Developing scalable and portable electronic structure methods with Cyclops Tensor Framework

Edgar Solomonik

ETH Zurich \[\Rightarrow\] August \[\Rightarrow\] University of Illinois

July 22, 2016
A stand-alone library for petascale tensor computations

Cyclops Tensor Framework (CTF)

- distributed-memory symmetric/sparse tensors as C++ objects

\[
\begin{align*}
\text{Matrix}<\text{int}> \quad & A(n, n, \text{AS|SP, World}(\text{MPI\_COMM\_WORLD})); \\
\text{Tensor}<\text{float}> \quad & T(\text{order, is\_sparse, dims, syms, ring, world}); \\
& T.\text{read}(\ldots); \ T.\text{write}(\ldots); \ T.\text{slice}(\ldots); \ T.\text{permute}(\ldots);
\end{align*}
\]

- parallel contraction/summation of tensors

\[
\begin{align*}
Z["abij"] &= V["ijab"]; \\
B["ai"] &= A["aiai"]; \\
T["abij"] &= T["abij"] \times D["abij"]; \\
W["mnij"] &= 0.5 \times W["mnef"] \times T["efij"]; \\
Z["abij"] &= R["mnje"] \times T3["abeimn"]; \\
M["ij"] &= \text{Function}<>()([](double x){ \text{return} \ 1./x; })(v["j"]); \\
\end{align*}
\]

- development (1500 commits) since 2011, open source since 2013

- fundamental part of Aquarius, CC4S, integrated into QChem and Psi4

https://github.com/solomonik/ctf
CTF is highly tuned for massively-parallel machines
- multidimensional tensor blocking and processor grids
- topology-aware mapping and collective communication
- performance-model-driven decomposition at runtime
- optimized redistribution kernels for tensor transposition
CCSD in Aquarius using CTF

Extracted from Aquarius (lead by Devin Matthews)
https://github.com/devinamatthews/aquarius

```plaintext
FMI["mi"] += 0.5*WMNEF["mnef"]*T2["efin"];  
WMNIJ["mnij"] += 0.5*WMNEF["mnef"]*T2["efij"];  
FAE["ae"] -= 0.5*WMNEF["mnef"]*T2["afmn"];  
WAMEI["amei"] -= 0.5*WMNEF["mnef"]*T2["afin"];  

Z2["abij"] = WMNEF["ijab"];  
Z2["abij"] += FAE["af"]*T2["fbij"];  
Z2["abij"] -= FMI["ni"]*T2["abnj"];  
Z2["abij"] += 0.5*WABEF["abef"]*T2["efij"];  
Z2["abij"] += 0.5*WMNIJ["mnij"]*T2["abmn"];  
Z2["abij"] -= WAMEI["amei"]*T2["ebmj"];  
```
Coupled cluster on IBM BlueGene/Q and Cray XC30

CCSD up to 55 (50) water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ\(^a\)

---

\(^a\)S., Matthews, Hammond, Demmel, JPDC, 2014

Cyclops Tensor Framework

https://github.com/solomonik/ctf
Comparison with NWChem

NWChem built using one-sided MPI, not necessarily best performance
- derives equations via Tensor Contraction Engine (TCE)
- generates contractions as blocked loops leveraging Global Arrays

Strong scaling CCSD on Edison

Strong scaling CCSDT on Edison
How does CTF achieve parallel scalability?

CTF algorithms address fundamental parallelization challenges:

- load balance
- communication costs
  - amount of data sent or received
  - number of messages sent or received
  - amount of data moved between memory and cache
  - amount of data moved between memory and disk
Balancing load via a cyclic data decomposition

for sparse tensors, a cyclic layout also provides a load-balanced distribution
CTF generalizes the most efficient matrix multiplication algorithms to tensor contractions

- the comm cost of matrix multiplication $C = AB$ of matrices with dims $m \times k$ and $k \times n$ on $p$ processors is

$$W = \begin{cases} 
O \left( \min_{p_1 p_2 p_3 = p} \left[ \frac{mk}{p_1 p_2} + \frac{kn}{p_2 p_3} + \frac{mn}{p_1 p_3} \right] \right) & : \text{dense} \\
O \left( \min_{p_1 p_2 p_3 = p} \left[ \frac{\text{nnz}(A)}{p_1 p_2} + \frac{\text{nnz}(B)}{p_2 p_3} + \frac{\text{nnz}(C)}{p_1 p_3} \right] \right) & : \text{sparse} 
\end{cases}$$

- communication-optimal algorithms require additional memory usage $M$,

$$W = \begin{cases} 
\Omega \left( \frac{mnk}{p \sqrt{M}} \right) & : \text{dense} \\
\Omega \left( \frac{\text{flops}(A,B,C)}{p \sqrt{M}} \right) & : \text{sparse} 
\end{cases}$$

- CTF selects best $p_1, p_2, p_3$ subject to memory usage constraints on $M$
Data redistribution and matricization

Transitions between contractions require redistribution and refolding

- CTF defines a base distribution for each tensor (by default, over all processors), which can also be user-specified
- before each contraction, the tensor data is redistributed globally and matricized locally
- 3 types of global redistribution algorithms are optimized and threaded
- matricization for sparse tensors corresponds to a conversion to a column-sparse-row matrix layout
- the cost of redistribution is part of the performance model used to select the contraction algorithm
A case-study of a naive sparse MP3 code

```cpp
Tensor<> Ea, Ei, Fab, Fij, Vabij, Vijab, Vabcd, Vijkl, Vaibj;
... // compute above 1-e an 2-e integrals

Tensor<> T(4, Vabij.lens, *Vabij.wrld);
T["abij"] = Vabij["abij"];

divide_EaEi(Ea, Ei, T);

Tensor<> Z(4, Vabij.lens, *Vabij.wrld);
Z["abij"] = Vijab["ijab"];
Z["abij"] += Fab["af"]*T["fbij"];
Z["abij"] -= Fij["ni"]*T["abnj"];  
Z["abij"] += 0.5*Vabcd["abef"]*T["efij"];  
Z["abij"] += 0.5*Vijkl["mnij"]*T["abmn"];  
Z["abij"] += Vaibj["amei"]*T["ebmj"];  

divide_EaEi(Ea, Ei, Z);

double MP3_energy = Z["abij"]*Vabij["abij"];  
```
A case-study of a naive sparse MP3 code

A naive dense version of division in MP2/MP3

```cpp
void divide_EaEi(Tensor<> & Ea, 
    Tensor<> & Ei, 
    Tensor<> & T){
    Tensor<> D(4, T.lens, *T.wrld);
    D["abij"] += Ei["i"];
    D["abij"] += Ei["j"];
    D["abij"] -= Ea["a"];
    D["abij"] -= Ea["b"];

    Transform<> div([](double & b){ b=1./b; });
    div(D["abij"]);
    T["abij"] = T["abij"]*D["abij"];}
```
A case-study of a naive sparse MP3 code

A sparsity-aware version of division in MP2/MP3 using CTF functions

```c
struct dp {
    double a, b;
    dp(int x=0){ a=0.0; b=0.0; }
    dp(double a_, double b_){ a=a_; b=b_; }
    dp operator +(dp const & p) const { return dp(a+p.a, b+p.b); }
};

Tensor<dp> TD(4, 1, T.lens, *T.wrld, Monoid<dp,false>());

TD["abij"] = Function<double,dp>(
    [](double d){ return dp(d, 0.0); }
  )(T["abij"]);

Transform<double,dp> ([](double d, dp & p){ p.b += d; })(Ei["i"], TD["abij"]);
... // similar for Ej, Ea, Eb

T["abij"] = Function<dp,double>([](dp p){ return p.a/p.b; })(TD["abij"]);
```

Cyclops Tensor Framework

https://github.com/solomonik/ctf

13/21
We study the time to solution of the sparse MP3 code, with (1) dense $V$ and $T$ (2) sparse $V$ and dense $T$ (3) sparse $V$ and $T$.
We study the scaling to larger problems of the sparse MP3 code, with (1) dense $V$ and $T$ (2) sparse $V$ and dense $T$ (3) sparse $V$ and $T$.
Can we get more cost savings from tensor symmetry?

We can exploit tensor symmetry (e.g. $A_{ij} = A_{ji}$) to reduce cost\(^1\)

- for order $d$ tensor, $d!$ less memory
- dot product $\sum_{ij} A_{ij} B_{ij} = 2 \sum_{i<j} A_{ij} B_{ij} + \sum_i A_{ii} B_{ii}$
- matrix-vector multiplication ($A_{ij} = A_{ji}$)\(^1\)

$$c_i = \sum_j A_{ij} b_j = \sum_j A_{ij} (b_i + b_j) - \left(\sum_j A_{ij}\right) b_i$$

$A_{ij} b_j \neq A_{ji} b_i$ but $A_{ij} (b_i + b_j) = A_{ji} (b_j + b_i) \rightarrow (1/2)n^2$ multiplies

- partially-symmetric case: $A_{ij}^{km} = A_{ji}^{km}$

$$c_{i}^{kl} = \sum_{jm} A_{ij}^{km} b_{j}^{ml} = \sum_j \left(\sum_m A_{ij}^{km} (b_{i}^{ml} + b_{j}^{ml})\right) - \sum_m \left(\sum_j A_{ij}^{km}\right) b_{i}^{ml}$$

- let $Z_{ij}^{kl} = \sum_m A_{ij}^{km}(b_{i}^{ml} + b_{j}^{ml})$ and observe $Z_{ij}^{kl} = Z_{ji}^{kl}$
- $Z_{ij}^{kl}$ can be computed using $(1/2)n^5$ multiplies and $(1/2)n^5$ adds

Symmetry preserving algorithms

By exploiting symmetry, reduce multiplies (but increase adds)\(^2\)

- rank-2 vector outer product

\[
C_{ij} = a_i b_j + a_j b_i = (a_i + a_j)(b_i + b_j) - a_i b_i - a_j b_j
\]

- squaring a symmetric matrix \(A\) (or \(AB + BA\))

\[
C_{ij} = \sum_k A_{ik} A_{kj} = \sum_k (A_{ik} + A_{kj} + A_{ij})^2 - \ldots
\]

- for symmetrized contraction of symmetric order \(s + \nu\) and \(\nu + t\) tensors

\[
\frac{(s + t + \nu)!}{s!t!\nu!} \text{ fewer multiplies}
\]

e.g. cases above are

- \(s = 1, t = 1, \nu = 0 \rightarrow \text{reduction by 2X}\)
- \(s = 1, t = 1, \nu = 1 \rightarrow \text{reduction by 6X}\)

\(^2\)S., Demmel; Technical Report, ETH Zurich, 2015.

Cyclops Tensor Framework  https://github.com/solomonik/ctf
Applications of symmetry preserving algorithms

Extensions and applications:

- algorithms (mostly) generalize to antisymmetric and Hermitian tensors
- cost reductions in partially-symmetric coupled cluster contractions:
  - 2X-9X for select contractions
  - approximately 1.3X for CCSD, 2.1X for CCSDT, 5.7X for CCSDTQ
    (depends on system size, factorization, spin treatment)
- for Hermitian tensors, multiplies cost 3X more than adds
  - Hermitian matrix multiplication and tridiagonal reduction (BLAS and LAPACK routines) with 25% fewer operations
- \((2/3)n^3\) multiplies for squaring a non symmetric matrix
- decompose symmetric contractions into smaller symmetric contractions

Further directions:

- high performance implementation
- generalization to other group actions
Ongoing and future work

CTF enhancements by expected time frame

- less than 3 months
  - contractions with output sparsity filtering (completing sparsity support)

- less than 2 years
  - automatic scheduling of many contractions (can already be done manually)
  - support for tensor networks and tensor decompositions
  - use of symmetry-preserving algorithms

- less than 5 years
  - advanced abstractions for tensor networks and tensor decompositions
  - optimization of data layouts across many contractions
  - tensor primitives beyond contractions: FFTs, diagonalization, etc.

Above subject to user input, direct collaboration is the shortest path to high performance for new types of applications
for input/contribution to results presented in this talk:

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

for funding:

- ETH Zurich Postdoctoral Fellowship
- DOE Computational Science Graduate Fellowship

for computational resources:

- NERSC (Lawrence Berkeley National Laboratory)
- ALCF (Argonne National Laboratory)
Cyclops Tensor Framework (CTF)

- distributed-memory symmetric/sparse tensors as C++ objects
  
  ```
  Matrix<int> A(n, n, AS|SP, World(MPI_COMM_WORLD));
  Tensor<float> T(order, is_sparse, dims, syms, ring, world);
  T.read(...); T.write(...); T.slice(...); T.permute(...);
  ```

- parallel contraction/summation of tensors
  
  ```
  Z["abij"] += V["ijab"];
  B["ai"] = A["aiai"];
  T["abij"] = T["abij"]*D["abij"];  
  W["mnij"] += 0.5*W["mnef"]*T["efij"];  
  Z["abij"] -= R["mnje"]*T3["abeimn"];  
  M["ij"] += Function<>([](double x){ return 1./x; })(v["j"]);  
  ```

- development (1500 commits) since 2011, open source since 2013

- fundamental part of Aquarius, CC4S, integrated into QChem and Psi4

https://github.com/solomonik/ctf
Symmetry preserving algorithm vs Strassen’s algorithm

Symmetry preserving alg. vs Strassen’s alg. (s=t=v=ω/3)

- Strassen’s algorithm
- Sym. preserving ω=6
- Sym. preserving ω=3

Cyclops Tensor Framework
https://github.com/solomonik/ctf
Our CCSD factorization

Credit to John F. Stanton and Jurgen Gauss

\[
\tau^{ab}_{ij} = t^{ab}_{ij} + \frac{1}{2} P^{a}_b P^{i}_j t^{a}_{i} t^{b}_{j},
\]

\[
\tilde{F}^{m}_e = f^{m}_e + \sum_{fn} v^{mn}_{ef} t^{f}_n,
\]

\[
\tilde{F}^{a}_e = (1 - \delta_{ae}) f^{a}_e - \sum_{m} \tilde{F}^{m}_e t^{a}_m - \frac{1}{2} \sum_{mnf} v^{mn}_{ef} t^{af}_{mn} + \sum_{fn} v^{an}_{ef} t^{f}_n,
\]

\[
\tilde{F}^{m}_i = (1 - \delta_{mi}) f^{m}_i + \sum_{e} \tilde{F}^{e}_m t^{e}_i + \frac{1}{2} \sum_{nef} v^{mn}_{ef} t^{ef}_{in} + \sum_{fn} v^{mn}_{if} t^{f}_n,
\]
\[ \tilde{W}^{mn}_{ei} = v^{mn}_{ei} + \sum_{f} v^{mn}_{ef} t^{f}_{i}, \]

\[ \tilde{W}^{mn}_{ij} = v^{mn}_{ij} + P^{i}_{j} \sum_{e} v^{mn}_{ie} t^{e}_{j} + \frac{1}{2} \sum_{ef} v^{mn}_{ef} t^{ef}_{ij}, \]

\[ \tilde{W}^{am}_{ie} = v^{am}_{ie} - \sum_{n} \tilde{W}^{mn}_{ei} t^{a}_{n} + \sum_{f} v^{ma}_{ef} t^{f}_{i} + \frac{1}{2} \sum_{nf} v^{mn}_{ef} t^{af}_{in}, \]

\[ \tilde{W}^{am}_{ij} = v^{am}_{ij} + P^{i}_{j} \sum_{e} v^{am}_{ie} t^{e}_{j} + \frac{1}{2} \sum_{ef} v^{am}_{ef} t^{ef}_{ij}, \]

\[ z^{a}_{i} = f^{a}_{i} - \sum_{m} \tilde{F}^{m}_{i} t^{a}_{m} + \sum_{e} f^{a}_{e} t^{e}_{i} + \sum_{em} v^{ma}_{ei} t^{e}_{m} + \sum_{em} v^{ae}_{im} \tilde{F}^{m}_{e} + \frac{1}{2} \sum_{efm} v^{am}_{ef} t^{ef}_{im}, \]

\[ z^{ab}_{ij} = v^{ab}_{ij} + P^{i}_{j} \sum_{e} v^{ab}_{ie} t^{e}_{j} + P^{a}_{b} P^{i}_{j} \sum_{me} \tilde{W}^{am}_{ie} t^{eb}_{m} - P^{a}_{b} \sum_{m} \tilde{W}^{am}_{ij} t^{b}_{m} \]

\[ + P^{a}_{b} \sum_{e} \tilde{F}^{a}_{e} t^{eb}_{ij} - P^{i}_{j} \sum_{m} \tilde{F}^{m}_{i} t^{ab}_{mj} + \frac{1}{2} \sum_{ef} v^{ab}_{ef} t^{ef}_{ij} + \frac{1}{2} \sum_{mn} \tilde{W}^{mn}_{ij} t^{ab}_{mn}, \]

Cyclops Tensor Framework

https://github.com/solomonik/ctf
Stability of symmetry preserving algorithms

Relative error of $c = A^T b$ with positive $A$ and alternating $b$

- Symmetry preserving algorithm relative error
- Direct evaluation algorithm relative error

Relative error of squaring a Householder transformation

- Symmetry preserving algorithm relative error
- Direct evaluation algorithm relative error
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
\( \nu \)-orbitals, \( \sigma \)-electrons

<table>
<thead>
<tr>
<th>kernel</th>
<th>% of time</th>
<th>complexity</th>
<th>architectural bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGEMM</td>
<td>45%</td>
<td>( O(\nu^4\sigma^2/p) )</td>
<td>flops/mem bandwidth</td>
</tr>
<tr>
<td>broadcasts</td>
<td>20%</td>
<td>( O(\nu^4\sigma^2/p\sqrt{M}) )</td>
<td>multicast bandwidth</td>
</tr>
<tr>
<td>prefix sum</td>
<td>10%</td>
<td>( O(p) )</td>
<td>allreduce bandwidth</td>
</tr>
<tr>
<td>data packing</td>
<td>7%</td>
<td>( O(\nu^2\sigma^2/p) )</td>
<td>integer ops</td>
</tr>
<tr>
<td>all-to-all-( \nu )</td>
<td>7%</td>
<td>( O(\nu^2\sigma^2/p) )</td>
<td>bisection bandwidth</td>
</tr>
<tr>
<td>tensor folding</td>
<td>4%</td>
<td>( O(\nu^2\sigma^2/p) )</td>
<td>memory bandwidth</td>
</tr>
</tbody>
</table>

Cyclops Tensor Framework
https://github.com/solomonik/ctf