A massively parallel library for matrix and tensor algorithms

Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

Argonne National Laboratory

Oct 25, 2017

A stand-alone library for parallel 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)

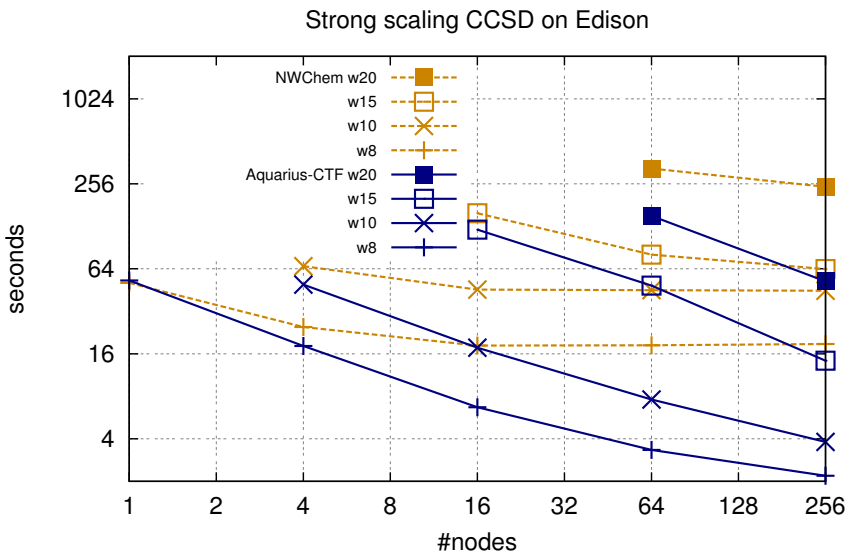
`https://github.com/devinamatthews/aquarius`

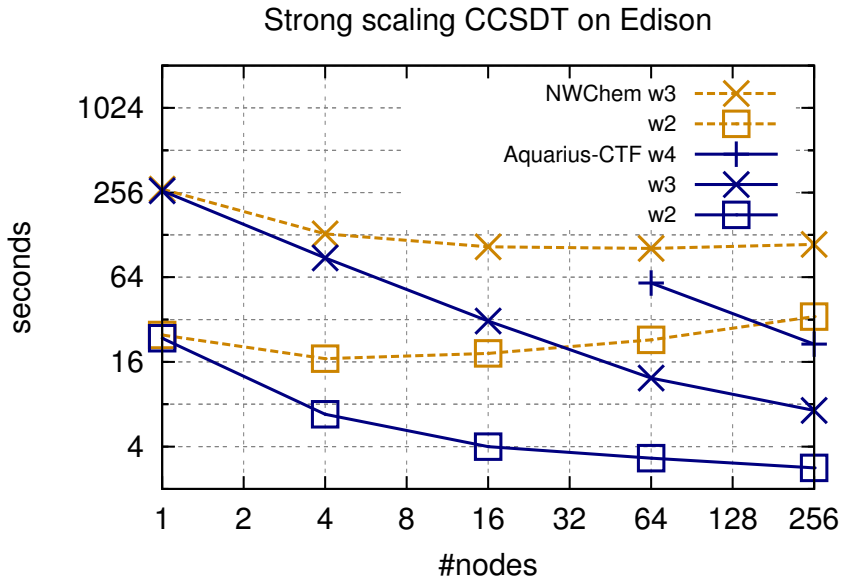
```
FMI["mi"]      += 0.5*WMNEF["mnef"]*T2["efin"];
WMNIJ["mnij"]  += 0.5*WMNEF["mnef"]*T2["efij"];
FAE["ae"]      -= 0.5*WMNEF["mnef"]*T2["afmn"];
WAMEI["amei"]  -= 0.5*WMNEF["mnef"]*T2["afin"];

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"];
```

Comparison with NWChem

NWChem built using one-sided MPI, not necessarily best performance

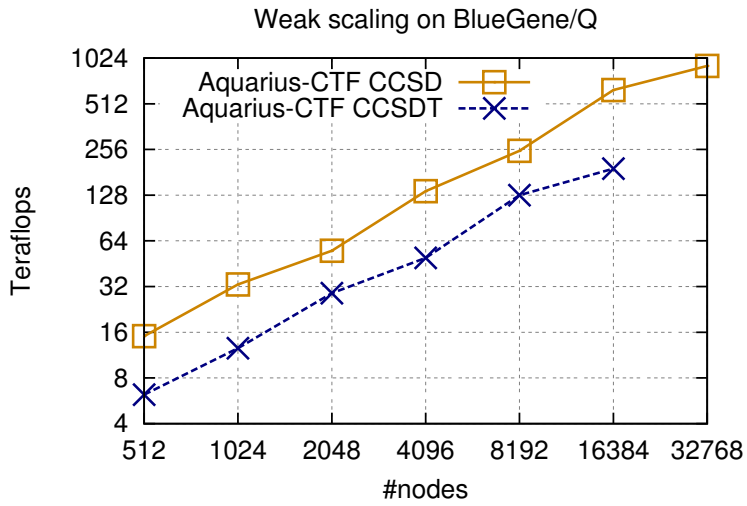
- derives equations via Tensor Contraction Engine (TCE)
- generates contractions as blocked loops leveraging (Global Arrays)

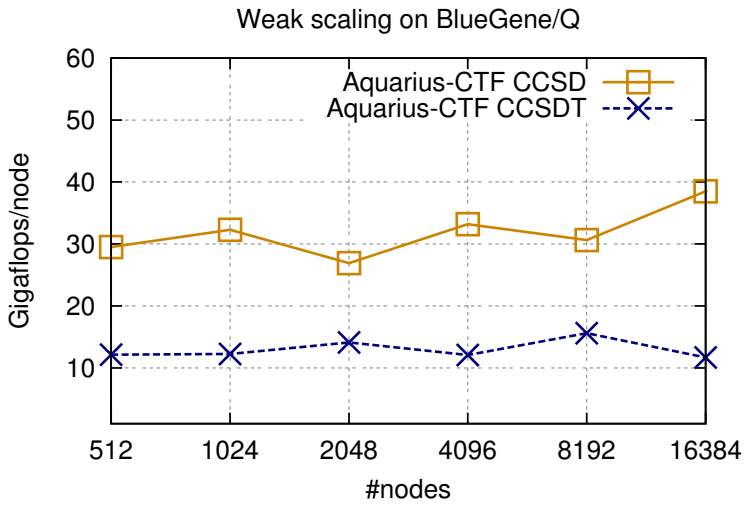


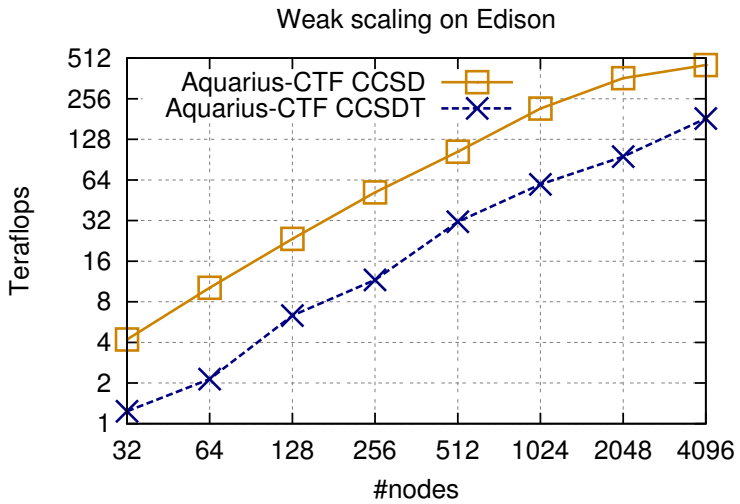


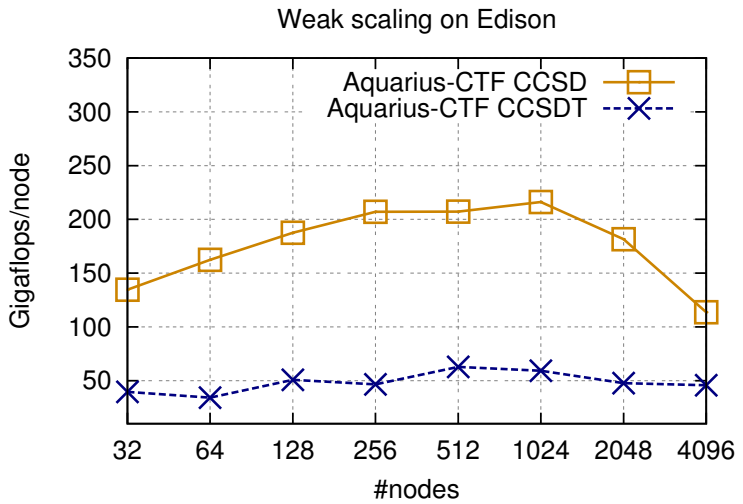
Performance of CTF for coupled cluster

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









Performance breakdown on BG/Q

Performance data (from circa 2013) 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

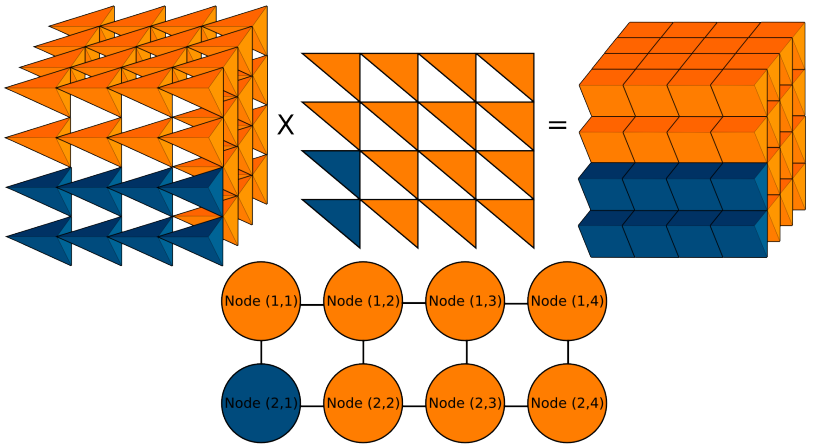
v-orbitals, o-electrons

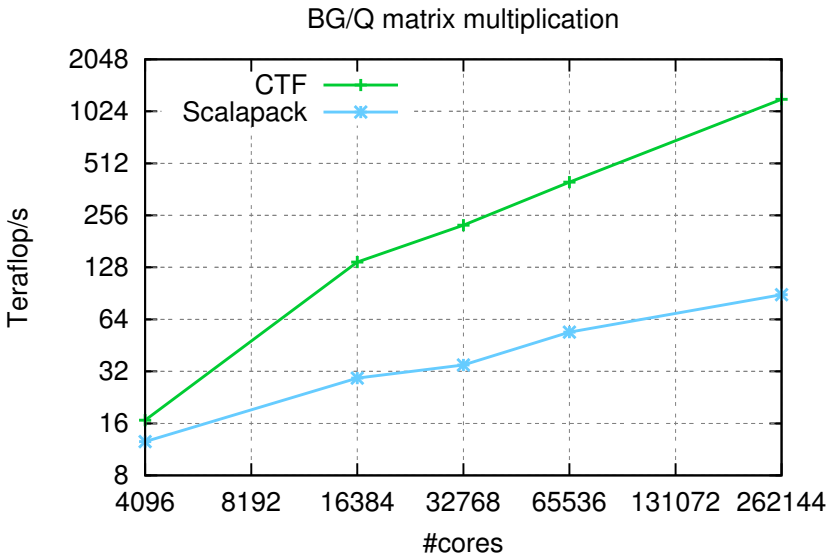
kernel	% of time	complexity	architectural bounds
DGEMM	45%	$O(v^4o^2/p)$	flops/mem bandwidth
broadcasts	20%	$O(v^4o^2/p\sqrt{M})$	multicast bandwidth
prefix sum	10%	$O(p)$	allreduce bandwidth
data packing	7%	$O(v^2o^2/p)$	integer ops
all-to-all-v	7%	$O(v^2o^2/p)$	bisection bandwidth
tensor folding	4%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

CTF parallel scalability

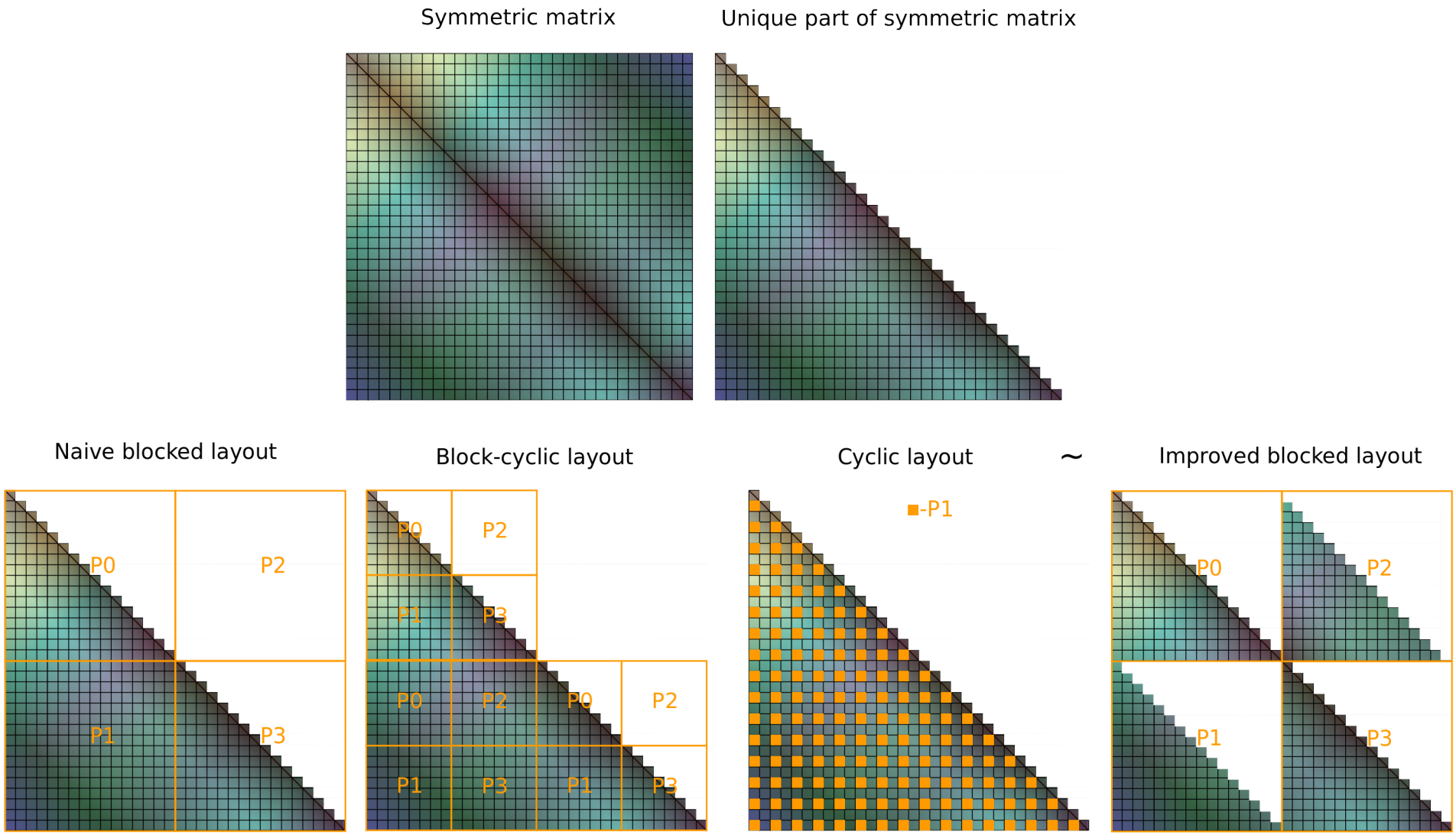
CTF is tuned for massively-parallel architectures

- 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





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

$$\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{t}$$

where the ith row of $\boldsymbol{A}$ is a set of observed parameters, $\boldsymbol{t}_i$ is the execution time of kernel with those parameters, and $\boldsymbol{x}$ are coefficients

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"];
```

MP3 dense division

A naive dense version of division in MP2/MP3

```
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"];
}
```

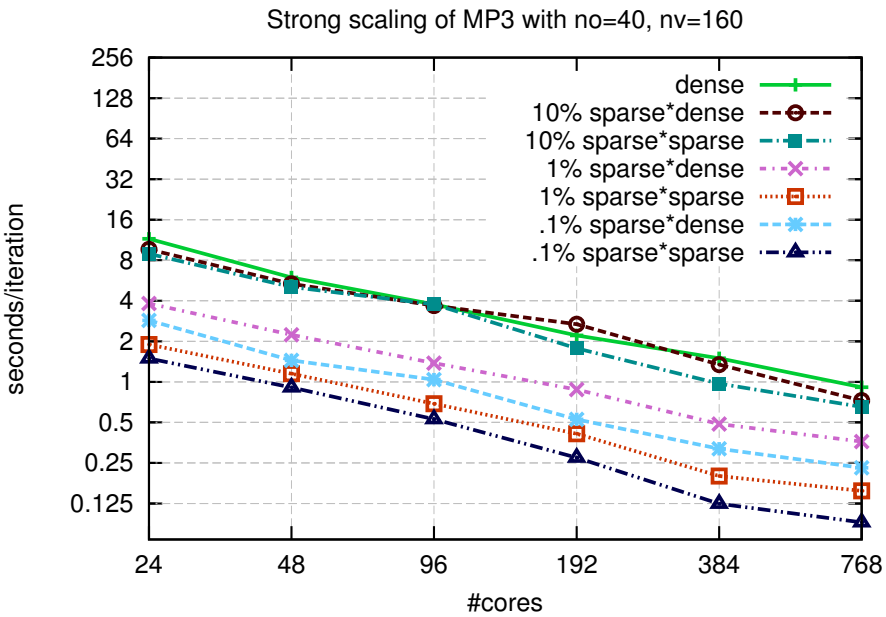
MP3 sparse division

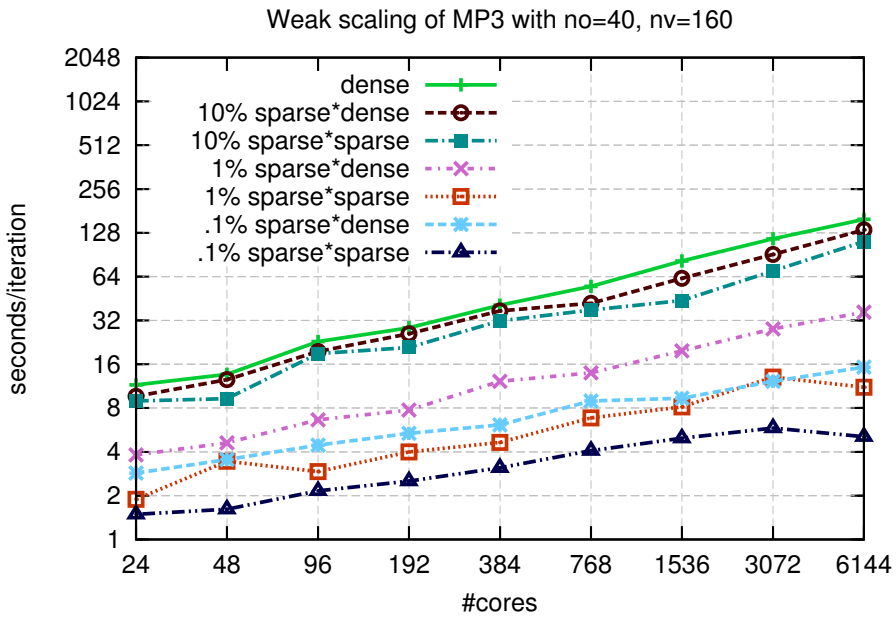
A sparsity-aware version of division in MP2/MP3 using CTF functions

```
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){ return 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"]);
```

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 centrality scores (no harder than unweighted)
- our formulation uses a set of starting vertices (many BFS runs), leveraging SpGEMM (sparse matrix times sparse matrix)

CTF code for betweenness centrality

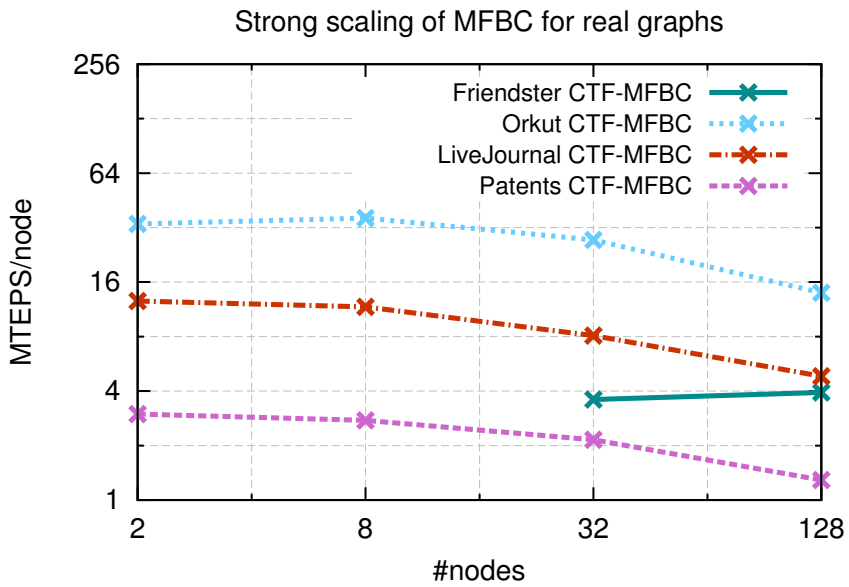
Betweenness centrality code snippet, for k of n nodes

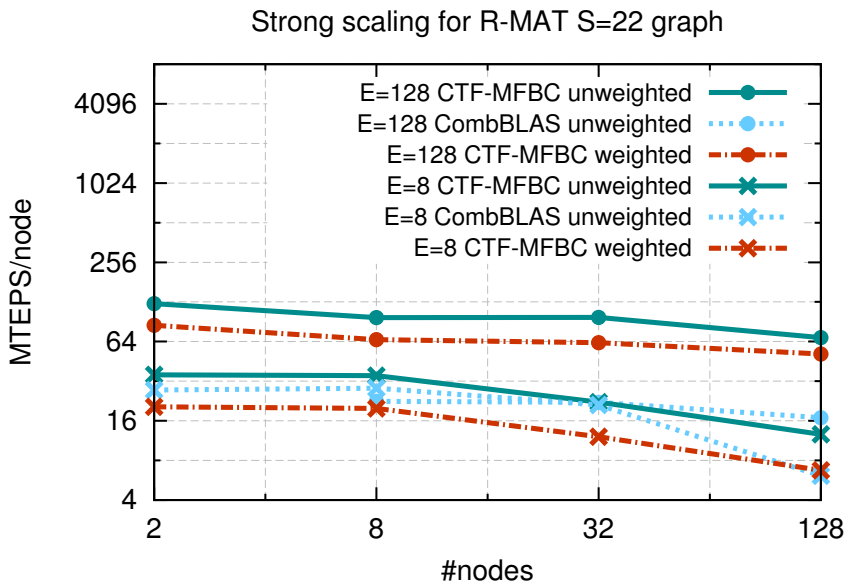
```
void btwn_central(Matrix<int> A, Matrix<path> P, int n, int k)
  Monoid<path> mon(...,
                   [](path a, path b){
                     if (a.w<b.w) return a;
                     else if (b.w<a.w) return b;
                     else return path(a.w, a.m+b.m);
                   }, ...);

  Matrix<path> Q(n,k,mon); // shortest path matrix
  Q["ij"] = P["ij"];

  Function<int,path> append([](int w, path p){
                       return path(w+p.w, p.m);
                     }; );

  for (int i=0; i<n; i++)
    Q["ij"] = append(A["ik"],Q["kj"]);
  ...
}
```

CTF performance 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
- by comparison, CombBLAS leverages semirings





Friendster has 66 million vertices and 1.8 billion edges (results on Blue Waters, Cray XE6)

Matrix and tensor factorizations

- hook-ups for conversion to ScaLAPACK format
  - arbitrary matrix factorization
  - tensor factorizations based on matrix algebra on unfoldings
- simpler interface-level support is in development
- native support for tensor networks/factorization planned
- long-term integration with communication-avoiding 3D algorithms
- CTF data layout abstractions make 3D grids easier to use

3D algorithms for dense linear algebra

For Cholesky ($\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$)with p processors, parallel cost is

$$F = O(n^3/p) \text{ flops}, \quad W = O(n^2/p^\delta) \text{ words}, \quad S = O(p^\delta) \text{ syncs}$$

for any $\delta = [1/2, 2/3]$.

Achieving similar costs for LU, QR, and the symmetric eigenvalue problem requires some algorithmic tweaks

triangular solve	square TRSM ✓[1]	rectangular TRSM ✓[2]
LU with pivoting	pairwise pivoting ✓[3]	tournament pivoting ✓[4]
QR factorization	Givens on square ✓[3]	Householder on rect. ✓[5]
SVD	singular values only ✓[5]	singular vectors X

✓ means costs attained (synchronization within polylog factors)

[1] B. Lipshitz, MS thesis 2013
[2] T. Wicky, E.S., T. Hoefler, IPDPS 2017
[3] A. Tiskin, FGCS 2007
[4] E.S., J. Demmel, EuroPar 2011
[5] E.S., G. Ballard, T. Hoefler, J. Demmel, SPAA 2017

CTF status and explorations

Much ongoing work and future directions in CTF

- 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)
- quantum circuit simulation (see paper on breaking 49-qubit simulation barrier, lead by IBM and LLNL)

Acknowledgements

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)

for computational resources:

- NERSC (Lawrence Berkeley National Laboratory)
- ALCF (Argonne National Laboratory)
- NCSA (National Center for Supercomputing Applications)

Backup slides

Applications of partially-symmetric tensor contractions

High-accuracy methods in computational quantum chemistry

- solve the multi-electron Schrödinger equation $\boldsymbol{H}|\Psi\rangle = E|\Psi\rangle$, where $\boldsymbol{H}$ is a linear operator, but Ψ is a function of *all* electrons
- use wavefunction ansatze like $\Psi \approx \Psi^{(k)} = e^{\boldsymbol{T}^{(k)}}|\Psi^{(k-1)}\rangle$ where $\Psi^{(0)}$ is a mean-field (averaged) function and $\boldsymbol{T}^{(k)}$ is an order $2k$ tensor, acting as a multilinear excitation operator on the electrons
- coupled-cluster methods use the above ansatze for $k \in \{2, 3, 4\}$ (CCSD, CCSDT, CCSDTQ)
- solve iteratively for $\boldsymbol{T}^{(k)}$, where each iteration has cost $O(n^{2k+2})$, dominated by contractions of partially antisymmetric tensors
- for example, a dominant contraction in CCSD ($k = 2$) is

$$\boldsymbol{Z}_{i\bar{c}}^{a\bar{k}} = \sum_{b=1}^{n}\sum_{j=1}^{n} \boldsymbol{T}_{ij}^{ab} \cdot \boldsymbol{V}_{b\bar{c}}^{j\bar{k}}$$

Our CCSD factorization

Credit to John F. Stanton and Jurgen Gauss

$$
\begin{aligned}
\tau_{ij}^{ab} &= t_{ij}^{ab} + \frac{1}{2} P_b^a P_j^i t_i^a t_j^b, \\
\tilde{F}_e^m &= f_e^m + \sum_{fn} v_{ef}^{mn} t_n^f, \\
\tilde{F}_e^a &= (1 - \delta_{ae}) f_e^a - \sum_m \tilde{F}_e^m t_m^a - \frac{1}{2} \sum_{mnf} v_{ef}^{mn} t_{mn}^{af} + \sum_{fn} v_{ef}^{an} t_n^f, \\
\tilde{F}_i^m &= (1 - \delta_{mi}) f_i^m + \sum_e \tilde{F}_e^m t_i^e + \frac{1}{2} \sum_{nef} v_{ef}^{mn} t_{in}^{ef} + \sum_{fn} v_{if}^{mn} t_n^f,
\end{aligned}
$$

Our CCSD factorization

$$\tilde{W}_{ei}^{mn} = v_{ei}^{mn} + \sum_f v_{ef}^{mn} t_i^f,$$

$$\tilde{W}_{ij}^{mn} = v_{ij}^{mn} + P_j^i \sum_e v_{ie}^{mn} t_j^e + \frac{1}{2} \sum_{ef} v_{ef}^{mn} \tau_{ij}^{ef},$$

$$\tilde{W}_{ie}^{am} = v_{ie}^{am} - \sum_n \tilde{W}_{ei}^{mn} t_n^a + \sum_f v_{ef}^{ma} t_i^f + \frac{1}{2} \sum_{nf} v_{ef}^{mn} t_{in}^{af},$$

$$\tilde{W}_{ij}^{am} = v_{ij}^{am} + P_j^i \sum_e v_{ie}^{am} t_j^e + \frac{1}{2} \sum_{ef} v_{ef}^{am} \tau_{ij}^{ef},$$

$$z_i^a = f_i^a - \sum_m \tilde{F}_i^m t_m^a + \sum_e f_e^a t_i^e + \sum_{em} v_{ei}^{ma} t_m^e + \sum_{em} v_{im}^{ae} \tilde{F}_e^m + \frac{1}{2} \sum_{efm} v_{ef}^{am} \tau_{im}^{ef}$$

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

$$z_{ij}^{ab} = v_{ij}^{ab} + P_j^i \sum_e v_{ie}^{ab} t_j^e + P_b^a P_j^i \sum_{me} \tilde{W}_{ie}^{am} t_{mj}^{eb} - P_b^a \sum_m \tilde{W}_{ij}^{am} t_m^b$$

$$+ P_b^a \sum_e \tilde{F}_e^a t_{ij}^{eb} - P_j^i \sum_m \tilde{F}_i^m t_{mj}^{ab} + \frac{1}{2} \sum_{ef} v_{ef}^{ab} \tau_{ij}^{ef} + \frac{1}{2} \sum_{mn} \tilde{W}_{ij}^{mn} \tau_{mn}^{ab},$$