Algorithms for contraction of tensors over a commutative ring

Edgar Solomonik\textsuperscript{1}, Devin Matthews\textsuperscript{3}, and James Demmel\textsuperscript{1,2}

\textsuperscript{1} Department of Electric Engineering and Computer Science, UC Berkeley
\textsuperscript{2} Department of Mathematics, UC Berkeley
\textsuperscript{3} Department of Chemistry, UT Austin

ETH Zürich

June 18, 2014
Motivation:

- to exploit permutational symmetry present in tensors within contractions that break the symmetry
- coupled-cluster computations, which use 4th, 6th, and 8th order partially-symmetric tensors
Introduction

Motivation:

- to exploit permutational symmetry present in tensors within contractions that break the symmetry
- coupled-cluster computations, which use 4th, 6th, and 8th order partially-symmetric tensors

Sample coupled-cluster contractions

- CCSD: \( Z_{ij}^{ab} = P(a, b)P(i, j) \sum_m \sum_e W_{ei}^{am} \cdot T_{mj}^{eb} \)
- CCSDT: \( Z_{ijk}^{abc} = P(a, bc)P(i, jk) \sum_m \sum_e W_{ei}^{am} \cdot T_{mjk}^{ebc} \)
- CCSDTQ: \( Z_{ijkl}^{abcd} = P(a, bcd)P(i, jkl) \sum_m \sum_e W_{ei}^{am} \cdot T_{mjkl}^{ebcd} \)

where \( P(\ldots, \ldots) \) denotes antisymmetrization of two index groups
Overview of result

New algorithms that lower the number of multiplications but require more additions

- relevant not only for coupled-cluster, but even some BLAS routines
- algorithms require that scalar operations are on a commutative ring
New algorithms that lower the number of multiplications but require more additions

- relevant not only for coupled-cluster, but even some BLAS routines
- algorithms require that scalar operations are on a commutative ring

Consider non-associative commutative ring \( \rho \) with multiplication cost \( \mu_\rho \) and with addition cost \( \nu_\rho \)

- on the usual sum-product ring over reals: \( \mu_\rho = \nu_\rho \)
- on the sum-product ring over complex numbers: \( \mu_\rho = 3\nu_\rho \)
- on a Jordan (commutative) ring of matrices: \( \mu_\rho \gg \nu_\rho \)
Outline

1. Introduction
2. Symmetry in matrix computations
   - Matrix-vector multiplication
   - Symmetrized outer product
   - Symmetric matrices on the Jordan ring
3. Symmetric tensor contractions
   - Arbitrary fully-symmetric contractions
4. Communication cost analysis
5. Numerical error analysis
6. Nonsymmetric tensor contractions as a Jordan ring
7. Summary and conclusion
Symmetric matrix times vector

- Let \( b \) be a vector of length \( n \) with elements in \( \mathcal{Q} \)
- Let \( A \) be an \( n \)-by-\( n \) symmetric matrix with elements in \( \mathcal{Q} \)

\[
A_{ij} = A_{ji}
\]

- We multiply matrix \( A \) by \( b \),

\[
c = A \cdot b
\]

\[
c_i = \sum_{j=1}^{n} A_{ij} \cdot b_j
\]

this corresponds to BLAS routine \( \text{symv} \) and has cost (ignoring low-order terms here and later)

\[
T_{\text{symv}}(\mathcal{Q}, n) = \mu_{\mathcal{Q}} \cdot n^2 + \nu_{\mathcal{Q}} \cdot n^2
\]

where \( \mu_{\mathcal{Q}} \) is the cost of multiplication and \( \nu_{\mathcal{Q}} \) of addition
Fast symmetric matrix times vector

We can perform \texttt{symv} using fewer element-wise multiplications,

\[ c_i = \sum_{j=1}^{n} A_{ij} \cdot (b_i + b_j) - \left( \sum_{j=1}^{n} A_{ij} \right) \cdot b_i \]

- \( A_{ij} \cdot (b_i + b_j) \) is symmetric, and can be computed with \( \binom{n}{2} \) element-wise multiplications
- \( \left( \sum_{j=1}^{n} A_{ij} \right) \cdot b_i \) may be computed with \( n \) multiplications
- The total cost of the new form is
  \[ T'_{\text{symv}}(\varrho, n) = \mu_\varrho \cdot \frac{1}{2} n^2 + \nu_\varrho \cdot \frac{5}{2} n^2 \]

- This formulation is cheaper when \( \mu_\varrho > 3 \nu_\varrho \)
- Form \texttt{symm} the formulation is cheaper when \( \mu_\varrho > \nu_\varrho \)
Symmetric rank-2 update

Consider a rank-2 outer product of vectors \( \mathbf{a} \) and \( \mathbf{b} \) of length \( n \) into symmetric matrix \( \mathbf{C} \)

\[
\mathbf{C} = \mathbf{a} \circ \mathbf{b}^T \equiv \mathbf{a} \cdot \mathbf{b}^T + \mathbf{b} \cdot \mathbf{a}^T
\]

\[
C_{ij} = a_i \cdot b_j + a_j \cdot b_i.
\]

- For floating point arithmetic, this is the BLAS routine \( \text{syr2} \)
- The routine may be computed from the nonsymmetric intermediate \( K_{ij} = a_i \cdot b_j \) with the cost

\[
T_{\text{syr2}}(\varrho, n) = \mu_\varrho \cdot n^2 + \nu_\varrho \cdot n^2.
\]
We may compute the rank-2 update via a symmetric intermediate quantity

\[ C_{ij} = (a_i + a_j) \cdot (b_i + b_j) - a_i \cdot b_i - a_j \cdot b_j. \]

- We can compute the symmetric \( Z_{ij} = (a_i + a_j) \cdot (b_i + b_j) \) in \( \binom{n}{2} \) multiplications.
- The total cost is then given to leading order by

\[ T'_{syr2}(\varrho, n) = \mu_{\varrho} \cdot \frac{1}{2} n^2 + \nu_{\varrho} \cdot \frac{5}{2} n^2. \]

- \( T'_{syr2}(\varrho, n) < T_{syr2}(\varrho, n) \) when \( \mu_{\varrho} > 3\nu_{\varrho} \)
- \( T'_{syr2K}(\varrho, n, K) < T_{syr2K}(\varrho, n, K) \) when \( \mu_{\varrho} > \nu_{\varrho} \)
Given symmetric matrices $A, B$ of dimension $n$ on non-associative commutative ring $\varrho$, we seek to compute the \textit{anticommutator} of $A$ and $B$

\begin{align*}
C &= A \circ B \equiv A \cdot B + B \cdot A \\
C_{ij} &= \sum_{k=1}^{n} (A_{ik} \cdot B_{jk} + A_{jk} \cdot B_{ik}) .
\end{align*}

The above equations requires $n^3$ multiplications and $n^3$ adds for a total cost of

$$T_{\text{syrm}}(\varrho, n) = \mu_\varrho \cdot n^3 + \nu_\varrho \cdot n^3 .$$

Note that $\circ$ defines a non-associative commutative ring (the Jordan ring) over the set of symmetric matrices.
We can combine the ideas from the fast routines for symv and syrk by forming a fully-symmetric intermediate $Z$,

$$
Z_{ijk} = (A_{ij} + A_{ik} + A_{jk}) \cdot (B_{ij} + B_{ik} + B_{jk})
$$

$$
\tilde{Z}_{ij} = \sum_k Z_{ijk}
$$

$$
V_{ij} = A_{ij} \cdot \left( \sum_k B_{ij} + B_{ik} + B_{jk} \right) + B_{ij} \cdot \left( \sum_k A_{ij} + A_{ik} + A_{jk} \right)
$$

$$
W_i = \sum_k A_{ik} \cdot B_{ik}
$$

$$
C_{ij} = \tilde{Z}_{ij} - V_{ij} - W_i - W_j
$$

The reformulation requires $\binom{n}{3}$ multiplications to leading order,

$$
T'_{\text{syrm}}(\varrho, n) = \mu_\varrho \cdot \frac{1}{6} n^3 + \nu_\varrho \cdot \frac{5}{3} n^3,
$$

which is faster than $T_{\text{syrm}}$ when $\mu_\varrho > (4/5) \nu_\varrho$. 

Tensor index notation

To generalize the fast symmetric algorithm, we introduce some convenient notation for symmetric index sets (ordered tuples)

\[ k \langle v \rangle = (k_1, k_2, \ldots, k_v). \]
Tensor index notation

To generalize the fast symmetric algorithm, we introduce some convenient notation for symmetric index sets (ordered tuples)

\[ k^{\langle v \rangle} = (k_1, k_2, \ldots k_v). \]

We define an ordered union of tuples \( k^{\langle d + f \rangle} = i^{\langle d \rangle} \cup j^{\langle f \rangle} \) as concatenate and sort, and the set of all possible pairs of \( d \)-and-\( f \)-tuples whose ordered union is \( k^{\langle d + f \rangle} \) (disjoint partition of \( k^{\langle d + f \rangle} \)) as,

\[ \chi^d_f(k^{\langle d + f \rangle}) = \{(i^{\langle d \rangle}, j^{\langle f \rangle}) | i^{\langle d \rangle} \cup j^{\langle f \rangle} = k^{\langle d + f \rangle}, \forall i^{\langle d \rangle}, j^{\langle f \rangle}\}. \]
Tensor index notation

To generalize the fast symmetric algorithm, we introduce some convenient notation for symmetric index sets (ordered tuples)

\[ k\langle v \rangle = (k_1, k_2, \ldots k_v). \]

We define an ordered union of tuples \( k\langle d + f \rangle = i\langle d \rangle \cup j\langle f \rangle \) as concatenate and sort, and the set of all possible pairs of \( d \)-and-\( f \)-tuples whose ordered union is \( k\langle d + f \rangle \) (disjoint partition of \( k\langle d + f \rangle \)) as,

\[ \chi_f^d(k\langle d + f \rangle) = \{(i\langle d \rangle, j\langle f \rangle) \mid i\langle d \rangle \cup j\langle f \rangle = k\langle d + f \rangle, \forall i\langle d \rangle, j\langle f \rangle\}. \]

Accordingly, we denote all possible ordered subsets as

\[ \chi^d(k\langle d + f \rangle) = \{a \mid \forall (a, b) \in \chi_f^d(k\langle d + f \rangle)\}. \]
Tensor index notation

To generalize the fast symmetric algorithm, we introduce some convenient notation for symmetric index sets (ordered tuples)

\[ k\langle v \rangle = (k_1, k_2, \ldots, k_v). \]

We define an ordered union of tuples \( k\langle d + f \rangle = i\langle d \rangle \cup j\langle f \rangle \) as concatenate and sort, and the set of all possible pairs of \( d\)-and-\( f\)-tuples whose ordered union is \( k\langle d + f \rangle \) (disjoint partition of \( k\langle d + f \rangle \)) as,

\[ \chi^d_f(k\langle d + f \rangle) = \{ (i\langle d \rangle, j\langle f \rangle) \mid i\langle d \rangle \cup j\langle f \rangle = k\langle d + f \rangle, \forall i\langle d \rangle, j\langle f \rangle \}. \]

Accordingly, we denote all possible ordered subsets as

\[ \chi^d(k\langle d + f \rangle) = \{ a \mid \forall (a, b) \in \chi^d_f(k\langle d + f \rangle) \}. \]

We omit the subscript and superscript on \( \chi \) when it is implicitly evident, i.e. \( (i\langle d \rangle, j\langle f \rangle) \in \chi(k\langle d + f \rangle). \)
Fully symmetric tensor contractions

For some \( s, t, v \geq 0 \), we seek to compute,

\[
C = A \circ B
\]

\[
C_i^{s+t} = \sum_{(j^{s}, l^{t}) \in \chi(i^{s+t})} \left( \sum_{k^{v}} A_{j^{s} \cup k^{v}} \cdot B_{k^{v} \cup l^{t}} \right),
\]

where \( A, B, \) and \( C \) are all fully symmetric with dimensions \( n \).
For some \( s, t, v \geq 0 \), we seek to compute,

\[
C = \mathcal{A} \circ \mathcal{B}
\]

\[
C_{i\langle s+t \rangle} = \sum_{(j\langle s \rangle, l\langle t \rangle) \in \chi(i\langle s+t \rangle)} \left( \sum_{k\langle v \rangle} A_{j\langle s \rangle \cup k\langle v \rangle} \cdot B_{k\langle v \rangle \cup l\langle t \rangle} \right),
\]

where \( \mathcal{A}, \mathcal{B}, \) and \( \mathcal{C} \) are all fully symmetric with dimensions \( n \). The standard method forms the partially-symmetric intermediate \( \bar{C} \),

\[
\bar{C}_{j\langle s \rangle, l\langle t \rangle} = \sum_{k\langle v \rangle} A_{j\langle s \rangle \cup k\langle v \rangle} \cdot B_{k\langle v \rangle \cup l\langle t \rangle}
\]

then symmetrizes \( \bar{C} \) to get \( C \), which is low-order, with a total cost of

\[
T_{\text{syctr}}'(\rho, n, s, t, v) = \mu_{\rho} \cdot \binom{n}{s} \binom{n}{t} \binom{n}{v} + \nu_{\rho} \cdot \binom{n}{s} \binom{n}{t} \binom{n}{v}.
\]
The fast algorithm for computing $C$ forms the following key intermediate, where $\omega = s + t + v$,

$$Z_{i\langle\omega\rangle} = \left( \sum_{j\langle s+v\rangle \in \chi(i\langle\omega\rangle)} A_{j\langle s+v\rangle} \right) \cdot \left( \sum_{l\langle t+v\rangle \in \chi(i\langle\omega\rangle)} B_{l\langle t+v\rangle} \right)$$

This intermediate costs $\binom{n}{\omega}$ multiplications to compute. Two other low-order intermediates need to be formed with cost $\binom{n}{\omega - 1}$. The leading order cost is dominated by forming $Z$ and accumulating it to $C$,

$$T'_{\text{syctr}}(\rho, n, s, t, v) = \mu_{\rho} \cdot \binom{n}{\omega} + \nu_{\rho} \cdot \binom{n}{\omega} \cdot \left[ \binom{\omega}{t} + \binom{\omega}{s} + \binom{\omega}{v} \right].$$
The fast algorithm for computing $C$ forms the following intermediates with $\binom{n}{\omega}$ multiplications (where $\omega = s + t + \nu$),

$$Z_{i\langle\omega\rangle} = \left( \sum_{j\langle s+\nu\rangle \in \chi(i\langle\omega\rangle)} A_{j\langle s+\nu\rangle} \right) \cdot \left( \sum_{l\langle t+\nu\rangle \in \chi(i\langle\omega\rangle)} B_{l\langle t+\nu\rangle} \right)$$

$$V_{i\langle\omega-1\rangle} = \left( \sum_{j\langle s+\nu\rangle \in \chi(i\langle\omega-1\rangle)} A_{j\langle s+\nu\rangle} \right) \cdot \left( \sum_{l\langle t+\nu\rangle \in \chi(i\langle\omega-1\rangle) \cup k\langle 1\rangle} B_{l\langle t+\nu\rangle} \right)$$

$$+ \left( \sum_{k_1} \sum_{j\langle s+\nu\rangle \in \chi(i\langle\omega-1\rangle) \cup k\langle 1\rangle} A_{j\langle s+\nu\rangle} \right) \cdot \left( \sum_{l\langle t+\nu\rangle \in \chi(i\langle\omega-1\rangle)} B_{l\langle t+\nu\rangle} \right)$$

$$W_{i\langle\omega-1\rangle} = \left( \sum_{j\langle s+\nu\rangle \in \chi(i\langle\omega-1\rangle)} A_{j\langle s+\nu\rangle} \right) \cdot \left( \sum_{l\langle t+\nu\rangle \in \chi(i\langle\omega-1\rangle)} B_{l\langle t+\nu\rangle} \right)$$

$$C_{i\langle s+t\rangle} = \sum_{k\langle \nu \rangle} Z_{i\langle s+t\rangle \cup k\langle \nu \rangle} - \sum_{k\langle \nu-1 \rangle} V_{i\langle s+t\rangle \cup k\langle \nu-1 \rangle}$$

$$- \sum_{j\langle s+t-1\rangle \in \chi(i\langle s+t\rangle)} \left( \sum_{k\langle \nu \rangle} W_{j\langle s+t-1\rangle \cup k\langle \nu \rangle} \right)$$
Reduction in operation count of fast algorithm with respect to standard

Reduction in operation count for different rings \( \frac{T}{T'} \)

\( \omega \)

\( \frac{T}{T'} \)

- (s+t=\omega) Jordan ring
- (s+t=\omega) complex ring
- (s+t=\omega) real ring

\( \omega \)

\( \frac{T}{T'} \)

- (s+t+v=\omega) Jordan ring
- (s+t+v=\omega) complex ring
- (s+t+v=\omega) real ring

\( \omega \)

\( \frac{T}{T'} \)

\( (s, t, v) \) values for left and right graph tabulated below

<table>
<thead>
<tr>
<th>( \omega )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left graph</td>
<td>(1, 0, 0)</td>
<td>(1, 1, 0)</td>
<td>(2, 1, 0)</td>
<td>(2, 2, 0)</td>
<td>(3, 2, 0)</td>
<td>(3, 3, 0)</td>
</tr>
<tr>
<td>Right graph</td>
<td>(1, 0, 0)</td>
<td>(1, 1, 0)</td>
<td>(1, 1, 1)</td>
<td>(2, 1, 1)</td>
<td>(2, 2, 1)</td>
<td>(2, 2, 2)</td>
</tr>
</tbody>
</table>
We consider communication bandwidth cost on a sequential machine with cache size $M$.

The intermediate formed by the standard algorithm may be computed via matrix multiplication with communication cost,

$$W(n, s, t, v, M) = \Theta \left( \frac{(n)}{s} \frac{(n)}{t} \frac{(n)}{v} \sqrt{M} + \left( \frac{n}{s + v} \right) + \left( \frac{n}{t + v} \right) + \left( \frac{n}{s + t} \right) \right).$$

The cost of symmetrizing the resulting intermediate is low-order or the same.
Communication cost of the fast algorithm

We can lower bound the cost of the fast algorithm using the Hölder-Brascamp-Lieb inequality.

An algorithm that blocks $Z$ symmetrically nearly attains the cost

$$W'(n, s, t, v, M) = O\left(\frac{n}{\sqrt{M}} \cdot \left[\binom{\omega}{t} + \binom{\omega}{s} + \binom{\omega}{v}\right] + \binom{n}{s + v} + \binom{n}{t + v} + \binom{n}{s + t}\right).$$

which is not far from the lower bound and attains it when $s = t = v$. 
Reduction in communication ($W/W'$)

- fast alg comm reduction for $s+t+v = \omega$
- fast alg comm reduction for $s+t = \omega$

Factor of reduction in communication volume

Graph showing the relationship between $\omega$ and the factor of reduction in communication volume.
Theoretical error bounds

We express error bounds in terms of $\gamma_n = \frac{n\epsilon}{1-n\epsilon}$, where $\epsilon$ is the machine precision.

Let $\Psi$ be the standard algorithm and $\Phi$ be the fast algorithm. The error bound for the standard algorithm arises from matrix multiplication

$$|| fl(\Psi(A, B)) - C ||_\infty \leq \gamma_m \cdot ||A||_\infty \cdot ||B||_\infty$$

where $m = \binom{n}{v} \left( \frac{\omega}{v} \right)$.

The following error bound holds for the fast algorithm

$$|| fl(\Phi(A, B)) - C ||_\infty \leq \gamma_m \cdot ||A||_\infty \cdot ||B||_\infty$$

where $m = 3 \binom{n}{v} \left( \frac{\omega}{t} \right) \left( \frac{\omega}{s} \right)$. 
We measure the error in computation of $A \cdot B^T - B \cdot A^T$ where $B = A + \epsilon \cdot \bar{B}$ for $\epsilon = 10^{-9}$ (antisymmetric rank-2 $K$ update).
Now we consider the computation of $\mathbf{A} \cdot \mathbf{B}^T - \mathbf{B} \cdot \mathbf{A}^T$ where 

$$\mathbf{B} = \mathbf{A} \cdot \mathbf{S} + \epsilon \cdot \bar{\mathbf{B}}$$

where $\mathbf{S}$ is a random symmetric matrix.
Typical definition of matrix ring

Matrices form an associative but noncommutative ring with multiplication defined as

\[ C = A \cdot B \equiv \sum_k A_{ik} \cdot B_{kj} = C_{ij} \quad \forall i, j \]

it is not commutative since

\[ D = B \cdot A \equiv \sum_k B_{ik} \cdot A_{kj} = \sum_k A_{kj} \cdot B_{ik} = D_{ij} \quad \forall i, j \]

on the other hand it is associative since

\[ A \cdot (B \cdot C) \equiv \sum_l A_{il} \cdot \left( \sum_k B_{lk} \cdot C_{kj} \right) \quad \forall i, j \]

\[ = \sum_k \left( \sum_l A_{il} \cdot B_{lk} \right) \cdot C_{kj} \quad \forall i, j \equiv (A \cdot B) \cdot C \]
A nonassociative commutative tensor ring

The existence of this ring can be deduced from the previously discussed symmetric contractions
Recall our definition of symmetric contractions of tensors $A, B$, into $C$,

$$C = A \circ B \equiv C_{i\langle s+t \rangle} = \sum_{(j\langle s \rangle, l\langle t \rangle) \in \chi(i\langle s+t \rangle)} \left( \sum_{k\langle v \rangle} A_{j\langle s \rangle \cup k\langle v \rangle} \cdot B_{k\langle v \rangle \cup l\langle t \rangle} \right),$$

the fast algorithm requires only that the operator $\cdot$ is commutative, which $\circ$ is, therefore the algorithm can be nested for symmetric $A, B$

$$C = A \circ B \equiv C_{i\langle s+t \rangle} = \sum_{(j\langle s \rangle, l\langle t \rangle) \in \chi(i\langle s+t \rangle)} \left( \sum_{k\langle v \rangle} A_{j\langle s \rangle \cup k\langle v \rangle} \circ B_{k\langle v \rangle \cup l\langle t \rangle} \right),$$

now, when $s + v + t = 1$, commutativity still holds, and the tensors (two vectors and a scalar) are all nonsymmetric, therefore we can nest nonsymmetric contractions by contracting "one index at a time".
A nonassociative commutative tensor ring

We can also explicitly define a nonassociative commutative tensor ring for tensors with dimensions $n$, and variable rank $r$, over a 'labeled' tensor set,

$$S^n = \{(r, Q, T) : r \in \{0, 1, \ldots\}, Q \subset \{a, b, \ldots\}, |Q| = r, \text{ and } T \text{ any rank } r \text{ tensor}\}$$

we now define addition of $A = (r_A, Q_A, V), B = (r_B, Q_B, W) \in S^n$ as

$$C = A \oplus B \equiv (r_C, Q_C, Z) \text{ where}$$

$$r_C = |Q_A \cup Q_B|$$

$$Q_C = Q_A \cup Q_B$$

$$Z_{Q_C} = V_{Q_A} + W_{Q_B} \quad \forall Q_C \in \{1, \ldots n\}^{r_C}, Q_A \in \{1, \ldots n\}^{r_A}, Q_B \in \{1, \ldots n\}^{r_B}$$

For example if $A = (2, \{i, j\}, V), B = (3, \{i, j, k\}, W) \in S^n$, then

$$C = A \oplus B = (3, \{i, j, k\}, Z) \text{ with}$$

$$Z_{ijk} = V_{ij} + W_{ijk}.$$
A nonassociative commutative tensor ring

We then define multiplication on $S^n$ according to the labels (in Einstein notation), of for $A = (r_A, Q_A, V), B = (r_B, Q_B, W) \in S^n$ as

$$C = A \odot B \equiv (r_C, Q_C, Z)$$

where

$$r_C = |Q_A \cup Q_B| - |Q_A \cap Q_B|$$

$$Q_C = (Q_A \cup Q_B) \setminus (Q_A \cap Q_B)$$

$$Z_{Q_C} = \sum_{Q_A \cap Q_B} V_{Q_A} \cdot W_{Q_B}$$

For example if $A = (2, \{i, k\}, V), B = (2, \{k, j\}, W) \in S^n$, then $C = A \odot B = (2, \{i, j\}, Z)$ with

$$Z_{ij} = \sum_k V_{ik} \cdot W_{kj}.$$  

Commutativity is evident but associativity is now lost e.g.,

$$\left( \sum_k A_{ik} \right) \cdot \left( \sum_k B_{lk} \cdot C_{kj} \right) \neq \left( \sum_k A_{ik} \cdot B_{lk} \right) \cdot \left( \sum_k C_{kj} \right)$$
Relation to Cyclops Tensor Framework

Our distributed memory tensor contraction library, ”Cyclops Tensor Framework” (CTF) behaves according to the definition of $\oplus$ and $\otimes$ on $S^n$. We employ Einstein notation to write C++ code for contractions like

$$Z[^{ij}] = V[^{ik}] * W[^{kj}];$$

where the sum over $k$ is implicit. Symmetrization is also done implicitly based on the symmetry of the output, so the CCSDT contraction from the first slide is implemented in Aquarius as

$$Z[^{abcijk}] = W[^{amei}] * Z[^{ebcmjk}];$$

where sums over $e$ and $m$ are implicit as well as antisymmetrizations $P(a, bc), P(i, jk)$ if $Z$ is defined to have two antisymmetric index groups

```cpp
int lens[6] = {n,n,n,n,n,n};
CTF_Tensor Z = CTF_Tensor(6,lens,syms,mpicomm);
```
Contraction $s$, $t$, $\nu$ values for CCSD

Table where $(s, t, \nu)$ are for the fast symmetric algorithm and $[s', t', \nu']$ are leftover indices (sometimes with unfolding):

<table>
<thead>
<tr>
<th>Operation</th>
<th>PPL:</th>
<th>Ring:</th>
<th>Index:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA*AA</td>
<td>[2,2,2]</td>
<td>(1,0,1), (1,0,1), [0,2,0]</td>
<td>(1,0,1)</td>
</tr>
<tr>
<td>AB*AB</td>
<td>[2,2,2]</td>
<td>[2,2,2]</td>
<td>[2,2,2]</td>
</tr>
<tr>
<td>AA*A,A</td>
<td>[2,2,2]</td>
<td>(1,0,1), (1,0,1), [0,2,0]</td>
<td>(1,0,1)</td>
</tr>
<tr>
<td>AA*A,B</td>
<td>[2,2,2]</td>
<td>(1,0,1), (1,0,1), [0,2,0]</td>
<td>(1,0,1)</td>
</tr>
<tr>
<td>AB*A,A</td>
<td>[2,2,2]</td>
<td>(1,0,1), (1,0,1), [0,2,0]</td>
<td>(1,0,1)</td>
</tr>
<tr>
<td>AB*A,B</td>
<td>[2,2,2]</td>
<td>[2,2,2]</td>
<td>[2,2,2]</td>
</tr>
<tr>
<td>AB*B,A</td>
<td>[2,2,2]</td>
<td>[2,2,2]</td>
<td>[2,2,2]</td>
</tr>
<tr>
<td>AB*B,B</td>
<td>[2,2,2]</td>
<td>[2,2,2]</td>
<td>[2,2,2]</td>
</tr>
<tr>
<td>AA*AA</td>
<td>(1,0,1), (1,0,1), [0,2,0]</td>
<td>(1,0,1)</td>
<td></td>
</tr>
<tr>
<td>AA*AB</td>
<td>(1,0,1), (1,0,1), [0,2,0]</td>
<td>(1,0,1)</td>
<td></td>
</tr>
<tr>
<td>AB*AA</td>
<td>(0,1,1), (0,1,1), [2,0,0]</td>
<td>(0,1,1)</td>
<td></td>
</tr>
<tr>
<td>AB*AB</td>
<td>[2,2,2]</td>
<td>[2,2,2]</td>
<td>[2,2,2]</td>
</tr>
<tr>
<td>AB*BB</td>
<td>(0,1,1), (0,1,1), [2,0,0]</td>
<td>(0,1,1)</td>
<td></td>
</tr>
</tbody>
</table>
Contraction \( s, t, \nu \) values for CCSD(T)

Table where \((s, t, \nu)\) are for the fast symmetric algorithm and \([s', t', \nu']\) are leftover indices (sometimes with unfolding):

- \(AA*AA: (1,2,0),(2,1,0),[0,0,1]\)
- \(AA*AB: (1,1,0),[2,2,1]\)
- \(AB*AA: (1,1,0),[2,2,1]\)
- \(AB*AB: (1,1,0),(1,1,0),[1,1,1]\)
- \(AB*BA: (1,1,0),(1,1,0),[1,1,1]\)
- \(AB*BB: (1,1,0),[2,2,1]\)
Table where \((s, t, \nu)\) are for the fast symmetric algorithm and \([s', t', \nu']\) are leftover indices (sometimes with unfolding):

<table>
<thead>
<tr>
<th>Tensor Configuration</th>
<th>Fast Symmetric Algorithm</th>
<th>Leftover Indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA*AA PPL</td>
<td>(1,0,2), [3,2,0]</td>
<td></td>
</tr>
<tr>
<td>AAB*AA PPL</td>
<td>[4,2,2]</td>
<td></td>
</tr>
<tr>
<td>AAB*AB PPL</td>
<td>(1,0,1), [3,2,1]</td>
<td></td>
</tr>
<tr>
<td>AAA*A,A RING</td>
<td>(2,0,1), (2,0,1), [0,2,0]</td>
<td></td>
</tr>
<tr>
<td>AAA*A,B RING</td>
<td>(2,0,1), (2,0,1), [0,2,0]</td>
<td></td>
</tr>
<tr>
<td>AAB*A,A RING</td>
<td>(1,0,1), (1,0,1), [2,2,0]</td>
<td></td>
</tr>
<tr>
<td>AAB*A,B RING</td>
<td>(1,0,1), (1,0,1), [2,2,0]</td>
<td></td>
</tr>
<tr>
<td>AAB*B,A RING</td>
<td>(2,1,0), (2,1,0), [0,0,2]</td>
<td></td>
</tr>
<tr>
<td>AAB*B,B RING</td>
<td>[4,2,2]</td>
<td></td>
</tr>
</tbody>
</table>
Contraction $s, t, \nu$ values for CCSDT(Q)

Table where $(s, t, \nu)$ are for the fast symmetric algorithm and $[s', t', \nu']$ are leftover indices (sometimes with unfolding):

<table>
<thead>
<tr>
<th>Tensor Product</th>
<th>$(s, t, \nu)$</th>
<th>$[s', t', \nu']$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA*AA:</td>
<td>$(2,2,0), (3,1,0), [0,0,1]$</td>
<td></td>
</tr>
<tr>
<td>AAA*AB:</td>
<td>$(2,1,0), [3,2,1]$</td>
<td></td>
</tr>
<tr>
<td>AAB*AA:</td>
<td>$(1,2,0), (2,1,0), [2,0,1]$</td>
<td></td>
</tr>
<tr>
<td>AAB*AB:</td>
<td>$(1,1,0), (1,1,0), (1,1,0), [2,0,1]$</td>
<td></td>
</tr>
<tr>
<td>AAB*BA:</td>
<td>$(2,1,0), (2,1,0), [1,1,1]$</td>
<td></td>
</tr>
<tr>
<td>AAB*BB:</td>
<td>$(1,1,0), [4,2,1]$</td>
<td></td>
</tr>
</tbody>
</table>
Contraction $s, t, \nu$ values for CCSDTQ

Table where $(s, t, \nu)$ are for the fast symmetric algorithm and $[s', t', \nu']$ are leftover indices (sometimes with unfolding):

AAAA*AA PPL: $(2,0,2), [4,2,0]$
AAAB*AA PPL: $(1,0,2), [5,2,0]$
AAAB*AB PPL: $(2,0,1), [4,2,1]$
AABB*AA PPL: $[6,2,2]$
AABB*AB PPL: $(1,0,1), (1,0,1), [4,2,0]$
AABB*BB PPL: $[6,2,2]$
AAAA*A,A RING: $(3,0,1), (3,0,1), [0,2,0]$
AAAA*A,B RING: $(3,0,1), (3,0,1), [0,2,0]$
AAAB*A,A RING: $(2,0,1), (2,0,1), [2,2,0]$
AAAB*A,B RING: $(2,0,1), (2,0,1), [2,2,0]$
AAAB*B,A RING: $(3,1,0), (3,1,0), [0,0,2]$
AAAB*B,B RING: $[6,2,2]$
AABB*A,A RING: $(1,0,1), (1,0,1), [4,2,0]$

...
Contraction $s, t, v$ values for CCSDTQ contd

Table where $(s, t, v)$ are for the fast symmetric algorithm and $[s', t', v']$ are leftover indices (sometimes with unfolding):

<table>
<thead>
<tr>
<th>Contracted Indices</th>
<th>Ring</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>AABB*A, B</td>
<td>(2, 1, 0), (2, 1, 0), [2, 0, 2]</td>
<td></td>
</tr>
<tr>
<td>AABB*B, A</td>
<td>(2, 1, 0), (2, 1, 0), [2, 0, 2]</td>
<td></td>
</tr>
<tr>
<td>AABB*B, B</td>
<td>(1, 0, 1), (1, 0, 1), [4, 2, 0]</td>
<td></td>
</tr>
<tr>
<td>AAA*A, A</td>
<td>(2, 2, 0), (2, 2, 0), [0, 0, 2]</td>
<td></td>
</tr>
<tr>
<td>AAA*A, AB</td>
<td>(2, 0, 1), (2, 0, 1), [0, 4, 0]</td>
<td></td>
</tr>
<tr>
<td>AAA*A, BB</td>
<td>(2, 0, 1), (2, 0, 1), [0, 4, 0]</td>
<td></td>
</tr>
<tr>
<td>AAB*A, A</td>
<td>(1, 2, 0), (1, 2, 0), [2, 0, 2]</td>
<td></td>
</tr>
<tr>
<td>AAB*A, AB</td>
<td>(1, 0, 1), (1, 0, 1), [2, 4, 0]</td>
<td></td>
</tr>
<tr>
<td>AAB*A, BB</td>
<td>(1, 2, 0), (1, 2, 0), [2, 0, 2]</td>
<td></td>
</tr>
<tr>
<td>AAB*B, A</td>
<td>(2, 2, 0), (2, 2, 0), [0, 0, 2]</td>
<td></td>
</tr>
<tr>
<td>AAB*B, AB</td>
<td>(2, 1, 0), (2, 1, 0), [0, 2, 2]</td>
<td></td>
</tr>
<tr>
<td>AAB*B, BB</td>
<td>[4, 4, 2]</td>
<td></td>
</tr>
</tbody>
</table>
Summary of results

The following table lists the leading order number of multiplications $F$ required by the standard algorithm and $F'$ by the fast algorithm for various cases of symmetric tensor contractions,

<table>
<thead>
<tr>
<th>$\omega$</th>
<th>$s$</th>
<th>$t$</th>
<th>$v$</th>
<th>$F$</th>
<th>$F'$</th>
<th>applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$n^2$</td>
<td>$n^2/2$</td>
<td>syr2, syr2k, her2, her2k</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$n^2$</td>
<td>$n^2/2$</td>
<td>symv, symm, hemv, hemm</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$n^3$</td>
<td>$n^3/6$</td>
<td>Jordan and Lie matrix rings</td>
</tr>
<tr>
<td>s+t+v</td>
<td>s</td>
<td>t</td>
<td>v</td>
<td>$\binom{n}{s}$ $\binom{n}{t}$ $\binom{n}{v}$</td>
<td>$\binom{n}{\omega}$</td>
<td>any symmetric tensor contraction</td>
</tr>
</tbody>
</table>

High-level conclusions:

- The fast symmetric contraction algorithms provide interesting potential arithmetic cost improvements for complex BLAS routines and partially symmetric tensor contractions.
- However, the new algorithms require more communication per flop, incur more numerical error, and usually unable to exploit fused-multiply-add units or blocked matrix multiplication primitives.
Communication cost of the fast algorithm

We can lower bound the cost of the fast algorithm using the Hölder-Brascamp-Lieb inequality.

\[
\hat{W}(n, s, t, v, M) = \Omega\left( \min_{m_A, m_B, m_C > 0, m_A \cdot (\omega) \cdot m_B + (\omega) \cdot m_C \leq M} \frac{\binom{n}{\omega}}{(m_A \cdot m_B \cdot m_C)^{1/2}} \right)
\cdot \left( (\omega) \cdot m_A + (\omega) \cdot m_B + (\omega) \cdot m_C \right).
\]

An algorithm that blocks \(Z\) symmetrically nearly attains the cost

\[
W'(n, s, t, v, M) = O\left( \frac{\binom{n}{\omega}}{\sqrt{M}} \cdot \left[ (\omega) + (\omega) + (\omega) \right] \right.
\]
\[
+ \binom{n}{s + v} + \binom{n}{t + v} + \binom{n}{s + t} \right).
\]

which is not far from the lower bound and attains it when \(s = t = v\).