Evolving the Tensor Contraction Engine for Next-Generation Multi-petaflop Supercomputers

Jeff Hammond[1] and Eugene DePrince[2]

Argonne National Laboratory
[1] Leadership Computing Facility (jhammond@anl.gov)
[2] Center for Nanoscale Materials (adeprince@anl.gov)

WOLFHPC – 31 May 2011

Argonne NATIONAL LABORATORY

Abstract

The Tensor Contraction Engine (TCE) is an enormously successful project in creating a domain-specific language for quantum many-body theory with an associated code generator for the massively-parallel computational chemistry package NWChem. This collection of tools has enabled hundreds of novel scientific simulations running efficiently on many of the largest supercomputers in the world. This talk will first recount five years of experience developing simulation capability with the TCE (specifically, response properties) and performance analysis of its execution on leadership-class supercomputers, summarizing its many successes with constructive criticism of its few shortcomings. Second, we will describe our recent investigation of quantum many-body methods on heterogeneous compute nodes, specifically GPUs attached to multicore CPUs, and how to evolve the TCE for the next generation of multi-petaflop supercomputers, all of which which feature multicore CPUs and many of which will be heterogeneous. We will describe new domain-specific libraries and high-level data structures that can couple to automatic code generation techniques for improved productivity and performance as well as our efforts to implement them.

What is TCE?

1. NWChem users know it as the coupled-cluster module that supports a kitchen sink of methods.
2. NWChem developers know it as the Python program that generates item 1.
3. People in this room probably know it as a multi-institutional collaboration that resulted in item 2 (among other things).

The practical TCE – NWChem many-body codes

What does it do?

1. GUI input quantum many-body theory e.g. CCSD.
2. Operator specification of theory.
3. Apply Wick's theory to transform operator expressions into array expressions.
4. Transform input array expression to operation tree using many types of optimization.
5. Produce Fortran+Global Arrays+NXTVAL implementation.

Developer can intercept at various stages to modify theory, algorithm or implementation.

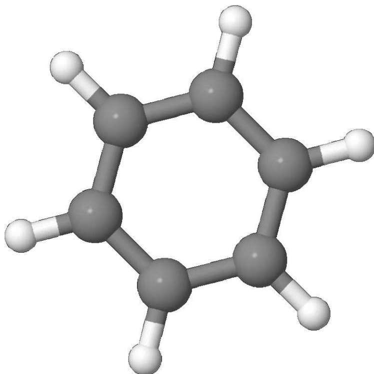
The practical TCE – Success stories

- First parallel implementation of many (most) CC methods.
- First truly generic CC code (not string-based): {RHF,ROHF,UHF}×CC{SD,SDT,SDTQ}×{T/Λ,EOM,LR/QR}
- Most of the largest calculations of their kind employ TCE: CR-EOMCCSD(T), CCSD-LR α, CCSD-QR β, CCSDT-LR α
- Reduces implementation time for new methods from years to hours, TCE codes are easy to verify.

Significant hand-tuning by Karol Kowalski and others at PNNL was required to make TCE run efficiently and scale to 1000 processors and beyond.

Before TCE

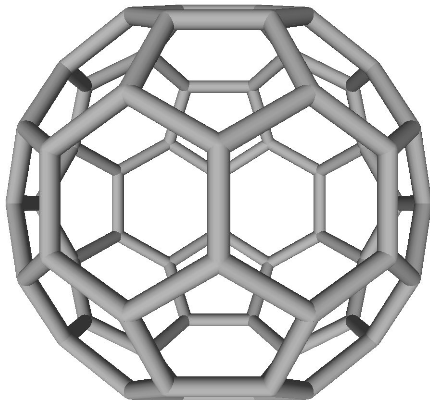
CCSD/aug-cc-pVDZ – 192 b.f. – days on 1 processor



Benzene is close to crossover point between *small* and *large*.

Linear response polarizability

CCSD/Z3POL – 1080 b.f. – 40 hours on 1024 processors

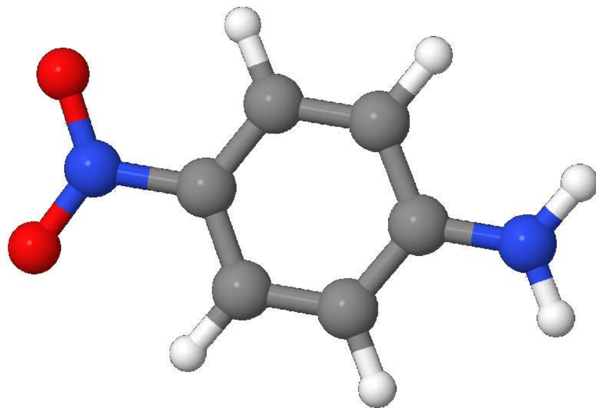


This problem is 20,000 times larger on the computer than benzene.

J. Chem. Phys. **129**, 226101 (2008).

Quadratic response hyperpolarizability

CCSD/d-aug-cc-pVTZ – 812 b.f. – 20 hours on 1024 processors

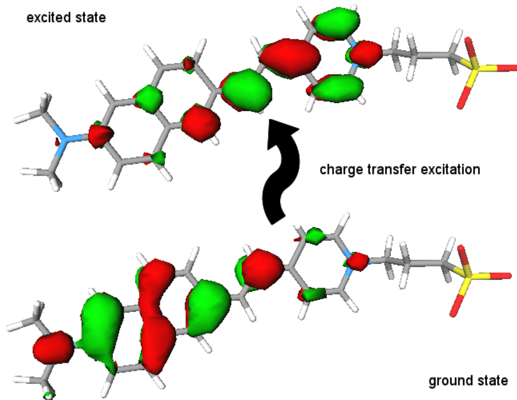


Lower levels of theory are not reliable for this system.

J. Chem. Phys. **130**, 194108 (2009).

Charge-transfer excited-states of biomolecules

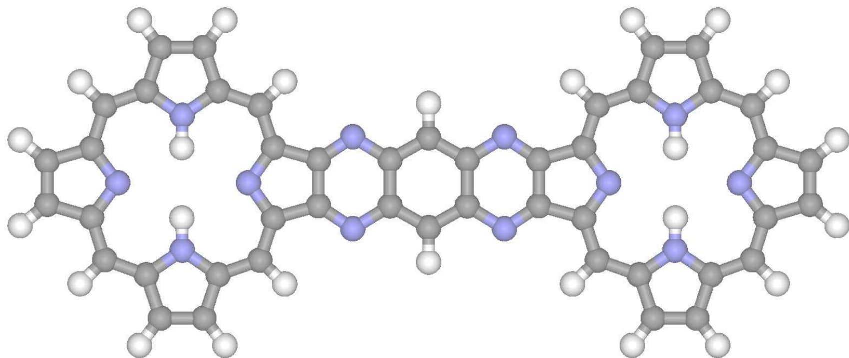
CR-EOMCCSD(T)/6-31G* – 584 b.f. – 1 hour on 256 cores



Lower levels of theory are wildly incorrect for this system.

Excited-state calculation of conjugated arrays

CR-EOMCCSD(T)/6-31+G* – 1096 b.f. – 15 hours on 1024 cores



Robert probably showed Karol's latest and greatest.

J. Chem. Phys. **132**, 154103 (2010).

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
-------------------------------------------
```

Only <25 KLOC are hand-written; ∼100 KLOC is utility code following TCE data-parallel template.

My thesis work

```
http://cloc.sourceforge.net v 1.53  T=13.0 s
-------------------------------------------
Language      files  blank  comment    code
-------------------------------------------
Fortran 77     5757      0    29098  983284
-------------------------------------------
SUM:           5757      0    29098  983284
-------------------------------------------
```

Total does not include ∼1M LOC that was reused (EOM).

CCSD quadratic response hyperpolarizability was derived, implemented and verified during a two week trip to PNNL. Over 100 KLOC were "written" in under an hour.

The practical TCE – Limitations

What does it NOT do?

- Relies upon external (read: hand-written) implementations of many procedures.
- Hand-written procedures define underlying data representation.
- Does not effectively reuse code (TCE needs own runtime). Of course, 4M LOC in F77 could be 4K LOC in C++.
- Ignores some obvious abstraction layers and hierarchical parallelism.

None of these shortcomings are intrinsic!

An instantiation of TCE is limited to the set of code transformations known to the implementer.

The practical TCE – Performance analysis

Performance TCE in NWChem cannot be understood independent of GA programming model.

- GA couldn't do block sparse so TCE does its own indexing. Table lookups are/were a bottleneck.
- Suboptimal data representation leads to nonlocal communication.
- Single-level tiling isn't ideal.
- Lack of abstraction for kernel prevents optimization; e.g. tensor permutations significant portion of wall time.

Time does not permit me to quantify performance issues.

Can *all* hand-optimizations can be back-ported into TCE?
Is this necessary? What are the challenges of an embedded DSL?

HPC circa 2012

Systems coming online in 2012 will have 200K+ cores with significant node-level parallelism both in the processor(s) and the NIC (e.g. Cray Gemini has 48 ports).

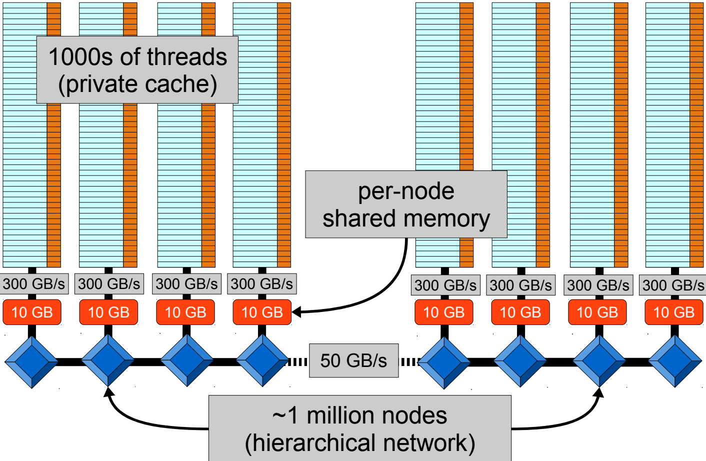
- BlueWaters (PERCS): 37,500+ sockets, 8 cores per socket.
- Mira (Blue Gene/Q): 49,152 nodes, 1 16-core CPU per node.
- Titan (Cray XK): 12,160 nodes, 1 AMD Bulldozer CPU and 1 NVIDIA Kepler GPU per node.

[Details from NCSA, Wikipedia and Buddy Bland's public slides.]

Node counts not increasing relative to current systems, so node-level parallelism is our primary challenge.

Process-only parallelism is not optimal for any of these machines. TCE 2.0 must address heterogeneity.

Exascale Architecture



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 (CCD) implementation

$$R^{ab}_{ij} = V^{ab}_{ij} + P(ia,jb)\left[T^{ae}_{ij}I^{b}_{e} - T^{ab}_{im}I^{m}_{j} + \frac{1}{2}V^{ab}_{ef}T^{ef}_{ij} + \frac{1}{2}T^{ab}_{mn}I^{mn}_{ij} - T^{ae}_{mj}I^{mb}_{ie} - I^{ma}_{ie}T^{eb}_{mj} + (2T^{ea}_{mi} - T^{ea}_{im})I^{mb}_{ej}\right]$$

$$\begin{aligned}
I^{a}_{b} &= (-2V^{mn}_{eb} + V^{mn}_{be})T^{ea}_{mn} \\
I^{i}_{j} &= (2V^{mi}_{ef} - V^{im}_{ef})T^{ef}_{mj} \\
I^{ij}_{kl} &= V^{ij}_{kl} + V^{ij}_{ef}T^{ef}_{kl} \\
I^{ia}_{jb} &= V^{ia}_{jb} - \frac{1}{2}V^{im}_{eb}T^{ea}_{jm} \\
I^{ia}_{bj} &= V^{ia}_{bj} + V^{im}_{be}(T^{ea}_{mj} - \frac{1}{2}T^{ae}_{mj}) - \frac{1}{2}V^{mi}_{be}T^{ae}_{mj}
\end{aligned}$$

Tensor contractions currently implemented as `GEMM` plus `PERMUTE`.

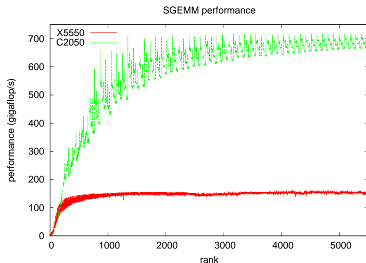
Hardware Details

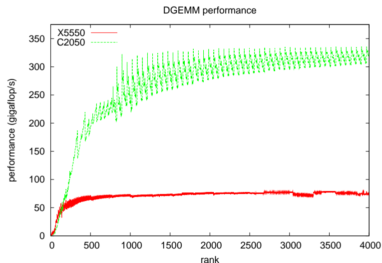
	CPU		GPU	
	X5550	2 X5550	C1060	C2050
processor speed (MHz)	2660	2660	1300	1150
memory bandwidth (GB/s)	32	64	102	144
memory speed (MHz)	1066	1066	800	1500
ECC available	yes	yes	no	yes
SP peak (GF)	85.1	170.2	933	1030
DP peak (GF)	42.6	83.2	78	515
power usage (W)	95	190	188	238

Note that power consumption is apples-to-oranges since CPU does not include DRAM, whereas GPU does.

Relative Performance of GEMM

GPU versus SMP CPU (8 threads):





CPU = 156.2 GF
GPU = 717.6 GF

CPU = 79.2 GF
GPU = 335.6 GF

We expect roughly 4-5 times speedup based upon this evaluation because `GEMM` *should* be 90% of the execution time.

CPU/GPU CCD

Iteration time in seconds

	our DP code			X5550		
	C2050	C1060	X5550	Molpro	TCE	GAMESS
C_8H_{10}	0.3	0.8	1.3	2.3	5.1	6.2
$C_{10}H_8$	0.5	1.5	2.5	4.8	10.6	12.7
$C_{10}H_{12}$	0.8	2.5	3.5	7.1	16.2	19.7
$C_{12}H_{14}$	2.0	7.1	10.0	17.6	42.0	57.7
$C_{14}H_{10}$	2.7	10.2	13.9	29.9	59.5	78.5
$C_{14}H_{16}$	4.5	16.7	21.6	41.5	90.2	129.3
C_{20}	8.8	29.9	40.3	103.0	166.3	238.9
$C_{16}H_{18}$	10.5	35.9	50.2	83.3	190.8	279.5
$C_{18}H_{12}$	12.7	42.2	50.3	111.8	218.4	329.4
$C_{18}H_{20}$	20.1	73.0	86.6	157.4	372.1	555.5

Our algorithm is most similar to GAMESS and does $\sim$4 times the flops as Molpro.

CPU+GPU CCSD

	Iteration time (s)						
	Hybrid	CPU	Molpro	NWChem	PSI3	TCE	GAMESS
C_8H_{10}	0.6	1.4	2.4	3.6	7.9	8.4	7.2
$C_{10}H_8$	0.9	2.6	5.1	8.2	17.9	16.8	15.3
$C_{10}H_{12}$	1.4	4.1	7.2	11.3	23.6	25.2	23.6
$C_{12}H_{14}$	3.3	11.1	19.0	29.4	54.2	64.4	65.1
$C_{14}H_{10}$	4.4	15.5	31.0	49.1	61.4	90.7	92.9
$C_{14}H_{16}$	6.3	24.1	43.1	65.0	103.4	129.2	163.7
C_{20}	10.5	43.2	102.0	175.7	162.6	233.9	277.5
$C_{16}H_{18}$	10.0	38.9	84.1	117.5	192.4	267.9	345.8
$C_{18}H_{12}$	14.1	57.1	116.2	178.6	216.4	304.5	380.0
$C_{18}H_{20}$	22.5	95.9	161.4	216.3	306.9	512.0	641.3

Statically distribute most diagrams between GPU and CPU, dynamically distribute leftovers.

More hybrid CCSD

molecule	Basis	o	v	Iteration time (s) Hybrid	CPU	Molpro	Speedup CPU	Molpro
CH_3OH	aTZ	7	175	2.5	4.5	2.8	1.8	1.1
benzene	aDZ	15	171	5.1	14.7	17.4	2.9	3.4
$C_2H_6SO_4$	aDZ	23	167	9.0	33.2	31.2	3.7	3.5
$C_{10}H_{12}$	DZ	26	164	10.7	39.5	56.8	3.7	5.3
$C_{10}H_{12}$	6-31G	26	78	1.4	4.1	7.2	2.9	5.1

Calculations are small because we are not using out-of-core or distributed storage, hence are limited by CPU main memory.

Physics- or array-based domain decomposition will lead to single-node tasks of this size.

Lessons learned

- **Do not GPU-ize legacy code!**
  Must redesign and reimplement (hopefully automatically).
- Verification is a pain.
- CC possess significant task-based parallelism.
- Threading ameliorates memory capacity and BW bottlenecks. (How many cores required to saturate STREAM BW?)
- `GEMM` and `PERMUTE` kernels both data-parallel, readily parallelizable via OpenMP or CUDA.
- Careful organization of asynchronous data movement hides entire PCI transfer cost for non-trivial problems.
- Näive data movement leads to 2x for CCSD; smart data movement leads to 8x.

Summary of GPU CC

- Implemented CCD on GPU and on CPU using CUDA/OpenMP and vendor BLAS. Implementation quality is very similar.
- Implemented CCSD on CPU+GPU using streams and mild dynamic load-balancing.
- Compared to legacy codes *as directly as possible*:
  - Apples-to-apples CPU vs. GPU is 4-5x (as predicted).
  - Apples-to-oranges us versus them shows 7-10x. Our CPU code is 2x, so again 4-5x is from GPU.

We have very preliminary MPI results using task parallelism plus `MPI_Allreduce`.

Load-balancing is the only significant barrier to GA+GPU implementation.

Suggestions for TCE 2.0

- Hierarchical task and data parallelism.
- Abstraction layers for high-level library insertion.
- Integrate with accurate hardware models.
- Runtime profile-guided optimization for iterative codes.
- Synergy between data structures and computation.
- Target low-level communication.
- Brew coffee.

Except for the last point, an evolutionary approach is sufficient for TCE to continue having a transformative impact on quantum chemistry codes in the post-petascale era.

Acknowledgments

Karol Kowalski

Saday, Robert, Sriram...

Dirac cluster at NERSC

Chemistry Details

Molecule	o	v
C_8H_{10}	21	63
$C_{10}H_8$	24	72
$C_{10}H_{12}$	26	78
$C_{12}H_{14}$	31	93
$C_{14}H_{10}$	33	99
$C_{14}H_{16}$	36	108
C_{20}	40	120
$C_{16}H_{18}$	41	123
$C_{18}H_{12}$	42	126
$C_{18}H_{20}$	46	138

- 6-31G basis set
- C_1 symmetry
- F and V from GAMESS via disk

Since January ...

- Integrated with PSI3 (GPL).
- No longer memory-limited by GPU.
- Working on GA-like one-sided.
- GPU one-sided R&D since 2009.

Numerical Precision versus Performance

Iteration time in seconds

molecule	C1060 SP	C1060 DP	C2050 SP	C2050 DP	X5550 SP	X5550 DP
C_8H_{10}	0.2	0.8	0.2	0.3	0.7	1.3
$C_{10}H_8$	0.4	1.5	0.2	0.5	1.3	2.5
$C_{10}H_{12}$	0.7	2.5	0.4	0.8	2.0	3.5
$C_{12}H_{14}$	1.8	7.1	1.0	2.0	5.6	10.0
$C_{14}H_{10}$	2.6	10.2	1.5	2.7	8.4	13.9
$C_{14}H_{16}$	4.1	16.7	2.4	4.5	12.1	21.6
C_{20}	6.7	29.9	4.1	8.8	22.3	40.3
$C_{16}H_{18}$	9.0	35.9	5.0	10.5	28.8	50.2
$C_{18}H_{12}$	10.1	42.2	5.6	12.7	29.4	50.3
$C_{18}H_{20}$	17.2	73.0	10.1	20.1	47.0	86.6

This the apples-to-apples CPU vs. GPU.