Developing scalable and portable electronic structure methods with Cyclops Tensor Framework

Edgar Solomonik

ETH Zurich $\underset{\text{August}}{\Rightarrow}$ University of Illinois

July 22, 2016

Cyclops Tensor Framework (CTF)

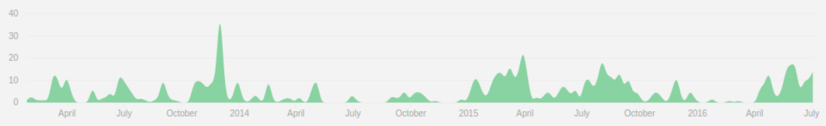
- distributed-memory symmetric/sparse tensors as C++ objects

```
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

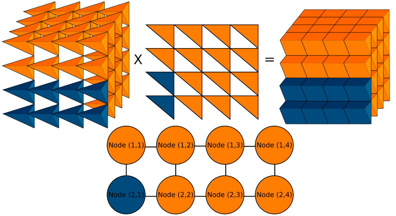


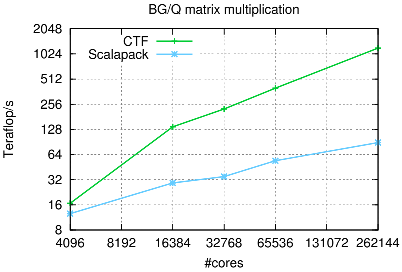
- fundamental part of Aquarius, CC4S, integrated into QChem and Psi4

CTF parallel scalability

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

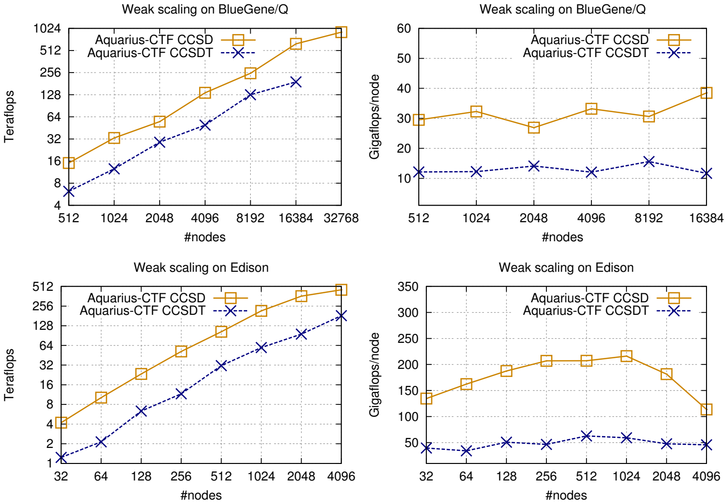
Extracted from Aquarius (lead by Devin Matthews)

https://github.com/devinamatthews/aquarius

```
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"];
```

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

CCSDT up to 10 water molecules with cc-pVDZ[a]

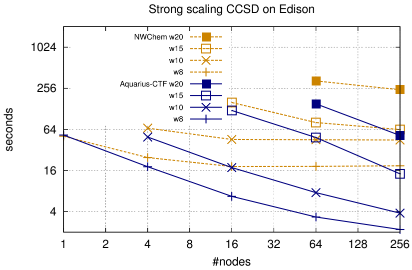


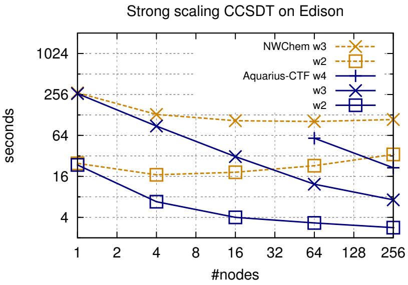
[a] S., Matthews, Hammond, Demmel, JPDC, 2014

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



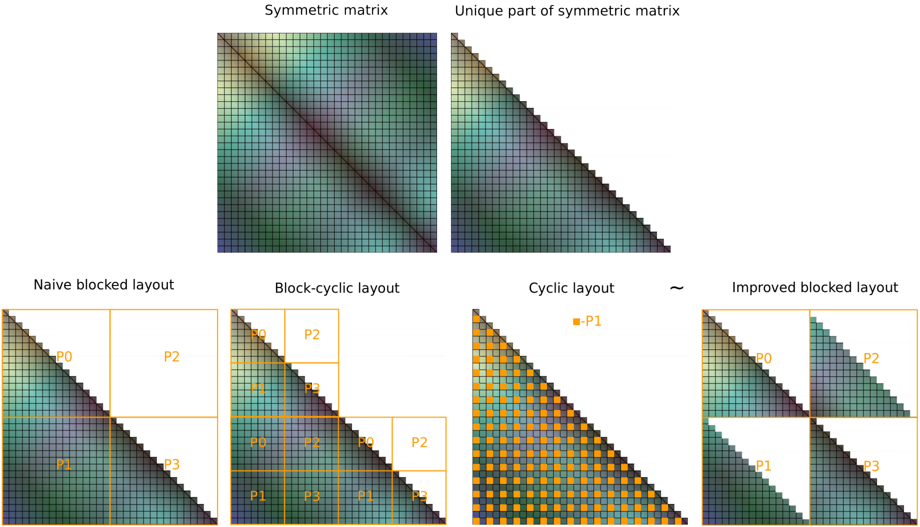


How does CTF achieve parallel scalability?

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

Communication avoiding algorithms

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

```
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

```
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"];
}
```

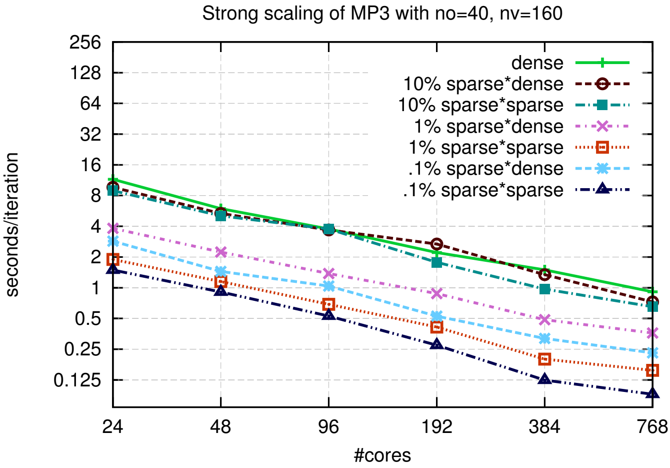
A case-study of a naive sparse MP3 code

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

```
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"]);
```

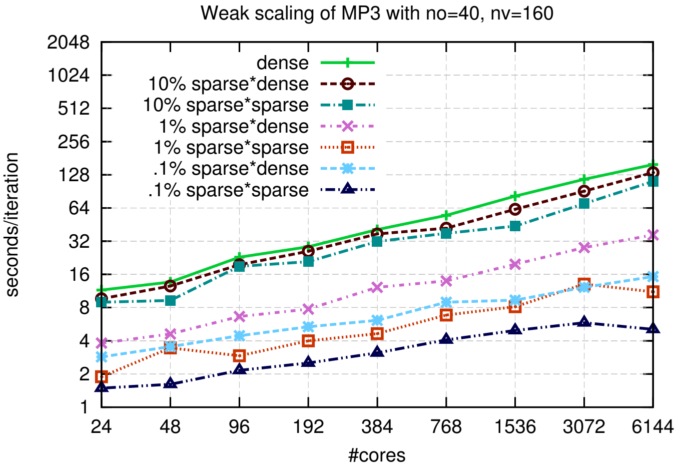
Strong scaling of CTF for naive sparse MP3

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



Weak scaling of CTF for naive sparse MP3

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]

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

- $A_{ij} b_j \neq A_{ji} b_i$ but $A_{ij}(b_i + b_j) = A_{ji}(b_j + b_i) \rightarrow (1/2)n^2$ multiplies
- partially-symmetric case: $A^{km}_{ij} = A^{km}_{ji}$

$$c^{kl}_i = \sum_{jm} A^{km}_{ij} b^{ml}_j = \quad \sum_j \left(\sum_m A^{km}_{ij}(b^{ml}_i + b^{ml}_j)\right) - \sum_m \left(\sum_j A^{km}_{ij}\right) b^{ml}_i$$

- let $Z^{kl}_{ij} = \sum_m A^{km}_{ij}(b^{ml}_i + b^{ml}_j)$ and observe $Z^{kl}_{ij} = Z^{kl}_{ji}$
- Z^{kl}_{ij} can be computed using $(1/2)n^5$ multiplies and $(1/2)n^5$ adds

[1] Noga, Valiron; Mol. Phys. 103:15-16, 2005. S., Demmel; Technical Report, ETH Zurich, 2015.

Symmetry preserving algorithms

By exploiting symmetry, reduce multiplies (but increase adds)[2]

- rank-2 vector outer product

$$C_{ij} = a_i b_j + a_j b_i \quad = \quad (a_i + a_j)(b_i + b_j) - a_i b_i - a_j b_j$$

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

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

- for symmetrized contraction of symmetric order $s + v$ and $v + t$ tensors

$$\frac{(s + t + v)!}{s!t!v!} \quad \text{fewer multiplies}$$

e.g. cases above are

- $s = 1, t = 1, v = 0 \rightarrow$ reduction by 2X
- $s = 1, t = 1, v = 1 \rightarrow$ reduction by 6X

[2] S., Demmel; Technical Report, ETH Zurich, 2015.

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 *nonsymmetric* 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

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)
- 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)

A stand-alone library for petascale tensor computations

Cyclops Tensor Framework (CTF)

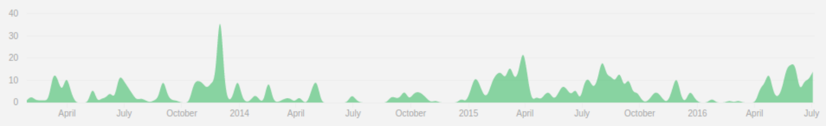
- distributed-memory symmetric/sparse tensors as C++ objects

```
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 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"]);
```

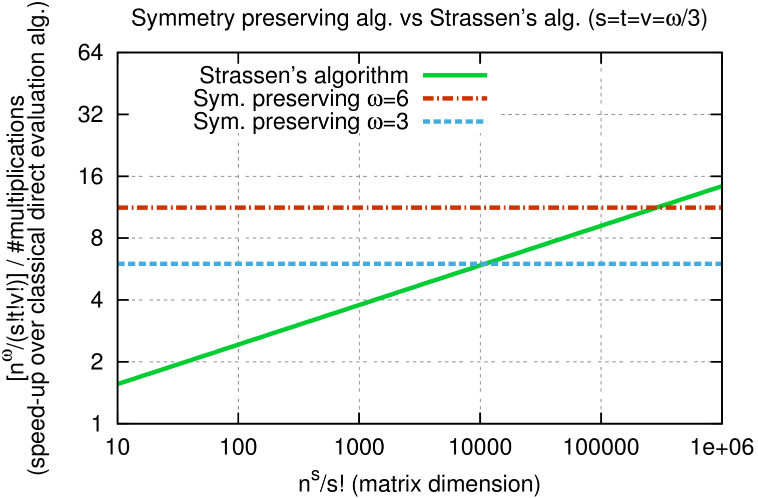
- development (1500 commits) since 2011, open source since 2013



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

Backup slides

Symmetry preserving algorithm vs Strassen's algorithm



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

$$\tilde{W}^{mn}_{ei} = v^{mn}_{ei} + \sum_f v^{mn}_{ef} t^f_i,$$

$$\tilde{W}^{mn}_{ij} = v^{mn}_{ij} + P^i_j \sum_e v^{mn}_{ie} t^e_j + \frac{1}{2} \sum_{ef} v^{mn}_{ef} \tau^{ef}_{ij},$$

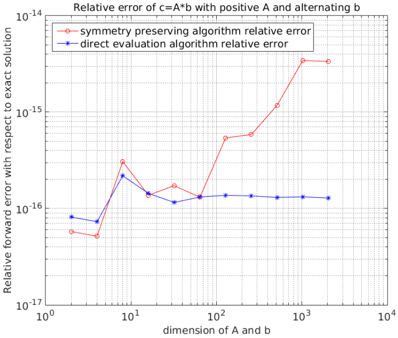
$$\tilde{W}^{am}_{ie} = v^{am}_{ie} - \sum_n \tilde{W}^{mn}_{ei} t^a_n + \sum_f v^{ma}_{ef} t^f_i + \frac{1}{2} \sum_{nf} v^{mn}_{ef} t^{af}_{in},$$

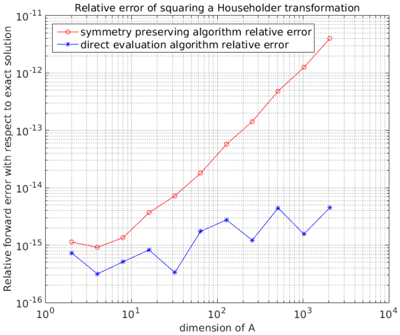
$$\tilde{W}^{am}_{ij} = v^{am}_{ij} + P^i_j \sum_e v^{am}_{ie} t^e_j + \frac{1}{2} \sum_{ef} v^{am}_{ef} \tau^{ef}_{ij},$$

$$z^a_i = f^a_i - \sum_m \tilde{F}^m_i t^a_m + \sum_e f^a_e t^e_i + \sum_{em} v^{ma}_{ei} t^e_m + \sum_{em} v^{ae}_{im} \tilde{F}^m_e + \frac{1}{2} \sum_{efm} v^{am}_{ef} \tau^{ef}_{im} - \frac{1}{2} \sum_{emn} \tilde{W}^{mn}_{ei} t^{ea}_{mn},$$

$$z^{ab}_{ij} = v^{ab}_{ij} + P^i_j \sum_e v^{ab}_{ie} t^e_j + P^a_b P^i_j \sum_{me} \tilde{W}^{am}_{ie} t^{eb}_{mj} - P^a_b \sum_m \tilde{W}^{am}_{ij} t^b_m + P^a_b \sum_e \tilde{F}^a_e t^{eb}_{ij} - P^i_j \sum_m \tilde{F}^m_i t^{ab}_{mj} + \frac{1}{2} \sum_{ef} v^{ab}_{ef} \tau^{ef}_{ij} + \frac{1}{2} \sum_{mn} \tilde{W}^{mn}_{ij} \tau^{ab}_{mn},$$

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