Leveraging sparsity and symmetry in parallel tensor contractions

Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

CCQ Tensor Network Workshop
Flatiron Institute, New York
Simons Foundation

Sep 15, 2017

@CS@Illinois
A stand-alone library for petascale tensor computations

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 generalized 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"]);
  ```

- NEW: Python! towards autoparallel numpy ndarray: einsum, slicing
Coupled cluster: an initial application driver

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

\[
\begin{align*}
FMI["mi"] &= 0.5 \times WMNEF["mnef"] \times T2["efin"] ; \\
WMNIJ["mnij"] &= 0.5 \times WMNEF["mnef"] \times T2["efij"] ; \\
FAE["ae"] &= -0.5 \times WMNEF["mnef"] \times T2["afmn"] ; \\
WAMEI["amei"] &= -0.5 \times WMNEF["mnef"] \times T2["afin"] ; \\
Z2["abij"] &= WMNEF["ijab"] ; \\
Z2["abij"] &= FAE["af"] \times T2["fbij"] ; \\
Z2["abij"] &= FMI["ni"] \times T2["abnj"] ; \\
Z2["abij"] &= 0.5 \times WABEF["abef"] \times T2["efij"] ; \\
Z2["abij"] &= 0.5 \times WMNIJ["mnij"] \times T2["abmn"] ; \\
Z2["abij"] &= WAMEI["amei"] \times T2["ebmj"] ;
\end{align*}
\]
Performance of CTF for coupled cluster

**CCSD** up to 55 (50) water molecules with cc-pVDZ

**CCSDT** up to 10 water molecules with cc-pVDZ

![Graphs showing weak scaling on BlueGene/Q and Edison for Aquarius-CTF CCSD and CCSDT](image)

compares well to **NWChem** (up to 10x speed-ups for CCSDT)
CTF parallel scalability

CTF is tuned for **massively-parallel** architectures
- multidimensional tensor blocking and processor grids
- topology-aware mapping and **collective communication**
- **performance-model-driven** decomposition at runtime
- optimized redistribution kernels for tensor transposition
- integrated with **HPTT** for fast local transposition

![Diagram](https://via.placeholder.com/150)

**BG/Q matrix multiplication**
- CTF
- Scalapack

![Graph](https://via.placeholder.com/150)
Symmetry and sparsity by cyclicity

For sparse tensors, a cyclic layout provides a load-balanced distribution
Transitions between contractions require redistribution and refolding

- base distribution for each tensor
  - default over all processors
  - or user can specify any processor grid mapping

- to contract, tensor is redistributed globally and matricized locally

- arbitrary sparsity supported via compressed-sparse-row (CSR)

- performance model used to select best contraction algorithm
  - model coefficients can be tuned for all kernels on a given architecture
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"];
Sparse MP3 code

Strong and weak scaling of sparse MP3 code, with
(1) dense $V$ and $T$ (2) sparse $V$ and dense $T$ (3) sparse $V$ and $T$

Strong scaling of MP3 with $no=40$, $nv=160$

Weak scaling of MP3 with $no=40$, $nv=160$
**Betweenness centrality** computes the relative importance of vertices in terms of the number of shortest paths that go through them.

- Can be computed via all-pairs shortest-path from distance matrix, but possible to do via less memory (Brandes’ algorithm).

**Unweighted graphs**
- **Breadth First Search (BFS)** for each vertex
- Back-propagation of centrality scores along BFS tree

**Weighted graphs**
- **SSSP** for each vertex (we use Bellman Ford with sparse frontiers)
- Back-propagation of betweenness centrality scores (no harder than unweighted)

Our formulation uses a set of starting vertices (many BFS runs), leveraging **sparse matrix times sparse matrix**.
Betweenness centrality is a measure of the importance of vertices in the shortest paths of a graph

- computed using **sparse matrix multiplication** (SpGEMM) with operations on special **monoids**
- **CTF** handles this in similar ways to CombBLAS

Friendster has 66 million vertices and 1.8 **billion edges** (results on Blue Waters, Cray XE6)
Much ongoing work and future directions in CTF

- recent: development of Python interface (einsum and slicing work)
- recent: hook-ups for conversion to ScaLAPACK format
- active: performance improvement for batched tensor operations
- active: simple interface for basic matrix factorizations
- active: tensor factorizations
- future: predefined output sparsity for contractions

Existing collaborations and external applications

- Aquarius (lead by Devin Matthews)
- QChem via Libtensor (integration lead by Evgeny Epifanovsky)
- QBall (DFT code, just matrix multiplication)
- CC4S (lead by Andreas Grüneis)
- early collaborations involving Lattice QCD and DMRG
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
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
Our CCSD factorization

\[ \tilde{W}_{ei}^{mn} = v_{ei}^{mn} + \sum_{f} v_{ef}^{mn} t_{i}^{f}, \]

\[ \tilde{W}_{ij}^{mn} = v_{ij}^{mn} + P_{j}^{i} \sum_{e} v_{ie}^{mn} t_{j}^{e} + \frac{1}{2} \sum_{ef} v_{ef}^{mn} \tau_{ij}^{ef}, \]

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

\[ \tilde{W}_{ij}^{am} = v_{ij}^{am} + P_{j}^{i} \sum_{e} v_{ie}^{am} t_{j}^{e} + \frac{1}{2} \sum_{ef} v_{ef}^{am} \tau_{ij}^{ef}, \]

\[ z_{i}^{a} = f_{i}^{a} - \sum_{m} \tilde{F}_{i}^{m} t_{m}^{a} + \sum_{e} f_{e}^{a} t_{i}^{e} + \sum_{em} v_{ei}^{ma} t_{m}^{e} + \sum_{em} v_{im}^{ae} \tilde{F}_{e}^{m} + \frac{1}{2} \sum_{efm} v_{ef}^{am} \tau_{im}^{ef}, \]

\[ - \frac{1}{2} \sum_{emn} \tilde{W}_{ei}^{mn} t_{mn}^{ea}, \]

\[ z_{ij}^{ab} = v_{ij}^{ab} + P_{j}^{i} \sum_{e} v_{ie}^{ab} t_{j}^{e} + P_{b}^{a} P_{j}^{i} \sum_{me} \tilde{W}_{ie}^{am} t_{mj}^{eb} - P_{b}^{a} \sum_{m} \tilde{W}_{ij}^{am} t_{m}^{b} \]

\[ + P_{b}^{a} \sum_{e} \tilde{F}_{e}^{a} t_{ij}^{eb} - P_{j}^{i} \sum_{m} \tilde{F}_{i}^{m} t_{mj}^{ab} + \frac{1}{2} \sum_{ef} v_{ef}^{ab} \tau_{ij}^{ef} + \frac{1}{2} \sum_{mn} \tilde{W}_{ij}^{mn} \tau_{mn}^{ab}, \]
Stability of symmetry preserving algorithms

Relative error of $c = A \cdot b$ 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, \( o \)-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^4o^2/p) )</td>
<td>flops/mem bandwidth</td>
</tr>
<tr>
<td>broadcasts</td>
<td>20%</td>
<td>( O(\nu^4o^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^2o^2/p) )</td>
<td>integer ops</td>
</tr>
<tr>
<td>all-to-all-( \nu )</td>
<td>7%</td>
<td>( O(\nu^2o^2/p) )</td>
<td>bisection bandwidth</td>
</tr>
<tr>
<td>tensor folding</td>
<td>4%</td>
<td>( O(\nu^2o^2/p) )</td>
<td>memory bandwidth</td>
</tr>
</tbody>
</table>