Cyclops Tensor Framework: reducing communication and eliminating load imbalance in massively parallel contractions

Edgar Solomonik[1], Devin Matthews[3],
Jeff Hammond[4], James Demmel[1,2]

[1] Department of EECS, UC Berkeley
[2] Department of Mathematics, UC Berkeley
[3] Department of Chemistry, UT Austin
[4] Leadership Computing Facility, Argonne National Laboratory

May 22, 2013

Outline

Electronic structure theory

Electronic structure calculations attempt to model the ground-state (and sometimes excited-state) energies of chemical systems, taking into account of quantum effects.
Density Functional Theory is the most common method

- cost is typically $O(n^3)$ for n electrons
- models system as a density functional, corrects for correlation
- good for metals and regular systems
- bad at molecules due to correlation effects on boundary

Coupled Cluster models electronic correlation explicitly

- cost is typically $O(n^{4+d})$, where $d \in \{2, 4, 6\}$
- the most accurate method used in practice

Coupled Cluster definition

Coupled Cluster (CC) is a method for computing an approximate solution to the time-independent Schrödinger equation of the form

$$\mathbf{H}|\Psi\rangle = E|\Psi\rangle,$$

CC rewrites the wave-function $|\Psi\rangle$ as an excitation operator $\hat{\mathbf{T}}$ applied to the Slater determinant $|\Phi_0\rangle$

$$|\Psi\rangle = e^{\hat{\mathbf{T}}}|\Phi_0\rangle$$

where $\hat{\mathbf{T}}$ is as a sum of $\hat{\mathbf{T}}_n$ (the n'th excitation operators)

$$\hat{\mathbf{T}}_{\mathrm{CCSD}} = \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2$$
$$\hat{\mathbf{T}}_{\mathrm{CCSDT}} = \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2 + \hat{\mathbf{T}}_3$$
$$\hat{\mathbf{T}}_{\mathrm{CCSDTQ}} = \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2 + \hat{\mathbf{T}}_3 + \hat{\mathbf{T}}_4$$

Coupled cluster (CCD) implementation

$e^{\hat{\mathbf{T}}_2}|\Phi_0\rangle$ turns into:

$$R_{ij}^{ab} = V_{ij}^{ab} + P(ia,jb)\Big[T_{ij}^{ae} I_e^b - T_{im}^{ab} I_j^m + \frac{1}{2} V_{ef}^{ab} T_{ij}^{ef} + \frac{1}{2} T_{mn}^{ab} I_{ij}^{mn} - T_{mj}^{ae} I_{ie}^{mb} - I_{ie}^{ma} T_{mj}^{eb} + (2T_{mi}^{ea} - T_{im}^{ea}) I_{ej}^{mb}\Big]$$

$$\begin{aligned}
I_b^a &= (-2V_{eb}^{mn} + V_{be}^{mn}) T_{mn}^{ea} \\
I_j^i &= (2V_{ef}^{mi} - V_{ef}^{im}) T_{mj}^{ef} \\
I_{kl}^{ij} &= V_{kl}^{ij} + V_{ef}^{ij} T_{kl}^{ef} \\
I_{jb}^{ia} &= V_{jb}^{ia} - \frac{1}{2} V_{eb}^{im} T_{jm}^{ea} \\
I_{bj}^{ia} &= V_{bj}^{ia} + V_{be}^{im}(T_{mj}^{ea} - \frac{1}{2} T_{mj}^{ae}) - \frac{1}{2} V_{be}^{mi} T_{mj}^{ae}
\end{aligned}$$

Tensor contractions

We define a tensor contraction between $\mathbf{A} \in \mathbb{R}^{\otimes k}$, $\mathbf{B} \in \mathbb{R}^{\otimes l}$ into $\mathbf{C} \in \mathbb{R}^{\otimes m}$ as

$$c_{i_1 \ldots i_m} = \sum_{j_1 \ldots j_{k+l-m}} a_{i_1 \ldots i_{m-l} j_1 \ldots j_{k+l-m}} \cdot b_{j_1 \ldots j_{k+l-m} i_{m-l+1} \ldots i_m}$$

Tensor contractions reduce to matrix multiplication via index folding (let $[ijk]$ denote a group of 3 indices folded into one),

$$c_{[i_1 \ldots i_{m-l}],[i_{m-l+1} \ldots i_m]} = \sum_{[j_1 \ldots j_{k+l-m}]} a_{[i_1 \ldots i_{m-l}],[j_1 \ldots j_{k+l-m}]} \cdot b_{[j_1 \ldots j_{k+l-m}],[i_{m-l+1} \ldots i_m]}$$

so here $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ can be treated simply as matrices.

Tensor symmetry

Tensors can have symmetry e.g.

$$a_{(ij)k} = a_{(ji)k} \quad \text{or} \quad a_{(ij)k} = -a_{(ji)k}$$

I will denote symmetric groups of indices as $(ab...)$. We now might face contractions like

$$c_{(ij)kl} = \sum_{pq} a_{(ij)(pq)} \cdot b_{(pk)(ql)}$$

where the computational graph G can be thought of as a 6D tensor with entries $g_{(ij)klpq} = (c_{(ij)kl}, a_{(ij)(pq)}, b_{(pk)(ql)})$. There are two things that can happen to symmetries during a contraction:

- preserved, e.g. $g_{(ij)klpq} = g_{(ji)klpq}$
- broken, e.g. $b_{(pk)(\mathbf{ql})} = b_{(pk)(\mathbf{lq})}$ but $g_{(ij)k\mathbf{l}p\mathbf{q}} \neq g_{(ij)k\mathbf{q}p\mathbf{l}}$

NWChem approach to contractions

A high-level description of NWChem's algorithm for tensor contractions:

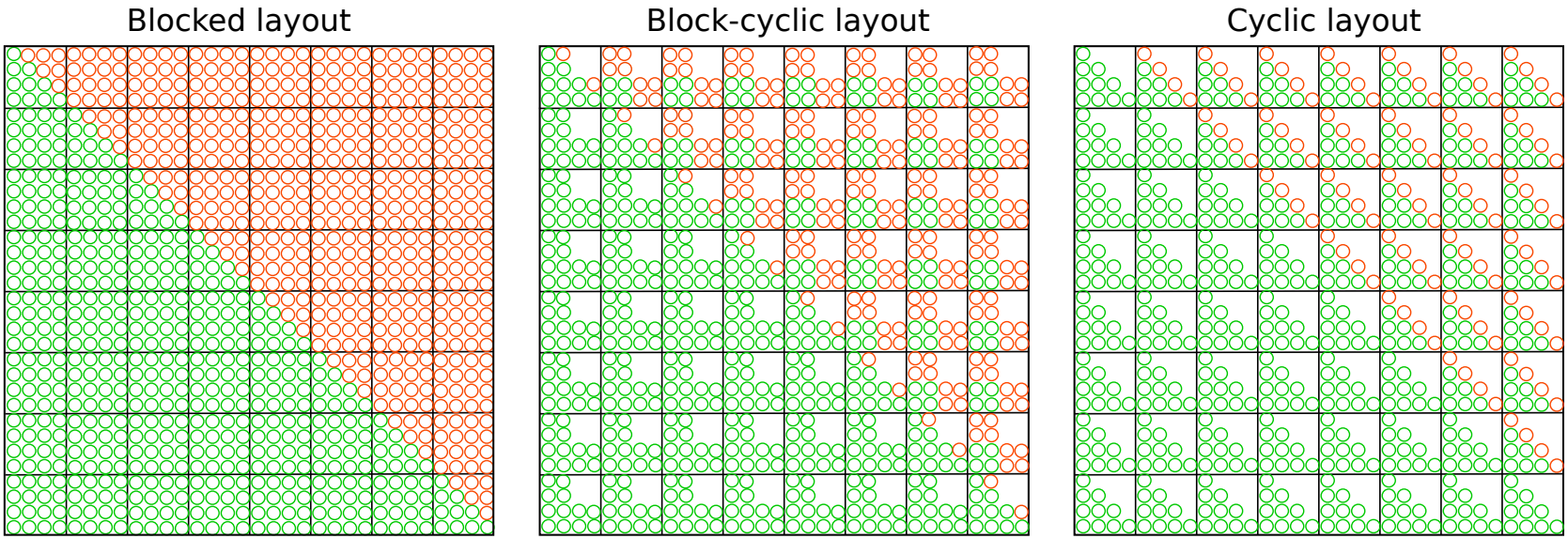
- data layout is abstracted away by the Global Arrays framework
- Global Arrays uses one-sided communication for data movement
- packed tensors are stored in blocks
- for each contraction, each process does a subset of the block contractions
- each block is transposed and unpacked prior to contraction
- dynamic load balancing is employed among processors

Cyclops Tensor Framework (CTF) approach to contractions

A high-level description of CTF's algorithm for tensor contractions:

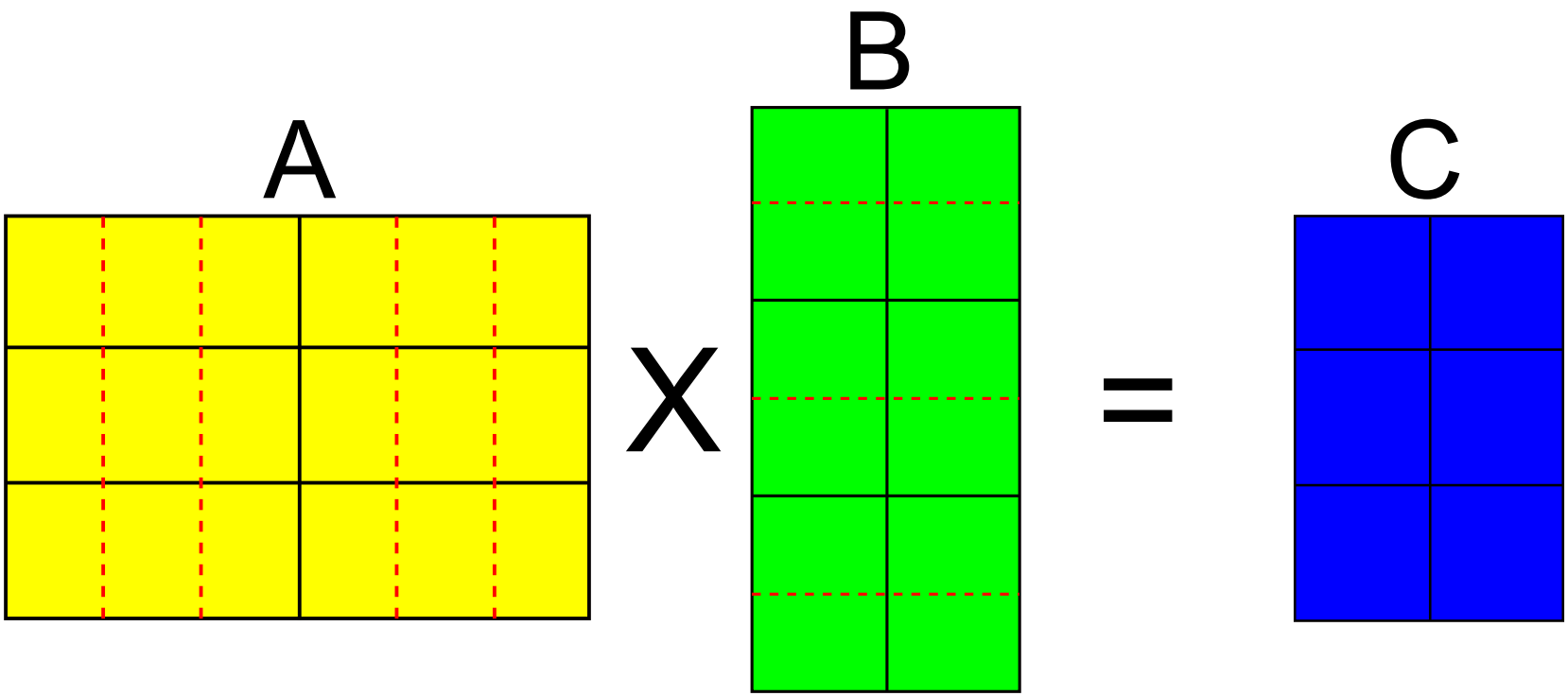
- packed tensors are decomposed cyclically among toroidal processor grids
- MPI collectives are used for all communication
- for each contraction, a distributed layout is selected based on internal performance models
- performance model considers all possible execution paths
- before contraction, tensors are redistributed to a new layout
- if there is enough memory, the tensors are (partially) unpacked
- all preserved symmetries and non-symmetric indices are folded in preparation for matrix multiplication
- nested distributed matrix multiply algorithms are used to perform the contraction in a load-balanced manner

Blocked vs block-cyclic vs cyclic decompositions

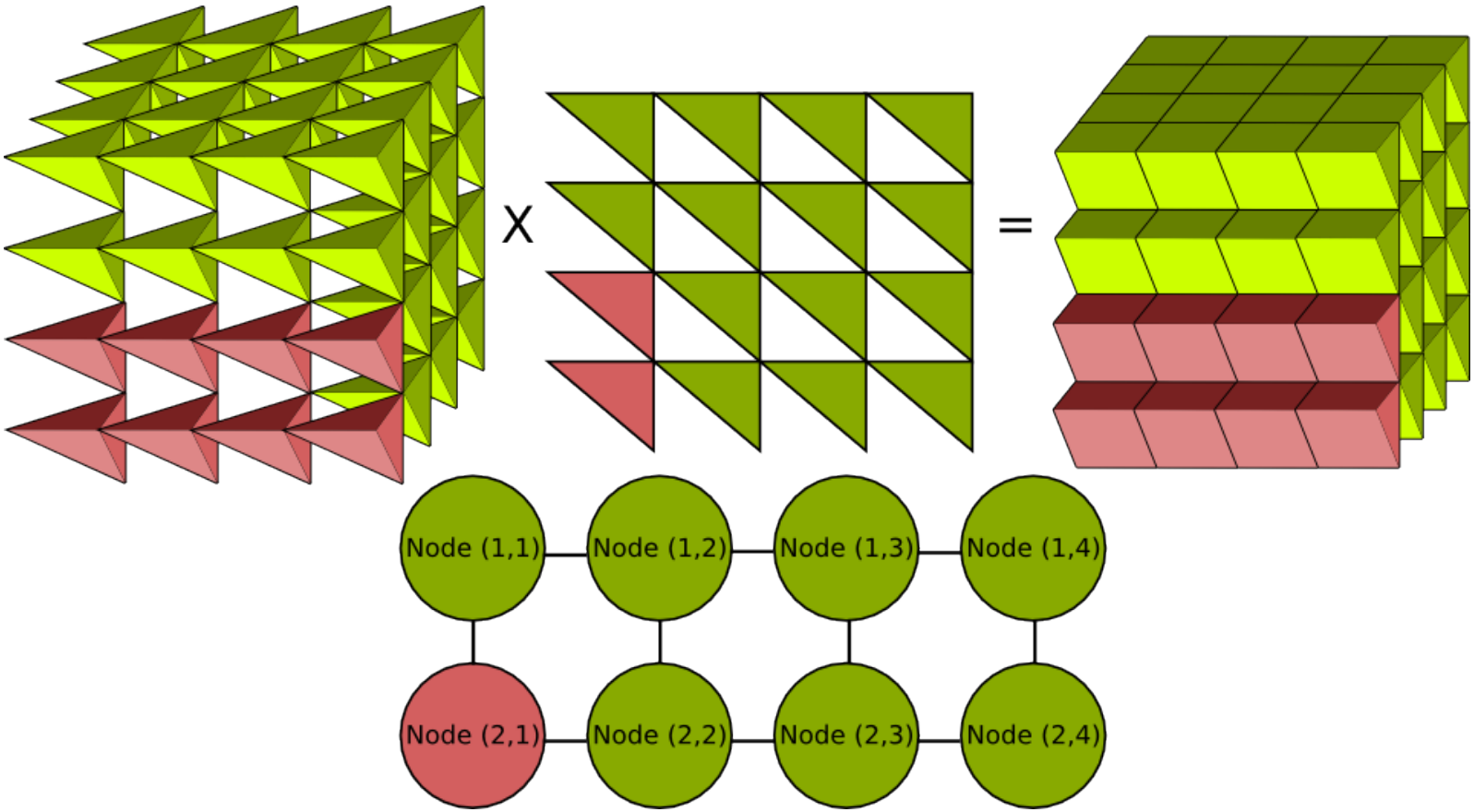


Virtualization

Matrix multiply on 2x3 processor grid. Red lines represent virtualized part of processor grid. Elements assigned to blocks by cyclic phase.



3D tensor mapping



A simple data layout

Replicated, distributed nonsymmetric tensor (processor grid)
of nonsymmetric tensors (virtual grid)
of symmetric tensors (folded broken symmetries)
of matrices (unfolded broken and folded preserved symmetries)

The layout typically changes for each tensor between each contraction.

Tensor redistribution

Our symmetric tensor data layout has a global ordering and a local ordering

- the local data is not in global order
- cannot compute local data index from global index
- cannot compute global data index from local index
- can iterate over local data and obtain global index
- can iterate over global data and obtain local index

Given these constraints, it is simplest to compute the global index of each piece of data and sort.

General data redistribution

We use an algorithm faster than sorting for redistribution

1. iterate over the local data and count where the data must be sent
2. communicate counts and compute prefix sums to obtain offsets
3. iterate over the local data **in global order** and bin it
4. exchange the data (MPI all to all v)
5. iterate over the new local data **in global order** and retrieve it from bins

This method is much faster, because it does not explicitly form and communicate keys for the data.

Threaded general redistribution

In order to hide memory latency and reduce integer operations it is imperative to thread the redistribution kernel

- prefix sums and counts are trivial to thread
- to thread the iterator over data, we must give each thread different global indices
- each thread moves the local data corresponding to a global index partition, preserving the ordering

Interface and code organization

- the CTF codebase is currently 31,345 lines of C++ code
- CTF provides functionality for general tensor contractions, including a contraction domain-specific language (DSL)
- Aquarius is a quantum chemistry package being developed by Devin Matthews
  - uses CTF for parallel tensor contraction execution
  - provides a DSL for spin-integrated tensor contractions
  - gives implementations of CC methods including other necessary components (e.g. SCF)
- efforts are underway to also integrate CTF into the QChem package

CCSD code using our domain specific language

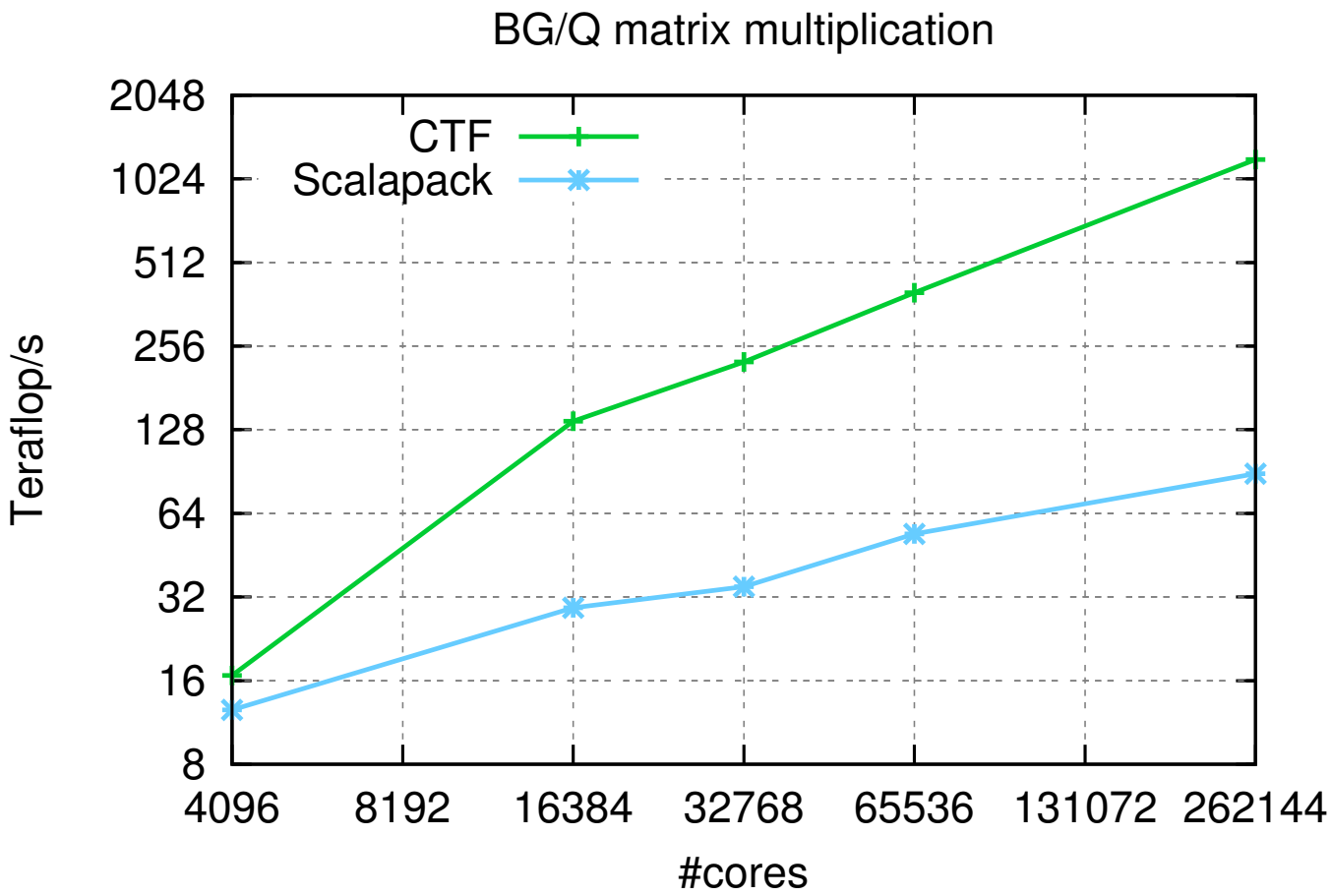
```
FVO["me"] = VABIJ["efmn"]*T1["fn"];
FVV["ae"] = -0.5*VABIJ["femn"]*T2["famn"];
FVV["ae"] -= FVO["me"]*T1["am"];
FVV["ae"] += VABCI["efan"]*T1["fn"];
FOO["mi"] = 0.5*VABIJ["efnm"]*T2["efni"];
FOO["mi"] += FVO["me"]*T1["ei"];
FOO["mi"] += VIJKA["mnif"]*T1["fn"];
WMNIJ["mnij"] = VIJKL["mnij"];
WMNIJ["mnij"] += 0.5*VABIJ["efmn"]*Tau["efij"];
WMNIJ["mnij"] += VIJKA["mnie"]*T1["ej"];
WMNIE["mnie"] = VIJKA["mnie"];
WMNIE["mnie"] += VABIJ["femn"]*T1["fi"];
WAMIJ["amij"] = VIJKA["jima"];
WAMIJ["amij"] += 0.5*VABCI["efam"]*Tau["efij"];
WAMIJ["amij"] += VAIBJ["amej"]*T1["ei"];
WMAEI["maei"] = -VAIBJ["amei"];
WMAEI["maei"] += 0.5*VABIJ["efmn"]*T2["afin"];
WMAEI["maei"] += VABCI["feam"]*T1["fi"];
WMAEI["maei"] -= WMNIE["nmie"]*T1["an"];
Z1["ai"] = 0.5*VABCI["efam"]*Tau["efim"];
Z1["ai"] -= 0.5*WMNIE["mnie"]*T2["aemn"];
Z1["ai"] += T2["aeim"]*FVO["me"];
Z1["ai"] -= T1["em"]*VAIBJ["amei"];
Z1["ai"] -= T1["am"]*FOO["mi"];
Z2["abij"] = VABIJ["abij"];
Z2["abij"] += FVV["af"]*T2["fbij"];
Z2["abij"] -= FOO["ni"]*T2["abnj"];
Z2["abij"] += VABCI["abej"]*T1["ei"];
Z2["abij"] -= WAMIJ["mbij"]*T1["am"];
Z2["abij"] += 0.5*VABCD["abef"]*Tau["efij"];
Z2["abij"] += 0.5*WMNIJ["mnij"]*Tau["abmn"];
Z2["abij"] += WMAEI["maei"]*T2["ebmj"];
E1["ai"]    = Z1["ai"]  *D1["ai"];
E2["abij"]  = Z2["abij"]*D2["abij"];
E1["ai"]   -= T1["ai"];
E2["abij"] -= T2["abij"];
T1["ai"]   += E1["ai"];
T2["abij"] += E2["abij"];

Tau["abij"]  = T2["abij"];
Tau["abij"] += 0.5*T1["ai"]*T1["bj"];

E_CCSD = 0.25*scalar(VABIJ["efmn"]*Tau["efmn"]);
```

Sequential and multi-threaded performance comparison

CCSD performance on a Xeon E5620, single threaded, Intel MKL. Entries are average time for one CCSD iteration, for the given number of virtual (n_v) and occupied (n_o) orbitals (electrons).

		$n_v = 110$ $n_o = 5$	$n_v = 94$ $n_o = 11$	$n_v = 71$ $n_o = 23$
NWChem	1 thread	6.80 sec	16.8 sec	49.1 sec
CTF	1 thread	23.6 sec	32.5 sec	59.8 sec
NWChem	8 threads	5.21 sec	8.60 sec	18.1 sec
CTF	8 threads	9.12 sec	9.37 sec	18.5 sec

A simple tensor contraction



Complex matrix multiplication (ZGEMM) of 32K-by-32K matrices benefits greatly from topology-aware mapping on BG/Q.

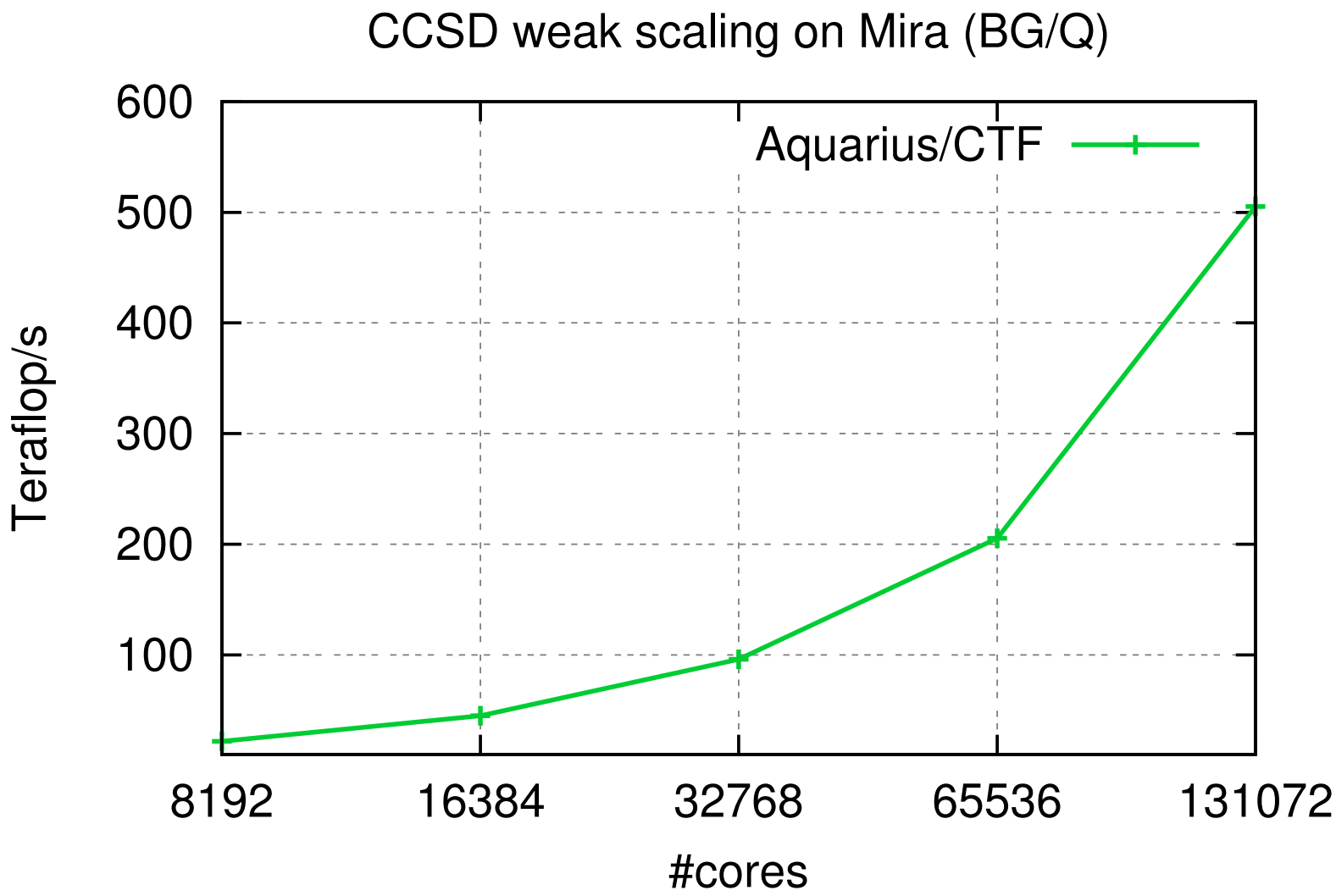
Comparison with NWChem on Cray XE6

CCSD iteration time on 64 nodes of Hopper:

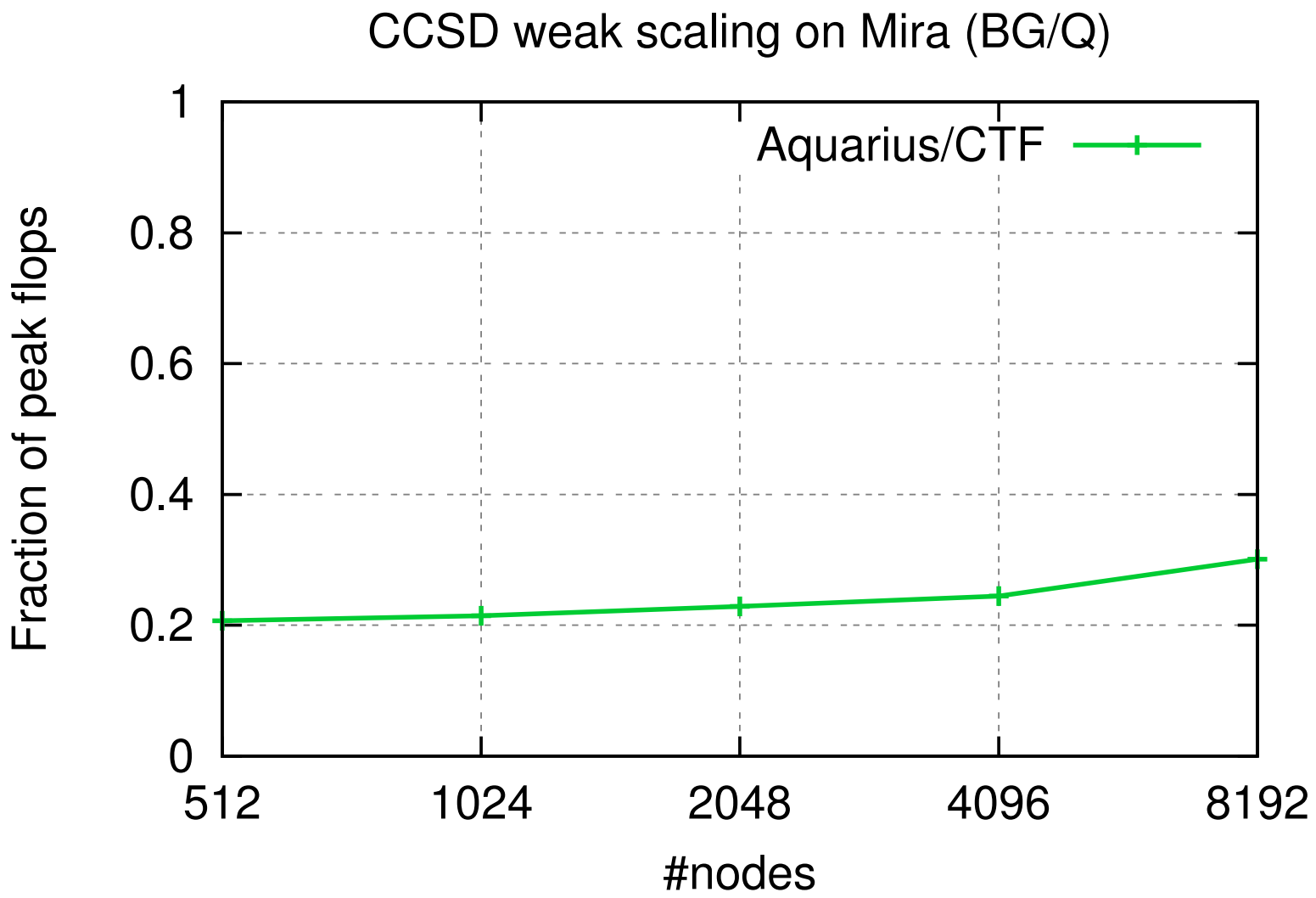
system	# electrons	# orbitals	CTF	NWChem
w5	25	205	14 sec	36 sec
w7	35	287	90 sec	178 sec
w9	45	369	127 sec	-
w12	60	492	336 sec	-

On 128 nodes, NWChem completed w9 in 223 sec, CTF in 73 sec.

Blue Gene/Q up to 1250 orbitals, 250 electrons



Coupled Cluster efficiency on Blue Gene/Q



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
n_v-orbitals, n_o-electrons, p-processors, M-local memory size

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

Performance breakdown on Cray XE6

Performance data for a CCSD iteration with 100 electrons and 500 orbitals on 256 nodes of Hopper
4 processes per node, 6 threads per process
Total time: 9 mins
v-orbitals, o-electrons

kernel	% of time	complexity	architectural bounds
DGEMM	21% $\Downarrow$ 24%	$O(v^4o^2/p)$	flops/mem bandwidth
broadcasts	32% $\Uparrow$ 12%	$O(v^4o^2/p\sqrt{M})$	multicast bandwidth
prefix sum	7% $\Downarrow$ 3%	$O(p)$	allreduce bandwidth
data packing	10% $\Uparrow$ 3%	$O(v^2o^2/p)$	integer ops
all-to-all-v	8%	$O(v^2o^2/p)$	bisection bandwidth
tensor folding	4%	$O(v^2o^2/p)$	memory bandwidth

Future Work

Cyclops Tensor Framework

- CCSDT and CCSDTQ implementation
- EOM-CC (excited state methods)
- sparse tensor support
- support for matlab-like tensor cuts (e.g. $A[2:10, 3:15, 10:15]$)
- improvements to the handling of broken symmetries

For more information and complete code see `ctf.cs.berkeley.edu`

Backup slides

Cyclic decomposition in CTF

Cyclical distribution is fundamental to CTF, hence the name Cyclops (cyclic-operations).
Given a vector **v** of length n on p processors

- in a blocked distribution process p_i owns $\{v_{i\cdot n/p+1}, \ldots v_{(i+1)\cdot n/p}\}$
- in a cyclic distribution process p_i owns $\{v_i, v_{2i}, \ldots v_{(n/p)i}\}$

A cyclic distribution is associated with a phase along each dimension (for the vector above this was p). The main advantage from this distribution is that each subtensor can retain packed structure with only minimal padding.
CTF assumes all subtensor symmetries have index relations of the form $\leq$ and not $<$, so in effect, diagonals are stored for skew-symmetric tensors.

Sequential tensor contractions

A cyclic distribution provides a vital level of abstraction, because each subtensor contraction becomes a packed contraction of the same sort as the global tensor contraction but of smaller size. Given a sequential packed contraction kernel, CTF can parallelize it automatically. Further, because each subcontraction is the same, the workload of each processor is the same. The actual sequential kernel used by CTF employs the following steps

1. if there is enough memory, unpack broken symmetries
2. perform a nonsymmetric transpose, to make all indices of non-broken symmetry be the leading dimensions
3. use a naive kernel to iterate though indices with broken symmetry and call BLAS GEMM for the leading dimensions

Multidimensional processor grids

CTF supports tensors and processor grids of any dimension because mapping a symmetric tensor to a processor grid of the same dimension preserves symmetric structure with minimal virtualization and padding. Processor grids are defined by

- a base grid, obtained from the physical topology or from factorizing the number of processors
- folding all possible combinations of adjacent processor grid dimensions

Tensors are contracted on higher dimensional processor grids by

- mapping an index shared by two tensors in the contraction to different processor grid dimensions
- running a distributed matrix multiplication algorithm for each such 'mismatched' index
- replicating data along some processor dimensions 'a la 2.5D'

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

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

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

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

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

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

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