Cyclops Tensor Framework

Edgar Solomonik

Department of EECS, Computer Science Division, UC Berkeley

March 17, 2014
A rank $r$ tensor is a $r$-dimensional representation of a dataset, for example,

- a rank one tensor is a vector (e.g. a set of nodes $V$)
- a rank two tensor is a matrix (e.g. a set of edges $E$ in a graph $E \subset V \times V$)

Graphically, a rank $r$ tensor is a set of all possible paths $P$ of length $r$ through vertices $V$

$$P \subset V \times \ldots \times V$$

Alternatively, $P$ may be thought of as a set of hypergraph edges with cardinality $r$

Programmatically, a rank $r$ tensor is an $r$-dimensional array
Given rank 4 tensors $\mathbf{T}$, $\mathbf{V}$, and $\mathbf{W}$ we may write perform tensor contraction as

$$W_{abij} = \sum_k \sum_l T_{abkl} \cdot V_{klij}$$

It is common to use raised and lowered index notation, which is sometimes related to the physical meaning of the indices,

$$W_{ij}^{ab} = \sum_{kl} T_{ab}^{kl} \cdot V_{ij}^{kl}$$

raised-indices are usually meant to be contracted with lowered indices.
Tensor folding

Since a tensor is a representation of any data set, we may always switch representations

- define transformation $\delta_{ab}^p$ to transform $a$ and $b$ into compound index $p$ ($\delta_{ab}^p = 1$ when $p = a + b \cdot n$ and 0 otherwise)
- graphically, folding corresponds to replacing edges with vertices ($W = V \times V$)

The contraction

$$W_{ij}^{ab} = \sum_{kl} T_{kl}^{ab} \cdot V_{ij}^{kl}$$

may be folded into matrix multiplication as follows

$$\delta_{ab}^p \cdot W_{ij}^{ab} \cdot \delta_{ij}^q = \sum_r \delta_{ab}^p \cdot T_{kl}^{ab} \cdot \delta_{rl}^{kl} \cdot \delta_{kl}^r \cdot V_{ij}^{kl} \cdot \delta_{ij}^q$$

$$W_{qp} = \sum_r T_{rp}^p \cdot V_{rq}^r$$
Reasons to use tensor representations

If all contractions can be folded into matrix multiplication, why use tensors of rank greater than two?

- permutational index symmetry: the tensors may express higher-dimensional structure
- expression of many different contractions with a single representation (each may require different folding)
- finding low-rank tensor decompositions, such as the CP (CANDECOMP/PARAFAC) decomposition

\[ T_{ijk} \approx \sum_{r}^{R} v_{ir} \cdot w_{jr} \cdot z_{kr} \]

This talk will not address low-rank decompositions.
Coupled Cluster (CC) is a numerical approximation scheme to the time-independent many-body Schrödinger equation

\[ |\psi\rangle = e^{T_1 + T_2 + T_3 + \ldots} |\phi_0\rangle \]

where \( T_k \) is a rank \( 2k \) 'amplitude' tensor which correlates sets of \( k \) electrons over sets of \( k \) basis-functions (captures \( k \)-electron excitations)

- the CCSD method is a truncation at \( T_1 + T_2 \)
- the CCSDT method also includes \( T_3 \)

The CC methods produce a set of nonlinear equations for the amplitude tensors which are solved iteratively via tensor contractions

Given a system of \( n \) electrons, the methods require \( O(n^{2k}) \) memory and \( O(n^{2k+2}) \) operations
Motivation and goals

Cyclops (cyclic-operations) Tensor Framework (CTF)
  - provides primitives for distributed memory tensor contractions
Motivation and goals

Cyclops (cyclic-operations) Tensor Framework (CTF)
- provides primitives for distributed memory tensor contractions
- takes advantage of thread (two-level) parallelism
Cyclops (cyclic-operations) Tensor Framework (CTF)
- provides primitives for distributed memory tensor contractions
- takes advantage of thread (two-level) parallelism
- exposes a simple domain specific language for contractions
Motivation and goals

Cyclops (cyclic-operations) Tensor Framework (CTF)
- provides primitives for distributed memory tensor contractions
- takes advantage of thread (two-level) parallelism
- exposes a simple domain specific language for contractions
- allows for efficient tensor redistribution and slicing
Motivation and goals

Cyclops (cyclic-operations) Tensor Framework (CTF)

- provides primitives for distributed memory tensor contractions
- takes advantage of thread (two-level) parallelism
- exposes a simple domain specific language for contractions
- allows for efficient tensor redistribution and slicing
- exploits permutational tensor symmetry efficiently
Motivation and goals

Cyclops (cyclic-operations) Tensor Framework (CTF)

- provides primitives for distributed memory tensor contractions
- takes advantage of thread (two-level) parallelism
- exposes a simple domain specific language for contractions
- allows for efficient tensor redistribution and slicing
- exploits permutational tensor symmetry efficiently
- uses only MPI, BLAS, and OpenMP and is a library
Define a parallel world

CTF relies on MPI (Message Passing Interface) for multiprocessor parallelism

- a set of processors in MPI corresponds to a communicator (MPI_Comm)
- MPI_COMM_WORLD is the default communicators containing all processes
- CTF_World dw(comm) defines an instance of CTF on any MPI communicator
Consider a rank four tensor $\mathbf{T}$ (in CC this is the 2-electron $\mathbf{T}_2$ amplitude)

$$
T^{ab}_{ij}
$$

where $\mathbf{T}$ is $m \times m \times n \times n$ antisymmetric in $ab$ and in $ij$

- CTF\_Tensor $\mathbf{T}(4,\{m,m,n,n\},\{AS,NS,AS,NS\},dw)$
Consider a rank four tensor $\mathbf{T}$ (in CC this is the 2-electron $T_2$ amplitude)

$$T_{ij}^{ab}$$

where $\mathbf{T}$ is $m \times m \times n \times n$ antisymmetric in $ab$ and in $ij$

- CTF_Tensor $T(4,\{m,m,n,n\},\{\text{AS,NS,AS,NS}\},\text{dw})$
- an 'AS' dimension is antisymmetric with the next
Define a tensor

Consider a rank four tensor $\mathbf{T}$ (in CC this is the 2-electron $T_2$ amplitude)

$$T_{ij}^{ab}$$

where $\mathbf{T}$ is $m \times m \times n \times n$ antisymmetric in $ab$ and in $ij$

- CTF_Tensor $\mathbf{T}(4,\{m,m,n,n\},\{\text{AS},\text{NS},\text{AS},\text{NS}\},\text{dw})$
- an 'AS' dimension is antisymmetric with the next
- symmetric (SY) and symmetric-hollow (SH) are also possible
Define a tensor

Consider a rank four tensor $T$ (in CC this is the 2-electron $T_2$ amplitude)

$$T_{ij}^{ab}$$

where $T$ is $m \times m \times n \times n$ antisymmetric in $ab$ and in $ij$

- CTF_Tensor $T(4,\{m,m,n,n\},\{AS,NS,AS,NS\},dw)$
- an 'AS' dimension is antisymmetric with the next
- symmetric (SY) and symmetric-hollow (SH) are also possible
- the first dimension of the tensor is mapped linearly onto memory
Define a tensor

Consider a rank four tensor $\mathbf{T}$ (in CC this is the 2-electron $T_2$ amplitude)

$$T_{ab}^{ij}$$

where $\mathbf{T}$ is $m \times m \times n \times n$ antisymmetric in $ab$ and in $ij$

- CTF_Tensor $\text{T}(4,\{m,m,n,n\},\{\text{AS},\text{NS},\text{AS},\text{NS}\},\text{dw})$
- an 'AS' dimension is antisymmetric with the next
- symmetric (SY) and symmetric-hollow (SH) are also possible
- the first dimension of the tensor is mapped linearly onto memory
- there are also obvious derived types for CTF_Tensor: CTF_Matrix, CTF_Vector, CTF_Scalar
CTF can express a tensor contraction like

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

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

- \( Z["\text{abij}"] += 2.0 \times F["\text{ak}"] \times T["\text{kbij}"] \)
CTF can express a tensor contraction like

\[ Z_{ij}^{ab} = Z_{ij}^{ab} + 2 \cdot P(a, b) \sum_k F^a_k \cdot T^{kb}_{ij} \]

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

- \( Z["\text{abij}"] += 2.0 \ast F["\text{ak}"] \ast T["\text{kbij}"] \)
- \( P(a, b) \) is applied implicitly if \( Z \) is antisymmetric in \( ab \)
CTF can express a tensor contraction like

\[ Z_{ij}^{ab} = Z_{ij}^{ab} + 2 \cdot P(a, b) \sum_k F^a_k \cdot T^{kb}_{ij} \]

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

- \( Z["abij"] += 2.0*F["ak"]*T["kbij"] \)
- \( P(a, b) \) is applied implicitly if \( Z \) is antisymmetric in \( ab \)
- \( Z, F, T \) should all be defined on the same world and all processes in the world must call the contraction bulk synchronously
CTF can express a tensor contraction like

\[ Z_{ij}^{ab} = Z_{ij}^{ab} + 2 \cdot P(a, b) \sum_{k} F_{ka} \cdot T_{kj}^{kb} \]

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

- \( Z[^{"abij"}] += 2.0 \cdot F[^{"ak"}] \cdot T[^{"kbij"}] \)
- \( P(a, b) \) is applied implicitly if \( Z \) is antisymmetric in \( ab \)
- \( Z, F, T \) should all be defined on the same world and all processes in the world must call the contraction bulk synchronously
- the beginning of the end of all for loops...
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - `T.write(int * indices, double * data)` (also possible to scale)

Matlab submatrix notation:

- `A[j:k, l:m]`
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - T.write(int * indices, double * data) (also possible to scale)
  - T.read(int * indices, double * data) (also possible to scale)
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - T.write(int * indices, double * data) (also possible to scale)
  - T.read(int * indices, double * data) (also possible to scale)
- Matlab submatrix notation: \( A[j : k, l : m] \)
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - T.write(int * indices, double * data) (also possible to scale)
  - T.read(int * indices, double * data) (also possible to scale)
- Matlab submatrix notation: $A[j : k, l : m]$
  - T.slice(int * offsets, int * ends) returns the subtensor
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - `T.write(int * indices, double * data)` (also possible to scale)
  - `T.read(int * indices, double * data)` (also possible to scale)

- Matlab submatrix notation: \( A[j : k, l : m] \)
  - `T.slice(int * offsets, int * ends)` returns the subtensor
  - `T.slice(int corner_off, int corner_end)` does the same
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - T.write(int * indices, double * data) (also possible to scale)
  - T.read(int * indices, double * data) (also possible to scale)

- Matlab submatrix notation: \(A[j : k, l : m]\)
  - T.slice(int * offsets, int * ends) returns the subtensor
  - T.slice(int corner_off, int corner_end) does the same
  - can also sum a subtensor from one tensor with a subtensor of another tensor
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - T.write(int * indices, double * data) (also possible to scale)
  - T.read(int * indices, double * data) (also possible to scale)

- Matlab submatrix notation: \( A[j : k, l : m] \)
  - T.slice(int * offsets, int * ends) returns the subtensor
  - T.slice(int corner_off, int corner_end) does the same
  - can also sum a subtensor from one tensor with a subtensor of another tensor
  - different subworlds can read different subtensors simultaneously
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - \( \text{T.write}(\text{int }* \text{ indices}, \text{double }* \text{ data}) \) (also possible to scale)
  - \( \text{T.read}(\text{int }* \text{ indices}, \text{double }* \text{ data}) \) (also possible to scale)
- Matlab submatrix notation: \( A[j : k, l : m] \)
  - \( \text{T.slice}(\text{int }* \text{ offsets, int }* \text{ ends}) \) returns the subtensor
  - \( \text{T.slice}(\text{int corner_off, int corner_end}) \) does the same
  - can also sum a subtensor from one tensor with a subtensor of another tensor
  - different subworlds can read different subtensors simultaneously
- Extract a subtensor of any permutation of the tensor
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - `T.write(int * indices, double * data)` (also possible to scale)
  - `T.read(int * indices, double * data)` (also possible to scale)

- Matlab submatrix notation: `A[j : k, l : m]`
  - `T.slice(int * offsets, int * ends)` returns the subtensor
  - `T.slice(int corner_off, int corner_end)` does the same
  - can also sum a subtensor from one tensor with a subtensor of another tensor
  - different subworlds can read different subtensors simultaneously

- Extract a subtensor of any permutation of the tensor
  - given mappings `P, Q`, does `B[i, j] = A[P[i], Q[j]]` via `permute()`
Access and write tensor data

CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - \( T.\text{write}(\text{int } \ast \text{ indices}, \text{double } \ast \text{ data}) \) (also possible to scale)
  - \( T.\text{read}(\text{int } \ast \text{ indices}, \text{double } \ast \text{ data}) \) (also possible to scale)

- Matlab submatrix notation: \( A[j : k, l : m] \)
  - \( T.\text{slice}(\text{int } \ast \text{ offsets}, \text{int } \ast \text{ ends}) \) returns the subtensor
  - \( T.\text{slice}(\text{int } \text{corner}_\text{off}, \text{int } \text{corner}_\text{end}) \) does the same
  - can also sum a subtensor from one tensor with a subtensor of another tensor
  - different subworlds can read different subtensors simultaneously

- Extract a subtensor of any permutation of the tensor
  - given mappings \( P, Q \), does \( B[i, j] = A[P[i], Q[j]] \) via permute()
  - \( P \) and \( Q \) may access only subsets of \( A \) (if \( B \) is smaller)
CTF takes away your data pointer

- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - `T.write(int * indices, double * data)` (also possible to scale)
  - `T.read(int * indices, double * data)` (also possible to scale)

- Matlab submatrix notation: `A[j : k, l : m]`
  - `T.slice(int * offsets, int * ends)` returns the subtensor
  - `T.slice(int corner_off, int corner_end)` does the same
  - can also sum a subtensor from one tensor with a subtensor of another tensor
  - different subworlds can read different subtensors simultaneously

- Extract a subtensor of any permutation of the tensor
  - given mappings `P, Q`, does `B[i, j] = A[P[i], Q[j]]` via permute()
  - `P` and `Q` may access only subsets of `A` (if `B` is smaller)
  - `B` may be defined on subworlds on the world on which `A` is defined and each subworld may specify different `P` and `Q`
Extracted from Aquarius (Devin Matthews’ code)

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

\[
\begin{align*}
Z(2)["abij"] &= WMNEF["ijab"]; \\
Z(2)["abij"] &= FAE["af"]*T(2)["fbij"]; \\
Z(2)["abij"] &= FMI["ni"]*T(2)["abnj"]; \\
Z(2)["abij"] &= 0.5*WABEF["abef"]*T(2)["efij"]; \\
Z(2)["abij"] &= 0.5*WMNIJ["mnij"]*T(2)["abmn"]; \\
Z(2)["abij"] &= WAMEI["amei"]*T(2)["ebmj"]; \\
\end{align*}
\]
Extracted from Aquarius (Devin Matthews’ code)

\[
\begin{align*}
Z(1)["ai"] & += 0.25*WMNEF["mnef"] * T(3)["aefimn"]; \\
Z(2)["abij"] & += 0.5*WAMEF["bmef"] * T(3)["aefijm"]; \\
Z(2)["abij"] & -= 0.5*WMNEJ["mnej"] * T(3)["abeinm"]; \\
Z(2)["abij"] & += FME["me"] * T(3)["abeijm"]; \\
Z(3)["abcijk"] & = WABEJ["bcek"] * T(2)["aeij"]; \\
Z(3)["abcijk"] & -= WAMIJ["bmjk"] * T(2)["acim"]; \\
Z(3)["abcijk"] & += FAE["ce"] * T(3)["abeijk"]; \\
Z(3)["abcijk"] & -= FMI["mk"] * T(3)["abcijm"]; \\
Z(3)["abcijk"] & += 0.5*WABEF["abef"] * T(3)["efcijk"]; \\
Z(3)["abcijk"] & += 0.5*WMNIJ["mnij"] * T(3)["abcmnk"]; \\
Z(3)["abcijk"] & -= WAMEI["amei"] * T(3)["ebcmjk"];
\end{align*}
\]
CCSD up to 55 water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ
Coupled Cluster on Cray XC30 Edison

CCSD up to 50 water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ
Comparison with NWChem

NWChem is a distributed-memory quantum chemistry method suite

- provides CCSD and CCSDT
- uses Global Arrays a Partitioned Global Address Space (PGAS) backend for tensor contractions
- derivation automatically done by Tensor Contraction Engine (TCE)

CCSD performance on Edison (thanks to Jeff Hammond for building NWChem and collecting data)

- NWChem 40 water molecules on 1024 nodes: 44 min
NWChem is a distributed-memory quantum chemistry method suite

- provides CCSD and CCSDT
- uses Global Arrays a Partitioned Global Address Space (PGAS) backend for tensor contractions
- derivation automatically done by Tensor Contraction Engine (TCE)

CCSD performance on Edison (thanks to Jeff Hammond for building NWChem and collecting data)

- NWChem 40 water molecules on 1024 nodes: 44 min
- CTF 40 water molecules on 1024 nodes: 9 min
A high-level description of NWChem’s algorithm for tensor contractions:

- data layout is abstracted away by the Global Arrays framework
- Global Arrays uses one-sided communication for data movement
- packed tensors are stored in blocks
- for each contraction, each process does a subset of the block contractions
- each block is transposed and unpacked prior to contraction
- dynamic load balancing is employed among processors
A high-level description of CTF’s algorithm for tensor contractions:

- packed tensors are decomposed cyclically among toroidal processor grids
- MPI collectives are used for all communication
- for each contraction, a distributed layout is selected based on internal performance models
- performance model considers all possible execution paths
- before contraction, tensors are redistributed to a new layout
- if there is enough memory, the tensors are (partially) unpacked
- all preserved symmetries and non-symmetric indices are folded in preparation for matrix multiplication
- nested distributed matrix multiply algorithms are used to perform the contraction in a load-balanced manner
In CTF, tensors are defined on a communicator (subset or full set of processors)

- the data pointer is hidden from the user
- the user can perform block-synchronous bulk writes and reads of index-value pairs
- to avoid communication, the user may read the current local pairs
- it is possible to perform overlapped writes (accumulate)
- CTF internal implementation (all parts threaded):
  1. bin keys by processor and redistribute
  2. bin key by virtual processor and then sort them
  3. iterate over the dense tensor, reading or writing keys along the way
  4. return keys to originating location if its a sparse read
Blocked vs block-cyclic vs cyclic decompositions

- **Blocked layout**: Red denotes padding / load imbalance.
- **Block-cyclic layout**: Green denotes fill (unique values).
- **Cyclic layout**: Red denotes padding / load imbalance.

- Green denotes fill (unique values).
- Red denotes padding / load imbalance.
Matrix multiply on 2x3 processor grid. Red lines represent virtualized part of processor grid. Elements assigned to blocks by cyclic phase.
3D tensor mapping

Node (1,1) — Node (1,2) — Node (1,3) — Node (1,4)

Node (2,1) — Node (2,2) — Node (2,3) — Node (2,4)
The mapping process

Do in parallel over all physical topologies (foldings of the original torus)

1. map longest physical torus dimension to longest tensor dimension and repeat
The mapping process

Do in parallel over all physical topologies (foldings of the original torus)

1. map longest physical torus dimension to longest tensor dimension and repeat
2. select virtualization factors to preserve symmetry (as well as to match the algorithmic requirements)
The mapping process

Do in parallel over all physical topologies (foldings of the original torus)

1. map longest physical torus dimension to longest tensor dimension and repeat
2. select virtualization factors to preserve symmetry (as well as to match the algorithmic requirements)
3. calculate the necessary memory usage and communication cost of the algorithm
The mapping process

Do in parallel over all physical topologies (foldings of the original torus)

1. map longest physical torus dimension to longest tensor dimension and repeat
2. select virtualization factors to preserve symmetry (as well as to match the algorithmic requirements)
3. calculate the necessary memory usage and communication cost of the algorithm
4. consider whether and what type of redistribution is necessary for the mapping
The mapping process

Do in parallel over all physical topologies (foldings of the original torus)

1. map longest physical torus dimension to longest tensor dimension and repeat
2. select virtualization factors to preserve symmetry (as well as to match the algorithmic requirements)
3. calculate the necessary memory usage and communication cost of the algorithm
4. consider whether and what type of redistribution is necessary for the mapping
5. select the best mapping based on a performance model
CTF must migrate tensors between different mappings between operations as well as for slice()

- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs
Redistribution amongst different mappings

CTF must migrate tensors between different mappings between operations as well as for slice()

- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs
- iterate over local sub-tensor in linear order and bin keys to determine counts
Redistribution amongst different mappings

CTF must migrate tensors between different mappings between operations as well as for slice()

- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs
- iterate over local sub-tensor in linear order and bin keys to determine counts
- iterate over local piece of old tensor in global order and place keys into bins
CTF must migrate tensors between different mappings between operations as well as for slice():

- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs
- iterate over local sub-tensor in linear order and bin keys to determine counts
- iterate over local piece of old tensor in global order and place keys into bins
- MPI all-to-all-v
CTF must migrate tensors between different mappings between operations as well as for `slice()`

- The key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs.
- Iterate over local sub-tensor in linear order and bin keys to determine counts.
- Iterate over local piece of old tensor in global order and place keys into bins.
- MPI all-to-all-v
- Iterate over local piece of new tensor in global order and retrieve keys from bins.
CTF must migrate tensors between different mappings between operations as well as for slice()  
- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs  
- iterate over local sub-tensor in linear order and bin keys to determine counts  
- iterate over local piece of old tensor in global order and place keys into bins  
- MPI all-to-all-v  
- iterate over local piece of new tensor in global order and retrieve keys from bins  
- kernel is threaded according to a global tensor partitioning
In some cases, it is necessary to change the assignment of the tensor dimensions to virtual grid dimensions without changing the virtual processor grid itself.

- In this case, CTF does not touch data within each block.
- Redistributed by block instead.
- Use MPI Isend and MPI Irecv for each sent and received block.
Local transposition

Once the data is redistributed into the new mapping, we fold the tensors locally within blocks

- turns all non-symmetric block contractions into matrix multiplication
- 'preserved' symmetries may be folded into one dimension, but broken ones cannot
- maps dimensions which have symmetry that cannot be folded into matrix multiplication to have the longest stride
Once the tensors are distributed accordingly, the contraction algorithm begins

1. replicate small tensors over some processor grid dimensions (2.5D/3D matrix multiplication algorithms)
2. nested distributed SUMMA (2D matrix multiplication algorithm)
3. call to iterate over virtual blocks
4. call to iterate over broken symmetric dimensions
5. call to DGEMM
Cyclops Tensor Framework is available at ctf.cs.berkeley.edu and github.com/solomonik/ctf
Cyclops Tensor Framework is available at ctf.cs.berkeley.edu and github.com/solomonik/ctf

Latest version: v1.1 (March 2014), v1.0 was released in December 2013, development started in June 2011
Cyclops Tensor Framework is available at ctf.cs.berkeley.edu and github.com/solomonik/ctf

Latest version: v1.1 (March 2014), v1.0 was released in December 2013, development started in June 2011

CCSD(T) and CCSDT(Q) methods in development
Collaborators:

- Devin Matthews, UT Austin (contributions to CTF, teaching me CC, and development of Aquarius on top of CTF)
- Jeff Hammond, Argonne National Laboratory (initiated project, provides continuing advice, and runs NWChem for me when my patience runs out)
- James Demmel and Kathy Yelick, UC Berkeley (advising)

Grants:

- Krell DOE Computational Science Graduate Fellowship