbf{r}}_I(\langle j \in \mathbb{N} : j \equiv p_1 \bmod d_1\rangle),$$

of mode I.

Figure 8 provides a visual depiction of the previous two distributions. They are equivalent to the *matrix* distributions (different distributions used for each mode) used in Appendix A.

Already, we see a pattern emerge. Each filter used to assign a subrange to a process is parameterized by two pieces of information: the process's location in $\mathcal{G}$ and the modes of the grid used for creating the filter. We can generalize the previous two distributions with a single symbol,

$$D^{\langle n\rangle}_{\langle h\rangle} = \langle j \in \mathbb{N} : j \equiv n \bmod d_h\rangle\,,$$

where h is either 0 or 1 indicating that the process's column or row mode index is to be used (respectively) for filtering, and n is the process's location within that mode. Given these two pieces of information, the exact subrange to assign process $\mathcal{G}_{\langle p_0,p_1\rangle}$ is completely specified. As a shorthand, we use the symbol $D_{\langle h\rangle}$ to indicate that process $\mathcal{G}_{\langle p_0,p_1\rangle}$ is assigned the subrange according to the filter $D^{\langle p_0,p_1\rangle(\langle h\rangle)}_{\langle h\rangle}$. It should come as no surprise that to specify the distribution where process $\mathcal{G}_{\langle p_0,p_1\rangle}$ is assigned the entire range of I is denoted by $D_{\langle\rangle}$ (no modes of the grid are required to specify the filter, thus no constraints).

Finally, it is useful to define the distribution which assigns indices of I based on viewing $\mathcal{G}$ as a vector of processes (instead of a rectangular grid). Given a process $\mathcal{G}_{\langle p_0,p_1\rangle}$, if we wish to assign it the subrange of I according to a column-major ordering of process locations, we would use the filter

$$D^{\langle p_0,p_1\rangle}_{\langle 0,1\rangle} = \langle j \in \mathbb{N} : j \equiv (p_0 + p_1 d_0) \bmod (d_0 d_1)\rangle\,.$$

To define the filter according to a row-major vector view of the processing grid, we would assign the index-values according to the filter

$$D^{\langle p_1,p_0\rangle}_{\langle 1,0\rangle} = \langle j \in \mathbb{N} : j \equiv (p_1 + p_0 d_1) \bmod (d_1 d_0)\rangle\,.$$

$\mathcal{G}_{\langle 0,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,6,12,\ldots\rangle)$	$\mathcal{G}_{\langle 0,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 2,8,14,\ldots\rangle)$	$\mathcal{G}_{\langle 0,2\rangle}$ $\overline{\mathbf{r}}_I(\langle 4,10,16,\ldots\rangle)$
$\mathcal{G}_{\langle 1,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 1,7,13,\ldots\rangle)$	$\mathcal{G}_{\langle 1,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 3,9,15,\ldots\rangle)$	$\mathcal{G}_{\langle 1,2\rangle}$ $\overline{\mathbf{r}}_I(\langle 5,11,17,\ldots\rangle)$

(a)

$\mathcal{G}_{\langle 0,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,6,12,\ldots\rangle)$	$\mathcal{G}_{\langle 0,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 1,7,13,\ldots\rangle)$	$\mathcal{G}_{\langle 0,2\rangle}$ $\overline{\mathbf{r}}_I(\langle 2,8,14,\ldots\rangle)$
$\mathcal{G}_{\langle 1,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 3,9,15,\ldots\rangle)$	$\mathcal{G}_{\langle 1,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 4,10,16,\ldots\rangle)$	$\mathcal{G}_{\langle 1,2\rangle}$ $\overline{\mathbf{r}}_I(\langle 5,11,17,\ldots\rangle)$

(b)

Figure 9: Example showing which subranges of mode I are assigned to a 2×3 processing grid using the distribution partitioning $D_{\langle 0,1\rangle}$ (a) and $D_{\langle 1,0\rangle}$ (b). The top entry of each box designates the process in $\mathcal{G}$, and the bottom entry corresponds to the subrange assigned.

The previous two distributions are equivalent to the *column-major* and *row-major* vector distributions, respectively, introduced in Appendix A[4]. We summarize the overall distribution for each process being assigned the distribution according to a column-major vector or row-major vector view of the grid as $D_{\langle 0,1\rangle}$ or $D_{\langle 1,0\rangle}$ respectively. The order of the integers 0 and 1 in both $D_{\langle 0,1\rangle}$ and $D_{\langle 1,0\rangle}$ describes the order in which $\mathcal{G}$ should be traversed when creating the associated filter. The relevance of the ordering will become for important when generalizing to computing grids of higher-order (greater than order-2 here). Figure 9 provides a visual depiction of the previous two distribution partitionings.

So far, the only information explicitly needed by a process to indicate the desired distribution is the sequence of modes of the grid used to generate the filter (given by the subscript of the distribution symbol). Given this information, the assigned subrange to each process can be viewed as a vectorization of certain modes of the processing grid followed by a cyclic distribution of $\overline{\mathbf{r}}_I$. For those familiar with Elemental, the previously mentioned distributions are the exact distributions utilized by that package.

3.4 Partitionings on an order-m_g mesh

With the previous examples given, we can discuss how distributions can be described if $\mathcal{G}$ is viewed as a higher-modal mesh (instead of just two-modal). In general, the same pattern of expressing a sequence of modes of the grid to use in creating the filter exists; we simply have more modes to choose from. For example, consider an $d_0 \times d_1 \times d_2$ processing grid. If we specify the distribution $D_{\langle 1,0,2\rangle}$, this means that process $\mathcal{G}_{\langle p_0,p_1,p_2\rangle}$ is assigned the subrange

$$\overline{\mathbf{r}}_I\left(D_{\langle 1,0,2\rangle}^{\langle p_1,p_0,p_2\rangle}\right) \text{ where } D_{\langle 1,0,2\rangle}^{\langle p_1,p_0,p_2\rangle} = \langle j \in \mathbb{N} : j \equiv (p_0 + p_1 d_1 + p_2 d_1 d_0) \bmod (d_1 d_0 d_2)\rangle,$$

of mode I.

An example of this distribution partitioning is provided in Figure 10.

So far, we have assumed that the process grid and tensor are aligned; i.e., processes mapping to zero in the filter are always assigned the zeroth index of a mode. In some cases, it is useful to align the processes according to a different alignment parameter, σ. We consolidate the above patterns observed for an arbitrary-

[4]The notion of this kind of distribution of vectors was first introduced in [10] and is the key idea behind PLAPACK and subsequently Elemental.

$\mathcal{G}_{\langle 0,0,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 0,24,48,\ldots\rangle)$	$\mathcal{G}_{\langle 0,1,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 1,25,49,\ldots\rangle)$	$\mathcal{G}_{\langle 0,2,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 2,26,50,\ldots\rangle)$
$\mathcal{G}_{\langle 1,0,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 3,27,51,\ldots\rangle)$	$\mathcal{G}_{\langle 1,1,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 4,28,52,\ldots\rangle)$	$\mathcal{G}_{\langle 1,2,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 5,29,53,\ldots\rangle)$
$\mathcal{G}_{\langle 2,0,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 6,30,54,\ldots\rangle)$	$\mathcal{G}_{\langle 2,1,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 7,31,55,\ldots\rangle)$	$\mathcal{G}_{\langle 2,2,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 8,32,56,\ldots\rangle)$
$\mathcal{G}_{\langle 3,0,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 9,33,57,\ldots\rangle)$	$\mathcal{G}_{\langle 3,1,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 10,34,58,\ldots\rangle)$	$\mathcal{G}_{\langle 3,2,0\rangle}$ $\overline{\mathbf{r}}_I(\langle 11,35,59,\ldots\rangle)$

$\mathcal{G}_{\langle 0,0,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 12,36,60,\ldots\rangle)$	$\mathcal{G}_{\langle 0,1,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 13,37,61,\ldots\rangle)$	$\mathcal{G}_{\langle 0,2,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 14,38,62,\ldots\rangle)$
$\mathcal{G}_{\langle 1,0,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 15,39,63,\ldots\rangle)$	$\mathcal{G}_{\langle 1,1,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 16,40,64,\ldots\rangle)$	$\mathcal{G}_{\langle 1,2,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 17,41,65,\ldots\rangle)$
$\mathcal{G}_{\langle 2,0,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 18,42,66,\ldots\rangle)$	$\mathcal{G}_{\langle 2,1,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 19,43,67,\ldots\rangle)$	$\mathcal{G}_{\langle 2,2,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 20,44,68,\ldots\rangle)$
$\mathcal{G}_{\langle 3,0,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 21,45,69,\ldots\rangle)$	$\mathcal{G}_{\langle 3,1,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 22,46,70,\ldots\rangle)$	$\mathcal{G}_{\langle 3,2,1\rangle}$ $\overline{\mathbf{r}}_I(\langle 23,47,71,\ldots\rangle)$

Figure 10: Example showing which subranges of mode I are assigned to a $4\times 3\times 2$ processing grid using the distribution partitioning $D_{\langle 1,0,2\rangle}$. The top entry of each box designates the process in $\mathcal{G}$, and the bottom entry corresponds to the subrange assigned.

modal processing grid with the notion of a *vectorized distribution partitioning*. We define this now and give an explicit example.

Definition 7 (**Vectorized distribution partitioning**). *Given the mode we wish to distribute, I, a processing grid viewed as a $d_0\times d_1\times\cdots\times d_{m_g-1}$ mesh, $\mathcal{G}$, an arbitrary process's location within $\mathcal{G}$ specified by the sequence $\overline{\mathbf{p}}=\langle p_0,\ldots,p_{m_g-1}\rangle$, and a sequence of unique modes of $\mathcal{G}$, $\overline{\mathbf{x}}=\langle h_0,h_1,\ldots,h_k\rangle$ where $h_i\in\{0,\ldots,m_g-1\}$, $i\neq j\implies h_i\neq h_j$, and $0\leq k<m_g$, we define the sequence*

$$D_{\overline{\mathbf{x}}}^{\overline{\mathbf{p}}(\overline{\mathbf{x}})}=\left\langle j\in\mathbb{N}:j\equiv\left(\sum_{\ell_0=0}^{|\overline{\mathbf{x}}|-1}\left(\overline{\zeta}_{\ell_0}\prod_{\ell_1=0}^{\ell_0-1}d_{\overline{\chi}_{\ell_1}}\right)+\sigma\right)\bmod\left(\prod_{\ell=0}^{|\overline{\mathbf{x}}|-1}d_{\overline{\chi}_\ell}\right)\right\rangle \tag{6}$$

for some alignment parameter $0\leq\sigma<\left(\prod_{\ell=0}^{|\overline{\mathbf{x}}|-1}d_{\overline{\chi}_\ell}\right)$.

We call the set of all $D_{\overline{\mathbf{x}}}^{\overline{\mathbf{p}}(\overline{\mathbf{x}})}$, given any valid choice of $\overline{\mathbf{p}}$, a vectorized distribution partitioning. The symbol $D_{\overline{\mathbf{x}}}$ will refer to the distribution where process $\mathcal{G}_{\langle p_0,p_1,\ldots,p_{m_g-1}\rangle}$ is assigned the subrange $\overline{\mathbf{r}}_I\left(D_{\overline{\mathbf{x}}}^{\overline{\mathbf{p}}(\overline{\mathbf{x}})}\right)$ of mode I.

Example 5. *We wish to distribute a mode I of dimension 20 on a process grid arranged in a $2\times 4\times 3$ mesh according to the distribution $D_{\langle 0,2\rangle}$ with alignment parameter $\sigma=4$. Then, process $\overline{\mathbf{p}}=\langle 0,3,1\rangle$ is assigned the subrange $\langle 0,\ldots,19\rangle\left(D_{\langle 0,2\rangle}^{\langle 0,1\rangle}\right)$ of the mode I where*

$$\begin{aligned}D_{\overline{\mathbf{x}}}^{\overline{\mathbf{p}}(\overline{\mathbf{x}})}=D_{\langle 0,2\rangle}^{\langle 0,1\rangle}&=\left\langle j\in\mathbb{N}:j\equiv\left(\sum_{\ell_0=0}^{2-1}\left(\overline{\zeta}_{\ell_0}\prod_{\ell_1=0}^{\ell_0-1}d_{\overline{\chi}_{\ell_1}}\right)+\sigma\right)\bmod\left(\prod_{\ell=0}^{2-1}d_{\overline{\chi}_\ell}\right)\right\rangle\\&=\langle j\in\mathbb{N}:j\equiv(0+1*2)+4\bmod(2*3)\rangle\\&=\langle j\in\mathbb{N}:j\equiv 6\bmod 6\rangle\,.\end{aligned}$$

In other words, under the distribution $D_{\langle 0,2\rangle}$, process $\mathcal{G}_{\langle 0,3,1\rangle}$ is assigned the subrange $\langle 0,6,12,18\rangle$ of I.

We will refer to a vectorized distribution partitioning as a *partitioning*. Further, as a partitioning is defined as a set, we refer to an element of a distribution partitioning as a *partition*. When all modes are distributed based on some partitioning, we say the tensor has been *distributed* based on a *distribution* (specified by how

all modes are distributed).

3.5 Distributions

Now that we have introduced the notion of a partitioning, we are ready to specify what conditions must be satisfied for a distribution of a tensor to be considered valid.

Given an order-m tensor,

$$\boldsymbol{\mathcal{A}}^{\boldsymbol{\iota}_0\boldsymbol{\iota}_1\cdots\boldsymbol{\iota}_{m-1}} \in \mathbb{R}^{L_{\iota_0}\times L_{\iota_1}\times\cdots\times L_{\iota_{m-1}}},$$

we wish to distribute among processes of $\mathcal{G}$ arranged as an $d_0 \times d_1 \times \cdots \times d_{m_g-1}$ mesh; we consider

$$\boldsymbol{\mathcal{A}}^{\boldsymbol{\iota}_0\boldsymbol{\iota}_1\cdots\boldsymbol{\iota}_{m-1}}\left(D_{\overline{\mathbf{x}}_0}, D_{\overline{\mathbf{x}}_1}, \ldots, D_{\overline{\mathbf{x}}_{m-1}}\right)$$

where $D_{\overline{\mathbf{x}}_k}$ is a vectorized distribution partitioning as defined previously, a valid distribution if

1. $\forall_{i,j\in\{0,\ldots,k-1\}}\forall_{\ell_0\in\overline{\mathbf{x}}_i}\forall_{\ell_1\in\overline{\mathbf{x}}_j}\ell_0 \neq \ell_1$ (No two partitionings share the same element)
2. $\forall_{\ell\in\{0,\ldots,m-1\}}\overline{\chi}_{\ell i} \neq \overline{\chi}_{\ell j}$ when $i \neq j$ (No element duplicated in any partitioning)
3. $\forall_{\ell\in\{0,\ldots,m-1\}}\forall_{i\in\{0,\ldots,|\overline{\mathbf{x}}_\ell|-1\}}\overline{\chi}_{\ell i} \in \{0,\ldots,m_g-1\}$. (All elements of partitionings valid)

If the above conditions hold, every element of $\boldsymbol{\mathcal{A}}$ will be assigned to at least one (perhaps many) process(es) in $\mathcal{G}$. The proof of this is left as future work.

The set of elements of $\boldsymbol{\mathcal{A}}$ under some distribution assigned to each process in $\mathcal{G}$ is the set of elements whose index within each mode of $\boldsymbol{\mathcal{A}}$ is an element of the assigned subrange of the corresponding mode. Another way to view this is the set of elements assigned to each process is defined as the set of elements whose location in $\boldsymbol{\mathcal{A}}$ is an element of the cross-product of subranges assigned to that process (preserving the same ordering of modes as the object being distributed). An example showing what elements of a matrix distributed on $\mathcal{G}$ viewed as an order-2 mesh is given below.

Example 6. *Consider a matrix, $\mathbf{A}^{MN} \in \mathbb{R}^{6\times 9}$, we wish to distribute on a processing grid, $\mathcal{G}$, viewed as a 2×3 mesh. Given that we wish to distribute $\mathbf{A}$ as*

$$\mathbf{A}^{MN}(D_{\langle 0\rangle}, D_{\langle 1\rangle}),$$

then the elements of $\mathbf{A}$ are distributed among processes according to the diagram

$\mathcal{G}_{\langle 0,0\rangle}$			$\mathcal{G}_{\langle 0,1\rangle}$			$\mathcal{G}_{\langle 0,2\rangle}$		
α_{00}	α_{03}	α_{06}	α_{01}	α_{04}	α_{07}	α_{02}	α_{05}	α_{08}
α_{20}	α_{23}	α_{26}	α_{21}	α_{24}	α_{27}	α_{22}	α_{25}	α_{28}
α_{40}	α_{43}	α_{46}	α_{41}	α_{44}	α_{47}	α_{42}	α_{45}	α_{48}
$\mathcal{G}_{\langle 1,0\rangle}$			$\mathcal{G}_{\langle 1,1\rangle}$			$\mathcal{G}_{\langle 1,2\rangle}$		
α_{10}	α_{13}	α_{16}	α_{11}	α_{14}	α_{17}	α_{12}	α_{05}	α_{18}
α_{30}	α_{33}	α_{36}	α_{31}	α_{34}	α_{37}	α_{32}	α_{25}	α_{38}
α_{40}	α_{53}	α_{56}	α_{51}	α_{54}	α_{57}	α_{52}	α_{45}	α_{58}

where $\mathcal{G}_{\langle p_0,p_1\rangle}$ corresponds to the process at location $\langle p_0, p_1\rangle$ in $\mathcal{G}$. Notice that process $\mathcal{G}_{\langle p_0,p_1\rangle}$ is assigned elements whose indices of mode M are in $\overline{\mathbf{r}}_M\left(D_{\langle 0\rangle}^{\langle p_0,p_1\rangle(\langle 0\rangle)}\right)$ and whose indices of mode N are in $\overline{\mathbf{r}}_N\left(D_{\langle 1\rangle}^{\langle p_0,p_1\rangle(\langle 1\rangle)}\right)$. This is exactly what is defined by the distribution $\mathbf{A}^{MN}(D_{\langle 0\rangle}, D_{\langle 1\rangle})$.

3.6 Canonical Distributions

We say a distribution is *canonical* if no duplication of elements among processes exists. Consider an order-m tensor, $\mathcal{A}$, we wish to distribute among processes of $\mathcal{G}$ viewed as an m_g-modal mesh. If $m > m_g$, then any valid distribution is canonical (fewer modes in $\mathcal{G}$ relative to modes in the object ensure no duplication of data).

If $m \leq m_g$, then any distribution involving all modes of $\mathcal{G}$ is canonical. For example, when distributing an order-2 tensor on $\mathcal{G}$ viewed as an order-3 mesh, distributions such as

$$\mathbf{A}^{IJ}(D_{\langle 0\rangle}, D_{\langle 1,2\rangle}),$$
$$\mathbf{A}^{IJ}(D_{\langle 0,1,2\rangle}, D_{\langle\rangle})$$

are canonical, but

$$\mathbf{A}^{IJ}(D_{\langle 0,1\rangle}, D_{\langle\rangle})$$

is not canonical (not all modes of $\mathcal{G}$ are used to describe the distribution).

When discussing how to derive algorithms for tensor contractions, we maintain the constraint that at the beginning of the algorithm, input and final output objects must be in canonical distributions. Additionally, any final output objects must be in canonical distributions at the end of the algorithm. This property is held if we first assign a single mode of $\mathcal{G}$ to each mode of the object in increasing order. If the order of the object is *greater* than the order of $\mathcal{G}$, $D_{\langle\rangle}$ partitionings are used once all modes of the grid have been exhausted. Consider mapping an order-3 tensor on a order-2 grid. The canonical distribution we use is

$$\boldsymbol{\mathcal{A}}^{IJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}).$$

If the order of the object is *less* than the order of $\mathcal{G}$, all unused modes of $\mathcal{G}$ are appended to the distribution partitioning assigned to the last mode of the object, i.e., for mapping a order-2 tensor to an order-3 processing grid, we will use the distribution

$$\mathbf{A}^{IJ}(D_{\langle 0\rangle}, D_{\langle 1,2\rangle}).$$

The necessary proofs for statements treated as fact is left as future work.

3.7 Elemental Distributions: Subset of the proposed

Elemental defines a set of distributions for matrices which can be viewed as special cases of the proposed notation. One assumption that Elemental makes for distributing matrices is that the mesh used is viewed as a 2-D mesh. In the following table, we show how the distributions used in Elemental are expressed in the proposed notation by relating the assigned subranges to processes.

Elemental distribution	Assigned subrange to $\mathcal{G}_{\langle p_0,p_1\rangle}$	Proposed distribution
$*$	$\langle j \in \mathbb{N}\rangle$	$D_{\langle\rangle}$
M_C	$\langle j \in \mathbb{N} : j \equiv p_0 \bmod d_0\rangle$	$D_{\langle 0\rangle}$
M_R	$\langle j \in \mathbb{N} : j \equiv p_1 \bmod d_1\rangle$	$D_{\langle 1\rangle}$
V_C	$\langle j \in \mathbb{N} : j \equiv (p_0 + p_1 d_0) \bmod (d_0 d_1)\rangle$	$D_{\langle 01\rangle}$
V_R	$\langle j \in \mathbb{N} : j \equiv (p_1 + p_0 d_1) \bmod (d_0 d_1)\rangle$	$D_{\langle 10\rangle}$

4 Redistributions

In the previous section, we described valid distributions for distributing an order-m tensor, $\mathcal{A}$, to a process grid arranged as an order-m_g mesh. In this section, we describe how, by design, data of $\mathcal{A}$ is redistributed on this mesh from one distribution to another utilizing the collective communications listed in Figure 5. Each subsection describes a different collective communication. The headers of each subsection should be

interpreted as defining the collective communication discussed, along with how distribution partitioning symbols of a valid distribution are affected. This is done for brevity; descriptions of how entire distributions are affected by the communication are provided within the subsection.

We will restrict our proposed language to only include the redistributions described in this section. Further, in this section we assume the alignment parameter for determining mode mappings is zero, that is $\sigma = 0$, an assumption that is easily lifted.

4.1 Allgather: $D_{\overline{\mathbf{y}}} \leftarrow D_{\overline{\mathbf{y}} \sqcup \langle h_\ell \rangle}$

Recall that the distribution of an order-m tensor is of the form

$$\boldsymbol{\mathcal{A}}^{\boldsymbol{l}_0 \cdots \boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{x}}_0}, D_{\overline{\mathbf{x}}_1}, \ldots, D_{\overline{\mathbf{x}}_{m-1}}).$$

In this subsection, we are concerned with redistributions of the form

$$\boldsymbol{\mathcal{A}}^{\boldsymbol{l}_0 \cdots \boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{x}}_0}, \ldots, D_{\overline{\mathbf{x}}_{k-1}}, D_{\overline{\mathbf{y}}}, D_{\overline{\mathbf{x}}_{k+1}}, \ldots, D_{\overline{\mathbf{x}}_{m-1}}) \leftarrow \boldsymbol{\mathcal{A}}^{\boldsymbol{l}_0 \cdots \boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{x}}_0}, \ldots, D_{\overline{\mathbf{x}}_{k-1}}, D_{\overline{\mathbf{y}} \sqcup \langle h_\ell \rangle}, D_{\overline{\mathbf{x}}_{k+1}}, \ldots, D_{\overline{\mathbf{x}}_{m-1}}).$$

Notice that the only change in the distribution of $\boldsymbol{\mathcal{A}}$ is that the distribution partitioning assigned to mode $\boldsymbol{l}_k$ has changed to $D_{\overline{\mathbf{y}}}$ from $D_{\overline{\mathbf{y}} \sqcup \langle h_\ell \rangle}$.

In this case, we are redistributing elements of $\boldsymbol{\mathcal{A}}$ such that process $\mathcal{G}_{\langle p_0, p_1, \ldots, p_{m_g - 1} \rangle}$ initially stores the subrange of $\boldsymbol{l}_k$ according to the distribution partition

$$D^{\overline{\mathbf{z}}}_{\overline{\mathbf{y}} \sqcup \langle h_\ell \rangle} = \left\langle j \in \mathbb{N} : j \equiv \left(\sum_{\ell_0 = 0}^{|\overline{\mathbf{y}}| - 1} \left(\overline{\zeta}_{\ell_0} \prod_{\ell_1 = 0}^{\ell_0 - 1} d_{\overline{\psi}_{\ell_1}} \right) + \overline{\zeta}_{|\overline{\mathbf{y}}|} \prod_{\ell_1 = 0}^{|\overline{\mathbf{y}}| - 1} d_{\overline{\psi}_{\ell_1}} \right) \bmod \left(d_{h_\ell} \prod_{\ell_1 = 0}^{|\overline{\mathbf{y}}| - 1} d_{\overline{\psi}_{\ell_1}} \right) \right\rangle,$$

where $\overline{\mathbf{z}} = \langle p_0, \ldots, p_{m_g - 1} \rangle (\overline{\mathbf{y}} \sqcup \langle h_\ell \rangle)$. Performing an allgather collective within the h_ℓ mode of the process grid does not affect any partitionings besides $D_{\overline{\mathbf{y}} \sqcup \langle h_\ell \rangle}$ as no other partitionings involve the h_ℓ mode of $\mathcal{G}$.

An allgather communication within the h_ℓ mode of the process grid will ensure that at the end of the communication process $\mathcal{G}_{\langle p_0, p_1, \ldots, p_{m_g - 1} \rangle}$ is assigned the subrange of $\boldsymbol{l}_k$ defined by

$$\bigcup_{\ell_0 = 0}^{d_{h_\ell} - 1} D^{\overline{\mathbf{z}}_{\ell_0}}_{\overline{\mathbf{y}} \sqcup \langle h_\ell \rangle},$$

where $\overline{\mathbf{z}}_i = \langle p_0, \ldots, p_{h_\ell - 1}, i, p_{h_\ell + 1}, \ldots, p_{m_g - 1} \rangle (\overline{\mathbf{y}} \sqcup \langle h_\ell \rangle)$.

We recognize the resulting distribution partitioning as $D^{\overline{\mathbf{z}}}_{\overline{\mathbf{y}}}$ where $\overline{\mathbf{z}} = \langle p_0, \ldots, p_{h_\ell - 1}, i, p_{h_\ell + 1}, \ldots, p_{m_g - 1} \rangle (\overline{\mathbf{y}})$ meaning the final distribution of the tensor is

$$\boldsymbol{\mathcal{A}}^{\boldsymbol{l}_0 \cdots \boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{x}}_0}, \ldots, D_{\overline{\mathbf{x}}_{k-1}}, D_{\overline{\mathbf{y}}}, D_{\overline{\mathbf{x}}_{k+1}}, \ldots, D_{\overline{\mathbf{x}}_{m-1}}).$$

For a proof, see Appendix B.

4.2 Scatter: $D_{\overline{\mathbf{x}} \sqcup \langle h_\ell \rangle} \leftarrow D_{\overline{\mathbf{x}}}$

Consider the order-m tensor $\boldsymbol{\mathcal{A}}^{\boldsymbol{l}_0 \cdots \boldsymbol{l}_{m-1}}$ distributed as

$$\boldsymbol{\mathcal{A}}^{\boldsymbol{l}_0 \cdots \boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{x}}_0}, \cdots, D_{\overline{\mathbf{x}}_{m-1}}).$$

Given a mode we wish to scatter along, h_ℓ, the result of scattering mode $\boldsymbol{l}_k$ is

$$\boldsymbol{\mathcal{A}}^{\boldsymbol{l}_0 \cdots \boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{x}}_0}, \ldots, D_{\overline{\mathbf{x}}_k \sqcup \langle h_\ell \rangle}, D_{\overline{\mathbf{x}}_{k+1}}, \ldots, D_{\overline{\mathbf{x}}_{m-1}}).$$

Beginning with a valid distribution, this communication is optimized by performing a local rearrangement of data as the elements required in the resulting distribution are a subset of those already stored by the

process. Having the scatter semantics clearly defined will help us later in describing the semantics of the reduce-scatter operation. In particular, scatter operations are useful when the entire tensor is stored on a single, lower order, hyper-plane of $\mathcal{G}$. An example of this is when all entries of a matrix are stored in a single column of processes in a 2-D process grid.

4.3 All-to-all: $(D_{\overline{\mathbf{x}}\sqcup\langle h_\ell\rangle}, D_{\overline{\mathbf{y}}}) \leftarrow (D_{\overline{\mathbf{x}}}, D_{\overline{\mathbf{y}}\sqcup\langle h_\ell\rangle})$

An all-to-all collective within mode h_ℓ of $\mathcal{G}$ can be viewed as performing an allgather followed by a scatter. Thus, an all-to-all communication within mode h_ℓ of the processing grid redistributes the data such that the symbol pair change

$$(D_{\overline{\mathbf{x}}\sqcup\langle h_\ell\rangle}, D_{\overline{\mathbf{y}}}) \leftarrow (D_{\overline{\mathbf{x}}}, D_{\overline{\mathbf{y}}\sqcup\langle h_\ell\rangle})$$

occurs.

4.4 Reduce-scatter: $\mathcal{B}^{\boldsymbol{l}_0}(D_{\overline{\mathbf{x}}\sqcup\langle h_\ell\rangle}) \leftarrow \mathcal{A}^{\boldsymbol{l}_0\boldsymbol{l}_1}(D_{\overline{\mathbf{x}}}, D_{\langle h_\ell\rangle})$

The goal of a reduction is to eliminate a mode of the distributed tensor tensor by accumulating elements. In this subsection, we first outline the effect of eliminating, via reduction, a mode of our tensor with a single reduce-scatter operation, and next describe the effect of performing the reduction using a multi-step version; the latter being viewed as a generalization of the former.

4.4.1 Single-step reduce-scatter mode elimination

Here, we are concerned with the global operation:

$$\mathcal{B}^{\boldsymbol{l}_0\ldots\boldsymbol{l}_{k-1}\boldsymbol{l}_{k+1}\ldots\boldsymbol{l}_{m-1}} = \sum_{\boldsymbol{l}_k} \mathcal{A}^{\boldsymbol{l}_0\ldots\boldsymbol{l}_{k-1}\boldsymbol{l}_k\boldsymbol{l}_{k+1}\ldots\boldsymbol{l}_{m-1}} \tag{7}$$

defined element-wise as

$$\beta_{i_0\ldots i_{k-1}i_{k+1}\ldots i_{m-1}} = \sum_{i_k=0}^{L_{\boldsymbol{l}_k}-1} \alpha_{i_0\ldots i_{k-1}i_k i_{k+1}\ldots i_{m-1}}.$$

As an example of this operation, consider the case where we wish to reduce the columns of a matrix together (forming a vector); i.e., given a matrix, $\mathbf{A}$, partitioned by columns,

$$\mathbf{A} = \left(\begin{array}{c|c|c|c} \mathbf{a}_0 & \mathbf{a}_1 & \ldots & \mathbf{a}_{k-1} \end{array}\right),$$

we wish to compute the vector

$$\mathbf{b} = \sum_{\ell=0}^{k-1} \mathbf{a}_\ell.$$

We can express this operation as

$$\mathbf{b}^I = \sum_K \mathbf{A}^{IK}.$$

Given $\mathcal{A}^{\boldsymbol{l}_0\ldots\boldsymbol{l}_{k-1}\boldsymbol{l}_k\boldsymbol{l}_{k+1}\ldots\boldsymbol{l}_{m-1}}$ distributed as

$$\mathcal{A}^{\boldsymbol{l}_0\ldots\boldsymbol{l}_{k-1}\boldsymbol{l}_k\boldsymbol{l}_{k+1}\ldots\boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{x}}_0}, \ldots, D_{\overline{\mathbf{x}}_{k-1}}, D_{\langle h_\ell\rangle}, D_{\overline{\mathbf{x}}_{k+1}}, \ldots, D_{\overline{\mathbf{x}}_{m-1}}),$$

we can form the appropriate output tensor $\mathcal{B}$ by performing a reduction in the h_ℓ mode of $\mathcal{G}$. We implement the reduction with a reduce-scatter collective. As a reduction is implemented as a reduce-scatter, we must

scatter the data resulting from the reduction. As we can choose how to arrange data to be scattered, the distribution associated with $\mathcal{B}$ is flexible. By arranging the data correctly, we can, in effect, scatter one of the modes remaining after the reduction. This means the resulting distribution of $\mathcal{B}$ is

$$\mathcal{B}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{y}}_0}, \dots, D_{\overline{\mathbf{y}}_{k-1}}, D_{\overline{\mathbf{y}}_{k+1}}, \dots, D_{\overline{\mathbf{y}}_{m-1}}),$$

where

$$\overline{\mathbf{y}}_j = \begin{cases} \overline{\mathbf{x}}_j \sqcup \overline{\mathbf{x}}_k & \text{if } j = \ell \\ \overline{\mathbf{x}}_j & \text{otherwise} \end{cases}$$

for some $(0 \leq \ell < m) \wedge (\ell \neq k)$.

4.4.2 Multi-step reduce-scatter mode elimination

Previously, we showed how a single reduce-scatter collective can be utilized to fully eliminate a mode of a tensor via reduction. Recall, the operation we are computing is a reduction of mode $\boldsymbol{l}_k$ expressed as

$$\mathcal{B}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}} = \sum_{\boldsymbol{l}_k} \mathcal{A}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}_k \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}},$$

where

$$\beta_{i_0 \dots i_{k-1} i_{k+1} \dots i_{m-1}} = \sum_{i_k=0}^{L_{\boldsymbol{l}_k}-1} \alpha_{i_0 \dots i_{k-1} i_k i_{k+1} \dots i_{m-1}}.$$

Expressed in this form, this performs the reduction in one step. By introducing a temporary $\mathcal{T}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}'_k \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}}$ where $L_{\boldsymbol{l}'_k} \setminus L_{\boldsymbol{l}_k}$, we could write the reduction in two steps, first forming $\mathcal{T}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}'_k \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}}$ defined element-wise as

$$\tau_{i_0 \dots i_{k-1} i'_k i_{k+1} \dots i_{m-1}} = \sum_{i_k = f(i'_k)}^{f(i'_k+1)-1} \alpha_{i_0 \dots i_{k-1} i_k i_{k+1} \dots i_{m-1}},$$

where $f(x) = (x+1)\frac{L_{\boldsymbol{l}_k}}{L_{\boldsymbol{l}'_k}}$, and then computing $\mathcal{B}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}}$ via a single reduce-scatter

$$\mathcal{B}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}} = \sum_{\boldsymbol{l}'_k} \mathcal{T}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}'_k \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}}.$$

In effect, the first reduction forms an intermediate that captures partial contributions to the final result. The second reduction then accumulates the partial results together to compute the final result. We can describe this two-step process as

$$\begin{aligned} \mathcal{T}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}^{(1)}_k \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}} &= \sum_{\boldsymbol{l}_k} \mathcal{A}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}_k \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}} \\ \mathcal{B}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}} &= \sum_{\boldsymbol{l}^{(1)}_k} \mathcal{T}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}^{(1)}_k \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}}, \end{aligned}$$

where $\boldsymbol{l}^{(1)}_k$ indicates that $\mathcal{T}^{\boldsymbol{l}_0 \dots \boldsymbol{l}_{k-1} \boldsymbol{l}^{(1)}_k \boldsymbol{l}_{k+1} \dots \boldsymbol{l}_{m-1}}$ represents partial contributions and requires further reduction to eliminate $\boldsymbol{l}^{(1)}_k$. Although this is technically not in the correct form (as the first reduction does not reduce over all indices of $\boldsymbol{l}_k$), we feel the meaning is understood as the output has a partially-reduced mode.

In general, we can view one single reduction as a series of reductions, which eventually will result in a tensor with an eliminated mode. For instance, we can view the reduction of $\boldsymbol{l}_k$ in $\mathcal{A}$ as the following series of reductions

$$
\begin{aligned}
\mathcal{T}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}_k^{(1)} \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}} &= \sum_{\boldsymbol{l}_k} \mathcal{A}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}_k \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}} \\
\mathcal{T}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}_k^{(2)} \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}} &= \sum_{\boldsymbol{l}_k^{(1)}} \mathcal{T}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}_k^{(1)} \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}} \\
&\vdots \\
\mathcal{B}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}} &= \sum_{\boldsymbol{l}_k^{(n)}} \mathcal{T}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}_k^{(n)} \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}}.
\end{aligned}
$$

We use the notation $\boldsymbol{l}_k^{(j)}$ to indicate that a mode of a tensor represents partial contributions associated with performing j reduction steps and is intended for elimination from the tensor via reduction. For brevity, we summarize all $\boldsymbol{l}_k^{(j)}$ modes with $\boldsymbol{l}'_k$, where it is understood that computational progress is made if there is no difference in inputs and outputs.

When viewing a reduction as a multi-step process, it is important to note that the number of useful reductions able to be performed is bounded by the length of the original mode being reduced. One can continue performing reduction steps past this point, however no new progress will be made.

Putting this together, given the tensor $\mathcal{A}$ distributed as

$$
\mathcal{A}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}_k \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{x}}_0}, \ldots, D_{\overline{\mathbf{x}}_{k-1}}, D_{\overline{\mathbf{x}}_k \sqcup \langle h_\ell \rangle}, D_{\overline{\mathbf{x}}_{k+1}}, \ldots, D_{\overline{\mathbf{x}}_{m-1}}),
$$

and performing the series of operations

$$
\begin{aligned}
\mathcal{T}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}'_k \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}} &= \sum_{\boldsymbol{l}_k} \mathcal{A}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}_k \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}} \\
\mathcal{B}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}} &= \sum_{\boldsymbol{l}'_k} \mathcal{T}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}'_k \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}},
\end{aligned}
$$

we know the form of the distribution of $\mathcal{B}$ as we must ensure that the semantics of this multi-step approach, at least, meet the semantics of performing a single-step reduction. In fact, the multi-step approach is more flexible than the single-step approach. However, for now, let us restrict ourselves to meeting the semantics of the single-step reduction.

We are now tasked with determining how $\mathcal{T}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}'_k \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}}$ is distributed. Then, $\mathcal{T}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}'_k \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}}$ must be distributed as

$$
\mathcal{T}^{\boldsymbol{l}_0 \ldots \boldsymbol{l}_{k-1} \boldsymbol{l}'_k \boldsymbol{l}_{k+1} \ldots \boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{y}}_0}, \ldots, D_{\overline{\mathbf{y}}_{k-1}}, D_{\overline{\mathbf{y}}_k}, D_{\overline{\mathbf{y}}_{k+1}}, \ldots, D_{\overline{\mathbf{y}}_{m-1}})
$$

where

$$
\overline{\mathbf{y}}_j = \begin{cases} \overline{\mathbf{x}}_j \sqcup \overline{\mathbf{x}}_k & \text{if } j = \ell \\ \overline{\mathbf{x}}_j & \text{otherwise} \end{cases}
$$

for some $(0 \leq \ell < m)$.

The important distinction between this definition and the definition in the single-step approach is that here the distribution of $\mathcal{T}$ can be the same as the distribution of $\mathcal{A}$. The difference is that here a mode of smaller dimension has replaced the mode we originally wished to eliminate. Another way to view this is that here computation, instead of redistribution of data, indicates progress of the algorithm. It is because of this added generality, relative to the single-step approach, that we previously stated that the multi-step approach is more flexible than the single-step approach for reducing a mode. Further, as this statement applies to all multi-step reductions, and each reduction can be viewed as a multi-step reduction, the space of possible distributions of the final output, $\mathcal{B}$, increases dramatically.

4.5 Elemental Redistributions: The order-2 case of the proposed

In the following table, we show how all of the redistributions in Elemental can be expressed as redistributions given in this paper. When translating from Elemental notation to the proposed, in general, the symbol C in Elemental maps to 0 in the proposed notation and R maps to 1. The table is templatized by a pair of variables x and y such that $x \neq y$ and $x, y \in \{0, 1, C, R\}$. When interpreted in the "Elemental Redistribution" column, $x, y \in \{C, R\}$. When interpreted in the "Proposed Redistribution" column, $x, y \in \{0, 1\}$.

Elemental Redistribution	Collective used	Proposed Redistribution
$(M_x, M_y) \leftrightarrow (M_x, *)$	$\leftarrow$ reduce-scatter, $\rightarrow$ allgather	$(D_{\langle x \rangle}, D_{\langle y \rangle}) \leftrightarrow (D_{\langle x \rangle}, D_{\langle\rangle})$
$(M_x, M_y) \leftrightarrow (*, M_y)$	$\leftarrow$ reduce-scatter, $\rightarrow$ allgather	$(D_{\langle x \rangle}, D_{\langle y \rangle}) \leftrightarrow (D_{\langle\rangle}, D_{\langle y \rangle})$
$(V_x, *) \leftrightarrow (M_x, *)$	$\leftarrow$ reduce-scatter, $\rightarrow$ allgather	$(D_{\langle xy \rangle}, D_{\langle\rangle}) \leftrightarrow (D_{\langle x \rangle}, D_{\langle\rangle})$
$(V_x, *) \leftrightarrow (M_x, M_y)$	$\leftrightarrow$ all-to-all	$(D_{\langle xy \rangle}, D_{\langle\rangle}) \leftrightarrow (D_{\langle x \rangle}, D_{\langle y \rangle})$
$(V_x, *) \leftrightarrow (V_y, *)$	$\leftrightarrow$ peer-to-peer	$(D_{\langle xy \rangle}, D_{\langle\rangle}) \leftrightarrow (D_{\langle yx \rangle}, D_{\langle\rangle})$
$(M_x, *) \leftrightarrow (*, *)$	$\leftarrow$ reduce-scatter, $\rightarrow$ allgather	$(D_{\langle x \rangle}, D_{\langle\rangle}) \leftrightarrow (D_{\langle\rangle}, D_{\langle\rangle})$
$(*, M_x) \leftrightarrow (*, *)$	$\leftarrow$ reduce-scatter, $\rightarrow$ allgather	$(D_{\langle\rangle}, D_{\langle x \rangle}) \leftrightarrow (D_{\langle\rangle}, D_{\langle\rangle})$
$(M_x, *) \leftrightarrow (*, M_x)$	$\leftrightarrow$ all-to-all	$(D_{\langle x \rangle}, D_{\langle\rangle}) \leftrightarrow (D_{\langle\rangle}, D_{\langle x \rangle})$

The last line in the above table may be unfamiliar to the reader. This redistribution is omitted in Elemental's list of redistributions, although it is valid.

5 Systematically deriving algorithms for tensor contractions

Up to this point, we have defined valid distributions and redistributions of our tensors. We have not commented on how these are used in deriving correct algorithms for general tensor contractions. In this section, we explain how one can derive different algorithms targeted for situations where exactly one tensor involved in the contraction is considered "large" and the other two "small". Each algorithm will avoid communicating the "large" tensor at the expense of communicating the "small" tensors to reduce the time devoted to communication. If the reader is familiar with Schatz et al. [22], these algorithms are referred to as the different "stationary" algorithmic variants.

We view the derivation process as a propagation of constraints. We will begin with fixed constraints on our objects; then by continually applying additional constraints, we will eventually arrive at a valid algorithm for the contraction.

One rule that we will not break during this propagation is that *during local computation, all modes, regardless of the tensor in which they appear, must be distributed in the same manner.* This is done to ensure correctness of the algorithms.

We will use the example given in the introduction as motivation for this section. Recall the operation in question was

$$\mathcal{X}^{ABJI} \mathrel{+}= \mathcal{V}^{ACIK} \mathcal{T}^{BCJK}.$$

For simplicity, we will describe algorithms for the equivalent operation

$$\mathcal{C}^{ABJI} \mathrel{+}= \mathcal{A}^{ACIK} \mathcal{B}^{BCJK}.$$

When deriving algorithms, we will assume our processing grid, $\mathcal{G}$, is of the same order as the highest-order tensor involved in the operation. We conjecture that with this constraint, we can ensure scalability in our algorithms.

5.1 Stationary C: Avoid communicating $\mathcal{C}$

For this algorithm, we wish to avoid communicating $\mathcal{C}$. Further, we would like an algorithm of the form:

Prep	$\mathcal{A}^{ACIK}(?,?,?,?) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(?,?,?,?) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(?,?,?,?) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Compute	$\mathcal{C}^{ABJI}(?,?,?,?) \mathrel{+}= \mathcal{A}^{ACIK}(?,?,?,?)\mathcal{B}^{BCJK}(?,?,?,?)$

In our algorithm templates, "?" indicates an unknown entry we must be resolved before we can arrive at a complete algorithm. In the Prep phase, we redistribute each tensor and prepare it for computation in the Compute phase. By definition, no communication of data occurs when transitioning from the Prep phase to the Compute phase.

As we wish not to communicate $\mathcal{C}$, we shouldn't redistribute $\mathcal{C}$, meaning the redistribution performed in the Prep phase is equivalent to the identity transformation. This implies the following modifications can be made to the template

Prep	$\mathcal{A}^{ACIK}(?,?,?,?) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(?,?,?,?) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Compute	$\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \mathrel{+}= \mathcal{A}^{ACIK}(?,?,?,?)\mathcal{B}^{BCJK}(?,?,?,?)$

As all modes, regardless of tensor, must be distributed in the same way during local computation, we can perform the following substitutions

Prep	$\mathcal{A}^{ACIK}(D_{\langle 0\rangle}, ?, D_{\langle 3\rangle}, ?) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(D_{\langle 1\rangle}, ?, D_{\langle 2\rangle}, ?) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Compute	$\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \mathrel{+}= \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, ?, D_{\langle 3\rangle}, ?)\mathcal{B}^{BCJK}(D_{\langle 1\rangle}, ?, D_{\langle 2\rangle}, ?)$

We are now tasked with determining how to distribute the remaining modes. Recall that we must ensure that no two modes use the same modes of the processing grid for distribution. Examining our template in its current state, we see all modes of $\mathcal{G}$ have been used to distribute modes. Therefore, we must assign the distribution partitioning $D_{\langle\rangle}$ to all remaining modes in the computation phase (and by extension in the Prep phase as well). This results in the following template

Prep	$\mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 2\rangle}, D_{\langle\rangle}) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Compute	$\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \mathrel{+}= \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle})\mathcal{B}^{BCJK}(D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 2\rangle}, D_{\langle\rangle})$

This leaves us with our final algorithm, which can be summarized as

1. Redistribute tensors $\mathcal{A}$ and $\mathcal{B}$
2. Perform local tensor contraction along the modes K and C.

A cost analysis of the above algorithm is given in Appendix C.1.

5.2 Stationary A: Avoid communicating $\mathcal{A}$

Now, we look at the same operation, but we wish to avoid communicating $\mathcal{A}$. As $\mathcal{A}$ contains some modes involved in the contraction which will be distributed in some way, we will have to perform a reduction to

compute the final results; otherwise we only have partially computed the final values. We use the notation $\widehat{\sum_h}$ to indicate a reduction across mode h of the processing grid. As we know we must perform a reduction after local computation, we would like an algorithm based on the following template:

Prep	$\mathcal{A}^{ACIK}(?,?,?,?) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(?,?,?,?) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(?,?,?,?) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Product	$\mathcal{T}^{AC'IK'BJ}(?,?,?,?,?,?) \mathrel{+}= \mathcal{A}^{ACIK}(?,?,?,?)\mathcal{B}^{BCJK}(?,?,?,?)$
Reduce	$\mathcal{T}^{AIK'BJ}(?,?,?,?,?) = \widehat{\sum_?}\mathcal{T}^{AC'IK'BJ}(?,?,?,?,?,?)$ $\mathcal{C}^{ABJI}(?,?,?,?) = \widehat{\sum_?}\mathcal{T}^{AIK'BJ}(?,?,?,?,?)$

Again, by definition, no communication occurs across boundaries of phases. In the above template, the tensors $\mathcal{T}^{AC'IK'BJ}$ and $\mathcal{T}^{AIK'BJ}$ represent intermediate objects which store the partial results of the overall computation. The order of their modes and reductions was arbitrarily chosen. We have two reduction steps since the global operation represents the contraction over two modes. To ensure correctness of our resulting algorithm, we must set each element of $\mathcal{T}^{AC'IK'BJ}$ to

$$\tau_{i_0 i_1 i_2 i_3 i_4 i_5} = \begin{cases} \gamma_{i_0 i_4 i_5 i_2} & \text{if } i_1 = 0 \wedge i_3 = 0 \\ 0 & \text{otherwise.} \end{cases}$$

and set all entries of $\mathcal{T}^{AIK'BJ}$ to zero. We can now proceed propagating constraints to the algorithm. In this example we do not wish to redistribute $\mathcal{A}$. Propagating this information results in

Prep	$\mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(?,?,?,?) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(?,?,?,?) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Product	$\mathcal{T}^{AC'IK'BJ}(?,?,?,?,?,?) \mathrel{+}= \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})\mathcal{B}^{BCJK}(?,?,?,?)$
Reduce	$\mathcal{T}^{AIK'BJ}(?,?,?,?,?) = \widehat{\sum_?}\mathcal{T}^{AC'IK'BJ}(?,?,?,?,?,?)$ $\mathcal{C}^{ABJI}(?,?,?,?) = \widehat{\sum_?}\mathcal{T}^{AIK'BJ}(?,?,?,?,?)$

for our template. Propagating the information that during local computations, all modes must be distributed in the same manner results in

Prep	$\mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(?, D_{\langle 1\rangle}, ?, D_{\langle 3\rangle}) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(?, ?, ?, ?) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Product	$\mathcal{T}^{AC'IK'BJ}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, ?, ?) \mathrel{+}= \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})\mathcal{B}^{BCJK}(?, D_{\langle 1\rangle}, ?, D_{\langle 3\rangle})$
Reduce	$\mathcal{T}^{AC'IK'BJ}(?, ?, ?, ?, ?) = \widehat{\sum_{?}}\mathcal{T}^{AC'IK'BJ}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, ?, ?)$ $\mathcal{C}^{ABJI}(?, ?, ?, ?) = \widehat{\sum_{?}}\mathcal{T}^{AIK'BJ}(?, ?, ?, ?, ?)$

for our template. Notice that we did not assign anything to modes of $\mathcal{C}$ in the reduce phase. This is because our constraint only tells us about local computations. As reductions are global operations, this constraint does not apply. The remaining modes of the local computation must be distributed as $D_{\langle\rangle}$. Propagating this information results in

Prep	$\mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(D_{\langle\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(?, ?, ?, ?) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Product	$\mathcal{T}^{AC'IK'BJ}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}, D_{\langle\rangle}) \mathrel{+}= \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})\mathcal{B}^{BCJK}(D_{\langle\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle})$
Reduce	$\mathcal{T}^{AIK'BJ}(?, ?, ?, ?, ?) = \widehat{\sum_{?}}\mathcal{T}^{AC'IK'BJ}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}, D_{\langle\rangle})$ $\mathcal{C}^{ABJI}(?, ?, ?, ?) = \widehat{\sum_{?}}\mathcal{T}^{AIK'BJ}(?, ?, ?, ?, ?)$

for our template. Now all that is left is determining what distributions to assign the remaining modes involved in the reductions. Recall that a reduce-scatter provides us options for resulting distributions. This is a valuable feature as we have flexibility in how $\mathcal{C}^{ABJI}$ must be distributed. One heuristic we can adopt is to avoid replicating any entries of $\mathcal{C}^{ABJI}$. Therefore, we would like to have $\mathcal{C}^{ABJI}$ distributed canonically. Further, if at all possible, we would like to avoid redistributing $\mathcal{C}^{ABJI}$ after the final reduction. One such distribution scheme is to distribute $\mathcal{C}^{ABJI}$ as $\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 3\rangle}, D_{\langle 2\rangle})$. This has the added benefit that the modes shared by $\mathcal{A}^{ACIK}$ are distributed equivalently. Propagating these constraint results in

Prep	$\mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(D_{\langle\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 3\rangle}, D_{\langle 2\rangle}) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Product	$\mathcal{T}^{AC'IK'BJ}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}, D_{\langle\rangle}) \mathrel{+}= \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})\mathcal{B}^{BCJK}(D_{\langle\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle})$
Reduce	$\mathcal{T}^{AIK'BJ}(?, ?, ?, ?, ?) = \widehat{\sum_{?}}\mathcal{T}^{AC'IK'BJ}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}, D_{\langle\rangle})$ $\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 3\rangle}, D_{\langle 2\rangle}) = \widehat{\sum_{?}}\mathcal{T}^{AIK'BJ}(?, ?, ?, ?, ?)$

for our template. An examination of the Reduce phase shows that the only modes involved in the reduction steps are C', K', B, and J. As these are the only modes affected by the reductions, the others must remain

under the same distribution[5]. Using this knowledge, and knowledge of how reduce-scatter collectives operate, we can modify our template resulting in

Prep	$\mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(D_{\langle\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 3\rangle}, D_{\langle 2\rangle}) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Product	$\mathcal{T}^{AC'IK'BJ}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}, D_{\langle\rangle}) \mathrel{+}= \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})\mathcal{B}^{BCJK}(D_{\langle\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle})$
Reduce	$\mathcal{T}^{AIK'BJ}(D_{\langle 0\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}) = \widehat{\sum_{1}}\mathcal{T}^{AC'IK'BJ}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}, D_{\langle\rangle})$ $\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 3\rangle}, D_{\langle 2\rangle}) = \widehat{\sum_{3}}\mathcal{T}^{AIK'BJ}(D_{\langle 0\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle})$

for our template, which is a resulting valid algorithm. Again, it should be noted that this is just one of many possible final distributions of $\mathcal{C}^{ABJI}$. Depending on the constraints of the environment, other choices may be better suited. Regardless, the derivation process remains the same. As our template is completely filled, we have a valid algorithm for computing the specified operation.

A cost analysis of the above algorithm is given in Appendix C.2.

5.3 Stationary B: Avoid communicating $\mathcal{B}$

The stationary B algorithm can be derived analogously to the stationary A algorithm.

5.4 Summary

In this section, we have shown how to derive three different variants of a specific tensor contraction. Although we only showed one example the ideas generalize to all contractions as long as the same assumptions about grid order are made. Generalizations such as one tensor being of lower order than the others follow the same derivation process starting with the same assumptions. Further, this derivation process applies to cases where the order of the computing grid is smaller than the largest tensor. In addition, this process supports the reduction across virtual modes of the processing grid (a reduction across multiple modes of the grid at once). For example, performing the reduction of an mode distributed as $D_{\langle 0,1\rangle}$ does not require any other special rules to support.

One disadvantage to all these algorithms is that they assume a large amount of extra memory is available to store each tensor. One can envision creating blocked versions (which partition some of the operands) of each algorithm to mitigate this effect.

6 Related work

Attempts from many different fields have been made to address the challenges associated with tensor computations in distributed-memory environments, all of which are useful towards to end goal of creating a well-designed library for high-performance tensor computation. Many of these approaches focus more on providing functionality to the computational community rather than focusing on understanding the underlying computation being performed. We detail the related work here.

[5]Though not technically correct, here we are assuming each reduction is performed as a single-step reduction. Assuming this, then the statement holds.

6.1 Tensor Contraction Engine

The *Tensor Contraction Engine* [6] (TCE) is a sophisticated project which, given an arbitrary tensor contraction expression, generates efficient code which implements the specified expression. By applying heuristics and expression rewrites, TCE is able to generate a search space of algorithms which reduce the computational complexity while retaining necessary properties such as having the data fit in memory. As a final step, the TCE generates code for the selected algorithm in terms of matrix-matrix multiplication which can achieve high-performance (given an appropriate library).

As we mentioned earlier in this paper, mapping tensor contractions to matrix-matrix multiplication can result in significant overhead due to the communication required to redistribute the data in preparation for computation. However, this final mapping is a result of the availability of high-performance libraries. There is no reason why, given a high-performance multi-linear algebra library, one could not replace the final step of TCE and implement the selected algorithm with the multi-linear algebra library, thereby reaping all benefits of the analysis of the TCE.

6.2 Cyclops Tensor Framework

The Cyclops Tensor Framework [23] (CTF) is a framework designed for applications based on tensor contractions which involve tensors with some form of symmetry. The CTF is designed for distributed-memory computing environments. By exploiting symmetry, CTF can significantly reduce the storage requirements while still providing an implementation for tensor contractions which provably achieves the communication lower bound for each tensor contraction. An elemental-cyclic distribution of entries is used to distribute only the unique portion of the symmetric tensors required in the operation. The general pattern for performing a tensor contraction is broken into a communication phase, which redistributes data among all processes using an all-to-all collective communication, and a computation phase which performs the actual computation. To ensure correctness with arbitrary processing grid meshes, CTF maps the problem to a virtual processing grid which then is mapped to the physical grid used.

6.3 SIAL

The Super Instruction Architecture (SIA) is an environment comprising a programming language, SIAL, and a runtime system, SIP, with the goal of providing portable and efficient code related to tensor computations for a wide array of computing environments including distributed-memory environments [20]. SIAL exposes commonly used abstractions in scientific computing, such as blocking, providing the user a useful method of describing how an algorithm proceeds without unnecessarily complicating the code. Programs written in SIAL are compiled to a bytecode which is then interpreted by an SIP virtual machine which handles the execution of the program. Additionally, the SIP handles difficulties associated with parallelism, thus hiding this aspect of the program from the user. Distributed-memory parallelism in the SIP is handled through the use of asynchronous communication routines to aid in effectively overlapping computation with communication. One important aspect of this project is that all computation fully computes a resulting block during local computation.

As an example of the use of SIAL, the Advanced Concepts in Electronic Structure library (ACES) [15] implements a suite of chemical methods of electronic structure theory.

6.4 NWChem

Similar to the ACES project described previously, the NWChem [25] package is designed to provide users access to many computational chemistry methods. NWChem is implemented through a distributed-memory framework built upon the Global Arrays toolkit (GA) [17, 14, 1]. The goal of GA is to provide the programmer a shared-memory interface to the user even though computing in a distributed-memory environment. When a user requests data at a particular global location, it is up to GA to ensure that the data is communicated correctly. The interface allows for all types of redistributions, and as a result does not attempt to enforce

regularity in the data which is communicated. As in the ACES project, the goal is not to devise a framework for the underlying computation, but a framework for expressing the computation needed to be performed.

6.5 CAST

Work by Rajbhandari et al. [19] provides a framework for the tensor contraction operation involving tensors which contain some form of symmetry. Similar to the proposed work, this work uses an elemental-cyclic wrapping of data onto the mesh and provides conditions which must be met to create a valid distribution. Using these constraints, and a model for computation, this work creates a space of algorithms which are then searched to find an efficient implementation. The algorithms created rely on a three-phase approach to computing a tensor contraction which is can be summarized as broadcasting, computing, and finally redistributing.

Indeed, many of the ideas in this work and the proposed work are shared: utilizing an elemental-cyclic distribution of data, defining similar valid distributions, and creating a space of algorithms which must be searched. The key difference, in our eyes, is that the proposed work formalizes the relationship between distributions of elemental-cyclic wrappings of data and the set of collective communications. By doing so, algorithms involving many more of the collective communications can be created and the algorithms required can be systematically derived and examined.

6.6 Other work

Work by Gao, et al. [12] takes a compiler approach to creating high-performance programs for tensor contractions. Similar to CTF, this work breaks the global computation into phases of communication and computation. The main difference is that this work allows for reduction-based algorithms (algorithms which utilize reduce-scatter collectives to perform part of the computation) thereby allowing some amount of flexibility. One significant drawback of this work is that presently, the approach taken is to serialize or group multiple modes of the tensors together as a single index and then apply a well-known high-performance linear algebra library to compute the result. As we have argued, this can incur additional cost due to communication which a native tensor library could potentially avoid.

7 Conclusion and Future Work

In this paper, we have proposed a notation for describing how tensors are distributed and redistributed on processing grids. Additionally, we have shown how one can systematically derive algorithms for the tensor contraction operation. As this work is preliminary, we conclude with ideas for future work.

Incorporate generalizations of "3D" algorithms. In this paper, we assumed that the processing grid used for computation is of the same order as the tensors involved in the computation. Looking at linear algebra, there are a class of algorithms where the processing grid is viewed as an order-3 mesh instead of an order-2 mesh commonly referred to as "3D" algorithms[2]. It is unclear how 3D algorithms (or generalizations thereof) fit into the proposed notation. As 3D algorithms provide benefits for parallel matrix-matrix computation, one can imagine similar benefits can be obtained for tensor contractions.

Automate algorithm derivation and selection. The proposed notation provides a systematic way to derive algorithms for tensor contractions. However, the process of deriving the algorithms and picking the optimal among the space of algorithms can potentially be time consuming, especially considering the complexity of many applications. As the goal of the notation is to provide a formal notation for all important features of the domain, one could envision creating a tool which can process and select the optimal algorithm, similar to how DxT has been applied to linear algebra [16].

Incorporate other tensor operations. In this paper, we only focus on the tensor contraction operation. There exist a slew of other tensor operations which are important for other domains. It would be interesting to see if/how one could similarly systematically derive algorithms for these operations as well using the proposed notation.

Incorporate symmetry. One significant drawback of computing with tensors storing all elements large amount of data required to express the problem. As one increases the order of the tensor being computed with, the amount of data required increases dramatically. If one can assume some symmetry within the tensor, this effect can be reduced significantly [21]. However, it is unclear how to incorporate this into the proposed notation.

Investigate utility of redistribution features. Looking at how high-performance matrix-matrix multiplication is performed in Elemental, the reduce-scatter operation is used to directly reduce the partial results to final results in one step. However, as we have argued in this paper, there is opportunity for a reduction of modes in multiple-steps. As this case does not appear in the matrix case, it would be interesting to investigate what, if any, benefits there are to performing a reduction as a multi-step process.

Further, other redistributions are not utilized in the matrix case, at least they are not described in the relevant papers such as using an all-to-all to redistribute between a specific set of distributions (mentioned in Section 4). Investigating the utility of such redistributions could provide benefits in the tensor case.

Expand distribution/redistribution rules The proposed notation is rich in the distributions and redistributions defined, however it is by no means complete. One could envision other rules which are simple to incorporate and useful in practice. Examining the patterns seen while executing different algorithms could potentially result in other rules created.

Provide a library. One assumption made in this paper is that in the end, the notation enables a high-performance implementation to exist. In theory, if the ideas presented are correctly implemented, a high-performance library for tensor contractions should be the result. However, at this point, it is unclear how difficult the ideas proposed are to implement.

Acknowledgments

We would like to than Tze Meng Low, Robert van de Geijn and Tamara G. Kolda for their contributions to many aspects of this paper. We thank the other members of the FLAME team for their support. This work was also partially sponsored by NSF grants ACI-1148125/1340293 (supplement) and CCF-1320112.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

References

[1] Global arrays webpage. http://hpc.pnl.gov/globalarrays/.

[2] R.C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39:39–5, 1995.

[3] Brett W. Bader and Tamara G. Kolda. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software*, 32(4):635–653, December 2006.

[4] Brett W. Bader and Tamara G. Kolda. Efficient matlab computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231, 2007.

[5] R J Bartlett. Many-body perturbation theory and coupled cluster theory for electron correlation in molecules. *Annual Review of Physical Chemistry*, 32(1):359–401, 1981.

[6] G. Baumgartner, A. Auer, D.E. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R.J. Harrison, S. Hirata, S. Krishnamoorthy, S. Krishnan, C. Lam, Q. Lu, M. Nooijen, R.M. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov. Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models. In *Proceedings of the IEEE*, volume 93, pages 276–292, 2005.

[7] Jehoshua Bruck, Ching tien Ho, Shlomo Kipnis, Eli Upfal, and Derrick Weathersby. Efficient algorithms for all-to-all communications in multi-port systems. In *IEEE Transactions on Parallel and Distributed Systems*, pages 298–309, 1997.

[8] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

[9] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 120–127. IEEE Comput. Soc. Press, 1992.

[10] C. Edwards, P. Geng, A. Patra, and R. van de Geijn. Parallel matrix distributions: have we been doing it all wrong? Technical Report TR-95-40, Department of Computer Sciences, The University of Texas at Austin, 1995.

[11] A. Einstein. Die Grundlage der allgemeinen Relativitätstheorie. *Annalen der Physik*, 354:769–822, 1916.

[12] Xiaoyang Gao, Swarup Kumar Sahoo, Chi-Chung Lam, J. Ramanujam, Qingda Lu, Gerald Baumgartner, and P. Sadayappan. Performance modeling and optimization of parallel out-of-core tensor contractions. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '05, pages 266–276, New York, NY, USA, 2005. ACM.

[13] B. Hendrickson, R. Leland, and S. Plimpton. An efficient parallel algorithm for matrix-vector multiplication. Technical report.

[14] Manojkumar Krishnan, Bruce Palmer, Abhinav Vishnu, Sriram Krishnamoorthy, Jeff Daily, and Daniel Chavarria. *The Global Arrays User Manual*, 2012.

[15] V. Lotrich, N. Flocke, M. Ponton, A.D. Yau, A. Perera, E. Deumens, and R.J. Bartlett. Parallel implementation of electronic structure energy, gradient and hessian calculations. *J. Chem. Phys.*, 128:194104, 2008.

[16] Bryan Marker, Jack Poulson, Don S. Batory, and Robert A. van de Geijn. Designing linear algebra algorithms by transformation: Mechanizing the expert developer. In *VECPAR*, 2012.

[17] Jarek Nieplocha, Bruce Palmer, Vinod Tipparaju, Manojkumar Krishnan, Harold Trease, and Edo Apra. Advances, applications and performance of the global arrays shared memory programming toolkit. *International Journal of High Performance Computing Applications*, 20(2):203–231, 2006.

[18] Jack Poulson, Bryan Marker, Robert A. van de Geijn, Jeff R. Hammond, and Nichols A. Romero. Elemental: A new framework for distributed memory dense matrix computations. *ACM Trans. Math. Softw.*, 39(2):13:1–13:24, February 2013.

[19] Samyam Rajbhandari, Akshay Nikam, Pai-Wei Lai, Kevin Stock, Sriram Krishnamoorthy, and P.Sadayappan. Framework for distributed contractions of tensors with symmetry. Technical Report 23, The Ohio State University and Pacific Northwest National Laboratory, 2013.

[20] B.A. Sanders, R. Bartlett, E. Deumens, V. Lotrich, and M. Ponton. A block-oriented language and runtime system for tensor algebra with very large arrays. In *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pages 1–11, 2010.

[21] Martin D. Schatz, Tze Meng Low, Robert A. van de Geijn, and FLAME Working Note #68 Tamara G. Kolda. Exploiting symmetry in tensors for high performance: an initial study. Technical Report TR-12-33, The University of Texas at Austin, Department of Computer Sciences, December 2012.

[22] Martin D. Schatz, Jack Poulson, and Robert van de Geijn. Parallel matrix multiplciation: 2d and 3d. FLAME Working Note #62 TR-12-13, The University of Texas at Austin, Department of Computer Sciences, JUNE 2012.

[23] Edgar Solomonik, Jeff Hammond, and James Demmel. A preliminary analysis of cyclops tensor framework. Technical Report UCB/EECS-2012-29, EECS Department, University of California, Berkeley, Mar 2012.

[24] G. W. Stewart. Communication and matrix computations on large message passing systems. *Parallel Computing*, 16:27–40, 1990.

[25] M. Valiev, E.J. Bylaska, N. Govind, K. Kowalski, T.P. Straatsma, H.J.J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T.L. Windus, and W.A. de Jong. Nwchem: A comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*, 181(9):1477 – 1489, 2010.

[26] Robert A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. The MIT Press, 1997.

A Elemental 2-D Examples

In this section, we describe Elemental's approach to expressing and deriving algorithms for matrix-vector and matrix-matrix operations.

A.1 Matrix-vector multiplication

Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{y} \in \mathbb{R}^m$, and label their individual elements so that

$$\mathbf{A} = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}, \quad \text{and} \quad \mathbf{y} = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix}.$$

Recalling that matrix-vector multiplication, $\mathbf{y} = \mathbf{A}\mathbf{x}$ (or $\mathbf{y}^I = \mathbf{A}^{IK}\mathbf{x}^K$ in tensor notation) is computed as

$$\begin{array}{rcccc} \psi_0 = & \alpha_{0,0}\chi_0 + & \alpha_{0,1}\chi_1 + \cdots + & \alpha_{0,n-1}\chi_{n-1} \\ \psi_1 = & \alpha_{1,0}\chi_0 + & \alpha_{1,1}\chi_1 + \cdots + & \alpha_{1,n-1}\chi_{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \psi_{m-1} = & \alpha_{m-1,0}\chi_0 + & \alpha_{m-1,1}\chi_1 + \cdots + & \alpha_{m-1,n-1}\chi_{n-1} \end{array}$$

we notice that element $\alpha_{i_0 i_1}$ multiplies χ_{i_1} and contributes to ψ_{i_0}. Thus we may summarize the interactions of the elements of $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{A}$ by

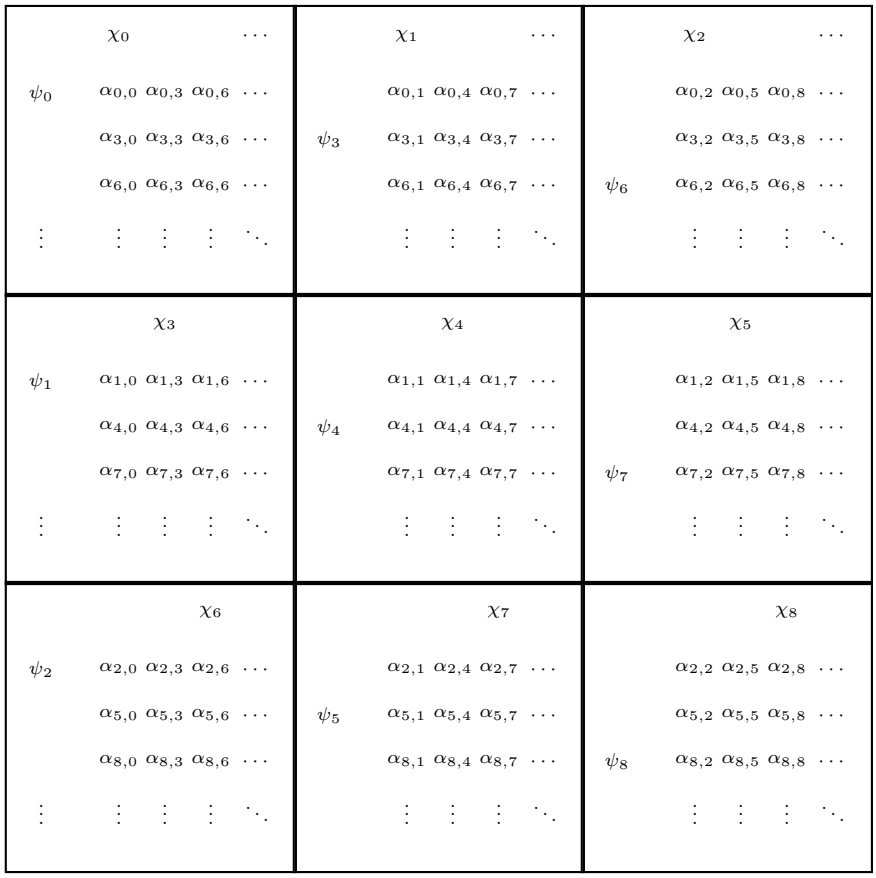

Figure 11: Distribution of $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{y}$ within a 3×3 mesh. Notice that redistributing a column of $\mathbf{A}$ in the same manner as $\mathbf{y}$ requires simultaneous scatters within rows of nodes while redistributing a row of $\mathbf{A}$ consistently with $\mathbf{x}$ requires simultaneous scatters within columns of nodes. Here the distribution of $\mathbf{x}$ and $\mathbf{y}$ are given by $\mathbf{x}(D_{\langle 1,0 \rangle})$ and $\mathbf{y}(D_{\langle 0,1 \rangle})$, respectively, and $\mathbf{A}$ by $\mathbf{A}(D_{\langle 0 \rangle}, D_{\langle 1 \rangle})$. While the presented mesh of nodes is square, none of the results depend on the mesh being square.

		$k-$index			
		χ_0	χ_1	$\cdots$	χ_{n-1}
i-index	ψ_0	$\alpha_{0,0}$	$\alpha_{0,1}$	$\cdots$	$\alpha_{0,n-1}$
	ψ_1	$\alpha_{1,0}$	$\alpha_{1,1}$	$\cdots$	$\alpha_{1,n-1}$
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	ψ_{m-1}	$\alpha_{m-1,0}$	$\alpha_{m-1,1}$	$\cdots$	$\alpha_{m-1,n-1}$

(8)

which is meant to indicate that χ_{i_1} must be multiplied by the elements in the i_1-th column of $\mathbf{A}$ while the i_0-th row of $\mathbf{A}$ contributes to ψ_{i_0}. The axes of (8) are meant to indicate the mode whose range each direction represents.

A.2 Two-Dimensional Elemental Cyclic Distribution

It is well established that (weakly) scalable implementations of dense linear algebra operations require nodes to be logically viewed as a two-dimensional mesh [24, 13].

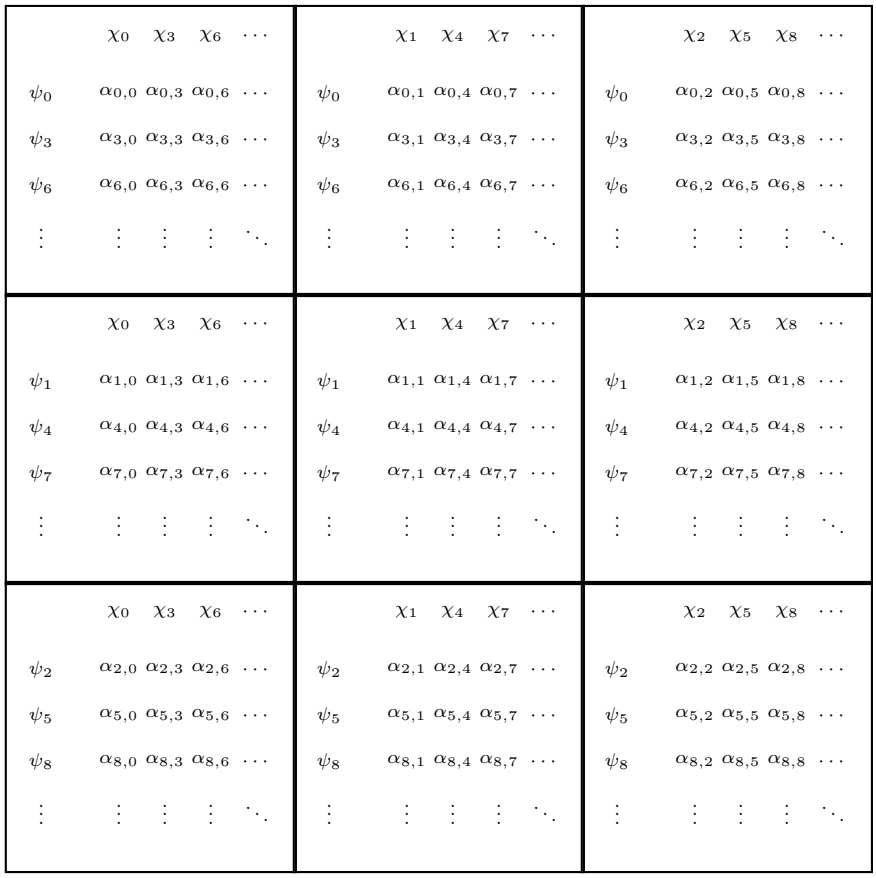

Figure 12: Vectors $\mathbf{x}$ and $\mathbf{y}$ respectively redistributed as row-projected and column-projected vectors. The column-projected vector $\mathbf{y}(D_{\langle 0\rangle})$ here is to be used to compute local results that will become contributions to a column vector $\mathbf{y}(D_{\langle 0,1\rangle})$ which will result from adding these local contributions within rows of nodes. By comparing and contrasting this figure with Figure 11 it becomes obvious that redistributing $\mathbf{x}(D_{\langle 1,0\rangle})$ to $\mathbf{x}(D_{\langle 1\rangle})$ requires an allgather within columns of nodes while $\mathbf{y}(D_{\langle 0,1\rangle})$ results from scattering $\mathbf{y}(D_{\langle 0\rangle})$ within process rows.

It is also well established that, to achieve load balance for a wide range of matrix operations, matrices should be cyclically "wrapped" onto this logical mesh. We start with these insights and examine the simplest of matrix distributions that result: 2D elemental cyclic distribution [18, 13].

Denoting the number of nodes by p, a $d_0 \times d_1$ mesh must be chosen such that $p = d_0 d_1$.

Matrix distribution The elements of $\mathbf{A}$ are assigned using an elemental cyclic (round-robin) distribution where $\alpha_{i_0 i_1}$ is assigned to node $(i_0 \bmod d_0, i_1 \bmod d_1)$. Thus, node $(\overline{\sigma}_0, \overline{\sigma}_1)$ stores submatrix

$$\mathbf{A}(\overline{\sigma}_0 : d_0 : m-1, \overline{\sigma}_1 : d_1 : n-1) = \begin{pmatrix} \alpha_{\overline{\sigma}_0,\overline{\sigma}_1} & \alpha_{\overline{\sigma}_0,\overline{\sigma}_1+d_1} & \cdots \\ \alpha_{\overline{\sigma}_0+d_0,\overline{\sigma}_1} & \alpha_{\overline{\sigma}_0+d_0,\overline{\sigma}_1+d_1} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix},$$

where the left-hand side of the expression uses the MATLAB convention for expressing submatrices, starting indexing from zero instead of one. This is illustrated in Figure 11.

Column-major vector distribution A *column-major* vector distribution views the $d_0 \times d_1$ mesh of nodes as a linear array of p nodes, numbered in *column-major* order. A vector is distributed with this distribution if it is assigned to this linear array of nodes in a round-robin fashion, one element at a time.

In other words, consider vector y. Its element ψ_{i_0} is assigned to node $(i_0 \bmod d_0, (i_0/d_0) \bmod d_1)$, where / denotes integer division. Or, equivalently in MATLAB-like notation, node $(\overline{\sigma}_0, \overline{\sigma}_1)$ stores subvector $\mathbf{y}(u(\overline{\sigma}_0, \overline{\sigma}_1) : p : m-1)$, where $u(\overline{\sigma}_0, \overline{\sigma}_1) = \overline{\sigma}_0 + \overline{\sigma}_1 d_0$ equals the rank of node $(\overline{\sigma}_0, \overline{\sigma}_1)$ when the nodes are viewed as a one-dimensional array, indexed in column-major order. The distribution of $\mathbf{y}$ is illustrated in Figure 11.

Row-major vector distribution Similarly, a *row-major* vector distribution views the $d_0 \times d_1$ mesh of nodes as a linear array of p nodes, numbered in *row-major* order. The vector is then assigned to this linear array of nodes in a round-robin fashion, one element at a time.

In other words, consider vector x. Its element χ_{i_1} is assigned to node $(i_1 \bmod d_1, (i_1/d_1) \bmod d_0)$. Or, equivalently, node $(\overline{\sigma}_0, \overline{\sigma}_1)$ stores subvector $\mathbf{x}(v(\overline{\sigma}_0, \overline{\sigma}_1) : p : n-1)$, where $v = \overline{\sigma}_0 d_1 + \overline{\sigma}_1$ equals the rank of node $(\overline{\sigma}_0, \overline{\sigma}_1)$ when the nodes are viewed as a one-dimensional array, indexed in row-major order. The distribution of $\mathbf{x}$ is illustrated in Figure 11.

A.3 Parallelizing matrix-vector operations

In the following discussion, we assume that $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{y}$ are distributed as discussed above[6]. At this point, we suggest comparing Eqn. (8) with Figure 11.

Computing $y := Ax$ The relation between these distributions of a matrix, column-major vector, and row-major vector is illustrated by revisiting the most fundamental of computations in linear algebra, $\mathbf{y} := \mathbf{A}\mathbf{x}$, already discussed in Section A.1. An examination of Figure 11 suggests that the elements of $\mathbf{x}$ must be gathered within columns of nodes (allgather within columns) leaving elements of $\mathbf{x}$ distributed as illustrated in Figure 12. Next, each node computes the partial contribution to vector $\mathbf{y}$ with its local matrix and copy of $\mathbf{x}$. Thus, in Figure 12, ψ_{i_0} in each node becomes a contribution to the final ψ_{i_0}. These must be added together, which is accomplished by a summation of contributions to $\mathbf{y}$ within rows of nodes. An experienced MPI programmer will recognize this as a reduce-scatter within each row of nodes.

Computing $\mathbf{A} := \mathbf{y}\mathbf{x}^T + \mathbf{A}$ A second commonly encountered matrix-vector operation is the rank-1 update: $\mathbf{A} := \alpha\mathbf{y}\mathbf{x}^T + \mathbf{A}$. We will discuss the case where $\alpha = 1$. Recall that

$$\mathbf{A} + \mathbf{y}\mathbf{x}^T = \begin{pmatrix} \alpha_{0,0} + \psi_0\chi_0 & \alpha_{0,1} + \psi_0\chi_1 & \cdots & \alpha_{0,n-1} + \psi_0\chi_{n-1} \\ \alpha_{1,0} + \psi_1\chi_0 & \alpha_{1,1} + \psi_1\chi_1 & \cdots & \alpha_{1,n-1} + \psi_1\chi_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1,0} + \psi_{m-1}\chi_0 & \alpha_{m-1,1} + \psi_{m-1}\chi_1 & \cdots & \alpha_{m-1,n-1} + \psi_{m-1}\chi_{n-1} \end{pmatrix},$$

which, when considering Figures 11 and 12, suggests the following parallel algorithm: All-gather of $\mathbf{y}$ within rows. All-gather of $\mathbf{x}$ within columns. Update of the local matrix on each node.

B Allgather redistribution communication pattern

In this section we show why, given a tensor $\mathcal{A}^{\boldsymbol{l}_0\boldsymbol{l}_1\cdots\boldsymbol{l}_{m-1}}$ distributed as

$$\mathcal{A}^{\boldsymbol{l}_0\boldsymbol{l}_1\cdots\boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{x}}_0}, \ldots, D_{\overline{\mathbf{x}}_{k-1}}, D_{\overline{\mathbf{y}}\sqcup\langle h_\ell\rangle}, D_{\overline{\mathbf{x}}_{k+1}}, \ldots, D_{\overline{\mathbf{x}}_{m-1}})$$

on an $d_0 \times d_1 \times \cdots \times d_{m_g-1}$ mesh of processes, an allgather communication within the h_ℓ mode of the grid results in the tensor distributed as

[6] We suggest the reader print copies of Figures 11 and 12 for easy referral while reading the rest of this section.

$$\boldsymbol{\mathcal{A}}^{\boldsymbol{l}_0\boldsymbol{l}_1\cdots\boldsymbol{l}_{m-1}}(D_{\overline{\mathbf{x}}_0},\ldots,D_{\overline{\mathbf{x}}_{k-1}},D_{\overline{\mathbf{y}}},D_{\overline{\mathbf{x}}_{k+1}},\ldots,D_{\overline{\mathbf{x}}_{m-1}})$$

Our argument lies in showing that, for any process location $\overline{\mathbf{s}} = \langle \overline{\sigma}_0, \overline{\sigma}_1, \ldots, \overline{\sigma}_{m_g-1} \rangle$ in our process grid,

$$D_{\overline{\mathbf{y}}}^{\overline{\mathbf{s}}(\overline{\mathbf{y}})} \iff \bigcup_{\ell_0=0}^{d_{h_\ell}-1} D_{\overline{\mathbf{y}}\sqcup\langle h_\ell\rangle}^{\overline{\mathbf{s}}(\overline{\mathbf{y}}\sqcup\langle h_\ell\rangle)}.$$

We are now tasked with showing that

$$\{j \in \mathbb{N} : j \equiv c_0 \bmod c_1\} \quad \iff$$
$$\textstyle\bigcup_{\ell_0=0}^{d_{h_\ell}} \{j \in \mathbb{N} : j \equiv (c_0 + \ell_0 c_1) \bmod (d_{h_\ell} c_1)\}$$

where $c_0 = \displaystyle\sum_{\ell_0=0}^{|\overline{\mathbf{x}}|-1} \left(\overline{\chi}_{\ell_0} \prod_{\ell_1=0}^{\ell_0-1} d_{\overline{\psi}_{\ell_1}} \right)$, $\overline{\mathbf{x}} = \overline{\mathbf{s}}(\overline{\mathbf{y}})$, and $c_1 = \displaystyle\prod_{\ell_1=0}^{|\overline{\mathbf{y}}|-1} d_{\psi_{\ell_1}}$.

$$\{j \in \mathbb{N} : j \equiv c_0 \bmod c_1\}$$

contains elements of the form $c_0 + nc_1$ where $n \in \mathbb{N}$. We first show that for any n, the set

$$\bigcup_{\ell_0=0}^{d_{h_\ell}} \{j \in \mathbb{N} : j \equiv (c_0 + \ell_0 c_1) \bmod (d_{h_\ell} c_1)\}$$

contains the element $c_0 + nc_1$. Consider the case where $0 \leq n < d_{h_\ell}$. The set in question contains such an element as it stores elements of the form $c_0 + \ell_0 c_1$ where $0 \leq \ell_0 < d_{h_\ell}$.

Consider the case where $n \geq d_{h_\ell}$. Then, the set in question contains the element $c_0 + nc_1$ as it can be rewritten as $c_0 + (d_{h_\ell} + j)c_1 = c_0 + (d_{h_\ell})c_1 + jc_1 \equiv c_0 + jc_1 \bmod d_{h_\ell} c_1$ for some j. By the previous argument, the set in question contains the needed element.

Showing the reverse direction is trivially true. Thus, the assertion holds.

C Algorithm cost analyses

In this section, we will provide a cost analyses (based on costs given in Figure 6) for the algorithms derived in Section 5. We assume our processing grid is arranged in a logical $d_0 \times d_1 \times d_2 \times d_3$ processing mesh. Further, we assume the dimension of mode I is denoted L_I.

C.1 Stationary C algorithm

Recall that our final derived stationary C algorithm was

$\boldsymbol{\mathcal{A}}^{ACIK}(D_{\langle 0\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}) \leftarrow \boldsymbol{\mathcal{A}}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\boldsymbol{\mathcal{B}}^{BCJK}(D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 2\rangle}, D_{\langle\rangle}) \leftarrow \boldsymbol{\mathcal{B}}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\boldsymbol{\mathcal{C}}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \leftarrow \boldsymbol{\mathcal{C}}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
$\boldsymbol{\mathcal{C}}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \mathrel{+}= \boldsymbol{\mathcal{A}}^{ACIK}(D_{\langle 0\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}) \boldsymbol{\mathcal{B}}^{BCJK}(D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 2\rangle}, D_{\langle\rangle})$

The redistribution

$$\boldsymbol{\mathcal{A}}^{ACIK}(D_{\langle 0\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}) \leftarrow \boldsymbol{\mathcal{A}}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$$

can be implemented as an all-to-all within mode-2 of the process grid, an all-to-all within mode-3, and an allgather within modes 1 and 2 of the process grid. The cost of this redistribution is

$$\log_2(d_2)\alpha + \beta\frac{L_AL_CL_IL_K}{d_0d_1d_2d_3}\frac{d_2-1}{d_2} + \log_2(d_3)\alpha + \beta\frac{L_AL_CL_IL_K}{d_0d_1d_2d_3}\frac{d_3-1}{d_3} + \log_2(d_1d_2)\alpha + \beta\frac{L_AL_CL_IL_K}{d_0d_1d_2d_3}\frac{d_1d_2-1}{d_1d_2} \quad = \\ \log_2(d_1d_2^2d_3)\alpha + \beta\frac{L_AL_CL_IL_K}{p}\left(\frac{d_2-1}{d_2} + \frac{d_3-1}{d_3} + \frac{d_1d_2-1}{d_1d_2}\right)$$

The redistribution

$$\mathcal{B}^{BCJK}(D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 2\rangle}, D_{\langle\rangle}) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$$

can be implemented as an all-to-all within mode-0, all-to-all within mode-1, and an allgather within modes 0 and 3. The cost of this redistribution is

$$\log_2(d_0)\alpha + \beta\frac{L_BL_CL_JL_K}{d_1d_2d_3}\frac{d_0-1}{d_0} + \log_2(d_1)\alpha + \beta\frac{L_BL_CL_JL_K}{d_0d_2d_3}\frac{d_1-1}{d_1} + \log_2(d_0d_3)\alpha + \beta\frac{L_BL_CL_JL_K}{d_0d_1d_2}\frac{d_0d_3-1}{d_0d_3} \quad = \\ \log_2(d_0^2d_1d_3)\alpha + \beta\frac{L_BL_CL_JL_K}{p}\left(\frac{d_0-1}{d_0} + \frac{d_1-1}{d_1} + \frac{d_0d_3-1}{d_0d_3}\right)$$

The computation has a cost of

$$\frac{2L_AL_CL_IL_KL_BL_J}{p}\gamma$$

In total, the algorithm has a cost of

$$\frac{2L_AL_CL_IL_KL_BL_J}{p}\gamma + 2\log_2\left(d_0d_2d_1d_3\right)\alpha + \beta\frac{L_CL_K}{p}\left(L_AL_I\left(\frac{d_2-1}{d_2} + \frac{d_3-1}{d_3} + \frac{d_1d_2-1}{d_1d_2}\right) + L_BL_J\left(\frac{d_0-1}{d_0} + \frac{d_1-1}{d_1} + \frac{d_0d_3-1}{d_0d_3}\right)\right)$$

C.2 Stationary A algorithm

Recall that our final derived stationary A algorithm was

Prep	$\mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{B}^{BCJK}(D_{\langle\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$ $\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 3\rangle}, D_{\langle 2\rangle}) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$
Product	$\mathcal{T}^{AC'IK'BJ}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}, D_{\langle\rangle}) \mathrel{+}= \mathcal{A}^{ACIK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})\mathcal{B}^{BCJK}(D_{\langle\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle})$
Reduce	$\mathcal{T}^{AIK'BJ}(D_{\langle 0\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}) = \widehat{\sum_1}\mathcal{T}^{AC'IK'BJ}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle\rangle}, D_{\langle\rangle})$ $\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 3\rangle}, D_{\langle 2\rangle}) = \widehat{\sum_3}\mathcal{T}^{AIK'BJ}(D_{\langle 0\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle})$

The redistribution

$$\mathcal{B}^{BCJK}(D_{\langle\rangle}, D_{\langle 1\rangle}, D_{\langle\rangle}, D_{\langle 3\rangle}) \leftarrow \mathcal{B}^{BCJK}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$$

can be implemented as an all-to-all within mode-0 of the grid and an allgather within modes 0 and 2 of the grid. The cost of this redistribution is

$$\alpha\log_2(d_0) + \beta\frac{L_BL_CL_JL_K}{p}\left(\frac{d_0-1}{d_0}\right) + \\ \alpha\log_2(d_0d_2) + \beta\frac{L_BL_CL_JL_K}{p}\left(\frac{d_0d_2-1}{d_0d_2}\right)$$

The redistribution

$$\mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 3\rangle}, D_{\langle 2\rangle}) \leftarrow \mathcal{C}^{ABJI}(D_{\langle 0\rangle}, D_{\langle 1\rangle}, D_{\langle 2\rangle}, D_{\langle 3\rangle})$$

can be implemented as an allgather within modes 2 and 3 of the grid. The cost of this redistribution is

$$\alpha \log_2(d_2 d_3) + \beta \frac{L_A L_B L_J L_I}{p} \left(\frac{d_2 d_3 - 1}{d_2 d_3} \right)$$

The local computation has a cost of

$$\frac{2 L_A L_C L_I L_K L_B L_J}{p} \gamma$$

The reduction within mode-1 of the grid has a cost of

$$\alpha \log_2(d_1) + (\beta + \gamma) \frac{L_A L_C L_I L_K L_B L_J}{p} \left(\frac{d_1 - 1}{d_1} \right)$$

The subsequent reduction within mode-3 of the grid has a cost of

$$\alpha \log_2(d_3) + (\beta + \gamma) \frac{L_A L_I L_K L_B L_J}{d_0 d_1 d_2} \left(\frac{d_3 - 1}{d_3} \right)$$

In total, the algorithm has a cost of

$$\begin{aligned}
&\tfrac{\gamma}{p} \left(2 L_A L_C L_I L_K L_B L_J + L_A L_C L_I L_K L_B L_J \tfrac{d_1 - 1}{d_1} + L_A L_I L_K L_B L_J \tfrac{d_3 - 1}{d_3} \right) + \\
&\alpha \log_2(d_1 (d_0 d_2 d_3)^2) + \\
&\beta \tfrac{L_B L_J}{p} \left(L_C L_K \left(\tfrac{d_0 - 1}{d_0} + \tfrac{d_0 d_2 - 1}{d_0 d_2} \right) + L_A L_I \left(\tfrac{d_2 d_3 - 1}{d_2 d_3} + L_K \left(\tfrac{d_3 - 1}{d_3} + L_C \tfrac{d_1 - 1}{d_1} \right) \right) \right)
\end{aligned}$$

D Best case flop to element ratio

We wish to determine what is the best flops to element ratio we can hope to attain when considering a tensor contraction. Here we show that the best case is $O(n^{\frac{m}{2}})$ where m is the order of the output tensor we wish to compute.

Theorem 8. *Given an order-$m_{\mathcal{C}}$ tensor $\mathcal{C}$ defined by the contraction of an order-$m_{\mathcal{A}}$ tensor $\mathcal{A}$ with an order-$m_{\mathcal{B}}$ tensor $\mathcal{B}$, then the maximum flop to number of elements involved ratio is $O(n^{\frac{m_{\mathcal{C}}}{2}})$ assuming all modes involved have dimension n.*

Proof: Without loss of generality, assume that $\mathcal{C}$ is computed as

$$\mathcal{C}^{\boldsymbol{l}_0 \boldsymbol{l}_1 \cdots \boldsymbol{l}_{m_{\mathcal{C}}-1}} = \mathcal{A}^{\boldsymbol{l}_0 \boldsymbol{l}_1 \cdots \boldsymbol{l}_{k_A - 1} \boldsymbol{s}_0 \boldsymbol{s}_1 \cdots \boldsymbol{s}_{S-1}} \mathcal{B}^{\boldsymbol{l}_{k_A} \boldsymbol{l}_{k_A+1} \cdots \boldsymbol{l}_{m_{\mathcal{C}}-1} \boldsymbol{s}_0 \boldsymbol{s}_1 \cdots \boldsymbol{s}_{S-1}}.$$

We know that the following conditions hold

$$\begin{aligned}
k_A + S &= m_{\mathcal{A}} \\
m_{\mathcal{C}} - k_A + S &= m_{\mathcal{B}}
\end{aligned}$$

Further, we know that the number of flops required to compute $\mathcal{C}$ is

$$\#flops = \underbrace{n^{m_{\mathcal{C}}}}_{\#elems} \underbrace{n^S}_{cost/element}$$

and the number of elements involved is

$$\#elems = \underbrace{n^{m_{\mathcal{C}}}}_{\mathcal{C}} + \underbrace{n^{m_{\mathcal{A}}}}_{\mathcal{A}} + \underbrace{n^{m_{\mathcal{B}}}}_{\mathcal{B}}.$$

Then the ratio of flops to elements involved is given by

$$R = \frac{\#flops}{\#elems} = \frac{n^{m_{\mathcal{C}}} n^S}{n^{m_{\mathcal{C}}} + n^{m_{\mathcal{A}}} + n^{m_{\mathcal{B}}}} = \frac{n^{m_{\mathcal{C}}} n^S}{n^{m_{\mathcal{C}}} + n^{k_A + S} + n^{m_{\mathcal{C}} - k_A + S}} = \frac{n^{m_{\mathcal{C}}}}{\frac{n^{m_{\mathcal{C}}}}{n^S} + n^{k_A} + n^{m_{\mathcal{C}} - k_A}}.$$

We wish to find the value k_A which results in R being maximized. This occurs at $k_A = \frac{m_{\mathfrak{e}}}{2}$ resulting in

$$R = \frac{n^{m_{\mathfrak{e}}}}{\frac{n^{m_{\mathfrak{e}}}}{n^S} + n^{\frac{m_{\mathfrak{e}}}{2}} + n^{\frac{m_{\mathfrak{e}}}{2}}} = \frac{n^{m_{\mathfrak{e}}}}{\frac{n^{m_{\mathfrak{e}}}}{n^S} + 2n^{\frac{m_{\mathfrak{e}}}{2}}}.$$

In the worst case for S, R is $O(n)$, however if $S \geq \frac{m_{\mathfrak{e}}}{2}$ then R is $O(n^{\frac{m_{\mathfrak{e}}}{2}})$. **endofproof**