Algorithms for contraction of tensors over a commutative ring

Edgar Solomonik[1], Devin Matthews[3], and James Demmel[1,2]

[1] Department of Electric Engineering and Computer Science, UC Berkeley
[2] Department of Mathematics, UC Berkeley
[3] Department of Chemistry, UT Austin

ETH Zürich

June 18, 2014

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

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

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

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

Consider non-associative commutative ring ϱ with multiplication cost μ_ϱ and with addition cost ν_ϱ

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

Outline

Symmetric matrix times vector

- Let $\mathbf{b}$ be a vector of length n with elements in ϱ
- Let $\mathbf{A}$ be an n-by-n symmetric matrix with elements in ϱ

$$A_{ij} = A_{ji}$$

- We multiply matrix $\mathbf{A}$ by $\mathbf{b}$,

$$\mathbf{c} = \mathbf{A} \cdot \mathbf{b}$$

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

this corresponds to BLAS routine `symv` and has cost (ignoring low-order terms here and later)

$$T_{\text{symv}}(\varrho, n) = \mu_\varrho \cdot n^2 + \nu_\varrho \cdot n^2$$

where μ_ϱ is the cost of multiplication and ν_ϱ of addition

Fast symmetric matrix times vector

We can perform `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

$$T'_{\mathrm{symv}}(\varrho, n) = \mu_\varrho \cdot \frac{1}{2} n^2 + \nu_\varrho \cdot \frac{5}{2} n^2$$

- This formulation is cheaper when $\mu_\varrho > 3\nu_\varrho$
- Form `symm` the formulation is cheaper when $\mu_\varrho > \nu_\varrho$

Symmetric rank-2 update

Consider a rank-2 outer product of vectors **a** and **b** of length n into symmetric matrix **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 `syr2`
- The routine may be computed from the nonsymmetric intermediate $K_{ij} = a_i \cdot b_j$ with the cost

$$T_{\mathrm{syr2}}(\varrho, n) = \mu_\varrho \cdot n^2 + \nu_\varrho \cdot n^2.$$

Fast symmetric rank-2 update

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

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

- $T'_{\mathrm{syr2}}(\varrho, n) < T_{\mathrm{syr2}}(\varrho, n)$ when $\mu_\varrho > 3\nu_\varrho$
- $T'_{\mathrm{syr2K}}(\varrho, n, K) < T_{\mathrm{syr2K}}(\varrho, n, K)$ when $\mu_\varrho > \nu_\varrho$

Symmetric-matrix by symmetric-matrix multiplication

Given symmetric matrices $\mathbf{A}, \mathbf{B}$ of dimension n on non-associative commutative ring ϱ, we seek to compute the *anticommutator* of $\mathbf{A}$ and $\mathbf{B}$

$$\mathbf{C} = \mathbf{A} \circ \mathbf{B} \equiv \mathbf{A} \cdot \mathbf{B} + \mathbf{B} \cdot \mathbf{A}$$

$$C_{ij} = \sum_{k=1}^{n} \left(A_{ik} \cdot B_{jk} + A_{jk} \cdot B_{ik} \right).$$

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

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

Note that $\circ$ defines a non-associative commutative ring (the Jordan ring) over the set of symmetric matrices.

Fast symmetric-matrix by symmetric-matrix multiplication

We can combine the ideas from the fast routines for `symv` and `syrk` by forming a fully-symmetric intermediate $\mathcal{Z}$,

$$Z_{ijk} = (A_{ij} + A_{ik} + A_{jk}) \cdot (B_{ij} + B_{ik} + B_{jk}) \qquad \bar{Z}_{ij} = \sum_k Z_{ijk}$$

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

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

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

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

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

which is faster than T_{syrmm} 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)

$$k\langle v\rangle = (k_1, k_2, \dots k_v).$$

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,

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

Tensor index notation

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

$$k\langle v\rangle = (k_1, k_2, \dots k_v).$$

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

$$\chi^d(k\langle d+f\rangle) = \{a | \forall (a,b) \in \chi_f^d(k\langle d+f\rangle)\}.$$

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,

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

Accordingly, we denote all possible ordered subsets as

$$\chi^d(k\langle d+f\rangle) = \{a | \forall (a,b) \in \chi_f^d(k\langle d+f\rangle)\}.$$

We omit the subscript and superscript on χ 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,

$$\mathcal{C} = \mathcal{A} \circ \mathcal{B}$$

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

where $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ are all fully symmetric with dimensions n.

Fully symmetric tensor contractions

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

$$\mathcal{C} = \mathcal{A} \circ \mathcal{B}$$

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

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

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

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

$$T'_{\text{syctr}}(\varrho, n, s, t, v) = \mu_\varrho \cdot \binom{n}{s}\binom{n}{t}\binom{n}{v} + \nu_\varrho \cdot \binom{n}{s}\binom{n}{t}\binom{n}{v}.$$

Fast fully-symmetric contraction algorithm

The fast algorithm for computing $\mathcal{C}$ forms the following key intermediate, where $\omega = s + t + v$,

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

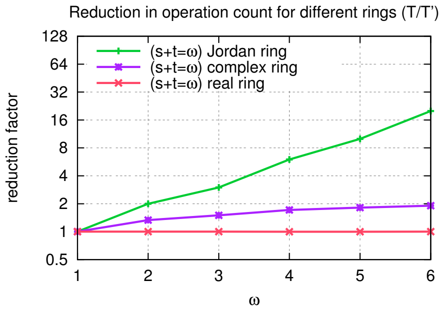
This intermediate costs $\binom{n}{\omega}$ multiplications to compute. Two other low-order intermediates need to be formed with cost $\binom{n}{\omega-1}$. The leading order cost is dominated by forming $\mathcal{Z}$ and accumulating it to $\mathcal{C}$,

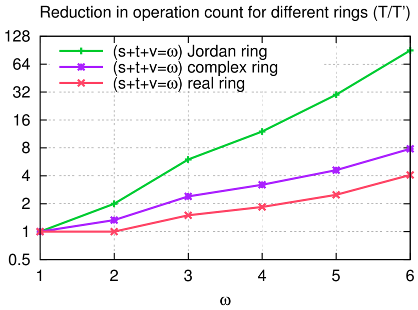
$$T'_{\mathrm{syctr}}(\varrho, n, s, t, v) = \mu_\varrho \cdot \binom{n}{\omega} + \nu_\varrho \cdot \binom{n}{\omega} \cdot \left[\binom{\omega}{t} + \binom{\omega}{s} + \binom{\omega}{v} \right].$$

The fast algorithm for computing $\mathcal{C}$ forms the following intermediates with $\binom{n}{\omega}$ multiplications (where $\omega = s + t + v$),

$$\begin{aligned}
Z_{i\langle\omega\rangle} =& \left(\sum_{j\langle s+v\rangle \in \chi(i\langle\omega\rangle)} A_{j\langle s+v\rangle} \right) \cdot \left(\sum_{l\langle t+v\rangle \in \chi(i\langle\omega\rangle)} B_{l\langle t+v\rangle} \right) \\
V_{i\langle\omega-1\rangle} =& \left(\sum_{j\langle s+v\rangle \in \chi(i\langle\omega-1\rangle)} A_{j\langle s+v\rangle} \right) \cdot \left(\sum_{k_1} \sum_{l\langle t+v\rangle \in \chi(i\langle\omega-1\rangle \cup k\langle 1\rangle)} B_{l\langle t+v\rangle} \right) \\
&+ \left(\sum_{k_1} \sum_{j\langle s+v\rangle \in \chi(i\langle\omega-1\rangle \cup k\langle 1\rangle)} A_{j\langle s+v\rangle} \right) \cdot \left(\sum_{l\langle t+v\rangle \in \chi(i\langle\omega-1\rangle)} B_{l\langle t+v\rangle} \right) \\
W_{i\langle\omega-1\rangle} =& \left(\sum_{j\langle s+v\rangle \in \chi(i\langle\omega-1\rangle)} A_{j\langle s+v\rangle} \right) \cdot \left(\sum_{l\langle t+v\rangle \in \chi(i\langle\omega-1\rangle)} B_{l\langle t+v\rangle} \right) \\
C_{i\langle s+t\rangle} =& \sum_{k\langle v\rangle} Z_{i\langle s+t\rangle \cup k\langle v\rangle} - \sum_{k\langle v-1\rangle} V_{i\langle s+t\rangle \cup k\langle v-1\rangle} \\
&- \sum_{j\langle s+t-1\rangle \in \chi(i\langle s+t\rangle)} \left(\sum_{k\langle v\rangle} W_{j\langle s+t-1\rangle \cup k\langle v\rangle} \right)
\end{aligned}$$

Reduction in operation count of fast algorithm with respect to standard





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

ω	1	2	3	4	5	6
Left graph	$(1,0,0)$	$(1,1,0)$	$(2,1,0)$	$(2,2,0)$	$(3,2,0)$	$(3,3,0)$
Right graph	$(1,0,0)$	$(1,1,0)$	$(1,1,1)$	$(2,1,1)$	$(2,2,1)$	$(2,2,2)$

Communication cost of the standard algorithm

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,

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

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

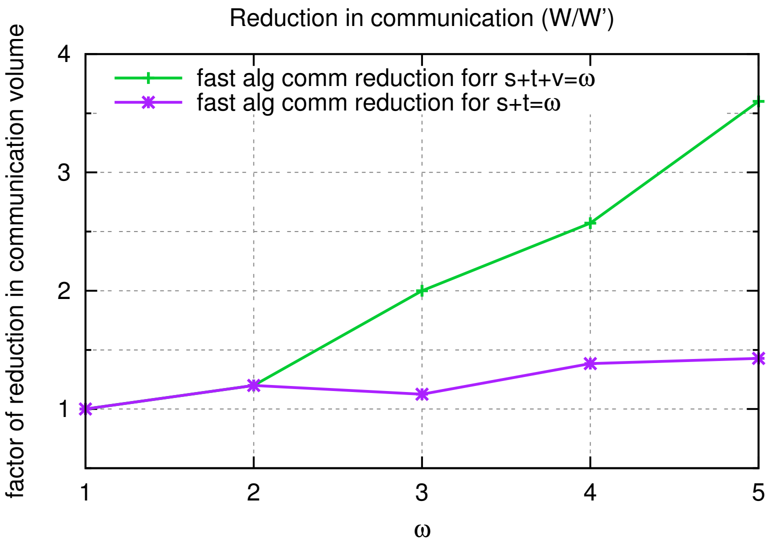
Communication cost of the fast algorithm

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

An algorithm that blocks $\mathcal{Z}$ symmetrically nearly attains the cost

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

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


Reduction in communication (W/W')
fast alg comm reduction forr s+t+v=ω
fast alg comm reduction for s+t=ω
factor of reduction in communication volume
ω
1
2
3
4
5
1
2
3
4

Theoretical error bounds

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

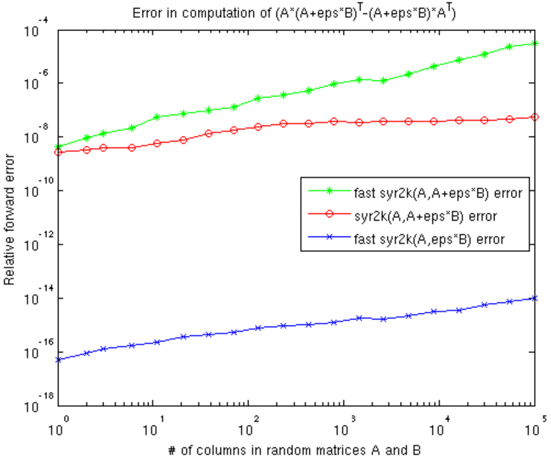
Let Ψ be the standard algorithm and Φ be the fast algorithm. The error bound for the standard algorithm arises from matrix multiplication

$$||fl\,(\Psi(\mathcal{A},\mathcal{B})) - \mathcal{C}||_\infty \le \gamma_m \cdot ||\mathcal{A}||_\infty \cdot ||\mathcal{B}||_\infty \;\text{ where }\; m = \binom{n}{v}\binom{\omega}{v}.$$

The following error bound holds for the fast algorithm

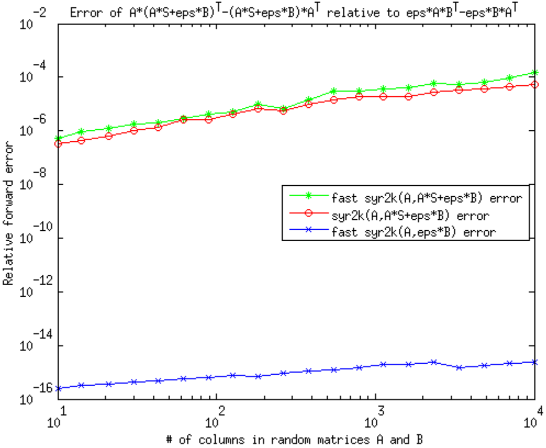
$$||fl\,(\Phi(\mathcal{A},\mathcal{B})) - \mathcal{C}||_\infty \le \gamma_m \cdot ||\mathcal{A}||_\infty \cdot ||\mathcal{B}||_\infty \;\text{ where }\; m = 3\binom{n}{v}\binom{\omega}{t}\binom{\omega}{s}.$$

Numerical test I



We measure the error in computation of $\mathbf{A} \cdot \mathbf{B}^T - \mathbf{B} \cdot \mathbf{A}^T$ where $\mathbf{B} = \mathbf{A} + \epsilon \cdot \bar{\mathbf{B}}$ for $\epsilon = 10^{-9}$ (antisymmetric rank-$2K$ update).

Numerical test II



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

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

it is not commutative since

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

on the other hand it is associative since

$$\begin{aligned} \mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C}) &\equiv \sum_l A_{il} \cdot \left(\sum_k B_{lk} \cdot C_{kj} \right) \quad \forall i, j \\ &= \sum_k \left(\sum_l A_{il} \cdot B_{lk} \right) \cdot C_{kj} \quad \forall i, j \equiv (\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C} \end{aligned}$$

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 $\mathcal{A}$, $\mathcal{B}$, into $\mathcal{C}$,

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

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

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

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,

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

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

$$\begin{aligned} C &= A \oplus B \equiv (r_C, Q_C, \mathcal{Z}) \text{ where} \\ r_C &= |Q_A \cup Q_B| \\ Q_C &= Q_A \cup Q_B \\ Z_{Q_C} &= V_{Q_A} + W_{Q_B} \quad \forall Q_C \in \{1, \ldots n\}^{r_C}, Q_A \in \{1, \ldots n\}^{r_A}, Q_B \in \{1, \ldots n\}^{r_B} \end{aligned}$$

For example if $A = (2, \{i, j\}, \mathcal{V}), B = (3, \{i, j, k\}, \mathcal{W}) \in S^n$, then $C = A \oplus B = (3, \{i, j, k\}, \mathcal{Z})$ 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 $A = (r_A, Q_A, \mathcal{V}), B = (r_B, Q_B, \mathcal{W}) \in S^n$ as

$$\begin{aligned} C = A \odot B &\equiv (r_C, Q_C, \mathcal{Z}) \text{ where} \\ r_C &= |Q_A \cup Q_B| - |Q_A \cap Q_B| \\ Q_C &= (Q_A \cup Q_B) \setminus (Q_A \cap Q_B) \\ Z_{Q_C} &= \sum_{Q_A \cap Q_B} V_{Q_A} \cdot W_{Q_B} \end{aligned}$$

For example if $A = (2, \{i, k\}, \mathbf{V}), B = (2, \{k, j\}, \mathbf{W}) \in S^n$, then $C = A \odot B = (2, \{i, j\}, \mathbf{Z})$ 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 antisymmetirzations $P(a, bc), P(i, jk)$ if Z is defined to have two antisymmetric index groups

```
int lens[6] = {n,n,n,n,n,n};
int syms[6] = {AS,AS,NS,AS,AS,NS};
CTF_Tensor Z = CTF_Tensor(6,lens,syms,mpicomm);
```

Contraction s, t, v values for CCSD

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

```
AA*AA PPL:     [2,2,2]
AB*AB PPL:     [2,2,2]
AA*A,A RING:   (1,0,1),(1,0,1),[0,2,0]
AA*A,B RING:   (1,0,1),(1,0,1),[0,2,0]
AB*A,A RING:   [2,2,2]
AB*A,B RING:   [2,2,2]
AB*B,A RING:   [2,2,2]
AB*B,B RING:   [2,2,2]
AA*AA INT:     (1,0,1),(1,0,1),[0,2,0]
AA*AB INT:     (1,0,1),(1,0,1),[0,2,0]
AB*AA INT:     (0,1,1),(0,1,1),[2,0,0]
AB*AB INT:     [2,2,2]
AB*BB INT:     (0,1,1),(0,1,1),[2,0,0]
```

Contraction s, t, v values for CCSD(T)

Table where (s, t, v) are for the fast symmetric algorithm and $[s', t', v']$ 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]
```

Contraction s, t, v values for CCSDT

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

```
AAA*AA PPL:    (1,0,2),[3,2,0]
AAB*AA PPL:    [4,2,2]
AAB*AB PPL:    (1,0,1),[3,2,1]
AAA*A,A RING:  (2,0,1),(2,0,1),[0,2,0]
AAA*A,B RING:  (2,0,1),(2,0,1),[0,2,0]
AAB*A,A RING:  (1,0,1),(1,0,1),[2,2,0]
AAB*A,B RING:  (1,0,1),(1,0,1),[2,2,0]
AAB*B,A RING:  (2,1,0),(2,1,0),[0,0,2]
AAB*B,B RING:  [4,2,2]
```

Contraction s, t, v values for CCSDT(Q)

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

```
AAA*AA:      (2,2,0),(3,1,0),[0,0,1]
AAA*AB:      (2,1,0),[3,2,1]
AAB*AA:      (1,2,0),(2,1,0),[2,0,1]
AAB*AB:      (1,1,0),(1,1,0),(1,1,0),[2,0,1]
AAB*BA:      (2,1,0),(2,1,0),[1,1,1]
AAB*BB:      (1,1,0),[4,2,1]
```

Contraction s, t, v values for CCSDTQ

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

```
AABB*A,B RING: (2,1,0),(2,1,0),[2,0,2]
AABB*B,A RING: (2,1,0),(2,1,0),[2,0,2]
AABB*B,B RING: (1,0,1),(1,0,1),[4,2,0]
AAA*A,AA RING: (2,2,0),(2,2,0),[0,0,2]
AAA*A,AB RING: (2,0,1),(2,0,1),[0,4,0]
AAA*A,BB RING: (2,0,1),(2,0,1),[0,4,0]
AAB*A,AA RING: (1,2,0),(1,2,0),[2,0,2]
AAB*A,AB RING: (1,0,1),(1,0,1),[2,4,0]
AAB*A,BB RING: (1,2,0),(1,2,0),[2,0,2]
AAB*B,AA RING: (2,2,0),(2,2,0),[0,0,2]
AAB*B,AB RING: (2,1,0),(2,1,0),[0,2,2]
AAB*B,BB RING: [4,4,2]
```

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,

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

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.

Backup slides

Communication cost of the fast algorithm

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

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

An algorithm that blocks $\mathcal{Z}$ symmetrically nearly attains the cost

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

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