Efficient Tensor Contraction Algorithms for Coupled Cluster

Edgar Solomonik

Department of Computer Science, ETH Zurich, Switzerland

20.6.2015

QESC 2015
Kobe, Japan
Outline

1 Cyclops Tensor Framework
   - Motivation
   - Interface
   - Coupled Cluster with CTF
   - Internal mechanism
   - Performance

2 Symmetry Preserving Algorithm
   - Instances in matrix computations
   - General symmetric contractions
   - Application to coupled-cluster

3 Conclusion
The problem

We want portable infrastructure and scalable algorithms for tensor-based electronic structure methods
We want portable infrastructure and scalable algorithms for tensor-based electronic structure methods

- the problem is not ‘ill-posed’, small perturbations to the equations of a method do not fundamentally change the computation
The problem

We want portable infrastructure and scalable algorithms for tensor-based electronic structure methods

- the problem is not ‘ill-posed’, small perturbations to the equations of a method do not fundamentally change the computation
- a ‘stable’ solution must provide a high-level abstraction that permits rapid manipulation of the algebra
The problem

We want portable infrastructure and scalable algorithms for tensor-based electronic structure methods

- the problem is not ‘ill-posed’, small perturbations to the equations of a method do not fundamentally change the computation
- a ‘stable’ solution must provide a high-level abstraction that permits rapid manipulation of the algebra
- scalability must be achieved both for intranode (shared memory) and internode (distributed memory) parallelism
Our solution

Cyclops (cyclic operations) Tensor Framework (CTF)
Our solution

Cyclops (cyclic operations) Tensor Framework (CTF)

- provides abstractions for symmetric tensors and symmetrized contractions
Our solution

Cyclops (cyclic operations) Tensor Framework (CTF)

- provides abstractions for symmetric tensors and symmetrized contractions
- selects best parallelization for each contraction based on runtime performance models
Our solution

Cyclops (cyclic operations) Tensor Framework (CTF)

- provides abstractions for symmetric tensors and symmetrized contractions
- selects best parallelization for each contraction based on runtime performance models
- leverages only portable building blocks: C++, MPI, BLAS, and OpenMP
Our solution

Cyclops (cyclic operations) Tensor Framework (CTF)

- provides abstractions for symmetric tensors and symmetrized contractions
- selects best parallelization for each contraction based on runtime performance models
- leverages only portable building blocks: C++, MPI, BLAS, and OpenMP
- optimized for distributed networks, shared memory, and accelerators
Our solution

Cyclops (\textit{cyclic operations}) Tensor Framework (CTF)

- provides abstractions for symmetric tensors and symmetrized contractions
- selects best parallelization for each contraction based on runtime performance models
- leverages only portable building blocks: C++, MPI, BLAS, and OpenMP
- optimized for distributed networks, shared memory, and accelerators
- open source, BSD license, https://github.com/solomonik/ctf
Distributed-memory tensor objects

CTF is orchestrated by bulk synchronous operations on a set of processors

CTF::World dw(MPI_COMM_WORLD);
Distributed-memory tensor objects

CTF is orchestrated by bulk synchronous operations on a set of processors

```cpp
CTF::World dw(MPI_COMM_WORLD);
```

CTF tensors are defined to be distributed over such worlds

```cpp
CTF::Tensor<T> T(4, {m, m, n, n}, {AS, NS, AS, NS}, dw);
```
Distributed-memory tensor objects

CTF is orchestrated by bulk synchronous operations on a set of processors

```cpp
CTF::World dw(MPI_COMM_WORLD);
```

CTF tensors are defined to be distributed over such worlds

```cpp
CTF::Tensor<> T(4, {m, m, n, n}, {AS, NS, AS, NS}, dw);
```

- an ‘AS’ dimension is antisymmetric with the next (also ‘SY’ and ‘SH’).
Distributed-memory tensor objects

CTF is orchestrated by bulk synchronous operations on a set of processors

```cpp
CTF::World dw(MPI_COMM_WORLD);
```

CTF tensors are defined to be distributed over such worlds

```cpp
CTF::Tensor<> T(4,{m,m,n,n},{AS,NS,AS,NS},dw);
```

- an ‘AS’ dimension is antisymmetric with the next (also ‘SY’ and ‘SH’)
- tensors are templated by the element type (double by default)
Distributed-memory tensor objects

CTF is orchestrated by bulk synchronous operations on a set of processors

```cpp
CTF::World dw(MPI_COMM_WORLD);
```

CTF tensors are defined to be distributed over such worlds

```cpp
CTF::Tensor<> T(4,{m,m,n,n},{AS,NS,AS,NS},dw);
```

- an ‘AS’ dimension is antisymmetric with the next (also ‘SY’ and ‘SH’)
- tensors are templated by the element type (double by default)
- custom algebraic structures (set, group, monoid, semiring, ring) may be defined by the user
Tensor algebra interface (credit: Devin Matthews)

CTF can express a tensor contraction like

\[ Z_{ij}^{ab} = \frac{1}{2} \cdot W_{ij}^{ab} + 2 \cdot P(a, b) \sum_k F_k^a \cdot T_{ij}^{kb} \]

where \( P(a, b) \) implies antisymmetrization of index pair \( ab \), as

\[
\begin{align*}
Z["abij"] &= 0.5*W["abij"]; \\
Z["abij"] &=+ 2.0*F["ak"]*T["kbij"]; \\
\end{align*}
\]
Tensor algebra interface (credit: Devin Matthews)

CTF can express a tensor contraction like

\[
Z_{ij}^{ab} = \frac{1}{2} \cdot W_{ij}^{ab} + 2 \cdot P(a, b) \sum_k F_{a}^{k} \cdot T_{ij}^{kb}
\]

where \(P(a, b)\) implies antisymmetrization of index pair \(ab\), as

\[
Z["abij"] = 0.5*W["abij"];
Z["abij"] += 2.0*F["ak"]*T["kbij"];
\]

- for loops and summations implicit in syntax
Tensor algebra interface (credit: Devin Matthews)

CTF can express a tensor contraction like

\[
Z_{ij}^{ab} = \frac{1}{2} \cdot W_{ij}^{ab} + 2 \cdot P(a, b) \sum_k F_k^a \cdot T_{ij}^{kb}
\]

where \( P(a, b) \) implies antisymmetrization of index pair \( ab \), as

\[
Z["abij"] = 0.5*W["abij"];
Z["abij"] += 2.0*F["ak"]*T["kbij"];
\]

- **for** loops and summations implicit in syntax
- \( P(a, b) \) is applied implicitly if \( Z \) is antisymmetric in \( ab \)
Tensor algebra interface (credit: Devin Matthews)

CTF can express a tensor contraction like

\[ Z_{ij}^{ab} = \frac{1}{2} \cdot W_{ij}^{ab} + 2 \cdot P(a, b) \sum_k F^a_k \cdot T^b_k \]

where \( P(a, b) \) implies antisymmetrization of index pair \( ab \), as

\[
\begin{align*}
Z["abij"] &= 0.5 \cdot W["abij"]; \\
Z["abij"] &= 2.0 \cdot F["ak"] \cdot T["kbij"]; \\
\end{align*}
\]

- **for** loops and summations implicit in syntax
- \( P(a, b) \) is applied implicitly if \( Z \) is antisymmetric in \( ab \)
- \( Z, F, T, W \) should all be defined on the same world and all processes in the world must call the contraction bulk synchronously
Tensor algebra interface (credit: Devin Matthews)

CTF can express a tensor contraction like

\[ Z_{ij}^{ab} = \frac{1}{2} \cdot W_{ij}^{ab} + 2 \cdot P(a, b) \sum_k F_k^a \cdot T_{ij}^{kb} \]

where \( P(a, b) \) implies antisymmetrization of index pair \( ab \), as

\[
\begin{align*}
Z["abij"] &= 0.5*W["abij"]; \\
Z["abij"] &=+ 2.0*F["ak"]*T["kbij"]; \\
\end{align*}
\]

- **for** loops and summations implicit in syntax
- \( P(a, b) \) is applied implicitly if \( Z \) is antisymmetric in \( ab \)
- \( Z, F, T, W \) should all be defined on the same world and all processes in the world must call the contraction bulk synchronously
- user-defined (mixed-type) scalar tensor functions can be applied instead of + and *
Quantum chemistry codes using CTF

- **Aquarius** was developed by Devin Matthews in conjunction with CTF.
Quantum chemistry codes using CTF

- **Aquarius** was developed by Devin Matthews in conjunction with CTF.
- **Libtensor** has been integrated with CTF by Evgeny Epifanovsky.
Quantum chemistry codes using CTF

- **Aquarius** was developed by Devin Matthews in conjunction with CTF
- **Libtensor** has been integrated with CTF by Evgeny Epifanovsky
- **Q-Chem** can leverage Libtensor and integration with CTF is almost complete
CCSD

Extracted from Aquarius (Devin Matthews' code, https://github.com/devinamatthews/aquarius)

\[
\begin{align*}
FMI["mi"] &\quad += 0.5*WMNEF["mnef"]*T(2)["efin"]; \\
WMNIJ["mnij"] &\quad += 0.5*WMNEF["mnef"]*T(2)["efij"]; \\
FAE["ae"] &\quad -= 0.5*WMNEF["mnef"]*T(2)["afmn"]; \\
WAMEI["amei"] &\quad -= 0.5*WMNEF["mnef"]*T(2)["afin"]; \\
\end{align*}
\]

\[
\begin{align*}
Z(2)["abij"] &\quad = WMNEF["ijab"]; \\
Z(2)["abij"] &\quad += FAE["af"]*T(2)["fbij"]; \\
Z(2)["abij"] &\quad -= FMI["ni"]*T(2)["abnj"]; \\
Z(2)["abij"] &\quad += 0.5*WABEF["abef"]*T(2)["efij"]; \\
Z(2)["abij"] &\quad += 0.5*WMNIJ["mnij"]*T(2)["abmn"]; \\
Z(2)["abij"] &\quad -= WAMEI["amei"]*T(2)["ebmj"]; \\
\end{align*}
\]
CCSDT

Extracted from Aquarius (Devin Matthews’ code)

\[
\begin{align*}
Z(1)["ai"] &= 0.25*WMNEF["mnef"]*T(3)["aefimn"];
\end{align*}
\]

\[
\begin{align*}
Z(2)["abij"] &= 0.5*WAMEF["bmef"]*T(3)["aefijm"];
Z(2)["abij"] &= -0.5*WMNEJ["mnej"]*T(3)["abeinm"];
Z(2)["abij"] &= \text{FME}["me"]*T(3)["abeijm"];
\end{align*}
\]

\[
\begin{align*}
Z(3)["abcijk"] &= \text{WABEJ}["bcek"]*T(2)["aeij"];
Z(3)["abcijk"] &= \text{WAMIJ}["bmjk"]*T(2)["acim"];
Z(3)["abcijk"] &= \text{FAE}["ce"]*T(3)["abeijk"];
Z(3)["abcijk"] &= \text{FMI}["mk"]*T(3)["abcijm"];
Z(3)["abcijk"] &= 0.5*\text{WABEF}["abef"]*T(3)["efciik"];
Z(3)["abcijk"] &= 0.5*\text{WMNIJ}["mnij"]*T(3)["abcmnk"];
Z(3)["abcijk"] &= \text{WAMEI}["amei"]*T(3)["ebcmjk"];
\end{align*}
\]
Tensor data input and output

- write, read, or accumulate data bulk synchronously by global index (coordinate format)
Tensor data input and output

- write, read, or accumulate data bulk synchronously by global index (coordinate format)
- input or output data from/to well-defined distributions faster
Tensor data input and output

- write, read, or accumulate data bulk synchronously by global index (coordinate format)
- input or output data from/to well-defined distributions faster
- extract contiguous tensor slices (to a subcommunicator if desired)
Tensor data input and output

- write, read, or accumulate data bulk synchronously by global index (coordinate format)
- input or output data from/to well-defined distributions faster
- extract contiguous tensor slices (to a subcommunicator if desired)
- extract permuted tensor slices (e.g. arbitrary subsets of rows and columns)
Tensor decomposition and mapping

CTF tensor decomposition

- cyclic layout used to preserve packed symmetric structure
Tensor decomposition and mapping

CTF tensor decomposition
- cyclic layout used to preserve packed symmetric structure
- overdecomposition employed to decouple the parallelization from the physical processor grid
Tensor decomposition and mapping

CTF tensor decomposition

- cyclic layout used to preserve packed symmetric structure
- overdecomposition employed to decouple the parallelization from the physical processor grid
Tensor decomposition and mapping

CTF tensor decomposition
- cyclic layout used to preserve packed symmetric structure
- overdecomposition employed to decouple the parallelization from the physical processor grid

CTF mapping logic
- arrange physical topology into all possible processor grids
Tensor decomposition and mapping

CTF tensor decomposition
- cyclic layout used to preserve packed symmetric structure
- overdecomposition employed to decouple the parallelization from the physical processor grid

CTF mapping logic
- arrange physical topology into all possible processor grids
- for each contraction autotune over all topologies and mappings
Tensor decomposition and mapping

CTF tensor decomposition
- cyclic layout used to preserve packed symmetric structure
- overdecomposition employed to decouple the parallelization from the physical processor grid

CTF mapping logic
- arrange physical topology into all possible processor grids
- for each contraction autotune over all topologies and mappings
- select best mapping based on performance models (communication cost, memory requirements, etc.)
Symmetric matrix representation

Symmetric matrix

Unique part of symmetric matrix
Blocked distributions of a symmetric matrix

Naive blocked layout

Block-cyclic layout
Cyclic distribution of a symmetric matrix

Cyclic layout ~ Improved blocked layout
Tensor contraction mapping visualization
The following three redistribution kernels are provided by CTF

- Sparse (key-value) redistribution (user input/output)
The following three redistribution kernels are provided by CTF

- Sparse (key-value) redistribution (user input/output)
  - performs (threaded) binning of key-value pairs and sends the pairs
Algorithms and optimization for tensor redistribution

The following three redistribution kernels are provided by CTF

- Sparse (key-value) redistribution (user input/output)
  - performs (threaded) binning of key-value pairs and sends the pairs
- Dense mapping-to-mapping redistribution between arbitrary decompositions

processors exchange blocks via point-to-point messages
The following three redistribution kernels are provided by CTF

- Sparse (key-value) redistribution (user input/output)
  - performs (threaded) binning of key-value pairs and sends the pairs

- Dense mapping-to-mapping redistribution between arbitrary decompositions
  - performs (threaded) binning by implicit index and sends pure data
The following three redistribution kernels are provided by CTF:

- **Sparse (key-value) redistribution (user input/output)**
  - performs (threaded) binning of key-value pairs and sends the pairs

- **Dense mapping-to-mapping redistribution between arbitrary decompositions**
  - performs (threaded) binning by implicit index and sends pure data

- **Block-to-block redistribution between similar distributions on different processor grids**
The following three redistribution kernels are provided by CTF

- **Sparse (key-value) redistribution (user input/output)**
  - performs (threaded) binning of key-value pairs and sends the pairs

- **Dense mapping-to-mapping redistribution between arbitrary decompositions**
  - performs (threaded) binning by implicit index and sends pure data

- **Block-to-block redistribution between similar distributions on different processor grids**
  - processors exchange blocks via point-to-point messages
Comparison with NWChem

NWChem is a commonly-used distributed-memory quantum chemistry method suite

- provides CCSD and CCSDT
- uses Global Arrays (GA) tensor partitioning and contraction
- Tensor Contraction Engine (TCE) factorizes CC equations and generated GA code
Coupled-cluster code on BlueGene/Q (Mira)

CCSD up to 55 water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ

Weak scaling on BlueGene/Q

Aquarius-CTF CCSD
Aquarius-CTF CCSDT

Gigaflops/node

#nodes
Coupled-cluster code on Cray XC30 (Edison)

CCSD up to 50 water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ
Symmetric-matrix–vector multiplication

- Consider symmetric $n \times n$ matrix $A$ and vectors $b, c$
Symmetric-matrix–vector multiplication

- Consider symmetric $n \times n$ matrix $A$ and vectors $b, c$
- $c = A \cdot b$ is usually done by computing a *nonsymmetric* intermediate matrix $W$,

$$W_{ij} = A_{ij} \cdot b_j$$

$$c_i = \sum_{j=1}^{n} W_{ij}$$

which requires $n^2$ multiplications and $n^2$ additions
Symmetric-matrix–vector multiplication

- Consider symmetric $n \times n$ matrix $A$ and vectors $b, c$
- $c = A \cdot b$ is usually done by computing a *nonsymmetric* intermediate matrix $W$,

$$W_{ij} = A_{ij} \cdot b_j$$

$$c_i = \sum_{j=1}^{n} W_{ij}$$

which requires $n^2$ multiplications and $n^2$ additions

- The *symmetry preserving algorithm* employs a *symmetric* intermediate matrix $Z$,

$$Z_{ij} = A_{ij} \cdot (b_i + b_j)$$

$$c_i = \sum_{j=1}^{n} Z_{ij} - \left( \sum_{j=1}^{n} A_{ij} \right) \cdot b_i$$

which requires $\frac{n^2}{2}$ multiplications and $\frac{5n^2}{2}$ additions
Symmetrized rank-two outer product

- Consider vectors $\mathbf{a}, \mathbf{b}$ of dimension $n$
Symmetry Preserving Algorithm

Instances in matrix computations

Symmetrized rank-two outer product

- Consider vectors $\mathbf{a}, \mathbf{b}$ of dimension $n$
- Symmetric matrix $\mathbf{C} = \mathbf{a} \cdot \mathbf{b}^T + \mathbf{b} \cdot \mathbf{a}^T$ is usually done by computing a nonsymmetric intermediate matrix $\mathbf{W}$,

  $$ W_{ij} = a_i \cdot b_j \quad \quad C_{ij} = W_{ij} + W_{ji} $$

  which requires $n^2$ multiplications and $n^2/2$ additions
Symmetrized rank-two outer product

- Consider vectors $\mathbf{a}, \mathbf{b}$ of dimension $n$
- Symmetric matrix $\mathbf{C} = \mathbf{a} \cdot \mathbf{b}^\text{T} + \mathbf{b} \cdot \mathbf{a}^\text{T}$ is usually done by computing a *nonsymmetric* intermediate matrix $\mathbf{W}$,

$$W_{ij} = a_i \cdot b_j \quad C_{ij} = W_{ij} + W_{ji}$$

which requires $n^2$ multiplications and $n^2/2$ additions
- The *symmetry preserving algorithm* employs a *symmetric* intermediate matrix $\mathbf{Z}$,

$$Z_{ij} = (a_i + a_j) \cdot (b_i + b_j) \quad C_{ij} = Z_{ij} - a_i \cdot b_i - a_j \cdot b_j$$

which requires $\frac{n^2}{2}$ multiplications and $2n^2$ additions
Symmetry Preserving Algorithm

Symmetrized matrix multiplication

Consider symmetric $n \times n$ matrices $A$, $B$, and $C$. 

Symmetry preserving algorithm employs a symmetric intermediate tensor $Z$ using $n^3/6$ multiplications and $7n^3/6$ additions.
Symmetrized matrix multiplication

- Consider symmetric $n \times n$ matrices $A$, $B$, and $C$
- $C = A \cdot B + B \cdot A$ is usually computed via a nonsymmetric intermediate order 3 tensor $W$,
  \[ W_{ijk} = A_{ik} \cdot B_{kj} \quad \bar{W}_{ij} = \sum_k W_{ijk} \quad C_{ij} = W_{ij} + W_{ji}. \]

which requires $n^3$ multiplications and $n^3$ additions.
Symmetrized matrix multiplication

- Consider symmetric $n \times n$ matrices $A$, $B$, and $C$
- $C = A \cdot B + B \cdot A$ is usually computed via a nonsymmetric intermediate order 3 tensor $W$,

$$W_{ijk} = A_{ik} \cdot B_{kj} \quad \bar{W}_{ij} = \sum_k W_{ijk} \quad C_{ij} = W_{ij} + W_{ji}.$$

which requires $n^3$ multiplications and $n^3$ additions.
- The symmetry preserving algorithm employs a symmetric intermediate tensor $Z$ using $n^3/6$ multiplications and $7n^3/6$ additions,

$$Z_{ijk} = (A_{ij} + A_{ik} + A_{jk}) \cdot (B_{ij} + B_{ik} + B_{jk}) \quad v_i = \sum_{k=1}^{n} A_{ik} \cdot B_{ik}$$

$$C_{ij} = \sum_{k=1}^{n} Z_{ijk} - n \cdot A_{ij} \cdot B_{ij} - v_i - v_j - \left(\sum_{k=1}^{n} A_{ik}\right) \cdot B_{ij} - A_{ij} \cdot \left(\sum_{k=1}^{n} B_{ik}\right)$$
Symmetry preserving algorithm generalization

- Any fully symmetrized contraction of two fully symmetric tensors with a total of \( \omega \) indices can be done with \( \frac{n^\omega}{\omega!} + O(n^{\omega-1}) \) multiplications.
Symmetry preserving algorithm generalization

- Any fully symmetrized contraction of two fully symmetric tensors with a total of $\omega$ indices can be done with $n^\omega/\omega! + O(n^{\omega-1})$ multiplications.
- Extensions to antisymmetric tensors and antisymmetrized contractions possible, but not for all cases.
Symmetry preserving algorithm generalization

- Any fully symmetrized contraction of two fully symmetric tensors with a total of $\omega$ indices can be done with $n^\omega/\omega! + O(n^{\omega-1})$ multiplications.
- Extensions to antisymmetric tensors and antisymmetrized contractions possible, but not for all cases.
- Extends to all complex/Hermitian cases.
Symmetry preserving algorithm generalization

- Any fully symmetrized contraction of two fully symmetric tensors with a total of $\omega$ indices can be done with $n^\omega/\omega! + O(n^{\omega-1})$ multiplications.
- Extensions to antisymmetric tensors and antisymmetrized contractions possible, but not for all cases.
- Extends to all complex/Hermitian cases.
- Also applicable to contractions of a tensor with itself, in particular $A^2$ for symmetric or antisymmetric matrix $A$ requires $n^3/6$ multiplications.
Symmetry preserving algorithm generalization

- Any fully symmetrized contraction of two fully symmetric tensors with a total of $\omega$ indices can be done with $n^\omega/\omega! + O(n^{\omega-1})$ multiplications.
- Extensions to antisymmetric tensors and antisymmetrized contractions possible, but not for all cases.
- Extends to all complex/Hermitian cases.
- Also applicable to contractions of a tensor with itself, in particular $A^2$ for symmetric or antisymmetric matrix $A$ requires $n^3/6$ multiplications.
- Nonsymmetric $A^2$ (or more generally $A \cdot B + B \cdot A$ for nonsymmetric matrices $A$, $B$) can be done in $2n^3/3$ operations.
Symmetry preserving algorithm generalization

- Any fully symmetrized contraction of two fully symmetric tensors with a total of $\omega$ indices can be done with $n^\omega/\omega! + O(n^{\omega-1})$ multiplications.
- Extensions to antisymmetric tensors and antisymmetrized contractions possible, but not for all cases.
- Extends to all complex/Hermitian cases.
- Also applicable to contractions of a tensor with itself, in particular $A^2$ for symmetric or antisymmetric matrix $A$ requires $n^3/6$ multiplications.
- Nonsymmetric $A^2$ (or more generally $A \cdot B + B \cdot A$ for nonsymmetric matrices $A$, $B$) can be done in $2n^3/3$ operations.
- Numerical stability confirmed via proof and experiments.
Symmetry preserving algorithm generalization

- Any fully symmetrized contraction of two fully symmetric tensors with a total of \( \omega \) indices can be done with \( n^\omega /\omega! + O(n^{\omega-1}) \) multiplications.
- Extensions to antisymmetric tensors and antisymmetrized contractions possible, but not for all cases.
- Extends to all complex/Hermitian cases.
- Also applicable to contractions of a tensor with itself, in particular \( A^2 \) for symmetric or antisymmetric matrix \( A \) requires \( n^3/6 \) multiplications.
- Nonsymmetric \( A^2 \) (or more generally \( A \cdot B + B \cdot A \) for nonsymmetric matrices \( A, B \)) can be done in \( 2n^3/3 \) operations.
- Numerical stability confirmed via proof and experiments.
- Communication cost lower and upper bounds derived.
The CCSD contraction

$$Z_{i\bar{c}}^{a\bar{k}} = \sum_b \sum_j T_{ij}^{ab} \cdot V_{j\bar{k}}^{b\bar{c}}$$

usually requires $2n^6$ total operations.
Application to CCSD

The CCSD contraction

\[ Z_{i\bar{c}}^{a\bar{k}} = \sum_b \sum_j T_{ij}^{ab} \cdot V_{j\bar{k}}^{b\bar{c}} \]

usually requires \(2n^6\) total operations.

The symmetry-preserving algorithm can be applied over the indices

\[ Z^a = \sum_b T^{ab} \cdot V_b \]

with each multiplication being a contraction over the other four indices \(i,j,\bar{c},\bar{k}\), which is more expensive than the addition operations, yielding \(n^6\) operations to leading order.
Application to CCSD(T) and CCSDT(Q)

The CCSD(T) contraction

\[
T_{ijkl}^{abc} = P(a, b)P(i, j) \sum_{l=1}^{n} T_{il}^{ac} \cdot W_{lj}^{lb}
\]

usually requires \(2n^7\) total operations.
Application to CCSD(T) and CCSDT(Q)

The CCSD(T) contraction

\[
T_{\overline{ab}\overline{c}ijk} = P(a, b)P(i, j) \sum_{\overline{l}=1}^{n} T_{\overline{a}\overline{c}i\overline{l}} \cdot W_{\overline{l}jk}
\]

usually requires \(2n^7\) total operations. The symmetry-preserving algorithm can be applied over the indices

\[
T^{ab} = P(a, b)T^a \cdot W^b \quad \text{and} \quad T_{ij} = P(i, j)T_i \cdot T_j
\]

with each multiplication in the latter being a contraction over the remaining three indices \(\overline{c}, \overline{k}, \text{ and } \overline{l}\), for a total of \(n^7/2\) leading order operations.
Application to CCSD(T) and CCSDT(Q)

The CCSD(T) contraction

\[ T^{ab\bar{c}}_{ijk} = P(a, b)P(i, j) \sum_{\bar{l}=1}^{n} T^{\bar{a}\bar{c}}_{\bar{i}\bar{l}} \cdot W^{\bar{l}b}_{jk} \]

usually requires $2n^7$ total operations.

The symmetry-preserving algorithm can be applied over the indices

\[ T^{ab} = P(a, b)T^a \cdot W^b \quad \text{and} \quad T_{ij} = P(i, j)T_i \cdot T_j \]

with each multiplication in the latter being a contraction over the remaining three indices $\bar{c}, \bar{k},$ and $\bar{l},$ for a total of $n^7/2$ leading order operations.

For a similar CCSDT(Q) contraction, which usually requires $n^9/2$ operations, the symmetry preserving algorithm achieves $n^9/18.$
Conclusion

Future work on symmetry-preserving algorithms

- full cost derivations for CC methods
Future work on symmetry-preserving algorithms

- full cost derivations for CC methods
- communication cost analysis for partially-symmetric contractions
Conclusion

Future work on symmetry-preserving algorithms

- full cost derivations for CC methods
- communication cost analysis for partially-symmetric contractions
- integration into CTF (the power of abstraction: one day you update the CTF version and your CC code becomes faster)
Conclusion

Future work on symmetry-preserving algorithms

- full cost derivations for CC methods
- communication cost analysis for partially-symmetric contractions
- integration into CTF (the power of abstraction: one day you update the CTF version and your CC code becomes faster)
Future work on symmetry-preserving algorithms
- full cost derivations for CC methods
- communication cost analysis for partially-symmetric contractions
- integration into CTF (the power of abstraction: one day you update the CTF version and your CC code becomes faster)

Future work on CTF
- iterative performance-model refinement via online learning
Future work on symmetry-preserving algorithms

- full cost derivations for CC methods
- communication cost analysis for partially-symmetric contractions
- integration into CTF (the power of abstraction: one day you update the CTF version and your CC code becomes faster)

Future work on CTF

- iterative performance-model refinement via online learning
- automatic multi-contraction scheduling
Conclusion

Future work on symmetry-preserving algorithms

- full cost derivations for CC methods
- communication cost analysis for partially-symmetric contractions
- integration into CTF (the power of abstraction: one day you update the CTF version and your CC code becomes faster)

Future work on CTF

- iterative performance-model refinement via online learning
- automatic multi-contraction scheduling
- sparse tensors
Acknowledgements

Contributors to mentioned work

- Devin Matthews (UT Austin)
- James Demmel (UC Berkeley)
- Jeff Hammond (Intel Corp.)
- Evgeny Epifanovsky (Q-Chem, Inc.)
- Torsten Hoefler (ETH Zurich)

Resources

- US DOE Computational Science Graduate Fellowship
- ETH Zurich Postdoctoral Fellowship
- Supercomputer allocations via NERSC and ANL