Framework for Distributed Contractions of Tensors with Symmetry

Samyam Rajbhandari*, Akshay Nikam*, Pai-Wei Lai*, Kevin Stock*, Sriram Krishnamoorthy,†, P. Sadayappan*
*Dept. of Computer Science and Engineering
The Ohio State University
Columbus OH, USA
{rajbhand,nikam,laip,stockk,saday}@cse.ohio-state.edu
†Computational Sciences and Mathematics Division
Pacific Northwest National Laboratory
Richland WA, USA
sriram@pnnl.gov

***Abstract*—Tensor contractions represent the most compute-intensive core kernels in ab initio computational quantum chemistry and nuclear physics. In this paper we develop a comprehensive framework for distributed tensor contractions on a torus network. The contraction algorithms are constructed using three basic data movement operations: Rotation, Recursive Broadcast and Reduction. A characterization is developed that classifies all possible mappings of the tensor elements and the computational iteration space onto processors, associating each mapping with the requisite data movement primitives. A cost model enables the selection of the algorithm with minimal communication overhead.**

We then develop an efficient approach to contract symmetric tensors. We introduce a novel approach that avoids data redistribution in contracting symmetric tensors while avoiding redundant storage and maintaining load balance. We present experimental results on two parallel supercomputers, for several symmetric contractions that appear in the CCSD coupled cluster quantum chemistry method.

I. Introduction

Tensor contractions are higher dimensional analogues of matrix matrix products, and comprise the computationally dominant operations in most many body methods in computational physics and chemistry, such as coupled cluster methods [2], [7]. Production parallel computational chemistry software suites such as ACES [8], GAMESS [10], NWChem [18] partition tensors into blocks that are distributed among nodes in a distributed-memory parallel computer in order to exploit parallelism in performing tensor contractions. Unlike the case for matrix-matrix multiplication, for which several efficient distributed-memory parallelization schemes have been developed [17], [14], [12], [19] that represent different space-time trade-offs, when these codes were developed, no communication optimized distributed parallel tensor contraction algorithms were known for tensors.

A significant complication in developing effective distributed tensor contraction algorithms is that the tensors used in computational models in quantum physics and chemistry exhibit symmetry over multiple dimensions, and exploitation of the symmetry is critical, both in order to save storage as well as avoid unnecessary arithmetic operations. Therefore all existing parallel quantum chemistry suites represent the distinct elements of symmetric tensors in a blocked fashion, distribute the blocks in some uniform fashion among nodes of a parallel system and dynamically perform required movement of blocks to the nodes where block-block contractions are performed. That strategy was practically satisfactory until recent times, but with the number of cores on parallel systems exceeding hundreds of thousands, the communication overhead of distributed tensor contractions now limits the scalability of coupled cluster methods [13].

Although some studies over the last two decades have studied the optimization of communication for parallel tensor contractions [5], [9], a significant limitation of most of these efforts is that symmetry in tensors was not exploited in the communication optimized schemes, and therefore not practically useful for quantum chemistry codes. The first distributed algorithm for symmetric tensors, called the Cyclops Tensor Framework (CTF), was recently developed by Solomonik et al. [17]. In this paper, we present a systematic framework to derive communication efficient implementations of tensor contractions expressions on distributed memory machines, with the CTF approach corresponding to one scheme in this classification. We identify three fundamental building block operations needed – recursive broadcast, rotation, and reduction. We determine the constraints in mapping the tensors to a multidimensional processor grid and bound the space of configurations to be considered. We develop a cost model to identify the most communication efficient configuration and the corresponding implementation. Multidimensional equivalents of 2.5D SUMMA for matrix multiplication are automatically derived in the framework.

Using the new framework, we then develop efficient distributed algorithms for contraction of symmetric tensors. These algorithms represent the first proposed approach to avoid the need for multiple redistributions when performing communication optimized contraction of symmetric tensors. Experimental results for representative contractions from the CCSD coupled cluster method are reported for a Cray XE6 and BlueGene/Q system, demonstrating effectiveness.

This paper makes the following contributions:

- It presents a systematic framework for developing efficient distributed tensor contraction algorithms for arbitrary dimensional dense and symmetric tensors. This represents an advance towards developing an understanding of parallel tensor contraction algorithms that compares with the comprehensiveness of our understanding of parallel matrix multiplication.

- It develops an approach to perform distributed contraction of symmetric tensors without requiring explicit transpose of the tensors during a contraction. To the best of our knowledge, this is the first such scheme to be developed. The system is planned to be made publicly available[1] later this year.
- It develops a cost model to determine the best distributed contraction scheme based on tensor dimensions and symmetry characteristics, processor dimensionality and communication parameters, as well as available memory. When additional collective memory beyond the minimal to hold all tensors is available, it is automatically exploited to select a *higher dimensional* scheme with lower communication overhead, analogous to the 2.5D and 3D algorithms known for matrix multiplication.
- It presents an experimental evaluation of the system, demonstrating its effectiveness.

II. RRR Framework

A. Overview

Consider the following tensor contraction:

$$C[o_{1\ldots k\ldots n}] = \sum_{i_1\ldots i_m} A[o_{1\ldots k}, i_{1\ldots m}] \times B[i_{1\ldots m}, o_{k+1\ldots n}]$$

A and B are input tensors that contract to produce tensor C. The labels for the indices only serve to match the dimensions involved in a contraction. $o_1, \ldots$ are the external indices, i.e., indices from the input tensors that appear in the result. $i_1, \ldots$ are contraction or internal indices that are common across both inputs and are summed over. Let the dimensionality A, B and C are d_A, d_B and d_C respectively. There are d_C external indices and $\frac{d_A+d_B-d_C}{2}$ contraction indices in this contraction. Hence the dimensionality of the iteration space or the computational space of this contraction is given by $d_C + \frac{d_A+d_B-d_C}{2}$. For a p-dimensional torus, the framework works as follows.

- Consider all possible mappings of the iteration space to the p-dimensional torus.
- For each mapping consider all possible contraction algorithms.
- For each contraction algorithm, determine the possible data space mapping of tensors.
- Model the total cost of contraction and memory requirement based on the algorithm and data space mapping.
- Choose the algorithm and distribution with the lowest cost that satisfies the memory constraints of the system.

The framework exhaustively searches through all iteration space mappings and data space mappings, and gives the optimal algorithm within this space. In this section we describe the first three steps above in detail. The remaining two will be explained in Section IV

[1]The system can be made available to reviewers if desired

B. Iteration Space Mapping

Consider all valid ways that a l-dimensional iteration space can be mapped to a t-dimensional torus network (excluding equivalent mappings because all network dimensions are equivalent). Each dimension of the iteration space corresponds to either an external or a contraction index. We will call them external and contraction iterator respectively.

- Each dimension of the iteration space must be either distributed along some dimension of the torus or serialized. A dimension i of the iteration space is distributed along dimension p_i of the grid if each node along p_i holds a range of iteration points along i in a tiled fashion. A dimension i of the iteration space is serialized if every node in the grid holds all the iteration points along i.
- No two dimensions of the iteration space can be mapped to the same dimension of the torus. Such a mapping has no meaning. An iteration space consists of all points in the product set given by the cartesian product of all the points in each dimension. It is not possible to form full cartesian product between two dimensions of the iteration space if they are both mapped to the same dimension of the torus.
- Iteration space must be replicated along those dimensions of the torus where no dimension of the iteration space is mapped. This implies redundant computation.

The mapping of the iteration space precisely defines where each computation of a tensor contraction occurs and therefore defines the data that needs to be present in each node. However, all the data required by the iteration space mapping at a particular node may not be immediately available. In this case data needs to be communicated. In the next sub-sections we formalize the communication operators and describe how contraction algoritms are constructed using these operators.

C. Data Communication Operators

Each dimension of the iteration space either corresponds to a contracting or an external iterator. In the iteration space mapping described above, each dimension is either distributed or serialized.

Distribution of a contracting iterator i along some dimension p_i of the torus implies that the mapping produces partial results along p_i. They have to be combined to obtain final result. We will refer to the communication operation for combining partial results as $Reduction$.

Serialization of a contracting iterator implies an owner compute model where results are computed entirely on the node. If the data required for this computation is already present on the node no communication is needed otherwise the data communication is necessary. We refer to this communication as $Recursive\ Broadcast$.

Similarly serialization of an external iterator e along some torus dimension p_e will result in communication if all the data corresponding to e is not already present on the node. We refer to the communication required here as $Rotation$.

$Reduction$, $Recursive\ Broadcast$ and $Reduction(RRR)$ are the basic communication operations that allows us to construct contraction algorithm for all input data distributions. In the next subsections we analyse the input distributions, precisely define the RRR and elaborate on how RRR works with different input distributions.

D. Distribution of input tensors

A tensor can be mapped to a torus in the same way as the iteration space. Consider all valid ways that a t-dimensional tensor can be mapped to a p-dimensional torus network (excluding equivalent mappings because all network dimensions are equivalent).

- Each dimension of a tensor must either be distributed along some dimension of the torus or serialized.
- No two dimensions of a tensor can be mapped to the same dimension of the torus.
- A tensor must be replicated along dimensions of the torus where no dimension of a tensor is mapped.

Under this distribution, the relations between indices of the input tensors A and B dictates the kind of data movement that is required.

1) Distribution of Contraction Indices: Let k be a contraction index. Let k_A and k_B be the corresponding contraction index in A and B. k_A and k_B can each be either serialized on every node of the torus or distributed along some dimension of the torus. All different possible distributions of $k_A,\ k_B$ are:

- Distributed, Distributed - Aligned (DDA)
- Distributed, Distributed - Orthogonal (DDO)
- Serialized, Distributed or viceversa (SD)
- Serialized, Serialized (SS)

DDA refers to the case when both k_A and k_B are distributed along the same dimension of the torus and DDO refers to distribution along seperate dimensions.

2) Distribution of External Indices: Let e_A be an external index in A. e_A can be either serialized on every node of the torus or distributed along some dimensions. All different possible mappings of e_A are:

- Distributed-Conflicting (DC)
- Distributed-Exclusive (DE)
- Serialized (S)

e_A is Distributed-Exclusive if no external index of B is distributed along the same dimension of torus as e_A, it is Distributed-Conflicting(DC) otherwise.

It is not necessary to classify the relation between the distributions of external and contraction indices. Notice that a contraction is defined exactly the same for every element of C as sum over products of matching contraction indices in A and B. Similarly each element of C corresponds to a unique element of the product set formed by the cartesian-product of the external indices independent of the contracting indices. Hence, the external indices and contracting indices describe orthogonal aspects of a contraction. Therefore, it is not necessary to study the relation between the distributions of contraction and external indices on the torus.

E. Using RRR with different input distributions

For each of the distributions classified above, RRR can be used for communication required for contraction. SS and S does not require communication while the rest does. We start by elaborating on distributions(DDA, DDO, SD and DC) that require communication.

Let p_x and p_y be two dimensions of p-dimensional torus and n_x and n_y be number of processors along p_x and p_y. Let $P(x, y)$ represent a general node whose coordinates are x along p_x and y along p_y.

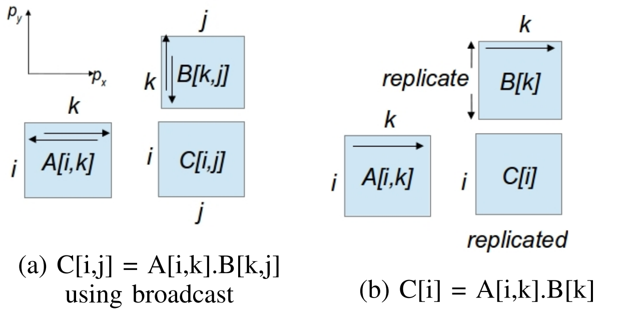


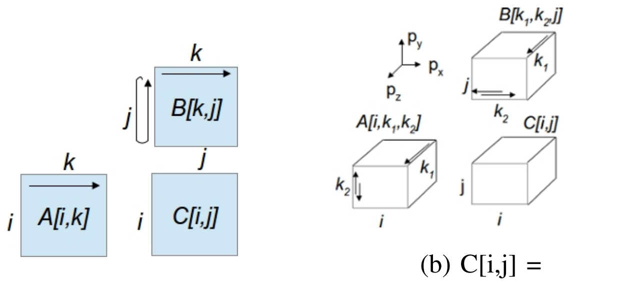
Figure 1. a) DDO Matrix Multiplication. The double arrows along p_x and p_y shows the dimension of the broadcast for k_A and k_B respectively. b) DDA vector product. Vector B[k] and C[i] are replicated along p_y and p_x respectively. The single arrow along p_x shows the dimension of reduction.

1) Recursive Broadcast with Distributed-Distributed-Orthogonal: Let contraction index k_A and k_B be distributed along processor dimension p_x and p_y respectively. Let k_A^m and k_B^m be range of values of k_A and k_B held at processor $P(m, y)$ and $P(x, m)$ respectively. Notice that except for the diagonal processors $P(m, m)$ the k_A and k_B data is not aligned. In order to contract k^m, k_A^m and k_B^m initially held at $P(m, y)$ and $P(x, m)$ needs to be held by all nodes. To do this $P(m, y)$ can broadcast k_A^m along p_x to $P(*, y)$ and $P(x, m)$ can broadcast k_B^m along p_y to $P(x, *)$. Now every processor $P(x, y)$ holds k_A^m and k_B^m and a local contraction k^m can be performed. This is done for each m so that entire contraction index k is contracted locally on each node.

This corresponds to serialization of the contracting iterator of the iteration space. This is the SUMMA algorithm for matrix multiplication. The algorithm is given in Listing. II-F and is depicted in Fig. 1.For a contraction involving multiple contraction index that are DDO, broadcasts are performed recursively along the dimensions of the torus where the contraction indices are distributed hence the name $Recursive\ Broadcast$.

2) Reduction with Distributed-Distributed-Aligned and Serialized-Distributed: Let contraction index k_A and k_B be distributed along processor dimension p_x. Let k_A^m and k_B^m be range of values of k_A and k_B held at processor $P(m, y)$. Notice that k_A^m and k_B^m are perfectly aligned on each node, i.e at node $P(m, y)$, k^m can be contracted since it has both k_A^m and k_b^m. After the local contraction each node along p_x i.e nodes $P(*, y)$ will hold partial result which can be summed using a $Reduction$. Notice that $Reduction$ can also be used for contraction indices that are Serialized-Distributed. Without loss of generality if k_A is serialized and k_B is distributed along p_x, then the nodes $P(m, y)$ that holds k_B^m also holds k_A^m since K_A is serialized. This is shown in Fig. 1

$Reduction$ corresponds to distribution of the contracting iterator since partial contractions are performed on each node along the dimension where contraction iterator is disributed.



(a) C[i,j] = A[i,k].B[k,j] Reduction and Rotation

(b) C[i,j] = A[i,k2,k1].B[k2,j,k1] Broadcast and Reduction

Figure 2. a) DC+DDA Matrix Multiplication. Both i and j are distributed along p_y and k_A and k_B is distributed along p_x. Hence, B is rotated along p_y and partial results are reduced along p_x. b) 2.5D SUMMA DDA+DDO Matrix Multiplication. $k2$ is DDO along p_x and p_y hece broadcast is done along these dimensions. $k1$ is DDA along p_z and hence reduction is done along p_z.

3) Rotation with Distributed-Conflicting: Let external indices e_A and f_B be distributed along p_x of the torus. Hence, e_A and f_B are conflicting. Let e_A^m and f_B^m be ranges of e_A and f_B held at some processor $P(m, y)$. Under such distribution there is no node which holds e_A^m and f_B^n where $m \neq n$. In otherwords a full cartesian product between e_A and e_B is not formed.

Since e_A and f_B are external indices that appears in output tensor C, these indices cannot be aligned and a full cartesian product needs to be formed between them. This can be done by rotating either A or B along p_x. Without loss of generality, during each step of rotation of B a processor $P(m, y)$ will recieve f_B^{m-1} from $P(m-1, y)$ and will send f_B^m to processor $P(m+1, y)$. Hence, after n_x steps of rotation, f_B will be completely serialized on each node. Hence, $Rotation$ corresponds to serialization of external iterator.Refer to Fig. 2.

4) Input distributions requiring no Communication: If the distribution of some contraction index k is SS, then entire range of k_A and k_B is available on each node. No communication is required to contract this index. Similarly if an external index e_A is serialized, the a full cartesian product is automatically formed between e_A and all other external indices, hence no communication is required w.r.t e_A.

F. Generating Contraction Algorithms and Input Distribution from Iteration Space Mapping

For a given iteration space map Fig. 3 shows how different algorithms and compatible data distribution can be derived. Each dimension of the iteration space corresponds to either a contraction iterator or an external iterator which is either distributed or serialized. When a contraction iterator is distributed it must correspond to a $Reduction$ and hence the contraction index in the tensors must me DDA. Notice that serialization of a contraction iterator can

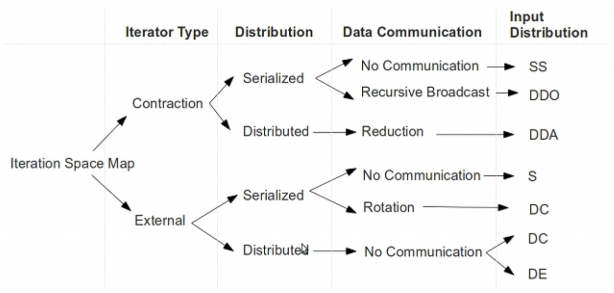


Figure 3. Scheme for deriving contraction algorithm and input distribution from iteration space mapping.

correspond to either no communication with SS distribution or $Recursive\ Broadcast$ with DDO. They however incur different communication cost(CCost) and differ in memory requirement(MReq). This will be elaborated in the Section IV of the paper. Similarly serialization of external iterator can either be done with no CCost corresponding to serialization of the external index or via $Rotation$ with DC distribution of external index. Lastly, the distribution of an external iterator always implies distribution of the corresponding external iterator which can be either DC or DE.

A general contraction algorithm for any iteration space mapping is given in Listing. 1. The algorithm works by first aligning all DDO contracting indices using $Recursive\ Broadcast$. Local contractions are performed once the indices are aligned. Once the DDO indices are contracted a reduction is done for indices that are DDA. This completes the algorithm in case where no external indices are DC, otherwise an input tensors with DC external indices corresponding to serialized external iterators are $rotated$ one step at a time. The RRR process is repeated until a full rotation has completed.

G. Distribution of the Output Tensor

The distribution of the output tensor C is determined by the distribution of the input tensors and the iterator space mapping. Indices of C corresponding to DE external indices of input tensors will have the same distribution in the output. An external index that is DC will be serialized if it corresponds to a serialized iterator otherwise it will stay distributed in the output. Alternatively, DC external indices with corresponding serialized external iteratrs may be distributed in the output if there exists some dimensions of the grid where no external index is mapped. As an example consider $C[i, j] = A[i, k].B[j, k]$. Let the tensors be mapped to a 2-D torus with dimensions p_x and p_y. Here i and j are overlapped along p_x and no external index are mapped along p_y. The contraction index is distributed along p_y. The contraction can be performed using $Reduction$ and $Rotation$. Index j can be redistributed along p_y in the output by reducing at the corresponding nodes along p_y for each j.

H. Dimension Scaling

The dimensionality of a torus grid varies from one machine to another. If this dimensionality is larger than that of

Listing 1. General tensor contraction algorithm

```
e_conflict = DC external indices from B
c_aligned = DDA contraction indices
c_ortho = DDO contracting indices
num_rotation =
    product(grid_size(x) for x in e_conflict)

for i in range(0, num_rotation):
    if c_aligned >= 1 and c_ortho >= 1:
        recursive_broadcast(c_ortho, c_aligned)
        reduction(c_aligned)
    else if c_aligned == 0 and c_ortho >= 1:
        recursive_broadcast(c_ortho)
    else if c_aligned >= 1 and c_ortho == 0:
            local_computation(c_aligned)
            reduction(c_aligned)
    rotate_one_step(e_conflict)
```

Listing 2. Recursive SUMMA

```
c_ortho = DDO contraction indices
r_summa(a, b, c_buf, c_orthogonal):
    k = c_ortho.pop()
    A_d, B_d = Dimension of k in A and B
    for x in range(0, len(k)):
        if A_d is not serialized:
            if my_rank == x:
                BCAST(A[k]) to A_d
            else: RECEIVE(A[k])
        if B_d is not serialized:
            if my_rank == x:
                BCAST(B[k]) to B_d
            else: RECEIVE(B[k])
        if is_empty(c_ortho):
            local_dgemm(a[k], b[k], c_buf)
        else:
            r_summa(a[k], b[k],
                c_buf, c_ortho)
```

Listing 3. Rotate A

```
e_conflict = DC external indices of A
rotate(a, e_conflict):
  k = e_conflict.pop();
  d_k = dimension where k is distributed
  for i in range(0,len(d_k)):
      next     = node_ID corresponding to
                 to the next node along d_k
      previous = node_ID corresponding to
                 previous node along d_k
      SEND a to next
      RECEIVE a from previous
      if !r_conflict.is_empty:
         rotate(a,e_conflict)
```

the tensors, then the tensors will have to be replicated along those extra dimensions. Furthermore, if the dimensionality of grid is larger than that of the iteration space then redundant computation may be done. One solution to such redundancy is to map a higher dimensional grid onto a lower dimension. However, algorithms developed for this embedded lower dimensional grid will not be able to use high level of internode connections of the higher dimensional grid.

An alternative and a more efficient solution is to upscale the dimensions of the input tensor, in other words dimension splitting. For instance a single dimension of a tensor i with a range $0-15$ may be splitted into two dimensions $i1$ and $i2$ with range $0-3$ and $0-3$ respectively. Dimension splitting will increase the dimension of input tensors if a contraction index is split or increase the dimensionality of both input and output tensors if an external index is split. Dimension splitting will also increase the dimensionality of the iteration space therefore avoid redundant computation or replication of tensors while fully taking advantage of the higher dimensional torus network. An example of such a contraction is 2.5D SUMMA where a matrix multiplication is performed in a 3D grid by dimension splitting and using a combination of $Broadcast$ and $Reduction$ show in Fig. 2.

III. Contraction Algorithm for Symmetric Tensors: CAST

A. Introduction to Symmetric Tensors

A tensor is symmetric with respect to a subset of its indices if permuting the indices within the subset does not change the value of the tensor. As an example, consider a tensor $v[a,b,i,j]$. We say that indices a and b are symmetric if $v[a,b,i,j] = v[b,a,i,j]$. A symmetric tensor can have multiple symmetry groups.For example, we say a, b and i, j are two symmetric groups of v if $v[a,b,i,j] = v[b,a,i,j] = v[b,a,j,i] = v[a,b,j,i]$. This tensor can be stored in a compact form $v[a<b, i<j]$ due to its symmetry. The symmetric properties imply that only $1/d!$ values of the full tensor need to be stored in the memory, where d is the dimensionality of a symmetric group. In case of v there are two symmetry groups of size 2, hence $\frac{1}{2} \times \frac{1}{2}$.

The primary difficulties in supporting a symmetric tensor contraction are two folds.

- Providing load balanced data distribution and computation.
- Providing efficient communication scheme. Dense tensor contraction algorithms do not work as only unique elements of symmetric tensors are stored.

The tensor distribution can be load balanced by using a cyclic or a block cyclic distribution where a symmetric tensor is divided into blocks which are then distributed in round robin fashion along the dimensions of a tori. For example consider Fig. 4. It shows block cyclic distribution of an 8x8 tensor grid of symmetric tensor A across a 2x2 physical processor grid. The 4 physical processors are marked in 4 different colored dots. Each processor holds several blocks in the grid. Let us index tensor blocks with small letters and physical grid with capital letters. Assuming $0 \leq k \leq 7$, consider column $k = 1$ of Ar. The corresponding section of A is distributed across the yellow and green processors. However, the column k data expected in the upper triangular part of A is stored at the blue processor instead of yellow due to symmetry.

This displacement of data in a block-cyclically distributed symmetric tensor makes deriving a communication efficient scheme for tensor contraction not so trivial. In this section we present a novel algorithm that expands on the recursive broadcast scheme to support symmetric tensors.

For ease of understanding we start with 2D symmetric matrix multiply. We then present a slightly more general

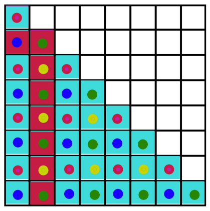


Figure 4. Block cyclic distribution

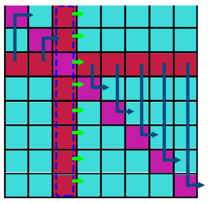


Figure 5. Communication pattern

4D symmetric tensor contraction. We finally present a completely general scheme for symmetric tensor contractions.

B. A Symmetric matrix Multiply

Consider a 8x8 physical processor grid as shown in Fig. 5. A symmetric tensor A is block cyclically distributed over this grid. The k^{th} column of A is stored at all the processors in the K^{th} row and column due to symmetry and the block cyclic distribution. These processors are marked in red in Fig. 5.

Recall that in SUMMA(broadcast for matrix-multiplication) for non-symmetric tensors, in order to contract column k of A with row k of B, each processor $P(Y, K)$ has to broadcast the data k_A it holds to all processors $P(Y, *)$. When A is symmetric, this data is held at $P(Y, K)$ and $P(K, Y)$. Hence, to achieve the broadcast, $P(Y, K)$ can first recieve the data from $P(K, Y)$ via $P(Y, Y)$ shown in pink in Fig. 5. Then it can broadcast the combined data to $P(Y, *)$. Similar process is repeated for B. We will define instigation as the process of collecting all the data required before doing the broadcast. The processors collecting the data are instigators and the ones sending the data are senders. Hence a broadcast scheme can be modified to work with symmetric tensors by adding an instigation before each broadcast.

C. Symmetric 4D contraction

$$C[i, j, d, l] = A[i, j, k, l] \times B[k, d] \tag{1}$$

Consider the above contraction in which A has symmetry between all four dimensions while B is nonsymmetric and k is the contracting index. Assume a 4D-physical processor grids. A is block cyclically distributed across four dimensions, while B is similarly distributed across two dimensions and replicated in the rest. Processors are referred as $P[I_1, I_2, I_3, I_4]$ as per their location in the grid.

Since there is a 4D-symmetry $[i > j > k > l]$ in A, for a particular value of k, the processor $P[I_1, I_2, K, I_4]$ require the blocks of A collectively represented by $A[i > j > l, k]$. These blocks can be grouped as $A[k > i > j > l], A[i > k > j > l], A[i > j > k > l]$ and $A[i > j > l > k]$.

All the aforementioned groups of blocks are located at processors $P[K, I_1, I_2, I_4]$, $P[I_1, K, I_2, I_4]$, $P[I_1, I_2, K, I_4]$ and $P[I_1, I_2, I_4, K]$ respectively. Processor $P[I_1, I_2, K, I_4]$ needs to instigate a data collection from all these processors before it can do a broadcast. The senders send their respective data via the diagonal processors. For instance, data held by $P[K, I_1, I_2, I_4]$ is sent via $P[I_2, I_2, I_3, I_4]$ and then via $P[I_2, I_3, I_3, I_4]$. We will refer to these diagonal processors as bouncers. In this contraction, for each value of the contraction index, an instigation is required as described above before a broadcast can be done for A. Since B is not symmetric, it can be directly broadcasted.

Listing 4. General CAST

```
CAST(A, B, C, cidxs):
    /* A,B,C : Tensors */
    /* cidxs : contraction indicies */
    c = cidxs.pop(0)
    for c in range(0, len(c)):
        if processor_id is instigator for A:
            Collect data in A_recieve
            Broadcast A_recv
        if processor_id is sender for A:
            Send data to instigator
        if processor_id is receiver for A:
            Receive broadcasted data A_recv
        /* Repeat for B */
        if len(cidxs) == 0:
            dgemm(A_recv, B_recv, C)
        else:
            CAST(A_recv, B_recv, C, cidxs)
```

D. Generalization of CAST

In the previous example the presence of a 4D symmetry group required data from four different physical nodes to be combined together before being broadcasted. In general, for a n-dimensional symmetry group, data from n physical nodes need to be bounced combined together. To clarify consider an n-dimensional tensor: $A[i_1, i_2, ..., i_k, ..., i_n]$ with n-dimensional symmetry $i_1 > i_2 > .... > i_k > .... > i_n$ stored at physical node $P[I_1, I_2, ..., I_k, ..., I_n]$where i_k is the contracting index. For some particular value of i_k, data collectively represented by

$$A[i_1 > i_2 > ....i_{k-1} > i_{k+1} > .... > i_n, i_k] \tag{2}$$

needs to be broadcast to $P[I_1, I_2, ..., I_{k-1}, *, I_{k+1}, ...I_n]$. The data represented by Eq. 2 has to be collected from the following processors

$$P[\Pi\left(I_1, I_2, ..., I_{k-1}, K, I_{k+1}, ...I_n\right)] \tag{3}$$

at $P[I_1, I_2, ..., I_{k-1}, K, I_{k+1}, ...I_n]$ where $K = I_k$ and Π gives all possible permutation of the indices such that I_{j-1} always comes before I_j where $1 \leq j \leq n$ and I_{k-1} always comes before I_{k+1} in sequence of indices. In other words the lexicographical ordering is preserved among all indices except K. This ensures that the relative ordering of the symmetric indices are preserved.

The general algorithm for contracting any symmetric tensors is: Let A be a tensor with m_A symmetry groups and let B be a tensor with m_B symmetry groups. Let $s^a_j | 0 < j < m_A$ represent the j^{th} symmetry group of A. Similarly $s^b_j | 0 < j < m_B$ represent the j^{th} symmetry group of B. Let ns^a and ns^b be the set of nonsymmetric indices in A and B respectively. Let the total number of contraction indices be n, denoted by $c_j|1 < j < n$. In order to identify the symmetry groups they belong to, let $c_1^{i_a}$ denote that the first contraction index that belongs to the i_a^{th} symmetry group in A. Let C^{i_a} be the set of all contraction indices belonging to the i_a^{th} symmetry group in A. Let CI be a list of all contraction indices.

Let the set of all indices in A be Ind_A and Ind_B be the set of all indices in B. A general tensor contraction is of the form:

$$C[(Ind_A - CI) \bigcup (Ind_B - CI)] = A[s_1^a, s_2^a, ...s_{m_A}^a, ns^A] \times B[s_1^b, s_2^b, ...s_{m_B}^b, ns^B] \tag{4}$$

Consider the physical grid where A and B are distributed: Let the set of indices s_j^a be distributed along dimensions S_j^a and indices s_j^b be distributed along dimensions S_j^b. The contraction index $c_k^{j_a}$ is distributed along dimension $C_k^{j_a} \in S_j^a$ and similarly for B. Now the distribution of A and B can be expressed as $P[S_1^a...S_{m_A}^a, nS^a]$ and $P[S_1^b...S_{m_B}^b, nS^b]$. Let the set of indices $s_j^a = i_1...i_{n_j^a}$ where $i_x \in s_j^a$. Similarly, the set of physical indices $S_j^a = \{I_1...I_n\}_j^a$.

For contraction index $c_k^{j_a}$ in A, the data collectively represented by

$$A[s_1^a, s_2^a, .., s_{j-1}^a, \{i_1 > i_2 > ... > i_{k-1} > i_{k+1} > ... > i_n\}_j^a, s_{j+1}^a.., s_{m_A}^a, nS^a, i_k] \tag{5}$$

where $i_k = c_k^{j_a}$ needs to be broadcasted among processors represented by Eq. (6). Using Eq. (3), the data is collected from Eq. (7) at processor Eq. (8)

$$P[S_1^a, .., S_{j-1}^a, \{I_1, .., I_{k-1}, *, I_{k+1}, ..., I_n\}_j^a, S_{j+1}^a.., S_{m_A}^a, nS^a] \tag{6}$$

$$P[S_1^a, S_2^a, .., S_{j-1}^a, \{\Pi\,(I_1, I_2, .., I_{k-1}, K, I_{k+1}, ..., I_n)\}_j^a, S_{j+1}^a.., S_{m_A}^a, nS^a] \tag{7}$$

$$P[S_1^a, S_2^a, .., S_{j-1}^a, \{I_1, I_2, .., I_{k-1}, K, I_{k+1}, ..., I_n\}_j^a, S_{j+1}^a.., S_{m_A}^a, nS^a] \tag{8}$$

For a given contraction index $c_i^{j_a}$, let us call the processors given by Eq. (6) `receivers`, Eq. (7) `senders` and Eq. (8) `instigator`. Let `A_recieve` be the data represented by Eq. (5). The pseudocode for a general contraction given by Eq. (4) can be represented as Listing. (4). The structure of this algorithm is very similar to that of recursive broadcast. In recursive broadcast for dense tensor, for a given iteration of the contraction index, processors holding the data can initiate a broadcast. In a symmetric tensor, none of the nodes initially hold all the data needed for the broadcast. An instigation is done to collect data from senders by instigators. Once this data is collected, the broadcast can proceed.

The ability to contract one iteration at a time makes it possible to contract only those iterations that preserves the ordering of contracting indicies. For example, if there are x number of contraction indicies belonging to the same symmetry group $c_1, c_2, ..c_n$ then, contraction needs to be performed only for $c_1 > c_2 > c_3... > c_x$. The result can then be multiplied by $x!$ to obtain the correct result.

E. Using a virtual grid for non-uniform physical grid

CAST works on perfectly square tori, i.e all the dimension of the physical grid along which symmetric indices are mapped have to be equal. The sizes do not have to be equal along different symmetry groups but within the same symmetry group. To overcome this restriction, we introduction a virtual grid. In this scheme, CAST will run on a perfectly square virtual grid which can be mapped to any rectangular physical grid. For instance, a 6x6 virtual grid may be mapped to a 6x3 physical grid where each physical node will be performing the work of two virtual nodes.

IV. Cost Model

In this section we present the cost model for RRR framework followed by the cost of CAST. The cost model for RRR framework predicts the optimal contraction algorithm and data space mapping within the framework. It works by iterating over all iteration space mapping. For each iteration space mapping, it considers all possible contraction algorithms and data space mappings given by Fig. 3. For each of these it generates a cost function based on the RRR components and chooses the one with the lowest cost under the memory constraints. CAST is a modification to $recursive\_broadcast$ in RRR framework. CAST involves an instigation before each broadcast. The cost of CAST is the sum of cost of instigations and the $recursive_broadcasts$.

In the following subsections we elaborate on the cost function and the memory requirement for a contraction algorithm and its data space mapping.

A. Cost function based on RRR

The general algorithm for any iteration space mapping is given by Listing. 1. The cost function is simply the cost of each RRR component of the general algorithm given by

$$\begin{aligned} Total\_cost = & num\_rotation \times \\ & (cost(recursive\_broadcast + local\_computation) + \\ & cost(reduction) + t_s + t_w \times m) \end{aligned}$$

Each component of the equation above is elaborated in the following subsections.

1) Cost of Rotation: $num\_rotation$ is the total number of times an input tensor is rotated due to presence of DC external index. If p_i is the torus dimension along which a DC external index is distributed and n_i is the number of processors along p_i then $num_rotation = \Pi n_i$. t_s and t_w are the latency and bandwidth of the torus network and m is the size per node of the input tensor that needs to be rotated.

2) Cost of Recursive Broadcasts and Local Computation: For a message size of m, the cost of broadcasting it along a torus dimension p_i with n_i processors is $log(n_i) \times (t_s + t_w \times m)$. Let the size of message per node be m_A and m_B for tensors A and B respectively. Let k be a DDO contraction index and let k_A and k_B be distributed along p_x and p_y

respectively. For simplicity let $n_x = n_y$. Then total cost for broadcasting and local computation will be

$$n_x \times log(n_x) \times m_A + n_x \times log(n_x) \times m_B + n_x \times \gamma(m_A, m_B) \quad (9)$$

where gamma gives the local computation time which is $total\_flops/flops\_per\_second$ based on machine parameters. For tensor contractions with multiple DDO contraction indices, this cost will be defined recursively. Let the total number of DDO contraction indices be b. Then the total cost of recursive broadcast and local computation is given by

$$cost\,(b) = n_x \times log\,(n_x) \times m_A + n_x \times log\,(n_x) \times m_B + n_x \times cost\,(b-1) \quad (10)$$

The base case is given by Eq. 9. The log terms that appears in the cost can be made linear using a pipelined version that we use for our implementation. For interested readers we refer to the SUMMA paper by Geijn.at.all. For matrix multiplication SUMMA algorithm corresponds to Broadcast and Local Computation.

3) Cost of Reduction: The cost of reduction has been studied rigorously and several algorithms exists. Here, we present the most well known cost. If a data of size m needs to be reduced amoung n processors, the cost is $log(n)(t_s + t_w \times m + \gamma(m))$ where $\gamma(m)$ gives the computational cost of adding two blocks of size m. t_s and t_w are latency and bandwidth of the torus.

B. Memory required for a contraction

The memory per node for a contraction is given by the distribution of the input tensors. Remember that each dimension of the tensor is either distributed or serialized. Let i be a dimension of the tensor distributed over dimension p_i of the torus. Let size of p_i be n_i. Let the total size of the tensor be S. So the memory required for this tensor per node is $\frac{S}{\Pi n_i}$. Hence the memory required per node is the sum pf memory required by input and output tensors. Additionally memory will be required for send and recieve buffers. The maximum buffer requirement will at most the memory required for the input tensors since RRR sends and recieves data to and from at most one processor at a time. In the experimental section of this paper we will compare the prediction of our cost model with actual results for a 3D tensor contraction.

C. Cost Model for CAST

In the case of a symmetric tensor, if the contraction indices are stored in full, a non-symmetric recursive broadcast scheme can be applied. At a single recursion level, representing a particular contraction index, the cost of communication is given by Eq. (11). This cost is derived from [19]. Consider distributed symmetric tensors A and B, with a contraction index stored in full. If the total size of these tensors distributed as described is M_A and M_B, their respective sizes per node would be $\frac{M_A}{P}$ and $\frac{M_B}{P}$ where P is the total number of nodes. If there are p number of nodes along the dimension of the virtual grid where the contraction index in A and B are distributed, then the total communication cost for the recursion level represented by this contraction index will be:

$$\begin{aligned} cost =& (k + 2p - 3)\left(\alpha + \frac{M_A \times p}{P \times k}\beta\right) \quad (11)\\ &+ (k + 2p - 3)\left(\alpha + \frac{M_B \times p}{P \times k}\beta\right) \end{aligned}$$

where k is the number of blocks along the contraction dimension that is block cyclically distributed among p virtual nodes. α is the latency term and β is the bandwidth term. Notice that we force the distribution dimension of the contraction index in both A and B to be of equal length p which is not necessary when the contraction index is stored in full but necessary otherwise. This is because all the symmetric indices are distributed with the same phase in CAST.

In the case when the contraction index is not stored in full, the `instigator` needs to collect data as described in Alg. (4). Without loss of generality, consider the data that needs to be collected by a virtual node $P[I1, I2, I3, K]$ for tensor $A[i1, i2, i3, k]$ where all the indices are symmetric. Using the diagonal bouncing scheme (which is essentially equivalent to a XY type routing scheme in a torus network), the data route can be expressed as

$$\begin{aligned} P[K, I1, I2, I3] &\rightarrow P[I1, I1, I2, I3] \rightarrow P[I1, I2, I2, I3] \quad (12)\\ &\rightarrow P[I1, I2, I3, I3] \rightarrow P[I1, I2, I3, K] \end{aligned}$$

Notice that all the `senders`, $P[\Pi\{K, I1, I2, I3\}]$ denoted by Eq. (3) follow a subsection of this route. Furthermore, notice that this path is unique to this `instigator` and no other `instigator` will use this path. This is true because the ordering among the indices are preserved, hence given a set of indices there can only be one such path that keeps the original ordering of the indices.

For a given iteration of the contraction index, the total data collected by the `instigator` is equal to the data that would have been at the `instigator` node if the contraction index was stored in full. This is $\frac{M_A \times p}{P \times k}$ for tensor A in Eq. (11). Furthermore, this is the maximum amount of data that flows through the unique route for each `instigator`. Hence, this implies that the cost of collecting data at the `instigator` for each iteration of the contraction index is given by $\alpha \times n + \frac{M_A \times p}{P \times k}\beta$. Therefore, the communication cost of CAST for a single level of recursion corresponding to a particular contraction index is

$$\begin{aligned} cost =& (2 \times k + 2p - 3)\left(\alpha + \frac{M_A \times p}{P \times k}\beta\right) \quad (13)\\ &+ (2 \times k + 2p - 3)\left(\alpha + \frac{M_B \times p}{P \times k}\beta\right) \quad (14)\\ &+ 2 \times k \times (n-1) \times \alpha \end{aligned}$$

The cost of communication at each recursion level can be added to obtain the total communication cost of performing CAST on the virtual grid.

1) Mapping square virtual grid to rectangular physical grid: CAST forces the symmetric dimensions of the virtual grid to have equal size, but that cannot be imposed on a physical grid. Without loss of generality, consider a symmetric tensor given by Eq. (2). The data distribution in the virtual grid is given by
$P[I_1, I_2, ..., I_k, ..., I_n]$, where all dimensions are of equal length. Given a physical grid whose size on each of the n dimensions are $f_1, f_2, ..., f_{n-1}, f_n$ respectively, preform a cyclic distribution of the virtual nodes to physical nodes. Given virtual nodes P_v and physical nodes P_p, let the length of each virtual grid dimension be p_v and the ratio between p_v and f_i be r_i. The data held at a virtual grid $P_v[I_1, I_2, ...I_N]$ maps to a physical node

$P_p[I_1\%f_1, I_2\%f_2, ...I_N\%f_n]$. For a given iteration of a contraction index i_k the data collected at physical node

$$P_p[I_1\%f_1, I_2\%f_2, K\%f_k, ...I_N\%f_n] \tag{15}$$

needs to be distributed among physical nodes

$$P_p[I_1\%f_1, I_2\%f_2, *, ...I_N\%f_n] \tag{16}$$

In fact, all virtual nodes of the form

$$P_v[I_1 + j_1 \times f_1, I_2 + j_2 \times f_2, ..., K, ..., I_n + j_n \times f_n] \tag{17}$$

map to the same physical node Eq. (15) where $-r_i < j_i < r_i$ such that $0 \leq I_i + j_i \times f_i \leq p_v$. Since $f_i \times r_i \leq p_v$, the total number of virtual nodes that maps to the same physical node is $\Pi_{i=1}^{n,i\neq k} r_i$. Since a single physical node has to perform the task of multiple virtual nodes, the communication cost of a single virtual node is scaled by this factor.

In the case where a contraction index is stored in full, Eq. (11) gives the communication cost of data in the physical grid along that index in A:
footnotesize

$$cost = \left(k + 2f_k^A - 3\right)\left(\alpha + \Pi_{i=1}^{n,i\neq k} r_i \times \frac{M_A \times p_v}{P_v \times k}\beta\right) \tag{18}$$

where k is the total number of blocks along i_k, f_k^A is the total number of physical processors along I_k for A and P_v is the size of the virtual grid. Eq. (11) can be simplified to :

$$cost = \left(k + 2f_k^A - 3\right)\left(\alpha + \frac{M_A \times p_v}{P_p \times k \times r_k}\beta\right) \tag{19}$$

where $r_k = \frac{p_v}{f_k}$. Now let us consider the case where the contraction index is not stored in full. In the earlier section, we described the uniqueness of the route taken by `senders` to send data to the `instigator`. Here, we show that the route taken by the data from `sender` in the physical grid to its `instigator` is unique for $\Pi_{i=1}^{n,i\neq k} r_i$ where k `instigators` where i_k is the contraction index and n is total dimension of the tensor and $r_i = p_v/f_i$. Consider the data that needs to be collected by a physical node:

$$P_p[I_1\%f_1, I_2\%f_2, ...K\%f_k] \tag{20}$$

All the `senders` will take a subsection of the route given by

$$\begin{gathered}
P_p[K\%f_1, I_1\%f_2, I_2\%f_3, ..., I_{k-1}\%f_k] \\
\downarrow \\
P_p[I_1\%f_1, I_1\%f_2, I_2\%f_3...., I_{k-1}\%f_k] \\
\downarrow \\
P_p[I_1\%f_1, I_2\%f_2, I_2\%f_3...., I_{k-1}\%f_k] \\
\downarrow \\
P_p[I_1\%f_1, I_2\%f_2, I_3\%f_3...., I_{k-1}\%f_k] \\
\downarrow \\
.... \\
P_p[I_1\%f_1, I_2\%f_2, ..., I_{k-1}\%f_{k-1}, I_{k-1}\%f_k] \\
\downarrow \\
P_p[I_1\%f_1, I_2\%f_2, ..., K\%f_k]
\end{gathered}$$

Each step of the route described above is unique to at most $MAX_{m=1}^n \left(\Pi_{i=1}^{n,i\neq m} r_i\right)$ virtual nodes. The total data collected by a single `instigator` in the virtual grid is equal to the data that would have been present at that node, if the contraction index was stored in full which is $\frac{M_A \times p}{P \times k}$. Hence the total data that flows through the route described above can be bounded by

$$MAX_{m=1}^n \left(\Pi_{i=1}^{n,i\neq m} r_i\right) \times \frac{M_A \times p}{P \times k} \tag{21}$$

The communication cost for A for a given contraction index is given by

$$\begin{aligned}
cost = &\left(k + 2f_k^A - 3\right)\left(\alpha + \frac{M_A \times p_v}{P_p \times k \times r_k}\beta\right) \\
&+ k \times MAX_{m=1}^n \left(\Pi_{i=1}^{n,i\neq m} r_i\right) \times \left(\alpha + \frac{M_A \times p_v}{P_v \times k}\beta\right)
\end{aligned} \tag{22}$$

which can be re-written as

$$\begin{aligned}
cost = &\left(k + 2f_k^A - 3\right)\left(\alpha + \frac{M_A \times p_v}{P_p \times k \times r_k}\beta\right) \\
&+ k\left(\Pi_{i=1}^n r_i \times \alpha + \frac{M_A \times p_v}{P_p \times k}\beta\right) \times \frac{1}{MIN_{i=1}^n (r_i)}
\end{aligned} \tag{23}$$

Notice that the communication cost is of the same order as performing a general SUMMA for the bandwidth term where the contraction index is stored in full. For a given virtual grid and physical grid r_i are constants.

V. Experiments

In this section we present results for three different experiments. We construct contraction algorithms for various iteration space mappings for a 3D tensor contraction within the RRR framework. We use the cost model to predict the fastest algorithm and compare theoretical predictions with the result. We then show comparision of communication time with Cyclops Tensor Framework(CTF). Finally we present scalability of CAST vs CTF using a wide range of symmetric tensor contractions.

A. *Cost Model Verification*

Consider the following contraction.

$$C[i, j, a, b] = A[i, j, k] \times B[k, a, b] \tag{24}$$

For simplicity of this example let the range of all the indices be n. Then $m_A = m_B = \frac{n^3}{p^3}$ and $m_C = \frac{n^4}{p^3}$.

Table I shows the iteration space mapping and data space mappings on to a 3D torus with p_x, p_y and p_z as its dimensions. Map_1 performs $rotation$ and $reduction$, Map_2 uses the $broadcast$ and $Map3$ does not need any communication. The memory requirement and total costs can be defined as per the cost model and is shown in Table II.

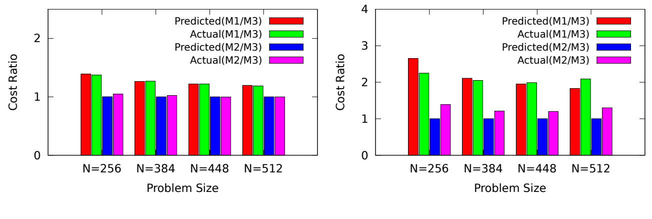


Figure 6. a) Actual vs Predicted on 256 BG/Q nodes. b)Actual vs Predicted on 256 Cray XE6 nodes.

For the purpose of model verification, we took the ratio of the time required to perform contractions using $Map1$, $Map2$ and $Map3$ on Cray XE6 and on BG/Q. The contractions were run on 256 nodes for 3D tensors A and B of size 256, 384 and 512 on each dimension. We used the cost model to predict the times . In Fig. 6, M1, M2 and M3

	i	j	a	b	k
Map1 1	p_x	p_y	s	s	p_z
Map1 2	p_x	p_y	p_z	s	s
Map1 3	p_x	p_y	p_z	s	s

$IterationSpaceMap$

PG	i1	i2	i3	s
A	i	j	k	
B	a	b	k	
C	i	j	a	b

$Map_1 : DC + DDA$

PG	i1	i2	i3	s
A	i	j	k	
B	r	k	a	b
C	i	j	a	b

$Map_2 : DE + S + DDO$

PG	i1	i2	i3	s	s
A	i	j	r		k
B	r	r	a	b	k
C	i	j	a	b	

$Map_3 : DE + S + SS$

Table I
ITERATION SPACE MAPPING OF EQ. 24 AND DATA SPACE MAPPING ONTO A 3D PHYSICAL TORUS. r DENOTES REPLICATION AND s DENOTES SERIALIZATION

Mapping	Memory required	Time Complexity
Map_1	$m_A + m_B + m_C$	$2\gamma p^2 \frac{n^5}{p^5} + 2p\left(m_C + \gamma\frac{n^4}{p^4}\right) + p^2\left(m_C + m_B\right)$
Map_2	$m_A + p \times m_B + m_C$	$2\gamma\frac{n^5}{p^3} + (3p-3)m_A + (3p-3)m_B \times p$
Map_3	$p \times m_A + p^2 \times m_B + m_C$	$2\gamma\frac{n^5}{p^3}$

Table II
PREDICTED TIME COMPLEXITIES FOR DIFFERENT MAPPINGS

represent $Map1$, $Map2$ and $Map3$. The cost model ranks M1>M2>M3 as the order for running time. The predicted and the actual values agree.

B. Comparision with CTF

The implementation of our framework is written in C/C++ and utilizes MPI for communication and BLAS for computation. Computational expensive routines are threaded using OpenMP. We compare our implementation with CTF [17] on seven symmetric representative tensor contractions from CCSD equations:

1	$C[i,j,a,b] = A[i,k,a>l] \times B[l,j,k>b]$
2	$C[i>j,a>b] = A[i>k,j>l] \times B[a>b,k>l]$
3	$C[i,a,j,b] = A[i>k,j>l] \times B[l,a,k,b]$
4	$C[i,a,j,b] = A[i>k,j>l] \times B[l,a,k>b]$
5	$C[i,a,j,b] = A[i>k,j>l] \times B[l>a,k>b]$
6	$C[i,a] = A[i>k] \times B[k>a]$
7	$C[i,j,a,b] = A[i,j,b>k] \times B[a>k]$

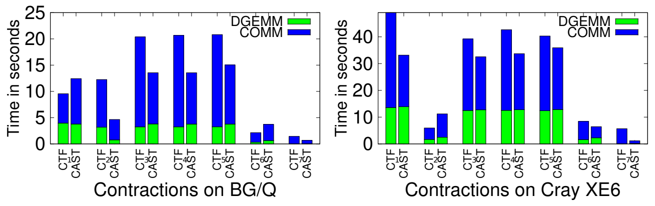


Figure 7. The figure on left shows the communication and computation time for seven contraction on 16,384 BG/Q nodes (262,144 cores) and the figure on the right shows times on 1,024 Cray XE6 nodes (24,576 cores).

Experimental results are collected on two state of the art supercomputer architectures, IBM Blue Gene/Q (BG/Q) and Cray XE6. The BG/Q consists of 16 cores (1.6GHz), 16 GB of memory per node and a 5D torus interconnect. The Cray XE6 supercomputer consisting of dual socket 12ncore AMD MagnyCours 2.1GHz and 32GB memory per node. Nodes are connected through a 3D torus Cray Gemini network, however, the scheduler on Cray XE6 does not support topology aware mapping. On each platform, we use vendor provided, optimized on node parallel BLAS implementation for benchmarking (IBM ESSL and Cray LibSci).

Fig.7 shows the communication and computation time taken by CAST and CTF for problem size N=384 on BGQ and N=256 on Cray XE6 for contractions 1,2,3,4,5 and 7, and N=32,768 for contraction 6. The larger size for contraction 6 was chosen because the total computational cost of contraction 6 is much smaller than the rest. On BG/Q, we report the result on 16,384 nodes with one MPI rank per node (total number of cores is 262,144). On Cray XE6, we ran on 1,024 nodes with four MPI ranks per node (total number of cores is 24,576). We use DGEMM to denote the time spent on BLAS computation, and COMM for rest of the time, the majority of which is communication.

C. Scalability of CAST

The seven contractions described in V-B can be categorized as 4D-4D (Contractions 1,2,3,4,5), 2D-2D (Contraction 6) and 4D-2D (Contraction 7). Of the 4D-4D contractions contraction 2 is the only one with symmetry in the external indices. Contractions 1, 3, 4 and 5 are similar contractions with varying degree of symmetry. Due to limited space we chose contractions 2, 3, 6 and 7 based on the aforementioned categorization to show scalability result of CAST compared to CTF. Fig.8 shows scalability of CAST and CTF on BG/Q. Notice that the slope of the scalability curve for CAST does not alter drastically for any contraction. For CTF the scalabilty trends are normal for contraction 6 and 7 but the slopes change directions for contractions 2 and 3 on 4,096 and 16,384 nodes. Based on the break down of different components of the ctf algorithm, at very large node size, packing and unpacking of data required to perform dgemm produces large overhead for contraction 2. Contraction 3 uses all reduce in its contraction algorithm to combine partial results. Doing all reduce on very large torus is expensive, which doesnt allow ctf to scale very well on 4096 and 16384 nodes. Fig.9 shows scalability of CAST and CTF on Cray XE6. The results show that both CAST and CTF have very similary slopes with different offsets. However, in terms of absolute performance CAST outperforms CTF on all but contraction 2. The primary reason for this is that CTF uses a fully cyclic distribution of tensors while CAST uses block-cyclic distribution. In cases where the resulting tensors are

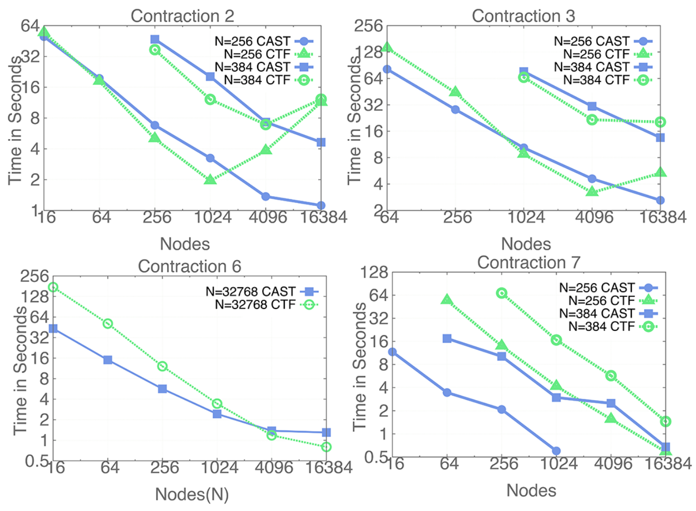


Figure 8. Scalability of contractions 2,3, 6 and 7 on upto 16,384 BG/Q nodes for N=256, 384 and 32768.

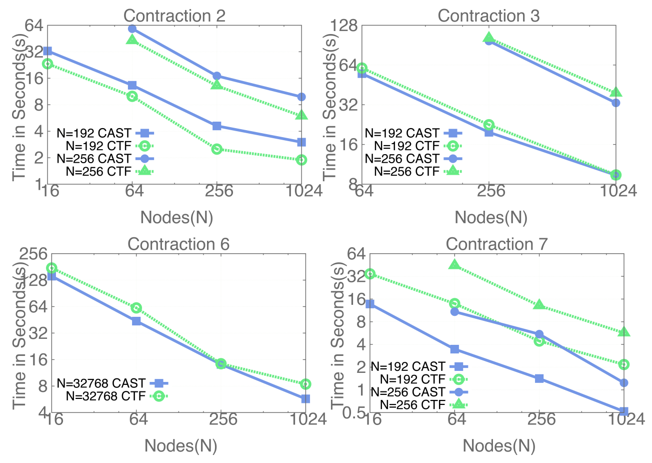


Figure 9. Scalability of contractions 2,3, 6 and 7 on upto 1024 Cray XE6 nodes for N=192, 256 and 32768.

symmetric, a block cyclic distribution is can cause small load imbalance which results in slightly lower performance.

Overall CAST outperforms CTF in most contractions, however both CAST and CTF both have strength and weaknesses that are apparent in Fig.7, 8, 9. The most optimal scheme could be something that combines the strengths of both these two schemes but that is beyond the scope of this paper.

VI. Related Work

Due to the prominent role of tensors in quantum chemistry, efficient execution of tensor contraction expressions have been extensively studied [3]. Efforts to minimize the number of operations in chains of tensor contraction expressions [11], minimize their memory requirement [6], and trade off increased computation cost to reduce space [4] have all been considered. None of these efforts directly attempted to minimize the communication costs.

Minimizing the communication volume using the Cannon's algorithm in the context of multiple tensor contractions under memory constrained was considered by Cociorva et al. [5]. Gao et al. [9] extended this algorithm by taking into account disk I/O costs. These algorithms only employed a 2D processor grid and also did not handle permutation symmetry in tensors.

Widely used implementations of tensor contractions, such as in the NWChem [18] computational chemistry suite, employ a simple dynamic load balancing scheme. In this scheme, the loops representing a tensor contraction expression are tiled and parallelized, akin to an OpenMP parallel loop. Each processor then dynamically determines the loop iterations to execute without concern for communication costs. While this was considered good enough in the past, recent work has shown that communication costs now impose fundamental limits on scalability of these methods. For example, Kowalski et al. [13] observed that while the computationally dominant ($O(N^7)$) noniterative triples calculation scales to hundreds of thousands of cores, the less expensive ($O(N^6)$) iterative portion does not scale beyond a few thousand cores. This disparity between the two sets of tensor contractions, has resulted in the less expensive iterative calculation requiring greater wallclock time than the noniterative calculation, thus becoming the computational bottleneck.

Solomonik et al. [17] created the first implementation of a communication optimized and load balanced distributed contraction algorithm for symmetric tensors. Their Cyclops Tensor Framework (CTF) uses a cyclic distribution of tensors across a multidimensional physical torus, thereby ensuring excellent load balancing of all tensors. Contractions involving symmetric tensors are achieved by using a number of passes of a generalization of the 2D SUMMA [19] algorithm, with dynamic redistribution of symmetric tensors in between passes. While effective, their approach is limited to employing SUMMA. When the data distribution does not match that required to employ the SUMMA algorithm, the tensors are transposed. Since a symmetric tensor stores only a portion of the actual data, tensor contractions with symmetry using CTF incur repeated redistribution costs to realign the dimensions being contracted.

Communication lower bounds and associated algorithms have been studied for various configurations of 2D matrix multiplication operations [12], [1]. In particular, 2D algorithms have been shown to incur asymptotically higher communication costs than algorithmic lower bound. 3D algorithms, which are communication optimal incur higher memory overheads. Solomonik and Demmel [16] presented the 2.5D algorithm, which is communication efficient [15] and considers a trade off between memory costs communication minimization.

Our approach differs in two significant ways from that of CTF: i) it considers a wider range of parallel implementations, and also considering additional communication schemes besides generalized 2D, 2.5D SUMMA, and ii) it avoids the need for multiple dynamic redistributions when contracting symmetric tensors.

VII. Conclusion

In this paper, we have presented a comprehensive framework for generating optimal algorithm and data mapping for contracting distributed tensors. We have also presented a novel and communication efficient algorithm for contracting tensors with symmetry. The cost model for our framework was verified and we showed scalability results for our contraction algorithm on BG/Q and Cray XE6. Comparisions with CTF revealed that CAST outperforms CTF in most situations, while still remaining competitive in the remaining.

References

[1] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Analysis Applications*, 32(3):866–901, 2011.

[2] R. Bartlett and M. Musia. Coupled-cluster theory in quantum chemistry. *Reviews of Modern Physics*, 79(1):291–352, 2007.

[3] G. Baumgartner, A. Auer, D. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Hirata, S. Krishnamoorthy, et al. Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models. *Proceedings of the IEEE*, 93(2):276–292, 2005.

[4] D. Cociorva, G. Baumgartner, C.-C. Lam, P. Sadayappan, J. Ramanujam, M. Nooijen, D. E. Bernholdt, and R. J. Harrison. Space-time trade-off optimization for a class of electronic structure calculations. In *PLDI*, pages 177–186, 2002.

[5] D. Cociorva, X. Gao, S. Krishnan, G. Baumgartner, C.-C. Lam, P. Sadayappan, and J. Ramanujam. Global communication optimization for tensor contraction expressions under memory constraints. In *IPDPS*, page 37, 2003.

[6] D. Cociorva, J. W. Wilkins, C.-C. Lam, G. Baumgartner, J. Ramanujam, and P. Sadayappan. Loop optimization for a class of memory-constrained computations. In *ICS*, pages 103–113, 2001.

[7] T. Crawford and H. Schaefer III. An Introduction to Coupled Cluster Theory for Computational Chemists. In *Reviews in Computational Chemistry*, volume 14, pages 33–136. 2000.

[8] E. Deumens, V. F. Lotrich, A. Perera, M. J. Ponton, B. A. Sanders, and R. J. Bartlett. Software design of aces iii with the super instruction architecture. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(6):895–901, 2011.

[9] X. Gao, S. K. Sahoo, C.-C. Lam, J. Ramanujam, Q. Lu, G. Baumgartner, and P. Sadayappan. Performance modeling and optimization of parallel out-of-core tensor contractions. In *PPOPP*, pages 266–276. ACM, 2005.

[10] M. S. Gordon and M. W. Schmidt. Advances in Electronic Structure Theory: GAMESS a Decade Later.

[11] A. Hartono, A. Sibiryakov, M. Nooijen, G. Baumgartner, D. E. Bernholdt, S. Hirata, C.-C. Lam, R. M. Pitzer, J. Ramanujam, and P. Sadayappan. Automated operation minimization of tensor contraction expressions in electronic structure calculations. In *ICCS*, pages 155–164, 2005.

[12] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, Sept. 2004.

[13] K. Kowalski, S. Krishnamoorthy, R. M. Olson, V. Tipparaju, and E. Apra. Scalable implementations of accurate excited-state coupled cluster theories: application of high-level methods to porphyrin-based systems. In *SC*. IEEE, 2011.

[14] B. Lipshitz, G. Ballard, J. Demmel, and O. Schwartz. Communication-avoiding parallel strassen: implementation and performance. SC '12, pages 101:1–101:11, 2012.

[15] E. Solomonik, A. Bhatele, and J. Demmel. Improving communication performance in dense linear algebra via topology aware collectives. In *SC*, page 77, 2011.

[16] E. Solomonik and J. Demmel. Communication-optimal parallel 2.5d matrix multiplication and lu factorization algorithms. In *Euro-Par 2011*, pages 90–109. 2011.

[17] E. Solomonik, D. Matthews, J. Hammond, and J. Demmel. Cyclops tensor framework: Reducing communication and eliminating load imbalance in massively parallel contractions. In *IPDPS*, pages 813–824, 2013.

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

[19] R. A. van de Geijn and J. Watts. Summa: scalable universal matrix multiplication algorithm. *Concurrency - Practice and Experience*, 9(4):255–274, 1997.