

Coupled-cluster theory on graphics processing units

Jeff Hammond¹ and Eugene DePrince²

Argonne National Laboratory

¹ Leadership Computing Facility (jhammond@anl.gov)

² Center for Nanoscale Materials (adeprince@anl.gov)

29 April 2011

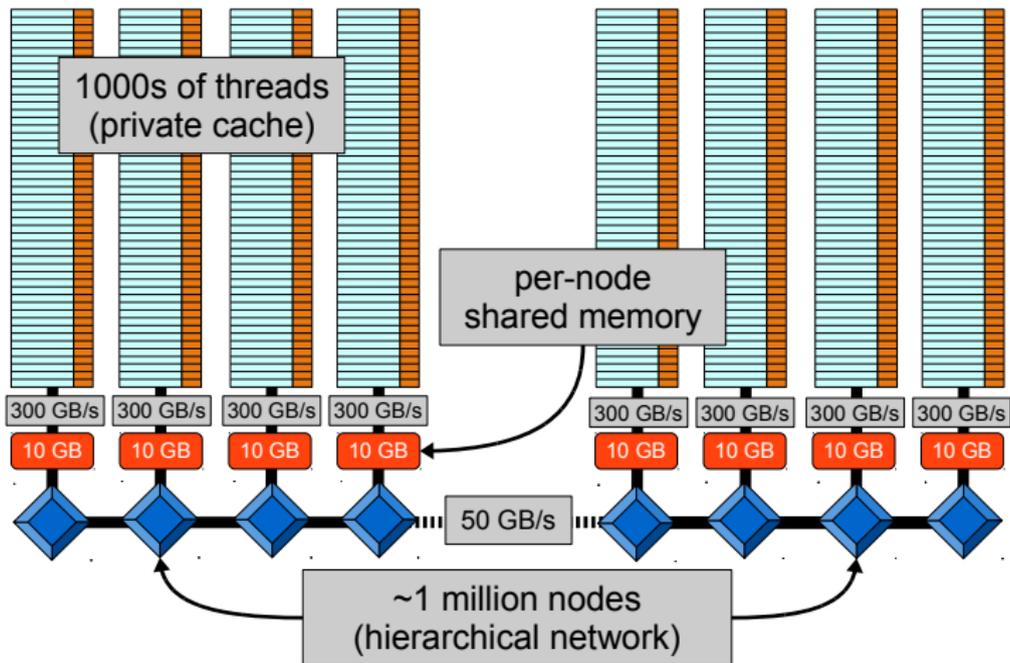


Abstract

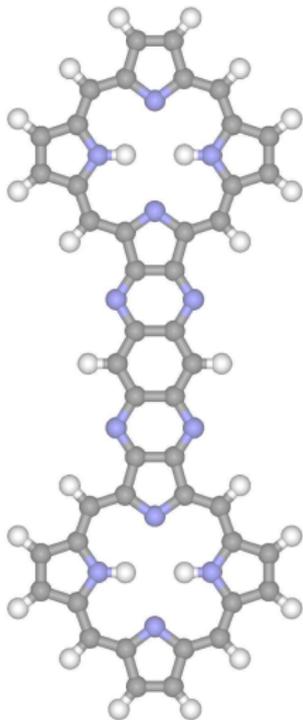
I will discuss our recent work on implementing the coupled-cluster method (CC) on GPUs. CC is an important quantum many-body method for molecular calculations and is considered the "gold standard" due to its accuracy and black box nature. Because CC can be implemented in terms of matrix-matrix multiplication ([SD]GEMM) operations, it is relatively straightforward to implement on GPUs using CUDA BLAS. However, due to the large number of terms, memory limitations of NVIDIA Fermi cards, and large variation in matrix dimensions, realizing a large speedup requires careful memory management and movement of data. This talk will present our earlier work on a GPU-only implementation of CCD as well as a more recent hybrid CCSD code, which attempts to dynamically optimize the overlap of computation and communication.

Motivation

Strawman Exascale Architecture



Wavefunction theory



- Second-order perturbation theory (MP2) is accurate via magical cancellation of error but can't do excited states.
- Quantum Monte Carlo (QMC) integration applied to the Schrödinger equation but can't do most properties.
- Coupled-cluster theory (CC) is infinite-order solution to many-body Schrödinger equation; it can do both excited-state and properties.
- CC applications limited to 10-100 atoms on terascale and petascale computers because of scaling: $\sim N_{basis}^x$ ($x=5-7$)

Motivation for CC with HPC

- Electronic excited-states and electric-field perturbations push the limits of conventional approximations in density functional theory (DFT) and are outside the scope of classical methods.
- Interesting chemical processes in biology and material science require model systems too large for a conventional computational resources.
- Answering many chemical questions requires large data sets which cannot be obtained in a reasonable amount time if done sequentially.

Coupled-cluster theory

Coupled-cluster theory

The coupled-cluster (CC) wavefunction ansatz is

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

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

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 Schrödinger 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

$$|CC\rangle = \exp(T)|0\rangle$$

$$T = T_1 + T_2 + \dots + T_n \quad (n \ll N)$$

$$T_1 = \sum_{ia} t_i^a \hat{a}_a^\dagger \hat{a}_i$$

$$T_2 = \sum_{ijab} t_{ij}^{ab} \hat{a}_a^\dagger \hat{a}_b^\dagger \hat{a}_j \hat{a}_i$$

$$\begin{aligned} |\Psi_{CCD}\rangle &= \exp(T_2)|\Psi_{HF}\rangle \\ &= (1 + T_2 + T_2^2)|\Psi_{HF}\rangle \end{aligned}$$

$$\begin{aligned} |\Psi_{CCSD}\rangle &= \exp(T_1 + T_2)|\Psi_{HF}\rangle \\ &= (1 + T_1 + \dots + T_1^4 + T_2 + T_2^2 + T_1 T_2 + T_1^2 T_2)|\Psi_{HF}\rangle \end{aligned}$$

Coupled-cluster theory

Projective solution of CC:

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

CCD is:

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

CCSD is:

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

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, \dots (a, b, c, \dots) run over the occupied (virtual) orbitals.

CCD Equations

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

$$I_b^a = (-2V_{eb}^{mn} + V_{be}^{mn}) T_{mn}^{ea}$$

$$I_j^i = (2V_{ef}^{mi} - V_{ef}^{im}) T_{mj}^{ef}$$

$$I_{kl}^{ij} = V_{kl}^{ij} + V_{ef}^{ij} T_{kl}^{ef}$$

$$I_{jb}^{ia} = V_{jb}^{ia} - \frac{1}{2} V_{eb}^{im} T_{jm}^{ea}$$

$$I_{bj}^{ia} = V_{bj}^{ia} + V_{be}^{im} (T_{mj}^{ea} - \frac{1}{2} T_{mj}^{ae}) - \frac{1}{2} V_{be}^{mi} T_{mj}^{ae}$$

Turning CC into GEMM 1

Some tensor contractions are trivially mapped to GEMM:

$$\begin{aligned} I_{kl}^{ij} &+ = V_{ef}^{ij} T_{kl}^{ef} \\ I_{(kl)}^{(ij)} &+ = V_{(ef)}^{(ij)} T_{(kl)}^{(ef)} \\ I_a^b &+ = V_c^b T_a^c \end{aligned}$$

Other contractions require reordering to use BLAS:

$$\begin{aligned} I_{bj}^{ia} &+ = V_{be}^{im} T_{mj}^{ea} \\ I_{bj,ia} &+ = V_{be,im} T_{mj,ea} \\ J_{bi,ja} &+ = W_{bi,me} U_{me,ja} \\ J_{bi}^{ja} &+ = W_{bi}^{me} U_{me}^{ja} \\ J_{(bi)}^{(ja)} &+ = W_{(bi)}^{(me)} U_{(me)}^{(ja)} \\ J_x^z &+ = W_x^y U_y^z \end{aligned}$$

Turning CC into GEMM 2

Reordering can take as much time as GEMM. 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.)

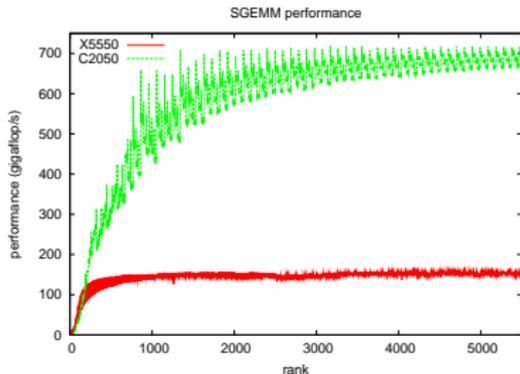
Hardware Details

	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

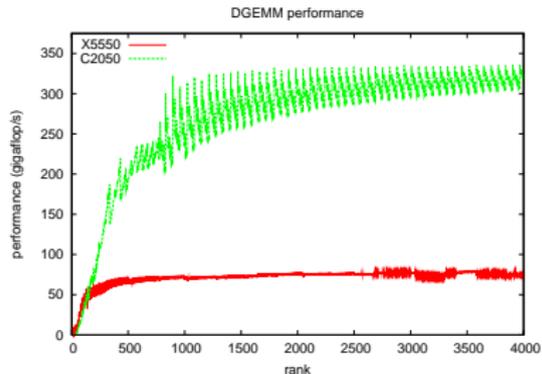
Note that power consumption is apples-to-oranges since CPU does not include DRAM, whereas GPU does.

Relative Performance of GEMM

GPU versus SMP CPU (8 threads):



Maximum:
CPU = 156.2 GF
GPU = 717.6 GF



Maximum:
CPU = 79.2 GF
GPU = 335.6 GF

We expect roughly 4-5 times speedup based upon this evaluation.

Molecule	o	v
C ₈ H 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
- C₁ symmetry
- F and V from PSI3.
- GPU code now runs from PSI3.

Without committing to anything, the intent is to release the code under GPL with PSI4.

CCD Algorithm

Copy V and F to GPU (cudaMemcpy)

WHILE ($|dT| > \text{eps}$) **DO**

IF v_{cd}^{ab} fits in GPU memory **THEN**

Update residual (cudaDgemm)

ELSE

FOR all tiles **DO**

copy v_{cd}^{ab} to GPU (cudaMemcpy)

contract $v_{ef}^{ab} t_{ij}^{ef}$ (cudaDgemm)

END DO

Update residual with remaining terms (cudaDgemm)

END IF

Update T with residual

Compute $|dT|$ (cudaDnrm2)

END WHILE

Copy amplitudes from GPU (cudaMemcpy)

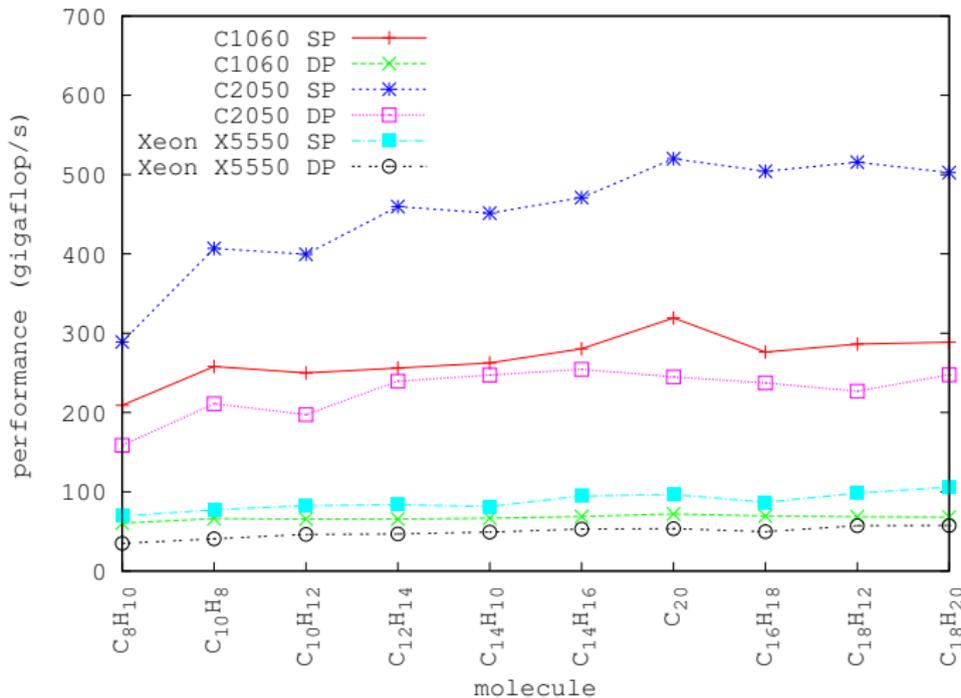
$E = t.v$

CCD Performance Results

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

CCD Performance Summary



Numerical Precision versus Performance

molecule	Iteration time in seconds					
	C1060		C2050		X5550	
	SP	DP	SP	DP	SP	DP
C ₈ H 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

Almost no work on single-precision or mixed-precision for standard CPU packages even though it is worth 2x.

CCSD Equations

$$t_{ij}^{ab} d_{ij}^{ab} = v_{ij}^{ab} + P(ia, jb) R_{ij}^{ab}$$

$$R_{ij}^{ab} + = \frac{1}{2} v_{ef}^{ab} C_{ij}^{ef}$$

$$R_{ij}^{ab} + = \frac{1}{2} C_{mn}^{ab} I_{ij}^{mn}$$

$$R_{ij}^{ab} - = t_{mj}^{ae} I_{ie}^{mb} + I_{ie}^{ma} t_{mj}^{eb}$$

$$R_{ij}^{ab} + = (2t_{mi}^{ea} - t_{im}^{ea}) I_{ej}^{mb}$$

$$R_{ij}^{ab} + = t_i^e I_{ej}^{ab}$$

$$R_{ij}^{ab} - = t_m^a I_{ij}^{mb}$$

$$R_{ij}^{ab} + = t_{ij}^{ae} I_e^b$$

$$R_{ij}^{ab} - = t_{im}^{ab} I_j^m$$

$$I_{kl}^{ij} = v_{kl}^{ij} + v_{ef}^{ij} C_{kl}^{ef} + P(ik / jl) t_k^e v_{el}^{ij}$$

$$I_{jb}^{ia} = v_{jb}^{ia} - \frac{1}{2} v_{eb}^{im} (t_{jm}^{ea} + 2t_j^e t_m^a) + v_{eb}^{ia} t_j^e - v_{jb}^{im} t_m^a$$

$$I_{ci}^{ab} = v_{ci}^{ab} - v_{ci}^{am} t_m^b - v_{ci}^{mb} t_m^a$$

$$I_{jk}^{ia} = v_{jk}^{ia} + v_{ef}^{ia} C_{jk}^{ef}$$

$$I_a^i = (2v_{je}^{im} - v_{ea}^{im}) t_m^e$$

$$I_j^i = I_j^i + I_e^i t_j^e$$

$$I_j^i = (2v_{je}^{im} - v_{ej}^{im}) t_m^e + (2v_{ef}^{mi} - v_{ef}^{im}) t_{mj}^{ef}$$

$$I_b^a = (2v_{be}^{am} - v_{be}^{ma}) t_m^e - (2v_{eb}^{mn} - v_{be}^{mn}) C_{mn}^{ea}$$

$$I_{bj}^{ia} = v_{bj}^{ia} - \frac{1}{2} v_{be}^{im} (t_{mj}^{ae} + 2t_m^a t_j^e) + v_{be}^{ia} t_j^e - v_{bj}^{im} t_m^a + \frac{1}{2} (2v_{be}^{im} - v_{eb}^{im}) t_{mj}^{ea}$$

$$t_i^a d_i^a = f_i^a + R_i^a$$

$$R_i^a + = I_e^a t_i^e$$

$$R_i^a - = I_i^m t_m^a$$

$$R_i^a + = I_e^m (2t_{mi}^{ea} - t_{im}^{ea})$$

$$R_i^a + = (2v_{ei}^{ma} - v_{ei}^{am}) t_m^e$$

$$R_i^a - = v_{ci}^{mn} (2t_{mn}^{ea} - t_{mn}^{ae})$$

$$R_i^a + = v_{ef}^{ma} (2t_{mi}^{ef} - t_{im}^{ef})$$

CCSD Algorithm

Guiding principles:

- Too many arrays to fit into GPU memory.
- Copy-in every iteration in CCD was not a problem
- Want multi-GPU, mixed CPU-GPU algorithms.

Design:

- Persistent buffers but push all large arrays every iteration.
- $O(N^6)$, $O(N^5)$ on GPU.
- $O(N^5)$, $O(N^4)$ on CPU.
- Dynamically schedule some diagrams each iteration to load-balance.
- Overlap computation and communication with CUDA streams (CUBLAS compatible now).

Hybrid CCSD

	Iteration time in seconds						
	Hybrid	CPU	Molpro	NWChem	PSI3	TCE	GAMESS
C ₈ H 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

We do at least twice as many flops as Molpro due to CC formalism.

More hybrid CCSD

molecule	Basis	o	v	Hybrid	CPU	Molpro	CPU	Molpro
CH ₃ OH	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 ₂ H ₆ SO 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

Molpro optimized for $v/o \gg 1$.

Our code doesn't favor any limit except $o, v \gg 1$.

The NWChem-GPU connection

- Automatically generated code: easy to rewrite (in principle).
- Data-parallel over tiles using Global Arrays.
- Näive dynamic load-balancing (shared counter).
- Standard GA programming model:
 check out, compute, check in

TCE CPU algorithm

```
for (P3,P4,H1,H2) in all (P,P,H,H) tiles:
  if (my_turn) and (nonzero_symmetry):
    allocate_and_zero buffer Jc
    for (P5,P6) in all (P,P) tiles:
      if (nonzero_symmetry):
        allocate_and_zero buffer Tc
        get Tb from global T
        reorder Tc
        allocate_and_zero buffer Ic
        get Ib from global I
        reorder Ic
        compute Jc += Tc*Ic
    reorder Jc
  acc Jc onto global J
```

TCE GPU algorithm

```
grab_from_pool_and_zero buffer pair (Tc,Tg)
grab_from_pool_and_zero buffer pair (Ic,Ig)
if (push_gpucompute_pull < cpucompute):
    iget_and_push Tg from global T
    iget_and_push Ig from global I
    igpu_reorder Tg
    igpu_reorder Ig
    gpucompute Jg += Tg*Ig
else:
    iget Tc from global T
    iget Ic from global I
    compute Jc += Tc*Ic
ireorder Jg
reorder Jc
pull_and_acc Jc += Jg
acc Jc onto global J
```

Evaluating the GPU algorithm

- Tilesizes not big enough to justify GPU all the time (bad).
- *Jg* stays on the GPU through each pass (good).
- Possibly good overlap of data movement (good).
- Can unroll inner loop for maximum overlap (good), but this doubles the memory required (bad).
- Threaded CPU code helps memory issues but GA/ARMCI not thread-safe (funneled or serialized should be okay).

Many of the GPU-oriented optimizations will help with multicore CPUs. Fully rewritten GPU TCE in NWChem will probably coincide with porting to BGQ for this reason.

Acknowledgments



Director's Fellowship (JRH)
Computational Fellowship (AED)
Breadboard cluster
Fusion cluster



Dirac cluster

Shameless promotion

What: Symposium on Application Accelerators in HPC

When: July 19-21, 2011

Where: Knoxville, Tennessee

Application Sessions:

Computational chemistry on accelerators (Chair: Jeff Hammond, ALCF)

Lattice QCD (Chair: Steven Gottlieb, Indiana University, Bloomington)

Weather and climate modeling (Chair: John Michalakes, NREL)

Bioinformatics (Chair: TBD)

Submissions:

Short paper (up to 4 pages, for a poster presentation)

Long paper (up to 10 pages, for an oral presentation)

Deadlines:

Submissions due: May 6, 2011

Presentation acceptance notification: June 6, 2011

Final papers due: June 30, 2011

Partial bibliography

MD on GPUs (of many more)

All major MD packages have or will soon have a GPU implementation.

- J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten. “Accelerating molecular modeling applications with graphics processors.” *J. Comp. Chem.*, 28 (16), 2618–2640, 2007.
- J. A. Anderson, C. D. Lorenz, and A. Travesset. “General purpose molecular dynamics simulations fully implemented on graphics processing units.” *J. Comp. Phys.*, 227 (10), 5342–5359, 2008.
- P. Friedrichs, M. S. and Eastman, V. Vaidyanathan, M. Houston, S. Legrand, A. L. Beberg, D. L. Ensign, C. M. Bruns, and V. S. Pande. “Accelerating molecular dynamic simulation on graphics processing units.” *J. Comp. Chem.*, 30 (6), 864–872, 2009.
- R. Yokota, T. Hamada, J. P. Bardhan, M. G. Knepley, and L. A. Barba. “Biomolecular electrostatics simulation by an FMM-based BEM on 512 GPUs.” *CoRR*, abs/1007.4591, 2010.

- K. Yasuda, "Accelerating density functional calculations with graphics processing unit." *J. Chem. Theo. Comp.*, 4 (8), 1230–1236, 2008.
- L. Genovese, M. Ospici, T. Deutsch, J.-F. Mehaut, A. Neelov, and S. Goedecker. "Density functional theory calculation on many-cores hybrid central processing unit-graphic processing unit architectures." *J. Chem. Phys.*, 131 (3), 034103, 2009.
- I. S. Ufimtsev and T. J. Martínez. "Quantum chemistry on graphical processing units. 2. Direct self-consistent-field (SCF) implementation." *J. Chem. Theo. Comp.*, 5 (4), 1004–1015, 2009; "Quantum chemistry on graphical processing units. 3. Analytical energy gradients, geometry optimization, and first principles molecular dynamics." *J. Chem. Theo. Comp.*, 5 (10), 2619–2628, 2009.
- C. J. Woods, P. Brown, and F. R. Manby. "Multicore parallelization of Kohn-Sham theory." *J. Chem. Theo. Comp.*, 5 (7), 1776–1784, 2009.
- P. Brown, C. J. Woods, S. McIntosh-Smith, and F. R. Manby. "A massively multicore parallelization of the Kohn-Sham energy gradients." *J. Comp. Chem.*, 31 (10), 2008–2013, 2010.
- R. Farnber, E. Bylaska, S. Baden and students. "(Car-Parrinello on GPGPUs)." Work in progress, 2011.

MP2 using the resolution-of-identity approximation involves a few large GEMMs.

- L. Vogt, R. Olivares-Amaya, S. Kermes, Y. Shao, C. Amador-Bedolla, and A. Aspuru-Guzik. “Accelerating resolution-of-the-identity second-order Møller-Plesset quantum chemistry calculations with graphical processing units.” *J. Phys. Chem. A*, 112 (10), 2049–2057, 2008.
- R. Olivares-Amaya, M. A. Watson, R. G. Edgar, L. Vogt, Y. Shao, and A. Aspuru-Guzik. “Accelerating correlated quantum chemistry calculations using graphical processing units and a mixed precision matrix multiplication library.” *J. Chem. Theo. Comp.*, 6 (1), 135–144, 2010.
- A. Koniges, R. Preissl, J. Kim, D. Eder, A. Fisher, N. Masters, V. Mlaker, S. Ethier, W. Wang, and M. Head-Gordon. “Application acceleration on current and future Cray platforms.” In *CUG 2010*, Edinburgh, Scotland, May 2010.

QMC is ridiculously parallel at the node-level hence implementation of the kernel implies the potential for scaling to many thousands of GPUs.

- A. G. Anderson, W. A. Goddard III, and P. Schröder. “Quantum Monte Carlo on graphical processing units.” *Comp. Phys. Comm.*, 177 (3), 298–306, 2007.
- A. Gothandaraman, G. D. Peterson, G. Warren, R. J. Hinde, and R. J. Harrison. “FPGA acceleration of a quantum Monte Carlo application.” *Par. Comp.*, 34 (4-5), 278–291, 2008.
- K. Esler, J. Kim, L. Shulenburger, and D. Ceperley. “Fully accelerating quantum Monte Carlo simulations of real materials on GPU clusters.” *Comp. Sci. Eng.*, 99 (PrePrints), 2010.

Since we started this project, two groups have implemented non-iterative triples corrections to CCSD on GPUs. These procedures involve a few very large GEMMs and a reduction. At least two other groups are working on CC on GPUs but have not reported any results.

- M. Melicherčík, L. Demovič, and P. N. Michal Pitoňák. "Acceleration of CCSD(T) computations using technology of graphical processing unit." 2010.
- W. Ma, S. Krishnaoorthy, O. Villa, and K. Kowalski. "GPU-based implementations of the regularized CCSD(T) method: applications to strongly correlated systems." Submitted to *J. Chem. Theo. Comp.*, 2010.
- A. E. DePrince and J. R. Hammond. "Coupled-cluster theory on graphics processing units I. The coupled-cluster doubles method." Submitted to *J. Chem. Theo. Comp.*, 2010.

Gaussian Integrals on GPUs

Quantum chemistry methods spend a lot of time generating matrix elements of operators in a Gaussian basis set. All published implementations are either closed-source (commercial) or the source is unpublished, otherwise we would be using these in our code.

- I. S. Ufimtsev and T. J. Martínez. “Quantum chemistry on graphical processing units. 1. Strategies for two-electron integral evaluation.” *J. Chem. Theo. Comp.*, 4 (2), 222–231, 2008.
- A. V. Titov, V. V. Kindratenko, I. S. Ufimtsev, and T. J. Martínez. “Generation of kernels for calculating electron repulsion integrals of high angular momentum functions on GPUs – preliminary results.” In *Proc. SAAHPC 2010*, pages 1–3, 2010.
- A. Asadchev, V. Allada, J. Felder, B. M. Bode, M. S. Gordon, and T. L. Windus. “Uncontracted Rys quadrature implementation of up to G functions on graphical processing units.” *J. Chem. Theo. Comp.*, 6 (3), 696–704, 2010.