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

L P N A @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

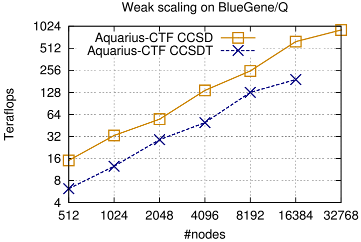
CCSD contractions from Aquarius (lead by Devin Matthews)

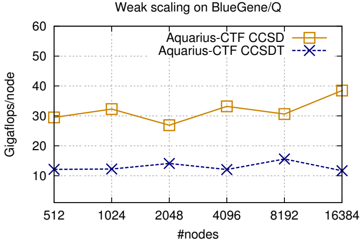
https://github.com/devinamatthews/aquarius

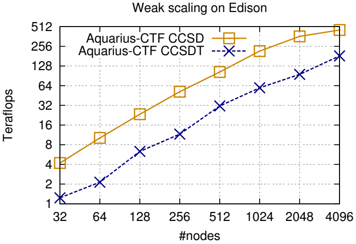
```
FMI["mi"]      += 0.5*WMNEF["mnef"]*T2["efin"];
WMNIJ["mnij"]  += 0.5*WMNEF["mnef"]*T2["efij"];
FAE["ae"]      -= 0.5*WMNEF["mnef"]*T2["afmn"];
WAMEI["amei"]  -= 0.5*WMNEF["mnef"]*T2["afin"];

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"];
```

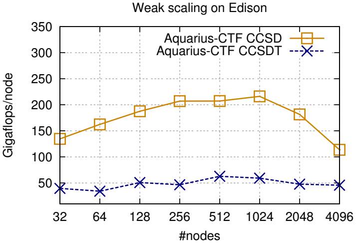
Performance of CTF for coupled cluster

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





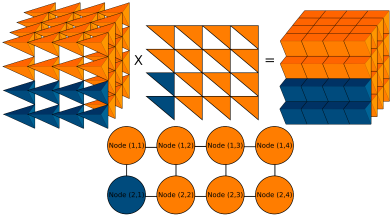


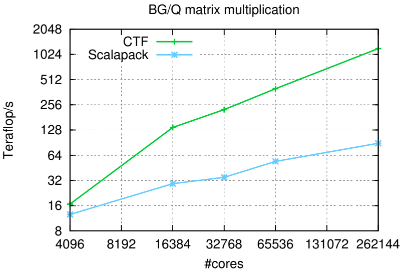


compares well to NWChem (up to **10x** speed-ups for CCSDT)

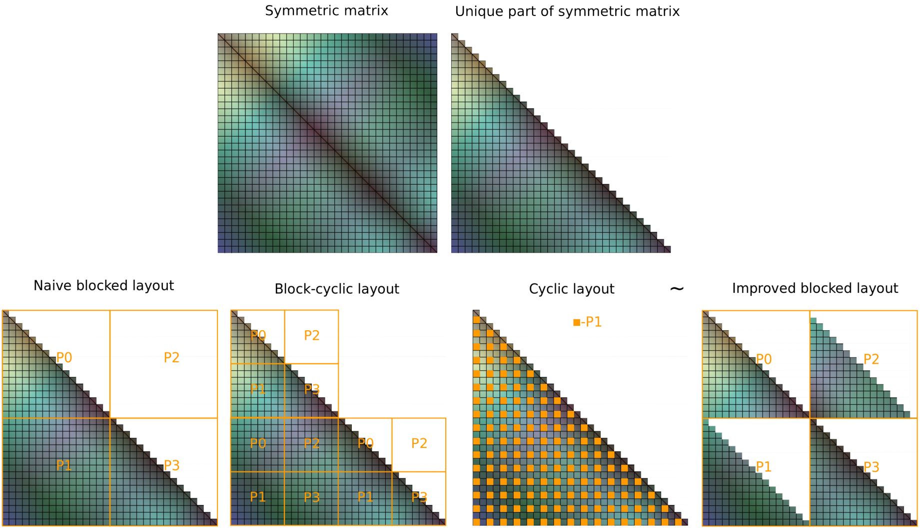
CTF is tuned for massively-parallel architectures

- 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





Symmetry and sparsity by cyclicity



for sparse tensors, a cyclic layout provides a load-balanced distribution

Data mapping and autotuning

Transitions between contractions require redistribution and refolding

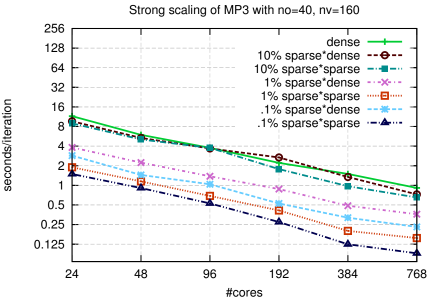
- base distribution for each tensor
  - default over all processors
  - or user can specify any processor grid mapping
- to contract, tensor is redistributed globally and matricized locally
- arbitrary sparsity supported via compressed-sparse-row (CSR)
- performance model used to select best contraction algorithm
  - model coefficients can be tuned for all kernels on a given architecture

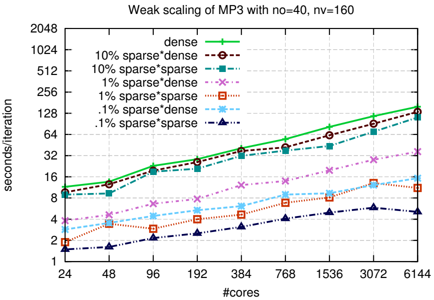
MP3 method

```
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***





Special operator application: betweenness centrality

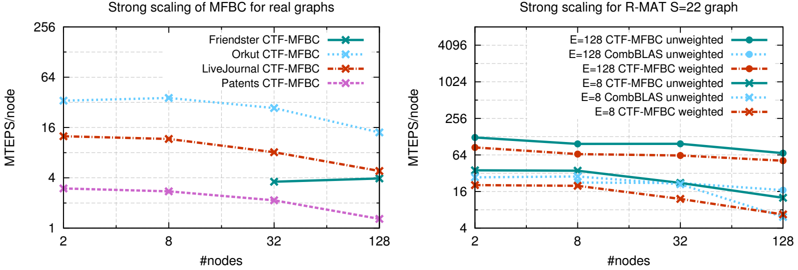
Betweenness centrality computes the relative importance vertices in terms of the number of shortest paths that go through them

- 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

CTF for betweenness centrality

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)

CTF status and explorations

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

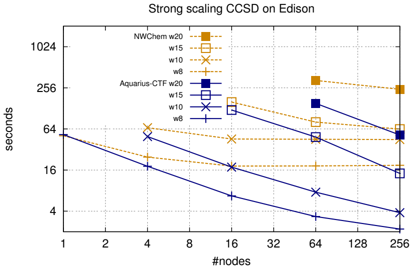
- Aquarius (lead by Devin Matthews)
- QChem via Libtensor (integration lead by Evgeny Epifanovsky)
- QBall (DFT code, just matrix multiplication)
- CC4S (lead by Andreas Grüneis)
- early collaborations involving Lattice QCD and DMRG

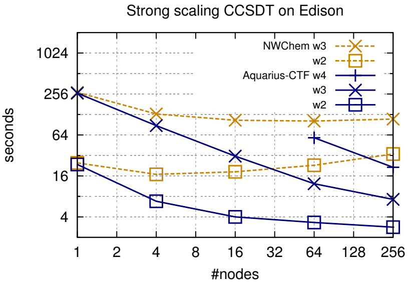
Backup slides

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



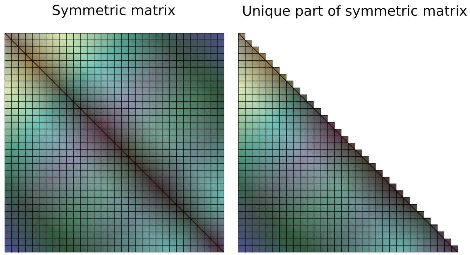


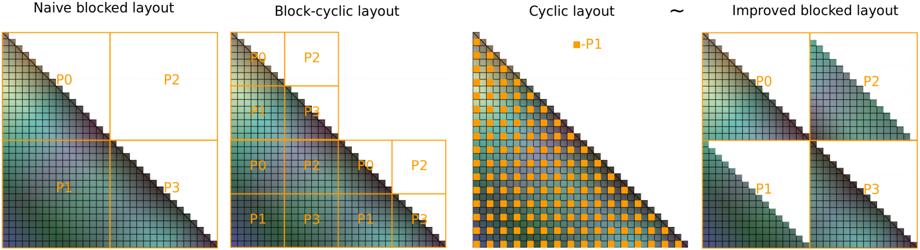
How does CTF achieve parallel scalability?

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



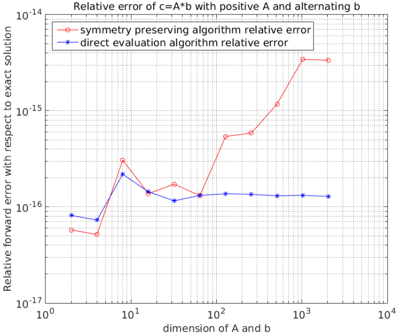


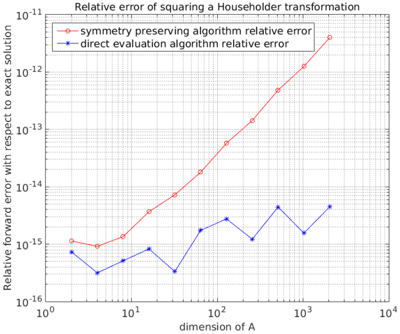
for sparse tensors, a cyclic layout also provides a load-balanced distribution

Our CCSD factorization

$$
\begin{aligned}
\tilde{W}^{mn}_{ei} &= v^{mn}_{ei} + \sum_f v^{mn}_{ef} t^f_i, \\
\tilde{W}^{mn}_{ij} &= v^{mn}_{ij} + P^i_j \sum_e v^{mn}_{ie} t^e_j + \frac{1}{2} \sum_{ef} v^{mn}_{ef} \tau^{ef}_{ij}, \\
\tilde{W}^{am}_{ie} &= v^{am}_{ie} - \sum_n \tilde{W}^{mn}_{ei} t^a_n + \sum_f v^{ma}_{ef} t^f_i + \frac{1}{2} \sum_{nf} v^{mn}_{ef} t^{af}_{in}, \\
\tilde{W}^{am}_{ij} &= v^{am}_{ij} + P^i_j \sum_e v^{am}_{ie} t^e_j + \frac{1}{2} \sum_{ef} v^{am}_{ef} \tau^{ef}_{ij}, \\
z^a_i &= f^a_i - \sum_m \tilde{F}^m_i t^a_m + \sum_e f^a_e t^e_i + \sum_{em} v^{ma}_{ei} t^e_m + \sum_{em} v^{ae}_{im} \tilde{F}^m_e + \frac{1}{2} \sum_{efm} v^{am}_{ef} \tau^{ef}_{im} \\
&- \frac{1}{2} \sum_{emn} \tilde{W}^{mn}_{ei} t^{ea}_{mn}, \\
z^{ab}_{ij} &= v^{ab}_{ij} + P^i_j \sum_e v^{ab}_{ie} t^e_j + P^a_b P^i_j \sum_{me} \tilde{W}^{am}_{ie} t^{eb}_{mj} - P^a_b \sum_m \tilde{W}^{am}_{ij} t^b_m \\
&+ P^a_b \sum_e \tilde{F}^a_e t^{eb}_{ij} - P^i_j \sum_m \tilde{F}^m_i t^{ab}_{mj} + \frac{1}{2} \sum_{ef} v^{ab}_{ef} \tau^{ef}_{ij} + \frac{1}{2} \sum_{mn} \tilde{W}^{mn}_{ij} \tau^{ab}_{mn},
\end{aligned}
$$

Stability of symmetry preserving algorithms





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
v-orbitals, o-electrons

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