Programming models for quantum chemistry applications

Jeff Hammond, James Dinan,
Edgar Solomonik and Devin Matthews

Argonne LCF and MCS, UC Berkeley and UTexas

8 May 2012

Abstract (for posterity)

Quantum chemistry applications have long been associated with irregular communication patterns and load-balancing, which motivated the development of Global Arrays (GA), the Distributed Data Interface (DDI) and, more recently, the Super Instruction Assembly Language (SIAL), which form the basis for essentially all parallel implementations of wavefunction-based quantum chemistry methods, as found in codes like NWChem, GAMESS, ACES III and others. In this talk, the mathematical and algorithmic fundamentals of a popular family of quantum chemistry methods known as coupled-cluster methods and various parallelization schemes associated with their implementation for supercomputers. First, the aforementioned runtimes (GA, DDI, SIAL) will be compared to Charm++ on various axes, including asynchronous communication, dynamic load-balancing, data decomposition, and topology awareness. Second, we describe the Cyclops Tensor Framework, which is a completely new approach to coupled-cluster methods that uses some of the key concepts found in Charm++. Finally, a case is made for using Charm++ to implement reduced-scaling coupled cluster methods.

Atomistic simulation in chemistry

1. *classical* molecular dynamics (MD) with empirical potentials
2. *quantum* molecular dynamics based upon **density**-function theory (DFT)
3. *quantum* chemistry with **wavefunctions** e.g. perturbation theory (PT), coupled-cluster (CC) or quantum monte carlo (QMC).

Classical molecular dynamics

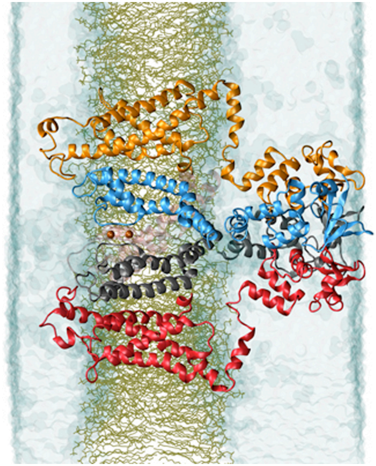


Image courtesy of Benoît Roux via ALCF.

- Solves Newton's equations of motion with empirical terms and classical electrostatics.
- Size: 100K-10M atoms
- Time: 1-10 ns/day
- Scaling: $\sim N_{atoms}$
- Math: N-body

Data from K. Schulten, et al. "Biomolecular modeling in the era of petascale computing." In D. Bader, ed., *Petascale Computing: Algorithms and Applications*.

Car-Parrinello molecular dynamics

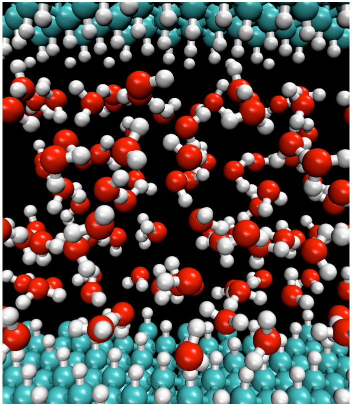


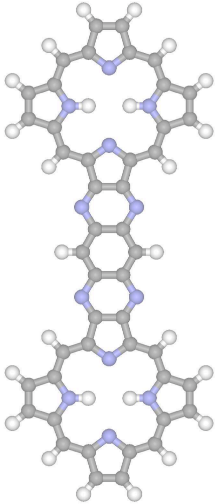
Image courtesy of Giulia Galli via ALCF.

- Forces obtained from solving an approximate single-particle Schrödinger equation.
- Size: 100-1000 atoms
- Time: 0.01-1 ps/day
- Scaling: $\sim N_{el}^{x}$ (x=1-3)
- Math: FFT, eigensolve.

F. Gygi, *IBM J. Res. Dev.* **52**, 137 (2008); E. J. Bylaska et al. *J. Phys.: Conf. Ser.* **180**, 012028 (2009).

Wavefunction theory

,



- MP2 is second-order PT and is accurate via magical cancellation of error.
- CC is infinite-order solution to many-body Schrödinger equation truncated via clusters.
- QMC is Monte Carlo integration applied to the Schrödinger equation.
- Size: 10-100 atoms, maybe 100-1000 atoms with MP2.
- Time: N/A (LOL)
- Scaling: $\sim N_{bf}^{x}$ (x=4-7)
- Math: DLA (tensors)

Image courtesy of Karol Kowalski and Niri Govind.

Basic Quantum Chemistry

The Fock build

Pseudocode for $F_{ij} = V^{ij}_{kl} D_{kl}$:

```
for i,j,k,l:
  if symmetry_criteria(i,j,k,l):
    if dynamic_load_balancer(me):
      if schwartz_criteria(i,j,k,l):
        Get block d(k,l) from D
        Compute v(i,j,k,l)
        f(i,j) += v(i,j,k,l) * d(k,l)
    Accumulate f(i,j) to F
```

Time to compute $v(i,j,k,l)$ varies wildly, Schwartz screening adds irregularity.

The SCF iterations

- Build Fock matrix, solve generalized eigenvalue problem, repeat until converged.
- Direct algorithms replaced out-of-core storage of V (Almlöf).
- Replicated F with allreduce is now common but not weak-scalable.
- Until MPI-3 is widely available, dynamic load-balancing is unpleasant.

Enter magic runtimes

- Global Arrays (GA) emerged before MPI-1 was settled, inspired by Linda and building upon TCGMSG, and was codesigned with NWChem *from the beginning*. ARMCI emerged later.
- DDI is a reimplementation of GA for GAMESS but lacks math abstractions (e.g. ScaLAPACK wrappers) that are probably unappreciated by most computer scientists.
- SIAL emerged much later as part of ACES III. Adopts many concepts from TCE but uses DSL-based abstraction to reduce runtime demands (MPI-1 and polling but could easily use ARMCI).

Magic runtime properties I

- Asynchrony: GA/ARMCI true passive-target progress, supports nonblocking; DDI has half the processes (oversubscribed 2x) in MPI polling loop; SIAL, like UPC and Charm++, doesn't need strong progress.
- Interoperability: GA/ARMCI works fine with MPI (dupes world now); DDI (ab)uses world; SIAL DSL seems incompatible with MPI but this is solvable.
- Load-balancing: GA and DDI use same (dumb) NXTVAL-style DLB, although Scioto and now Tascel address this. SIAL has both static and dynamic algorithms.

Magic runtime properties II

- Hierarchical parallelism: no support for topology-aware anything except for intra/internode. To be fair, `MPI_{Cart,Graph}_create` aren't perfect.
- Data-distribution: GA supports standard, user-defined and chemistry-specific distributions; DDI was 1D last time I looked; SIAL supernumber concept is basically identical to TCE tiling and hashing.
- Phases: GA doesn't support MSA-style explicit epochs (yet) but user can implement caching (QMCPACK/Einspline and Jim's IPDPS 2012) and replication. Breaking BSP via GA sync bypass is *special*. . .

Coupled-cluster theory

Coupled-cluster theory

The coupled–cluster (CC) wavefunction ansatz is

$$|CC\rangle = e^T|HF\rangle$$

where $T = T_1 + T_2 + \cdots + T_n$.

T is an excitation operator which promotes n electrons from occupied orbitals to virtual orbitals in the Hartree-Fock Slater determinant.

Inserting $|CC\rangle$ into the Schödinger equation:

$$\hat{H}e^T|HF\rangle = E_{CC}e^T|HF\rangle \qquad \hat{H}|CC\rangle = E_{CC}|CC\rangle$$

Coupled-cluster theory

$$
\begin{aligned}
|CC\rangle &= \exp(T)|0\rangle \\
T &= T_1 + T_2 + \cdots + T_n \quad (n \ll N) \\
T_1 &= \sum_{ia} t_i^a \hat{a}_a^\dagger \hat{a}_i \\
T_2 &= \sum_{ijab} t_{ij}^{ab} \hat{a}_a^\dagger \hat{a}_b^\dagger \hat{a}_j \hat{a}_i \\
|\Psi_{CCD}\rangle &= \exp(T_2)|\Psi_{HF}\rangle \\
&= (1 + T_2 + T_2^2)|\Psi_{HF}\rangle \\
|\Psi_{CCSD}\rangle &= \exp(T_1 + T_2)|\Psi_{HF}\rangle \\
&= (1 + T_1 + \cdots + T_1^4 + T_2 + T_2^2 + T_1 T_2 + T_1^2 T_2)|\Psi_{HF}\rangle
\end{aligned}
$$

Coupled-cluster theory

Projective solution of CC:

$$\begin{aligned} E_{CC} &= \langle HF|e^{-T}He^{T}|HF\rangle \\ 0 &= \langle X|e^{-T}He^{T}|HF\rangle \quad (X = S, D, \ldots) \end{aligned}$$

CCD is:

$$\begin{aligned} E_{CC} &= \langle HF|e^{-T_2}He^{T_2}|HF\rangle \\ 0 &= \langle D|e^{-T_2}He^{T_2}|HF\rangle \end{aligned}$$

CCSD is:

$$\begin{aligned} E_{CC} &= \langle HF|e^{-T_1-T_2}He^{T_1+T_2}|HF\rangle \\ 0 &= \langle S|e^{-T_1-T_2}He^{T_1+T_2}|HF\rangle \\ 0 &= \langle D|e^{-T_1-T_2}He^{T_1+T_2}|HF\rangle \end{aligned}$$

Notation

$$\begin{aligned} H &= H_1 + H_2 \\ &= F + V \end{aligned}$$

F is the Fock matrix. CC only uses the diagonal in the canonical formulation.

V is the fluctuation operator and is composed of two-electron integrals as a 4D array.

V has 8-fold permutation symmetry in V_{pq}^{rs} and is divided into six blocks: V_{ij}^{kl}, V_{ij}^{ka}, V_{ia}^{jb}, V_{ij}^{ab}, V_{ia}^{bc}, V_{ab}^{cd}.

Indices $i, j, k, \ldots$ $(a, b, c, \ldots)$ run over the occupied (virtual) orbitals.

CCD Equations

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

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

Turning CC into GEMM 1

Some tensor contractions are trivially mapped to GEMM:

$$\begin{aligned} I^{ij}_{kl} &\mathrel{+}= V^{ij}_{ef} T^{ef}_{kl} \\ I^{(ij)}_{(kl)} &\mathrel{+}= V^{(ij)}_{(ef)} T^{(ef)}_{(kl)} \\ I^{b}_{a} &\mathrel{+}= V^{b}_{c} T^{c}_{a} \end{aligned}$$

Other contractions require reordering to use BLAS:

$$\begin{aligned} I^{ia}_{bj} &\mathrel{+}= V^{im}_{be} T^{ea}_{mj} \\ I_{bj,ia} &\mathrel{+}= V_{be,im} T_{mj,ea} \\ J_{bi,ja} &\mathrel{+}= W_{bi,me} U_{me,ja} \\ J^{ja}_{bi} &\mathrel{+}= W^{me}_{bi} U^{ja}_{me} \\ J^{(ja)}_{(bi)} &\mathrel{+}= W^{(me)}_{(bi)} U^{(ja)}_{(me)} \\ J^{z}_{x} &\mathrel{+}= W^{y}_{x} U^{z}_{y} \end{aligned}$$

Turning CC into GEMM 2

Reordering can take as much time as GEMM in the node-level implementation (e.g. NWChem). Why?

Routine	flops	mops	pipelined
GEMM	$O(mnk)$	$O(mn + mk + kn)$	yes
reorder	0	$O(mn + mk + kn)$	no

Increased memory bandwidth on GPU makes reordering less expensive (compare matrix transpose).

(There is a chapter in my thesis with profiling results and more details if anyone cares.)

Tensor Contraction Engine

Tensor Contraction Engine

What does it do?

1. GUI input quantum many-body theory e.g. CCSD.
2. Operator specification of theory (as in a theory paper).
3. Apply Wick's theory to transform operator expressions into array expressions (as in a computational paper).
4. Transform input array expression to operation tree using many types of optimization (i.e. compile).
5. Generate F77/GA/NXTVAL implementation for NWChem or C++/MemoryGrp for MPQC or F90/.. for UTChem.

Developer can intercept at various stages to modify theory, algorithm or implementation (may be painful).

TCE Input

We get 73 lines of serial F90 or 604 lines of parallel F77 from this:

```
1/1 Sum(g1 g2 p3 h4) f(g1 g2) t(p3 h4) {g1+ g2}{p3+ h4}
1/4 Sum(g1 g2 g3 g4 p5 h6) v(g1 g2 g3 g4) t(p5 h6) {g1+
g2+ g4 g3}{p5+ h6}
1/16 Sum(g1 g2 g3 g4 p5 p6 h7 h8) v(g1 g2 g3 g4) t(p5 p6
h7 h8) {g1+ g2+ g4 g3}{p5+ p6+ h8 h7}
1/8 Sum(g1 g2 g3 g4 p5 h6 p7 h8) v(g1 g2 g3 g4) t(p5 h6)
t(p7 h8) {g1+ g2+ g4 g3}{p5+ h6} {p7+ h8}
```

LaTeX equivalent of the first term:

$$\sum_{g_1,g_2,p_3,h_4} f_{g_1,g_2} t_{p_3,h_4} \{g_1^\dagger g_2\}\{p_3^\dagger h_4\}$$

Summary of TCE module

```
http://cloc.sourceforge.net v 1.53  T=30.0 s
---------------------------------------------
Language      files  blank  comment     code
---------------------------------------------
Fortran 77    11451   1004   115129  2824724
---------------------------------------------
SUM:          11451   1004   115129  2824724
---------------------------------------------
```

Perhaps $<$25 KLOC are hand-written; $\sim$100 KLOC is utility code following TCE data-parallel template.

Expansion from TCE input to massively-parallel F77 is $\sim$ 200 (drops with language abstractions).

TCE template

Pseudocode for $R_{i,j}^{a,b} = T_{i,j}^{c,d} * V_{a,b}^{c,d}$:

```
for i,j in occupied blocks:
   for a,b in virtual blocks:
      for c,d in virtual blocks:
         if symmetry_criteria(i,j,a,b,c,d):
            if dynamic_load_balancer(me):
               Get block t(i,j,c,d) from T
               Permute t(i,j,c,d)
               Get block v(a,b,c,d) from V
               Permute v(a,b,c,d)
               r(i,j,c,d) += t(i,j,c,d) * v(a,b,c,d)
      Permute r(i,j,a,b)
      Accumulate r(i,j,a,b) block to R
```

TCE profile

`ccsd_t2_8` (DGEMM-like):

timer	min	max	avg
dgemm	68.605	91.296	81.282
ga_acc	0.042	0.070	0.050
ga_get	5.845	7.779	6.679
nxtask	0.012	28.710	13.638
tce_sort4	6.184	8.174	7.347
tce_sortacc4	7.892	11.042	9.290

nxtask timings are a Blue Gene/P artifact and should be ignored.

Observations about the TCE template

1. Blocking get means no overlap.
   (fix with double buffering but memory usage increases)
2. Dynamic load balancing is **global** shared counter.
   (see next talk)
3. Get+Permute of t(i,j,c,d)/v(a,b,c,d) for all (a,b)/(i,j).
   (data-affinity + reuse or global permute)
4. Permute is a nasty operation.
   (need fused contraction at DGEMM speed)

There are an uncountable number of missing optimizations in any scientific code. NWChem is certainly not special in this regard.

Some of these issues hurt more on Blue Gene than COTS. . .

TCE Template for MMM

Pseudocode for $C^i_j = A^i_k * B^k_j$:

```
for i in I blocks:
   for j in J blocks:
      for k in K blocks:
          if dynamic_load_balancer(me):
             Get block a(i,k) from A
             Get block b(k,j) from B
             c(i,j) += a(i,k) * b(k,j)
      Accumulate c(i,j) block to C
```

This is clearly not the best way to do MMM!

A better way

- Adopt the TCE node kernel approach in parallel: tensor contraction = permute + matmul.
- Parallel permute = parallel sorting = well-understood.
- Parallel matmul = well-understood.

Therefore, parallel tensor contractions are solved, up to the implementation details and future algorithm developments in sorting and matmul.

All existing TCE technology for higher-level optimizations are still valid. Our abstraction provides a much cleaner performance model for TCE to target.

Cyclops Tensor Framework

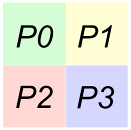
Edgar Solomonik (PPL alum) develops CTF.

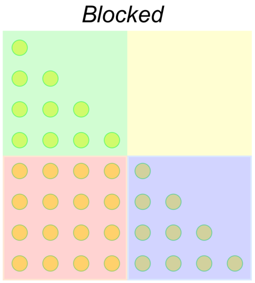
Upper-level interface and CC codes by Devin Matthews (UTexas).

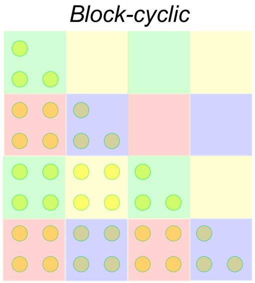
Cyclops Tensor Framework

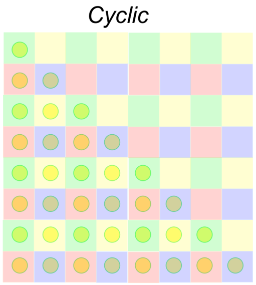
- Can we apply the absolute state-of-the-art in dense matrix algorithms to tensors without much difficulty (and thereby capture all previous develops with respect to communication minimization, topology-awareness, numerical precision and fault-detection)?
- Can we *completely eliminate* the irregularity associated with permutation symmetry and irregular blocking that create a signficant load-balancing challenge?
- Can we do it without sacrificing any of the productivity of high-level abstractions as found in TCE?

Data decompositions



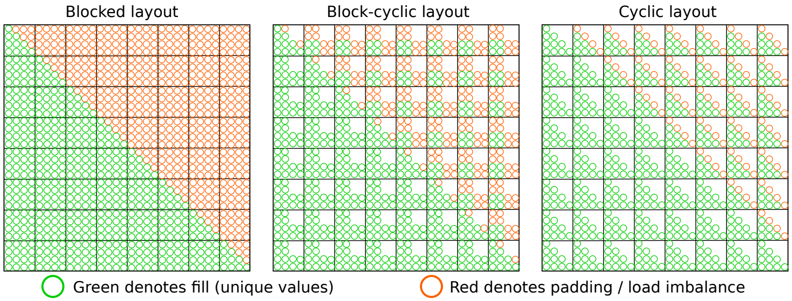






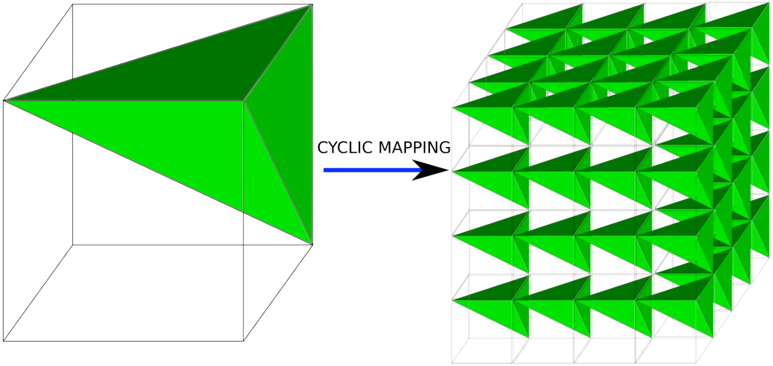
Data decompositions

Triangle of squares or square of triangles?
i.e. DGEMM vs. topo (triangle network topology anyone?)


Blocked layout
Block-cyclic layout
Cyclic layout
◯ Green denotes fill (unique values)
◯ Red denotes padding / load imbalance

If you fold the triangle into a rectangle, what does the *communication* topology look like?

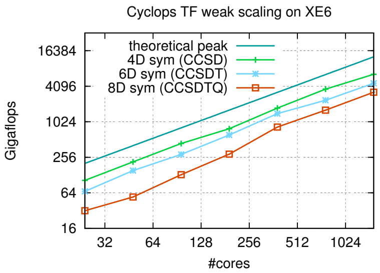
Data decompositions

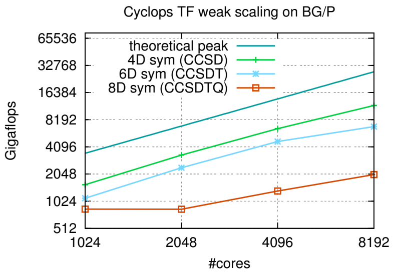


Calculate memory overhead for square tiling in 4D, 6D, 8D.
Triangle-pack surface tiles and lose DGEMM. . .

Performance

Note: 6D and 8D are actually *more* difficult cases than found in CCSDT and CCSDTQ, but they represent that CTF can exploit all the symmetry available in both the inner and outer indices.





Have not had time to look at BGP 8D scaling issues; likely due to weird dimensions.

Science

CCD is already running. This is what the code looks like:

```
void calcE(DistTensor & T2AA,    DistTensor & T2AB,    DistTensor & T2BB,
           DistTensor & TTAA,    DistTensor & TTAB,    DistTensor & TTBB,
           DistTensor & VABIJAA, DistTensor & VABIJAB, DistTensor & VABIJBB,
           DistTensor & D2AA,    DistTensor & D2AB,    DistTensor & D2BB,
           DistTensor & Z2AA,    DistTensor & Z2AB,    DistTensor & Z2BB,
           DistTensor & E_CCD)
{
      TTAA["ABIJ"] = Z2AA["ABIJ"]*D2AA["ABIJ"];
      TTAB["AbIj"] = Z2AB["AbIj"]*D2AB["AbIj"];
      TTBB["abij"] = Z2BB["abij"]*D2BB["abij"];

      E_CCD  = VABIJAA["ABIJ"]*TTAA["ABIJ"];
      E_CCD += VABIJAB["AbIj"]*TTAB["AbIj"];
      E_CCD += VABIJBB["abij"]*TTBB["abij"];

      TTAA["ABIJ"] -= T2AA["ABIJ"];
      TTAB["AbIj"] -= T2AB["AbIj"];
      TTBB["abij"] -= T2BB["abij"];

      T2AA["ABIJ"] += TTAA["ABIJ"];
      T2AB["AbIj"] += TTAB["AbIj"];
      T2BB["abij"] += TTBB["abij"];

}
```

Can you find the code corresponding to $E_{CCD} = V_{ij}^{ab} * T_{ij}^{ab}$?

Summary

- CTF is perfectly statically load-balanced thanks to cyclic distribution.
- CTF can immediately utilize SUMMA, Cannon, Strassen, 2.5D, etc. dense MMM.
- No automatic code generation but Devin's interface makes writing CC almost trivial. Each permutation-unique term is *one line of code*.

We're not out of the woods yet. . .

- Nontrivial integer computation in parallel redistribution code.
- Still exploring virtualization dimensions and thread parallelism.

Not bad given that this project began in May 2011 and is unfunded except for DOE-CSGF (Edgar and Devin).

Reduced-scaling coupled-cluster

Acknowledgments

Argonne
NATIONAL LABORATORY

DOE CSGF