Fast tensor contractions for Coupled Cluster

Edgar Solomonik

Department of EECS, Computer Science Division, UC Berkeley

Feb 7, 2014

Motivation and goals

Cyclops (cyclic-operations) Tensor Framework

- provide primitives for distributed memory tensor contractions

Motivation and goals

Cyclops (cyclic-operations) Tensor Framework

- provide primitives for distributed memory tensor contractions
- take advantage of thread (two-level) parallelism

Motivation and goals

Cyclops (cyclic-operations) Tensor Framework

- provide primitives for distributed memory tensor contractions
- take advantage of thread (two-level) parallelism
- expose a simple domain specific language for contractions

Motivation and goals

Cyclops (cyclic-operations) Tensor Framework

- provide primitives for distributed memory tensor contractions
- take advantage of thread (two-level) parallelism
- expose a simple domain specific language for contractions
- allow for efficient tensor redistribution and slicing

Motivation and goals

Cyclops (cyclic-operations) Tensor Framework

- provide primitives for distributed memory tensor contractions
- take advantage of thread (two-level) parallelism
- expose a simple domain specific language for contractions
- allow for efficient tensor redistribution and slicing
- exploit permutational tensor symmetry efficiently

Motivation and goals

Cyclops (cyclic-operations) Tensor Framework

- provide primitives for distributed memory tensor contractions
- take advantage of thread (two-level) parallelism
- expose a simple domain specific language for contractions
- allow for efficient tensor redistribution and slicing
- exploit 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

Define a tensor

A tensor is a multidimensional array, e.g.

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

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)

Define a tensor

A tensor is a multidimensional array, e.g.

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

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

Define a tensor

A tensor is a multidimensional array, e.g.

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

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

Define a tensor

A tensor is a multidimensional array, e.g.

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

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},{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

A tensor is a multidimensional array, e.g.

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

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
- there are also obvious derived types for CTF_Tensor: CTF_Matrix, CTF_Vector, CTF_Scalar

Contract tensors

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["abij"] += 2.0*F["ak"]*T["kbij"]

Contract tensors

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["abij"] += 2.0*F["ak"]*T["kbij"]
- $P(a,b)$ is applied implicitly if **Z** is antisymmetric in ab

Contract tensors

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["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

Contract tensors

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["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
- 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)

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)
  - `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

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.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)

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()
  - 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

Write a Coupled Cluster code

Extracted from Aquarius (Devin Matthews' code)

```
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"];

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"];
```

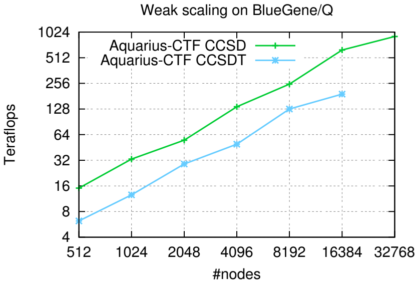
Write more Coupled Cluster code

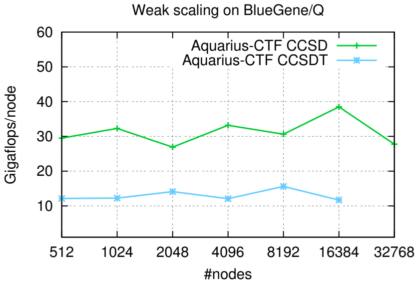
Extracted from Aquarius (Devin Matthews' code)

```
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"];
```

Run your Coupled Cluster code on a IBM supercomputer

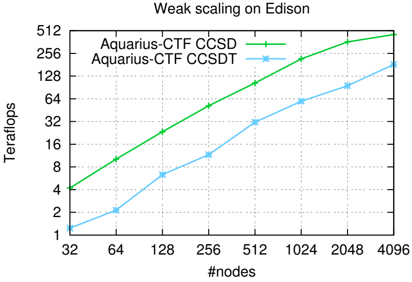
CCSD up to 55 water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ

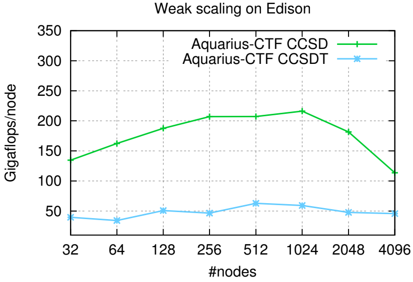




Run your Coupled Cluster code on the computer next door (Edison)

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





Run your Coupled Cluster code faster than NWChem

NWChem is a distributed-memory quantum chemistry method suite

- provides CCSD and CCSDT

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

Run your Coupled Cluster code faster than 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

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

Run your Coupled Cluster code faster than 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)

Run your Coupled Cluster code faster than 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

Run your Coupled Cluster code faster than 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
- CTF 40 water molecules on 1024 nodes: 9 min

Ongoing development in CTF

Lots of room for improvement, ongoing effort

- Multi-contraction scheduler being developed by Richard Lin (UCB)

Ongoing development in CTF

Lots of room for improvement, ongoing effort

- Multi-contraction scheduler being developed by Richard Lin (UCB)
- Better performance models and their external exposure

Ongoing development in CTF

Lots of room for improvement, ongoing effort

- Multi-contraction scheduler being developed by Richard Lin (UCB)
- Better performance models and their external exposure
- Improvements to tensor slicing and usage thereof for CCSD(T) and CCSDT(Q)

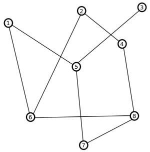
Ongoing development in CTF

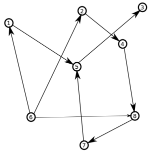
Lots of room for improvement, ongoing effort

- Multi-contraction scheduler being developed by Richard Lin (UCB)
- Better performance models and their external exposure
- Improvements to tensor slicing and usage thereof for CCSD(T) and CCSDT(Q)
- Faster symmetric tensor contraction algorithms...

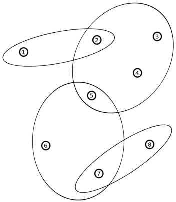
Graphs and hypergraphs

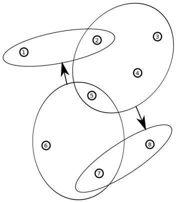
Examples of a undirected graph and directed graph





Examples of a undirected hypergraph and directed hypergraph





Matrices are graphs

A graph $G = (V, E)$ is a set of vertices V and a set of edges E.

- we can associate a weight w_{ij} for each edge $(i, j) \in E$.

Matrices are graphs

A graph $G = (V, E)$ is a set of vertices V and a set of edges E.

- we can associate a weight w_{ij} for each edge $(i, j) \in E$.
- the weights w_{ij} correspond to a (typically sparse) matrix $\mathbf{W}$

Matrices are graphs

A graph $G = (V, E)$ is a set of vertices V and a set of edges E.

- we can associate a weight w_{ij} for each edge $(i, j) \in E$.
- the weights w_{ij} correspond to a (typically sparse) matrix $\mathbf{W}$
- the rows and columns of $\mathbf{W}$ correspond to vertices and its entries to edges

Matrices are graphs

A graph $G = (V, E)$ is a set of vertices V and a set of edges E.

- we can associate a weight w_{ij} for each edge $(i, j) \in E$.
- the weights w_{ij} correspond to a (typically sparse) matrix $\mathbf{W}$
- the rows and columns of $\mathbf{W}$ correspond to vertices and its entries to edges
- $\mathbf{W}$ is symmetric if G is undirected

Matrices are graphs

A graph $G = (V, E)$ is a set of vertices V and a set of edges E.

- we can associate a weight w_{ij} for each edge $(i, j) \in E$.
- the weights w_{ij} correspond to a (typically sparse) matrix $\mathbf{W}$
- the rows and columns of $\mathbf{W}$ correspond to vertices and its entries to edges
- $\mathbf{W}$ is symmetric if G is undirected
- similarly we can represent any matrix as a graph with the connectivity of the graph corresponds to the sparsity of the matrix

Its useful to connect matrices with graphs

Graphs give a natural visualization for

- any mesh

Matrices allow for numerical computation on graphs

Its useful to connect matrices with graphs

Graphs give a natural visualization for

- any mesh
- the interaction list in a molecular dynamics simulation (pairlist)

Matrices allow for numerical computation on graphs

Its useful to connect matrices with graphs

Graphs give a natural visualization for

- any mesh
- the interaction list in a molecular dynamics simulation (pairlist)

Matrices allow for numerical computation on graphs

- iterative numerical solvers, often compute

$$\mathbf{x}^k = \mathbf{A} \cdot \mathbf{x}^{k-1}$$

Its useful to connect matrices with graphs

Graphs give a natural visualization for

- any mesh
- the interaction list in a molecular dynamics simulation (pairlist)

Matrices allow for numerical computation on graphs

- iterative numerical solvers, often compute

$$\mathbf{x}^k = \mathbf{A} \cdot \mathbf{x}^{k-1}$$

- direct particle methods may be written in above form (for certain definition of $\cdot$), where $\mathbf{x}$ are particles and $\mathbf{A}$ corresponds to forces

Semirings

We typically work with the semiring $c + a \cdot b$, but we could employ the tropical semiring $\min(c, a + b)$

- let $\mathbf{y} = y \oplus \mathbf{A} \odot \mathbf{x}$ denote matrix vector multiplication on the tropical semiring, so

$$y_i = \min(y_i, \min_k(A_{ik} + x_k))$$

Semirings

We typically work with the semiring $c + a \cdot b$, but we could employ the tropical semiring $\min(c, a + b)$

- let $\mathbf{y} = y \oplus \mathbf{A} \odot \mathbf{x}$ denote matrix vector multiplication on the tropical semiring, so

$$y_i = \min(y_i, \min_k(A_{ik} + x_k))$$

- $\mathbf{x}^k = \mathbf{A} \odot \mathbf{x}^{k-1}$ gives the shortest paths $\mathbf{x}^n$ between one vertex and the rest (Ford-Fulkerson algorithm)

Semirings

We typically work with the semiring $c + a \cdot b$, but we could employ the tropical semiring $\min(c, a + b)$

- let $\mathbf{y} = y \oplus \mathbf{A} \odot \mathbf{x}$ denote matrix vector multiplication on the tropical semiring, so

$$y_i = \min(y_i, \min_k(A_{ik} + x_k))$$

- $\mathbf{x}^k = \mathbf{A} \odot \mathbf{x}^{k-1}$ gives the shortest paths $\mathbf{x}^n$ between one vertex and the rest (Ford-Fulkerson algorithm)
- the closure of a matrix $\mathbf{A}$ is $\mathbf{A}^* = \mathbf{I} + \mathbf{A} + \mathbf{A}^2 \ldots$, for a numerical $\mathbf{A}$ under the $(+, \cdot)$ semiring, it can be computed by Gaussian Elimination $\mathbf{A}^* = (\mathbf{I} - \mathbf{A})^{-1}$

Semirings

We typically work with the semiring $c + a \cdot b$, but we could employ the tropical semiring $\min(c, a + b)$

- let $\mathbf{y} = y \oplus \mathbf{A} \odot \mathbf{x}$ denote matrix vector multiplication on the tropical semiring, so

$$y_i = \min(y_i, \min_k(A_{ik} + x_k))$$

- $\mathbf{x}^k = \mathbf{A} \odot \mathbf{x}^{k-1}$ gives the shortest paths $\mathbf{x}^n$ between one vertex and the rest (Ford-Fulkerson algorithm)
- the closure of a matrix $\mathbf{A}$ is $\mathbf{A}^* = \mathbf{I} + \mathbf{A} + \mathbf{A}^2 \ldots$, for a numerical $\mathbf{A}$ under the $(+, \cdot)$ semiring, it can be computed by Gaussian Elimination $\mathbf{A}^* = (\mathbf{I} - \mathbf{A})^{-1}$
- the closure of a matrix $\mathbf{B}$ corresponding to graph G on the tropical semiring $\mathbf{B}^* = \mathbf{I} \oplus \mathbf{B} \oplus \mathbf{B}^2 \ldots$ gives all shortest paths in G

Tensors are hypergraphs

Consider a $n \times n \times m \times m$ tensor

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

which is symmetric in *ab* and *ij*

- represent it as a directed hypergraph $H = (V, E)$ with edges of the form $(\{a, b\}, \{i, j\}) \in E$

Tensors are hypergraphs

Consider a $n \times n \times m \times m$ tensor

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

which is symmetric in ab and ij

- represent it as a directed hypergraph $H = (V, E)$ with edges of the form $(\{a, b\}, \{i, j\}) \in E$
- H automatically expresses the symmetry of $\mathbf{T}$ since sets are invariant under permutation of elements

Tensors are hypergraphs

Consider a $n \times n \times m \times m$ tensor

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

which is symmetric in ab and ij

- represent it as a directed hypergraph $H = (V, E)$ with edges of the form $(\{a, b\}, \{i, j\}) \in E$
- H automatically expresses the symmetry of $\mathbf{T}$ since sets are invariant under permutation of elements
- If we can divide V into two disjoint subsets $V_1 \cup V_2 = V$ (occupied and unoccupied orbitals), where all edges go from V_1 to V_2, H is a bipartite hypergraph

Tensors are hypergraphs

Consider a $n \times n \times m \times m$ tensor

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

which is symmetric in ab and ij

- represent it as a directed hypergraph $H = (V, E)$ with edges of the form $(\{a, b\}, \{i, j\}) \in E$
- H automatically expresses the symmetry of $\mathbf{T}$ since sets are invariant under permutation of elements
- If we can divide V into two disjoint subsets $V_1 \cup V_2 = V$ (occupied and unoccupied orbitals), where all edges go from V_1 to V_2, H is a bipartite hypergraph
- We can represent any fully-symmetric tensor of dimension d as a hypergraph where all edges have cardinality d

Coupled Cluster as a hypergraph computation

Coupled Cluster iteratively refines the bipartite hypergraph H corresponding to $\mathbf{T}$

- the integrals $\mathbf{V}$ may also be interpreted as hyperedges in H, but not bipartite

Coupled Cluster as a hypergraph computation

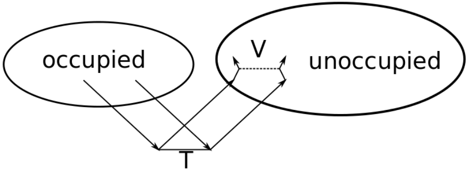
Coupled Cluster iteratively refines the bipartite hypergraph H corresponding to $\mathbf{T}$

- the integrals $\mathbf{V}$ may also be interpreted as hyperedges in H, but not bipartite
- each diagram is a contribution to a path through $\mathbf{V}$ of the form $\mathbf{T}$

Coupled Cluster as a hypergraph computation

Coupled Cluster iteratively refines the bipartite hypergraph H corresponding to $\mathbf{T}$

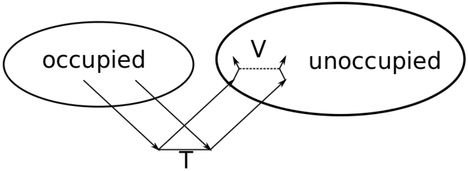
- the integrals $\mathbf{V}$ may also be interpreted as hyperedges in H, but not bipartite
- each diagram is a contribution to a path through $\mathbf{V}$ of the form $\mathbf{T}$
- for instance $Z^{ab}_{ij} = \sum_{cd} V^{ab}_{cd} T^{cd}_{ij}$ may be visualized as



Coupled Cluster as a hypergraph computation

Coupled Cluster iteratively refines the bipartite hypergraph H corresponding to **T**

- the integrals **V** may also be interpreted as hyperedges in H, but not bipartite
- each diagram is a contribution to a path through **V** of the form **T**
- for instance $Z_{ij}^{ab} = \sum_{cd} V_{cd}^{ab} T_{ij}^{cd}$ may be visualized as



- Speculation: switch semirings and have CC compute shortest paths in a hypergraph

Symmetric matrix times vector

Now lets consider a special case of semirings: commutative rings

- Let **b** be a vector of length n

Symmetric matrix times vector

Now lets consider a special case of semirings: commutative rings

- Let **b** be a vector of length n
- Let **A** be a n-by-n symmetric matrix with elements

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

Symmetric matrix times vector

Now lets consider a special case of semirings: commutative rings

- Let **b** be a vector of length n
- Let **A** be a n-by-n symmetric matrix with elements

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

- Typically, we say the symmetry of **A** is broken and compute

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

Symmetric matrix times vector

Now lets consider a special case of semirings: commutative rings

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

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

- Typically, we say the symmetry of $\mathbf{A}$ is broken and compute

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

- If $\cdot$ is an operator on a ring, we can use half the number of multiplications

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

Symmetrized product

We can apply a similar trick for the symmetrized outer product

- Let **a** and **b** be vectors of length n
- Compute symmetric matrix **A**

$$\mathbf{C} = \mathbf{a} \cdot \mathbf{b}^T + \mathbf{b} \cdot \mathbf{a}^T$$

$$C_{i \leq j} = a_i \cdot b_j + a_j \cdot b_i$$

- If $\cdot$ is an operator on a commutative ring, we can use half the multiplications,

$$C_{i \leq j} = (a_i + a_j) \cdot (b_i + b_j) - a_i \cdot b_i - a_j \cdot b_j.$$

A commutative ring of symmetric matrices

Given n-by-n symmetric matrices $\mathbf{A}, \mathbf{B}$ define commutative ring $\otimes$

$$\mathbf{A} \otimes \mathbf{B} = \mathbf{A} \cdot \mathbf{B} + \mathbf{B} \cdot \mathbf{A}$$

- note that the product is still symmetric, unlike $\mathbf{A} \cdot \mathbf{B}$

A commutative ring of symmetric matrices

Given n-by-n symmetric matrices $\mathbf{A}, \mathbf{B}$ define commutative ring $\otimes$

$$\mathbf{A} \otimes \mathbf{B} = \mathbf{A} \cdot \mathbf{B} + \mathbf{B} \cdot \mathbf{A}$$

- note that the product is still symmetric, unlike $\mathbf{A} \cdot \mathbf{B}$
- the operator $\otimes$ may be applied using $n^3/3! = n^3/6$ multiplications

$$w_i = \sum_{k=1}^{n} A_{ik} \quad x_i = \sum_{k=1}^{n} B_{ik} \quad y_i = \sum_{k=1}^{n} A_{ik} \cdot B_{ik}$$

$$Z_{i \le j \le k} = (A_{ij} + A_{ik} + A_{jk}) \cdot (B_{ij} + B_{ik} + B_{jk})$$

$$C_{i \le j} = C_{i \le j} + \sum_{k=1}^{n} Z_{ijk} - n \cdot A_{ij} \cdot B_{ij} - y_i - y_j - w_i \cdot B_{ij} - A_{ij} \cdot x_j$$

General fast symmetric tensor contractions

Given *fully* symmetric **A**, **B**, and **C**, compute $\mathbf{C} = A \otimes B$

$$C_{i_1 \ldots i_{s+t}} = \sum_{((j_1 \ldots j_s),(l_1 \ldots l_t)) \in \chi_s(i_1 \ldots i_{s+t})} \left(\sum_{k_1 \ldots k_v} A^{k_1 \ldots k_v}_{j_1 \ldots j_s} \cdot B^{l_1 \ldots l_t}_{k_1 \ldots k_v} \right).$$

Typically computed by (implicitly) forming partially-symmetric $\bar{\mathbf{C}}$

$$\bar{C}^{l_1 \ldots l_t}_{j_1 \ldots j_s} = \sum_{k_1 \ldots k_v} A^{k_1 \ldots k_v}_{j_1 \ldots j_s} \cdot B^{l_1 \ldots l_t}_{k_1 \ldots k_v}.$$

This requires $\frac{n^{s+t+v}}{s!t!v!}$ multiplications, via fully symmetric intermediates it becomes,

$$\binom{n}{s+t+v} \approx \frac{n^{s+t+v}}{(s+t+v)!}$$

General symmetric contraction algorithm

Compute $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$, using the notation $(j_1 \dots j_s, k_1 \dots k_t) \in \{i_1 \dots i_{s+t+v}\}$ to denote a partition into two disjoint sets:

$$Z_{i_1 \leq \dots i_{s+t+v}} = \sum_{(j_1 \dots j_s, k_1 \dots k_v) \in \{i_1 \dots i_{s+t+v}\}} A^{k_1 \dots k_v}_{j_1 \dots j_s} \cdot \sum_{(l_1 \dots l_t, k_1 \dots k_v) \in \{i_1 \dots i_{s+t+v}\}} B^{l_1 \dots l_t}_{k_1 \dots k_v}$$

$$W_{i_1 \leq \dots i_{s+t+v-1}} = \sum_{(j_1 \dots j_s, k_1 \dots k_v) \in \{i_1 \dots i_{s+t+v-1}\}} A^{k_1 \dots k_v}_{j_1 \dots j_s} \cdot \sum_{(l_1 \dots l_t, k_1 \dots k_v) \in \{i_1 \dots i_{s+t+v-1}\}} B^{l_1 \dots l_t}_{k_1 \dots k_v}$$

$$V_{i_1 \leq \dots i_{s+t+v-1}} = \sum_{(j_1 \dots j_s, k_1 \dots k_{v-1}) \in \{i_1 \dots i_{s+t+v-1}\}} \sum_{k_v} A^{k_1 \dots k_v}_{j_1 \dots j_s}$$
$$\cdot \sum_{(l_1 \dots l_t, k_1 \dots k_{v-1}) \in \{i_1 \dots i_{s+t+v-1}\}} \sum_{k_v} B^{l_1 \dots l_t}_{k_1 \dots k_v}$$

$$C_{i_1 \dots i_{s+t}} = \sum_{k_1 \dots k_v} Z_{i_1 \dots i_{s+t}, k_1, \dots k_v} - n \cdot \sum_{k_1 \dots k_{v-1}} W_{i_1 \dots i_{s+t}, k_1, \dots k_{v-1}}$$
$$- \sum_{k_1 \dots k_{v-1}} V_{i_1 \dots, i_{s+t}, k_1, \dots k_{v-1}}$$

Any tensor is a fully symmetric tensor

Realizing that a vector is a symmetric tensor, we may express any tensor as a nested symmetric tensor

- A nonsymmetric matrix A_{ij} is a vector of vectors $\bar{a}$ where each element $\bar{\bar{a}} = \bar{a}_i$ is a vector with $\bar{\bar{a}}_j = A_{ij}$

Any tensor is a fully symmetric tensor

Realizing that a vector is a symmetric tensor, we may express any tensor as a nested symmetric tensor

- A nonsymmetric matrix A_{ij} is a vector of vectors $\bar{a}$ where each element $\bar{\bar{a}} = \bar{a}_i$ is a vector with $\bar{\bar{a}}_j = A_{ij}$
- The partially symmetric matrix C^{ab}_{ij} may then be written as a two-level nest of symmetric matrices

Any tensor is a fully symmetric tensor

Realizing that a vector is a symmetric tensor, we may express any tensor as a nested symmetric tensor

- A nonsymmetric matrix A_{ij} is a vector of vectors $\bar{a}$ where each element $\bar{\bar{a}} = \bar{a}_i$ is a vector with $\bar{\bar{a}}_j = A_{ij}$
- The partially symmetric matrix C_{ij}^{ab} may then be written as a two-level nest of symmetric matrices
- Therefore, we can compute a contraction like

$$C_{abij} = P(a, b) \sum_{ck} A_{acik} \cdot B_{cbkj}$$

where **A** is symmetric in ac, **B** is symmetric in cb in $n^6/6$ operations

Any tensor is a fully symmetric tensor

Realizing that a vector is a symmetric tensor, we may express any tensor as a nested symmetric tensor

- A nonsymmetric matrix A_{ij} is a vector of vectors $\bar{a}$ where each element $\bar{\bar{a}} = \bar{a}_i$ is a vector with $\bar{\bar{a}}_j = A_{ij}$
- The partially symmetric matrix C^{ab}_{ij} may then be written as a two-level nest of symmetric matrices
- Therefore, we can compute a contraction like

$$C_{abij} = P(a,b) \sum_{ck} A_{acik} \cdot B_{cbkj}$$

  where **A** is symmetric in ac, **B** is symmetric in cb in $n^6/6$ operations
- Unfortunately contractions of the above form do not exist in Coupled Cluster theory and cannot be written using raised and lowered index notation

Limited application of fast symmetric contraction to Coupled Cluster

For some CC contractions, we can at least gain a factor of two

- Consider the contraction

$$Z_{ij}^{ab} = P(i,j)P(a,b)\sum_{klcd} T_{ik}^{ac} V_{cd}^{kl} T_{lj}^{db}$$

Limited application of fast symmetric contraction to Coupled Cluster

For some CC contractions, we can at least gain a factor of two

- Consider the contraction

$$Z_{ij}^{ab} = P(i,j)P(a,b)\sum_{klcd} T_{ik}^{ac} V_{cd}^{kl} T_{lj}^{db}$$

- which may be done by forming a nonsymmetric(!) W_{il}^{ad}

$$W_{il}^{ad} = \sum_{ck} T_{ik}^{ac} V_{cd}^{kl}$$

Limited application of fast symmetric contraction to Coupled Cluster

For some CC contractions, we can at least gain a factor of two

- Consider the contraction

$$Z_{ij}^{ab} = P(i,j)P(a,b)\sum_{klcd} T_{ik}^{ac} V_{cd}^{kl} T_{lj}^{db}$$

- which may be done by forming a nonsymmetric(!) W_{il}^{ad}

$$W_{il}^{ad} = \sum_{ck} T_{ik}^{ac} V_{cd}^{kl}$$

- Defining vector $\bar{W}^a$ with elements $W_{il}^{ad} \in \bar{W}^a$ for all d, i, l, and similarly vector $\bar{V}_c$ and symmetric matrix $\bar{T}^{ac}$, we may compute $\bar{\mathbf{W}} = \bar{\mathbf{T}} \otimes \bar{\mathbf{V}}$,

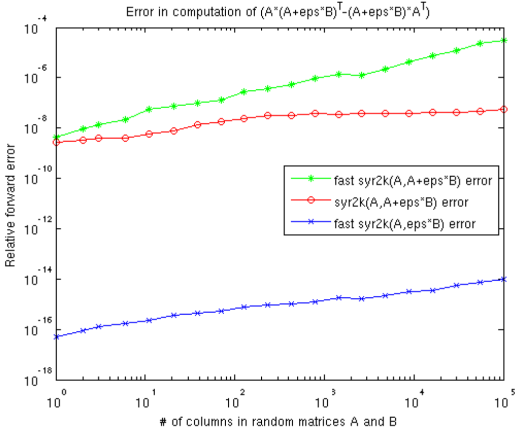
$$\bar{W}^a = \sum_c \bar{T}^{ac} \cdot \bar{V}_c$$

using half the multiplications, resulting in $n^2/2$ calls to subcontraction

$$\tilde{W}_{il}^{d} = \sum_k \tilde{T}_{ik} \cdot \tilde{V}_d^{kl}$$

Disclaimer: numerical characteristics

The fast contraction algorithms have different numerical characteristics in floating-point precision



Rewind

Stepping back from hypergraphs and rings...

- Cyclops Tensor Framework is available at `ctf.cs.berkeley.edu` and `github.com/solomonik/ctf`

Rewind

Stepping back from hypergraphs and rings...

- Cyclops Tensor Framework is available at `ctf.cs.berkeley.edu` and `github.com/solomonik/ctf`
- CTF v1.0 was released in December and the master branch is even faster

Rewind

Stepping back from hypergraphs and rings...

- Cyclops Tensor Framework is available at `ctf.cs.berkeley.edu` and `github.com/solomonik/ctf`
- CTF v1.0 was released in December and the master branch is even faster
- Looking for collaborative effort on CCSD(T) and CCSDT(Q) methods

Rewind

Stepping back from hypergraphs and rings...

- Cyclops Tensor Framework is available at `ctf.cs.berkeley.edu` and `github.com/solomonik/ctf`
- CTF v1.0 was released in December and the master branch is even faster
- Looking for collaborative effort on CCSD(T) and CCSDT(Q) methods
- Hopefully the hypergraph and fast contraction algorithms lead to some insight towards better understanding of CC

Collaborators and acknowledgements

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

Backup slides