Efficient algorithms for symmetric tensor contractions

Edgar Solomonik

[1] Department of EECS, UC Berkeley

Oct 22, 2013

Motivation

The goal is to design a parallel numerical library for tensor algebra

- motivation comes from high-accuracy quantum chemistry applications
- desire to support user-defined tensor types
- algebraic rings allow definition of tensor algebra for any type

This talk will cover two major topics

- Part I: Symmetric matrix computations, tensor contraction preliminaries
- Part II: Cyclops Tensor Framework

Outline

Care for type definitions (algebraic ring)

We consider vectors and matrices on any commutative ring $\varrho = (R, +, \times)$

- The following are equivalent in our notation $a \times b = a \cdot b = ab$

Care for type definitions (algebraic ring)

We consider vectors and matrices on any commutative ring $\varrho = (R, +, \times)$

- The following are equivalent in our notation $a \times b = a \cdot b = ab$
- Rings ensure associativity of $+$ and distributivity of $\times$

Care for type definitions (algebraic ring)

We consider vectors and matrices on any commutative ring $\varrho = (R, +, \times)$

- The following are equivalent in our notation $a \times b = a \cdot b = ab$
- Rings ensure associativity of $+$ and distributivity of $\times$
- Commutativity of the ring ensures $\forall a, b \in R : \quad a \cdot b = b \cdot a$

Care for type definitions (algebraic ring)

We consider vectors and matrices on any commutative ring $\varrho = (R, +, \times)$

- The following are equivalent in our notation $a \times b = a \cdot b = ab$
- Rings ensure associativity of $+$ and distributivity of $\times$
- Commutativity of the ring ensures $\forall a, b \in R : \quad a \cdot b = b \cdot a$
- Floating point arithmetic does not define a ring due to non-associativity of rounding

Care for type definitions (algebraic ring)

We consider vectors and matrices on any commutative ring $\varrho = (R, +, \times)$

- The following are equivalent in our notation $a \times b = a \cdot b = ab$
- Rings ensure associativity of $+$ and distributivity of $\times$
- Commutativity of the ring ensures $\forall a, b \in R : \quad a \cdot b = b \cdot a$
- Floating point arithmetic does not define a ring due to non-associativity of rounding
- Multiplication of n-by-n matrices of reals defines a non-commutative ring with $R = \mathbb{R}^{n \times n}$

Care for type definitions (algebraic ring)

We consider vectors and matrices on any commutative ring $\varrho = (R, +, \times)$

- The following are equivalent in our notation $a \times b = a \cdot b = ab$
- Rings ensure associativity of $+$ and distributivity of $\times$
- Commutativity of the ring ensures $\forall a, b \in R : \quad a \cdot b = b \cdot a$
- Floating point arithmetic does not define a ring due to non-associativity of rounding
- Multiplication of n-by-n matrices of reals defines a non-commutative ring with $R = \mathbb{R}^{n \times n}$
- We denote the computational cost of addition $(+)$ on ring ϱ as α_ϱ and of multiplication $(\times)$ on ϱ as β_ϱ

Symmetric matrix times vector

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

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

- We can multiply matrix **A** by **b**,

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

$$c_i = \sum_{j=1}^{n} A_{ij} b_j \qquad (1)$$

this corresponds to BLAS routine `symv`

$$T_{\mathrm{symv}}(\varrho, n) = \alpha_\varrho \cdot n^2 + \beta_\varrho \cdot n^2$$

Fast symmetric matrix times vector

Evidently, we can perform `symv` using fewer element-wise multiplications,

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

- This transformation requires an additive inverse on ϱ
- $A_{ij} \cdot (b_i + b_j)$ is symmetric, and can be computed with $n(n+1)/2$ element-wise multiplications
- $\left(\sum_{j=1}^{n} A_{ij} \right) b_i$ may be computed with n multiplications
- The total cost of Equation 2 is

$$T'_{\text{symv}}(\varrho, n) = \alpha_\varrho \cdot \frac{5}{2} n^2 + \beta_\varrho \cdot \frac{1}{2} n^2$$

- This formulation is cheaper when $\beta_\varrho > 3\alpha_\varrho$

Symmetric matrix times nonsymmetric matrix

- Now consider the multiplication of **A** by another n-by-K matrix **B**

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$$

$$C_{ik} = \sum_{j=1}^{n} A_{ij} B_{jk} \tag{2}$$

- This operation corresponds to the BLAS routine `symm`
- We can compute this with K matrix vector products for a cost

$$T_{\text{symm}}(\varrho, n, K) = K \cdot T_{\text{symv}}(\varrho, n) = \alpha_\varrho \cdot Kn^2 + \beta_\varrho \cdot Kn^2.$$

Fast symmetric matrix times nonsymmetric matrix

- Using the new symmetric-vector product algorithm, we have the reformulation

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

- The cost of $\sum_{j=1}^{n} A_{ij}$ is amortized over the K vectors
- The total new cost is given to leading order by

$$T'_{\text{symm}}(\varrho, n, K) = \alpha_\varrho \cdot \frac{3}{2} K n^2 + \beta_\varrho \cdot \frac{1}{2} K n^2.$$

- The reformulation is cheaper when $n, K \gg 1$ and $\alpha_\varrho < \beta_\varrho$

Symmetric rank-2 update

Consider a rank-2 outer product of vectors **a** and **b** of length n into symmetric matrix **C**

$$\mathbf{C} = \mathbf{a} \cdot \mathbf{b}^T + \mathbf{b} \cdot \mathbf{a}^T$$
$$C_{i \leq j} = a_i \cdot b_j + a_j \cdot b_i. \tag{3}$$

- For floating point arithmetic, this is the BLAS routine `syr2`
- The routine may be computed from the intermediate $Z_{ij} = a_i \cdot b_j$ with the cost

$$T_{\text{syr2}}(\varrho, n) = \alpha_\varrho \cdot n^2 + \beta_\varrho \cdot n^2.$$

Fast symmetric rank-2 update

We may compute the rank-2 update via a symmetric intermediate quantity

$$C_{i\leq j} = (a_i + a_j)\cdot(b_i + b_j) - a_i \cdot b_i - a_j \cdot b_j. \qquad (4)$$

- We can compute the symmetric $Z_{i\leq j} = (a_i + a_j)\cdot(b_i + b_j)$ in $n(n+1)/2$ multiplications
- The total cost is then given to leading order by

$$T'_{\text{syr2}}(\varrho, n) = \alpha_\varrho \cdot \frac{5}{2}n^2 + \beta_\varrho \cdot \frac{1}{2}n^2.$$

- $T'_{\text{syr2}}(\varrho, n) < T_{\text{syr2}}(\varrho, n)$ when $\beta_\varrho > 3\alpha_\varrho$

Symmetric rank-$2K$ update

Consider a rank-K symmetric update of matrix $\mathbf{A}$ and $\mathbf{B}$ of size n-by-K,

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B}^T + \mathbf{B} \cdot \mathbf{A}^T$$

$$C_{i\le j} = \sum_{k=1}^{K} (A_{ik} \cdot B_{jk} + A_{jk} \cdot B_{ik}) \,. \qquad (5)$$

- For floating point arithmetic, this is the BLAS routine `syr2k`
- Symmetric matrix $\mathbf{C}$ may be computed via K rank-2 updates for a cost

$$T_{\text{syr2K}}(\varrho, n, K) = K \cdot T_{\text{syr2}}(\varrho, n) = \alpha_\varrho \cdot Kn^2 + \beta_\varrho \cdot Kn^2.$$

Fast symmetric rank-$2K$ update

Using a symmetric intermediate we can compute the rank-$2K$ update using fewer ring multiplications

$$C_{i\leq j} = \sum_{k=1}^{K} \left((A_{ik} + A_{jk}) \cdot (B_{ik} + B_{jk}) - A_{ik} \cdot B_{ik} - A_{jk} \cdot B_{jk} \right).$$

- Here we amortize the cost of the last two terms due to having $K \gg 1$ rank-2 updates
- The cost requires half the leading order multiplications of the old algorithm

$$T'_{\text{syr2K}}(\varrho, n, K) = \alpha_\varrho \cdot \frac{3}{2} K n^2 + \beta_\varrho \cdot \frac{1}{2} K n^2.$$

- The reformulation is cheaper when $n, K \gg 1$ and $\alpha_\varrho < \beta_\varrho$

Symmetric-matrix by symmetric-matrix multiplication

Given symmetric matrices $\mathbf{A}, \mathbf{B}$ of dimension n with elements on commutative ring $\varrho = (R, +, \times)$, we seek to compute

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

$$C_{i \leq j} = \sum_{k=1}^{n} (A_{ik} \cdot B_{jk} + A_{jk} \cdot B_{ik}) \,. \tag{6}$$

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

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

This routine is a bit more extravagant and is not in the BLAS

Fast symmetric-matrix by symmetric-matrix multiplication

We can combine the ideas from the fast routines for `symv` and `syrk` by forming an intermediate Z_{ijk} which is symmetric in all three indices,

$$
\begin{aligned}
w_i =& \sum_{k=1}^{n} A_{ik} \quad x_i = \sum_{k=1}^{n} B_{ik} \quad y_i = \sum_{k=1}^{n} A_{ik} \cdot B_{ik} \\
Z_{i\le j\le k} =& (A_{ij} + A_{ik} + A_{jk}) \cdot (B_{ij} + B_{ik} + B_{jk}) \\
C_{i\le j} =& \sum_{k=1}^{n} Z_{ijk} - n \cdot A_{ij} \cdot B_{ij} - y_i - y_j - w_i \cdot B_{ij} - A_{ij} \cdot x_j \qquad (7)
\end{aligned}
$$

The reformulation requires a sixth of the multiplications to leading order,

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

The ring of symmetric matrices

The set of real symmetric matrices $\mathbb{S} \subset \mathbb{R}^{n \times n}$ is not closed under normal matrix multiplications, because the product of two symmetric matrices is not generally symmetric. However, we can define a ring $\varsigma = (\mathbb{S}, +, \odot)$ with $+$ being regular addition $\odot$ defined as

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

$$C_{i \leq j} = \sum_{k=1}^{n} \left(A_{ik} B_{kj} + A_{jk} B_{ki} \right).$$

This ring anticommutator multiplication preserves symmetry and is commutative since $\mathbf{A} \odot \mathbf{B} = \mathbf{B} \odot \mathbf{A}$.

Assuming the cost of real adds and multiplications is the same (as it is for the floating point representation of reals), we define the element-wise costs of ring ς as $\alpha_\varsigma = n^2$ for each addition and $\beta_\varsigma = 2n^3$ for each multiplication by the symmetric operator $\odot$.

Symmetric matrix multiplication on the ring ς

Define symmetric m-by-m matrices $\mathcal{A}$ and $\mathcal{B}$ whose elements are elements of the ring ς. $\mathcal{A}$ and $\mathcal{B}$ are 4D tensors of reals which are partially symmetric in two pairs of indices. Consider,

$$\mathcal{C} = \mathcal{A} \odot \mathcal{B} + \mathcal{B} \odot \mathcal{A}$$

$$\mathbf{C}^{p\le q} = \sum_{r=1}^{m} (\mathbf{A}^{pr} \odot \mathbf{B}^{rq} + \mathbf{A}^{qr} \odot \mathbf{B}^{rp})$$

$$= \sum_{r=1}^{m} (\mathbf{A}^{pr} \cdot \mathbf{B}^{rq} + \mathbf{B}^{pr} \cdot \mathbf{A}^{rq} + \mathbf{A}^{qr} \cdot \mathbf{B}^{rp} + \mathbf{B}^{qr} \cdot \mathbf{A}^{rp})$$

$$C^{p\le q}_{i\le j} = \sum_{r=1}^{m} \sum_{k=1}^{n} \left(A^{pr}_{ik} \cdot B^{rq}_{kj} + A^{pr}_{jk} \cdot B^{rq}_{ki} + A^{qr}_{ik} \cdot B^{rp}_{kj} + A^{qr}_{jk} \cdot B^{rp}_{ki} \right)$$

Using the new symmetric-matrix by symmetric-matrix multiplication algorithm lowers the leading cost of this tensor contraction from $2m^3n^3$ to $\frac{1}{3}m^3n^3$.

Tensor contractions in the real world

The fast symmetric 4D tensor contraction algorithm does not apply to any contractions needed by Coupled Cluster. One of the leading order terms in Coupled Cluster is similar for 4D tensors $\mathcal{T}$ and $\mathcal{V}$

$$\mathcal{W} = \mathcal{T} \cdot \mathcal{V} \cdot \mathcal{T}^T + \mathcal{T}^T \cdot \mathcal{V} \cdot \mathcal{T}$$

$$W_{ij}^{mn} = \sum_{klrs} T_{ik}^{mr} \cdot V_{kl}^{rs} \cdot T_{lj}^{sn} + T_{jk}^{mr} \cdot V_{kl}^{rs} \cdot T_{li}^{sn} + \ldots$$

Here we adapt the standard raised and lowered index tensor notation, where upper indices are contracted with lower indices and rewrite the above as

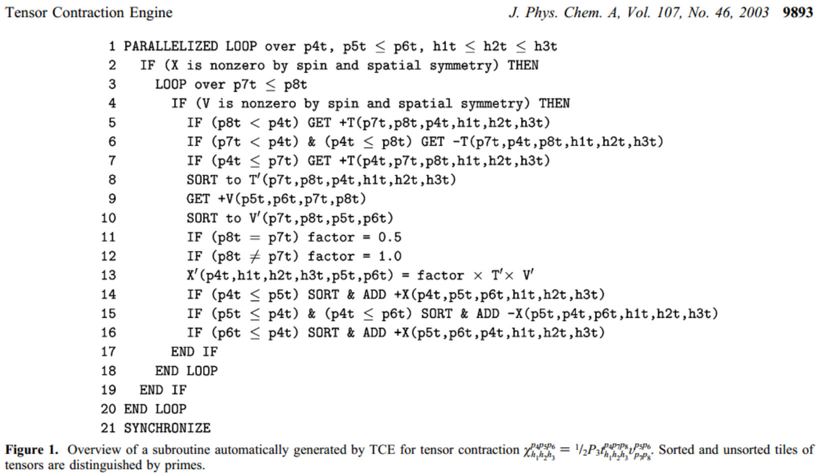
$$W_{ij}^{mn} = \sum_{klrs} T_{ik}^{mr} \cdot V_{rs}^{kl} \cdot T_{lj}^{sn} + T_{jk}^{mr} \cdot V_{rs}^{kl} \cdot T_{li}^{sn} + \ldots$$

NWChem

NWChem is a scientific package for electronic structure calculations

- Tensor Contraction Engine (TCE)
  - compiler technology which factorizes multi-term contractions
  - synthesizes Global Arrays code for each contraction
- Global Arrays
  - distributed memory PGAS runtime system
  - provides one-sided get and put operations
- Uses load-balancing to handle tensor symmetry efficiently

TCE



```
PARALLELIZED LOOP over p4t, $p5t \leq p6t$, $h1t \leq h2t \leq h3t$
  IF (X is nonzero by spin and spatial symmetry) THEN
    LOOP over $p7t \leq p8t$
      IF (V is nonzero by spin and spatial symmetry) THEN
        IF ($p8t < p4t$) GET +T(p7t,p8t,p4t,h1t,h2t,h3t)
        IF ($p7t < p4t$) & ($p4t \leq p8t$) GET -T(p7t,p4t,p8t,h1t,h2t,h3t)
        IF ($p4t \leq p7t$) GET +T(p4t,p7t,p8t,h1t,h2t,h3t)
        SORT to $T'$(p7t,p8t,p4t,h1t,h2t,h3t)
        GET +V(p5t,p6t,p7t,p8t)
        SORT to $V'$(p7t,p8t,p5t,p6t)
        IF ($p8t = p7t$) factor = 0.5
        IF ($p8t \neq p7t$) factor = 1.0
        $X'$(p4t,h1t,h2t,h3t,p5t,p6t) = factor $\times$ $T' \times V'$
        IF ($p4t \leq p5t$) SORT & ADD +X(p4t,p5t,p6t,h1t,h2t,h3t)
        IF ($p5t \leq p4t$) & ($p4t \leq p6t$) SORT & ADD -X(p5t,p4t,p6t,h1t,h2t,h3t)
        IF ($p6t \leq p4t$) SORT & ADD +X(p5t,p6t,p4t,h1t,h2t,h3t)
      END IF
    END LOOP
  END IF
END LOOP
SYNCHRONIZE
```

Figure 1. Overview of a subroutine automatically generated by TCE for tensor contraction $\chi^{p_4p_5p_6}_{h_1h_2h_3} = {}^1/_2 P_3 t^{p_4p_7p_8}_{h_1h_2h_3} v^{p_5p_6}_{p_7p_8}$. Sorted and unsorted tiles of tensors are distinguished by primes.

NWChem approach to contractions

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

Cyclops Tensor Framework (CTF)

Bulk synchronous distributed memory C++ MPI/OpenMP/BLAS-based library for tensor contractions

- does not require language or compiler support

Cyclops Tensor Framework (CTF)

Bulk synchronous distributed memory C++ MPI/OpenMP/BLAS-based library for tensor contractions

- does not require language or compiler support
- provides primitives for distributed memory tensor data

Cyclops Tensor Framework (CTF)

Bulk synchronous distributed memory C++ MPI/OpenMP/BLAS-based library for tensor contractions

- does not require language or compiler support
- provides primitives for distributed memory tensor data
- automatically supports symmetry in tensors

Cyclops Tensor Framework (CTF)

Bulk synchronous distributed memory C++ MPI/OpenMP/BLAS-based library for tensor contractions

- does not require language or compiler support
- provides primitives for distributed memory tensor data
- automatically supports symmetry in tensors
- uses internal data mapping and redistribution

Cyclops Tensor Framework (CTF)

Bulk synchronous distributed memory C++ MPI/OpenMP/BLAS-based library for tensor contractions

- does not require language or compiler support
- provides primitives for distributed memory tensor data
- automatically supports symmetry in tensors
- uses internal data mapping and redistribution
- exposes sparse remote data access and tensor slicing

Cyclops Tensor Framework (CTF)

Bulk synchronous distributed memory C++ MPI/OpenMP/BLAS-based library for tensor contractions

- does not require language or compiler support
- provides primitives for distributed memory tensor data
- automatically supports symmetry in tensors
- uses internal data mapping and redistribution
- exposes sparse remote data access and tensor slicing
- uses C++ templated interface which corresponds to Einstein notation

$$\mathtt{W}[\text{``}\mathtt{MnIj}\text{''}] = \mathtt{V}[\text{``}\mathtt{MnEf}\text{''}] * \mathtt{T}[\text{``}\mathtt{EfIj}\text{''}];$$

$$\mathtt{Z}[\text{``}\mathtt{AbIj}\text{''}] = \mathtt{W}[\text{``}\mathtt{MnIj}\text{''}] * \mathtt{T}[\text{``}\mathtt{AbMn}\text{''}];$$

Cyclops Tensor Framework (CTF)

Bulk synchronous distributed memory C++ MPI/OpenMP/BLAS-based library for tensor contractions

- does not require language or compiler support
- provides primitives for distributed memory tensor data
- automatically supports symmetry in tensors
- uses internal data mapping and redistribution
- exposes sparse remote data access and tensor slicing
- uses C++ templated interface which corresponds to Einstein notation

$$\texttt{W}[``\texttt{MnIj}"] = \texttt{V}[``\texttt{MnEf}"] * \texttt{T}[``\texttt{EfIj}"];$$

$$\texttt{Z}[``\texttt{AbIj}"] = \texttt{W}[``\texttt{MnIj}"] * \texttt{T}[``\texttt{AbMn}"];$$

- W, V, T, Z are tCTF_Tensor objects, $\texttt{W}[``\texttt{MnIj}"]$ is a tCTF_Idx_Tensor object

Cyclops Tensor Framework (CTF) approach to contractions

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

Multidimensional processor grids

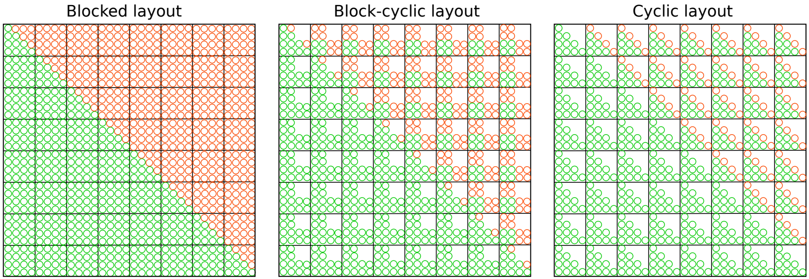
CTF supports tensors and processor grids of any dimension because mapping a symmetric tensor to a processor grid of the same dimension preserves symmetric structure with minimal virtualization and padding. Processor grids are defined by

- a base grid, obtained from the physical topology or from factorizing the number of processors
- folding all possible combinations of adjacent processor grid dimensions

Tensors are contracted on higher dimensional processor grids by

- mapping an index shared by two tensors in the contraction to different processor grid dimensions
- running a distributed matrix multiplication algorithm for each such 'mismatched' index
- replicating data along some processor dimensions 'a la 2.5D'

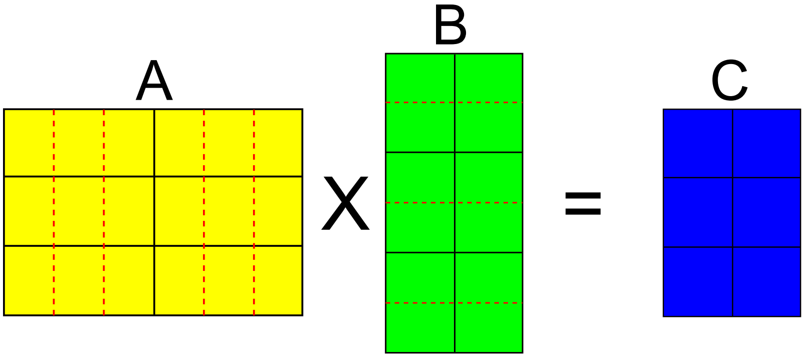
Blocked vs block-cyclic vs cyclic decompositions



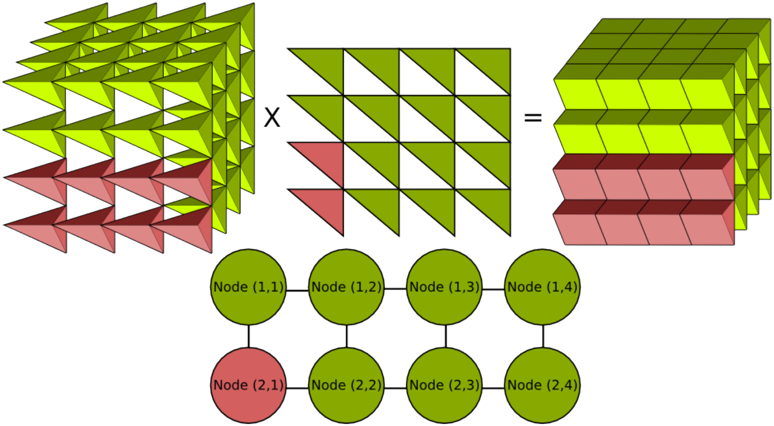
○ Green denotes fill (unique values) ○ Red denotes padding / load imbalance

Virtualization

Matrix multiply on 2x3 processor grid. Red lines represent virtualized part of processor grid. Elements assigned to blocks by cyclic phase.



3D tensor mapping



A simple data layout

Replicated, distributed nonsymmetric tensor (processor grid)
  of nonsymmetric tensors (virtual grid)
    of symmetric tensors (folded broken symmetries)
      of matrices (unfolded broken and folded preserved symmetries)

The layout typically changes for each tensor between each contraction.

Tensor redistribution

Our symmetric tensor data layout has a global ordering and a local ordering

- the local data is not in global order
- cannot compute local data index from global index
- cannot compute global data index from local index
- can iterate over local data and obtain global index
- can iterate over global data and obtain local index

Given these constraints, it is simplest to compute the global index of each piece of data and sort.

- interface: $A[\text{vector of indices}] = \text{vector of values}$

General data redistribution

We use an algorithm faster than sorting for redistribution

1. iterate over the local data and count where the data must be sent
2. communicate counts and compute prefix sums to obtain offsets
3. iterate over the local data **in global order** and bin it
4. exchange the data (MPI all to all v)
5. iterate over the new local data **in global order** and retrieve it from bins

This method is much faster, because it does not explicitly form and communicate keys for the data.

Threaded general redistribution

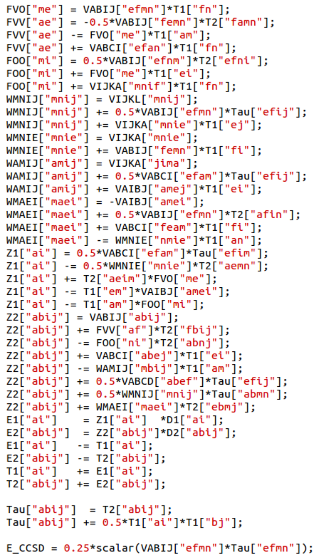
In order to hide memory latency and reduce integer operations it is imperative to thread the redistribution kernel

- prefix sums and counts are trivial to thread
- to thread the iterator over data, we must give each thread different global indices
- each thread moves the local data corresponding to a global index partition, preserving the ordering

Interface and code organization

- the CTF codebase is currently 30K+ lines of C++ code
- CTF provides functionality for general tensor contractions and summations, including a contraction domain-specific language (DSL)
- Aquarius is a quantum chemistry package being developed by Devin Matthews
  - uses CTF for parallel tensor contraction execution
  - provides a DSL for spin-integrated tensor contractions
  - gives implementations of CC methods including other necessary components (e.g. SCF)
- efforts are underway to also integrate CTF into the QChem package

CCSD code using our domain specific language



```
FVO["me"] = VABIJ["efmn"]*T1["fn"];
FVV["ae"] = -0.5*VABIJ["femn"]*T2["famn"];
FVV["ae"] -= FVO["me"]*T1["am"];
FVV["ae"] += VABCI["efan"]*T1["fn"];
FOO["mi"] = 0.5*VABIJ["efnm"]*T2["efni"];
FOO["mi"] += FVO["me"]*T1["ei"];
FOO["mi"] += VIJKA["mnif"]*T1["fn"];
WMNIJ["mnij"] = VIJKL["mnij"];
WMNIJ["mnij"] += 0.5*VABIJ["efmn"]*Tau["efij"];
WMNIJ["mnij"] += VIJKA["mnie"]*T1["ej"];
WMNIE["mnie"] = VIJKA["mnie"];
WMNIE["mnie"] += VABIJ["femn"]*T1["fi"];
WAMIJ["amij"] = VIJKA["jima"];
WAMIJ["amij"] += 0.5*VABCI["efam"]*Tau["efij"];
WAMIJ["amij"] += VAIBJ["amej"]*T1["ei"];
WMAEI["maei"] = -VAIBJ["amei"];
WMAEI["maei"] += 0.5*VABIJ["efmn"]*T2["afin"];
WMAEI["maei"] += VABCI["feam"]*T1["fi"];
WMAEI["maei"] -= WMNIE["nmie"]*T1["an"];
Z1["ai"] = 0.5*VABCI["efam"]*Tau["efim"];
Z1["ai"] -= 0.5*WMNIE["mnie"]*T2["aemn"];
Z1["ai"] += T2["aeim"]*FVO["me"];
Z1["ai"] -= T1["em"]*VAIBJ["amei"];
Z1["ai"] -= T1["am"]*FOO["mi"];
Z2["abij"] = VABIJ["abij"];
Z2["abij"] += FVV["af"]*T2["fbij"];
Z2["abij"] -= FOO["ni"]*T2["abnj"];
Z2["abij"] += VABCI["abej"]*T1["ei"];
Z2["abij"] -= WAMIJ["mbij"]*T1["am"];
Z2["abij"] += 0.5*VABCD["abef"]*Tau["efij"];
Z2["abij"] += 0.5*WMNIJ["mnij"]*Tau["abmn"];
Z2["abij"] += WMAEI["maei"]*T2["ebmj"];
E1["ai"]    = Z1["ai"]  *D1["ai"];
E2["abij"]  = Z2["abij"]*D2["abij"];
E1["ai"]   -= T1["ai"];
E2["abij"] -= T2["abij"];
T1["ai"]   += E1["ai"];
T2["abij"] += E2["abij"];

Tau["abij"]  = T2["abij"];
Tau["abij"] += 0.5*T1["ai"]*T1["bj"];

E_CCSD = 0.25*scalar(VABIJ["efmn"]*Tau["efmn"]);
```

Tensor slicing

CTF provides functionality to read/write blocks to tensors, commonly known as slicing. We can give a simple example of slicing for matrices in Matlab notation,

$$A[1:n/2, 1:n/2] = B[n/2+1:n, n/2+1:n]$$

In CTF the general interface is
*void slice(int * offsets, int * ends, double beta,*
    *tCTF_Tensor & A, int * offsets_A, int * ends_A, double alpha);*
Each tensor in CTF is defined on a CTF_World, which corresponds to a MPI communicator.

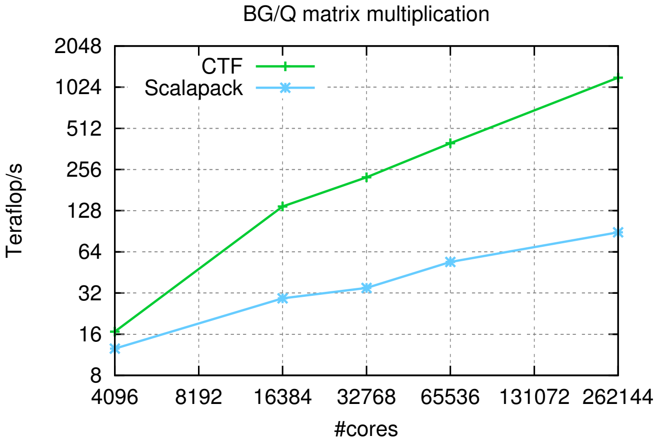
- CTF allows slicing of tensors between different worlds!
- Also plan to generalize slices to "permutation-slices"

Sequential and multi-threaded performance comparison

CCSD performance on a Xeon E5620, single threaded, Intel MKL. Entries are average time for one CCSD iteration, for the given number of virtual (n_v) and occupied (n_o) orbitals (electrons).

		$n_v = 110$	$n_v = 94$	$n_v = 71$
		$n_o = 5$	$n_o = 11$	$n_o = 23$
NWChem	1 thread	6.80 sec	16.8 sec	49.1 sec
CTF	1 thread	23.6 sec	32.5 sec	59.8 sec
NWChem	8 threads	5.21 sec	8.60 sec	18.1 sec
CTF	8 threads	9.12 sec	9.37 sec	18.5 sec

A simple tensor contraction



Complex matrix multiplication (ZGEMM) of 32K-by-32K matrices benefits greatly from topology-aware mapping on BG/Q.

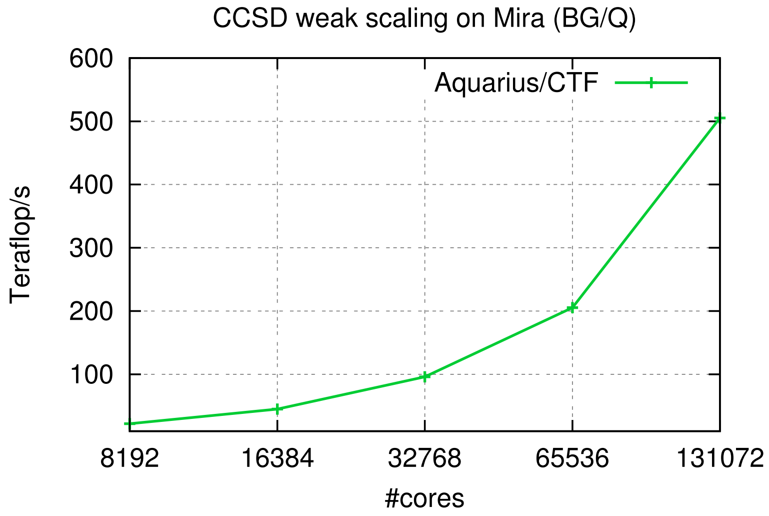
Comparison with NWChem on Cray XE6

CCSD iteration time on 64 nodes of Hopper:

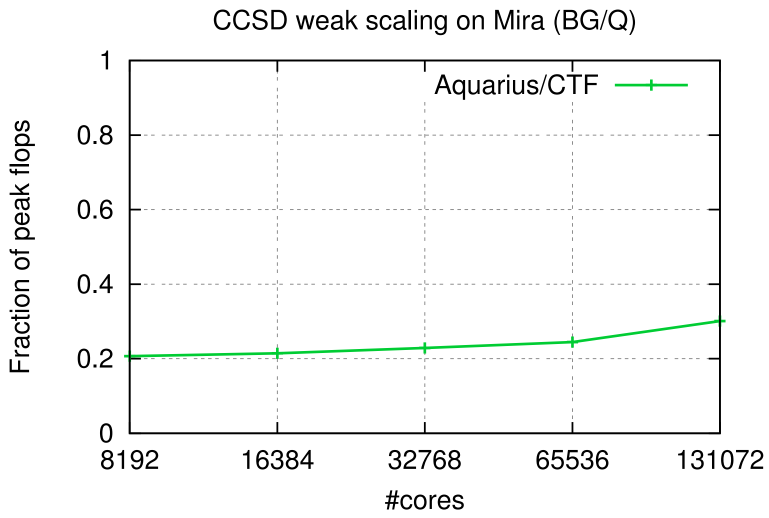
system	# electrons	# orbitals	CTF	NWChem
w5	25	205	14 sec	36 sec
w7	35	287	90 sec	178 sec
w9	45	369	127 sec	-
w12	60	492	336 sec	-

On 128 nodes, NWChem completed w9 in 223 sec, CTF in 73 sec.

Blue Gene/Q up to 1250 orbitals, 250 electrons



Coupled Cluster efficiency on Blue Gene/Q



Performance breakdown on BG/Q

Performance data for a CCSD iteration with 200 electrons and 1000 orbitals on 4096 nodes of Mira

4 processes per node, 16 threads per process

Total time: 18 mins

n_v-orbitals, n_o-electrons, p-processors, M-local memory size

kernel	% of time	complexity	architectural bounds
DGEMM	45%	$O(n_v^4 n_o^2/p)$	flops/mem bandwidth
broadcasts	20%	$O(n_v^4 n_o^2/p\sqrt{M})$	multicast bandwidth
prefix sum	10%	$O(p)$	allreduce bandwidth
data packing	7%	$O(n_v^2 n_o^2/p)$	integer ops
all-to-all-v	7%	$O(n_v^2 n_o^2/p)$	bisection bandwidth
tensor folding	4%	$O(n_v^2 n_o^2/p)$	memory bandwidth

Performance breakdown on Cray XE6

Performance data for a CCSD iteration with 100 electrons and 500 orbitals on 256 nodes of Hopper
4 processes per node, 6 threads per process
Total time: 9 mins
v-orbitals, *o*-electrons

kernel	% of time	complexity	architectural bounds
DGEMM	21% $\Downarrow$ 24%	$O(v^4o^2/p)$	flops/mem bandwidth
broadcasts	32% $\Uparrow$ 12%	$O(v^4o^2/p\sqrt{M})$	multicast bandwidth
prefix sum	7% $\Downarrow$ 3%	$O(p)$	allreduce bandwidth
data packing	10% $\Uparrow$ 3%	$O(v^2o^2/p)$	integer ops
all-to-all-v	8%	$O(v^2o^2/p)$	bisection bandwidth
tensor folding	4%	$O(v^2o^2/p)$	memory bandwidth

CCSDT performance

Table: CCSDT iteration time for systems of 3 and 4 waters with the aug-cc-pVDZ basis set:

#nodes	w3 CTF	w3 NWChem	w4 CTF	w4 NWChem
32	332 sec	868 sec	-	-
64	236 sec	775 sec	1013 sec	-
128	508 sec	414 sec	745 sec	-

So far, the largest CCSDT computation we have completed on Mira is a 5-water system with 180 virtual orbitals on 4096 nodes of Mira. Each CCSDT iteration took roughly 30 minutes and 21% of the execution time was spent in sequential matrix multiplication.

Summary, conclusions, and future work

Fast algorithms for symmetric tensor contractions:

- exploit symmetry in `symv`, `symm`, `syr2`, and `syr2k` to reduce the number of multiplications by 2
- exploit symmetry in ring matrix multiplication, to save a factor of six in multiplications
- Factorization of more general physical tensor equations requires careful consideration of intermediate quantities

Cyclops Tensor Framework (CTF)

- parallel numerical library for tensor algebra which provides 'vectorized' symmetric tensor contraction routines
- CTF provides slicing and sparse write primitives for data extraction and manipulation
- For code and more information and complete code see `ctf.cs.berkeley.edu`

Future direction: sparse tensor contractions, faster algorithms

Backup slides