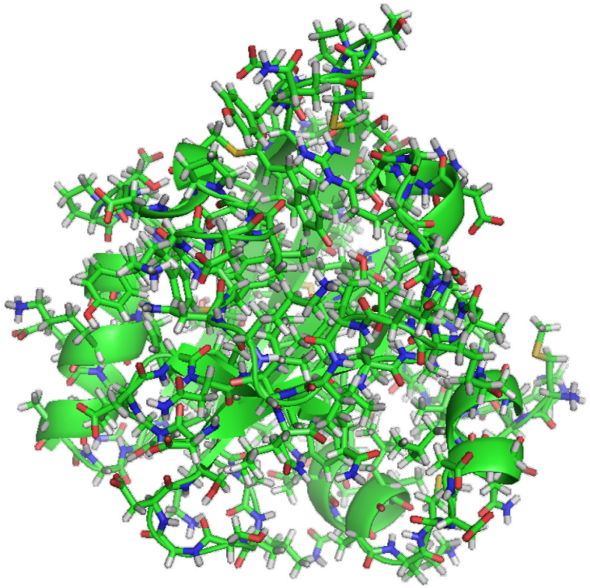
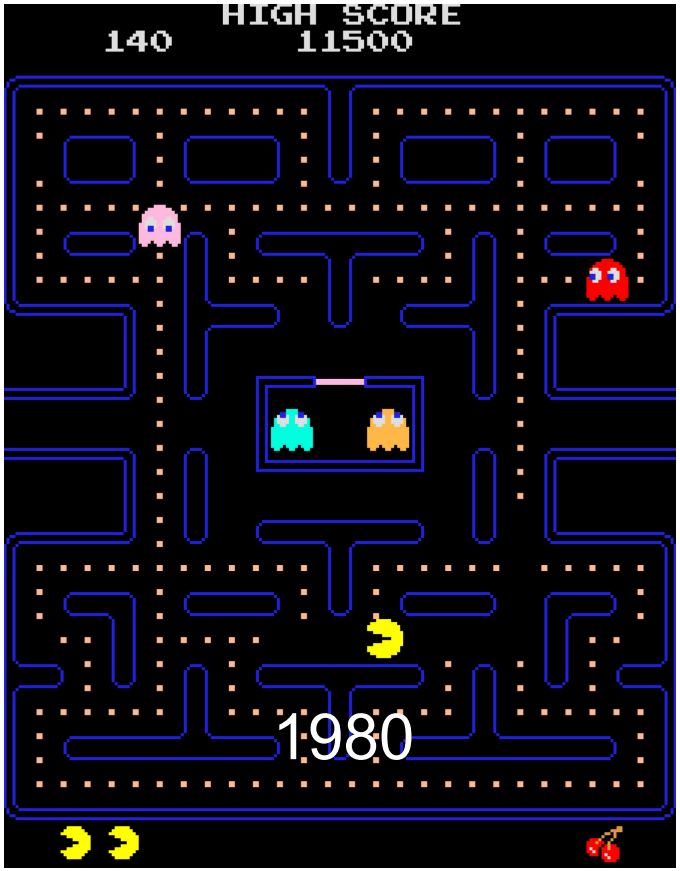
GPU Accelerated Quantum Chemistry with TeraChem



Turning Hardware Constraints into Performance Advantages

Nathan Luehr, Ivan Ufimtsev, Alexey Titov, Ed Hohenstein, Heather Kulik, and Todd Martinez

A (Very) Brief History of Gaming



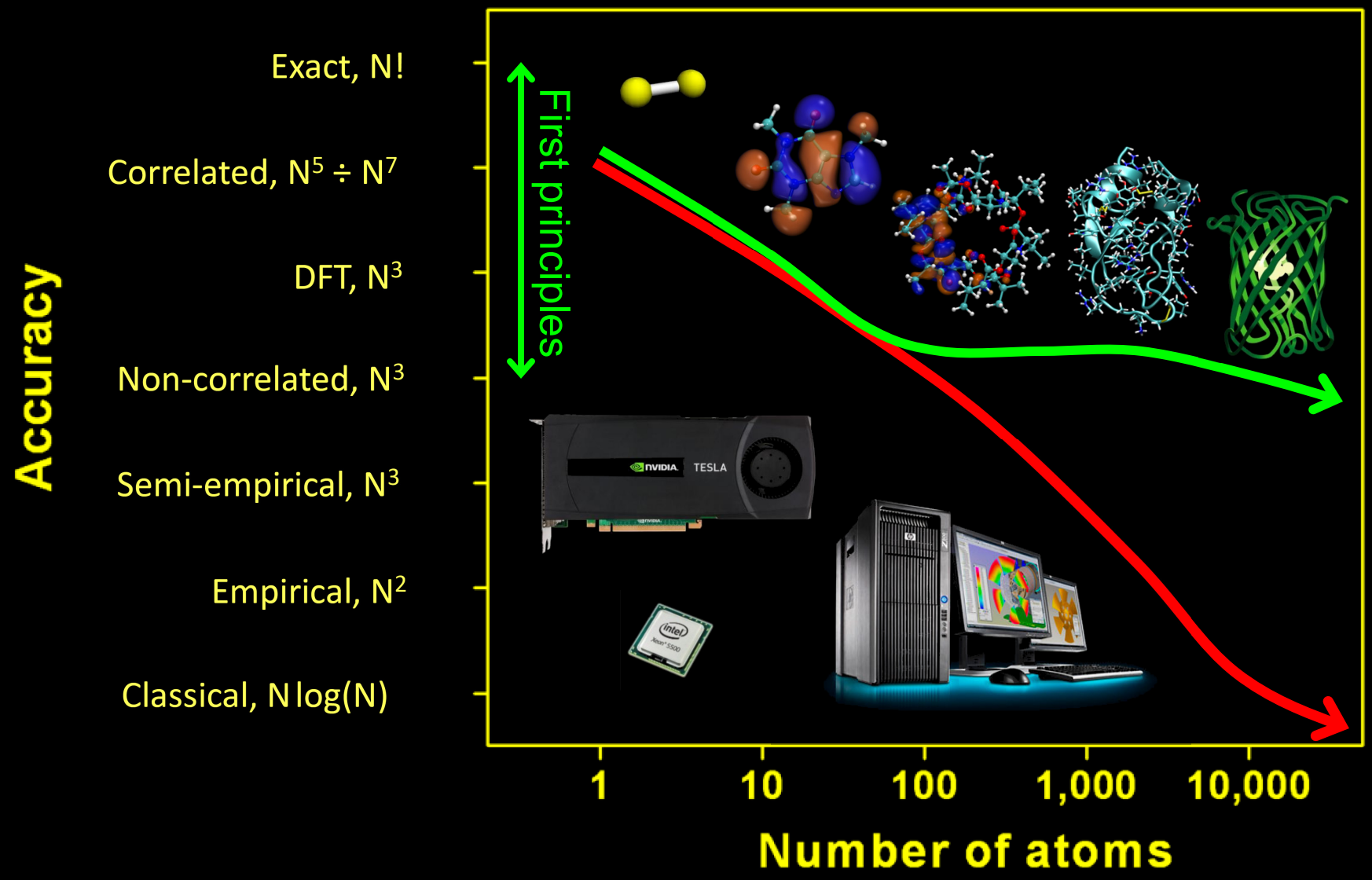
Gamers want realistic and immersive games.

GPUs exist because games need a few operations to be extremely fast.

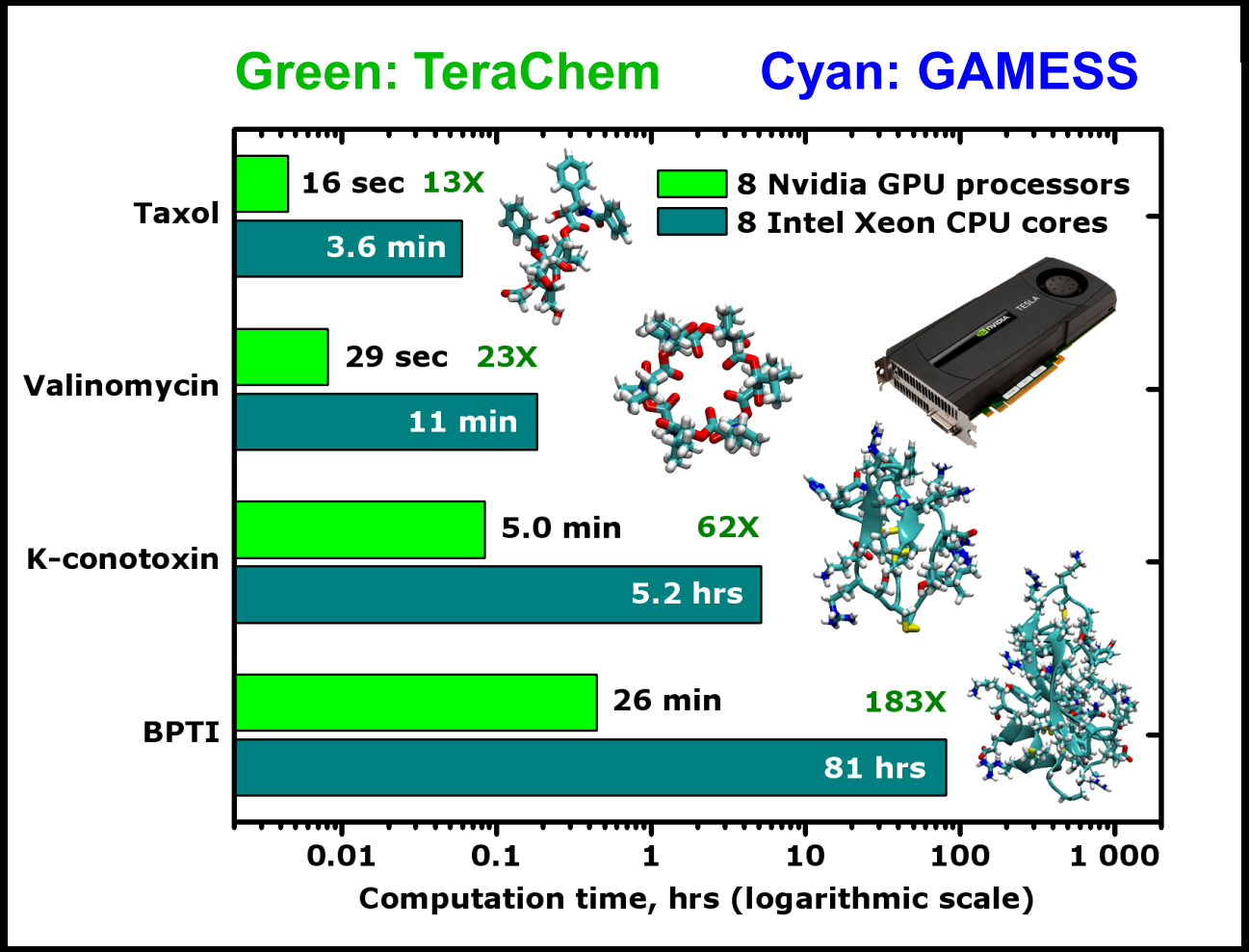
Games *are* physics simulations – Chemical simulation depends on many of the same operations as modern games.

CUDA and OpenCL offer abstracted access to GPU hardware.

Molecular modeling on graphical processors



TeraChem/GPU is over 100X faster than GAMESS/CPU



RHF/3-21G

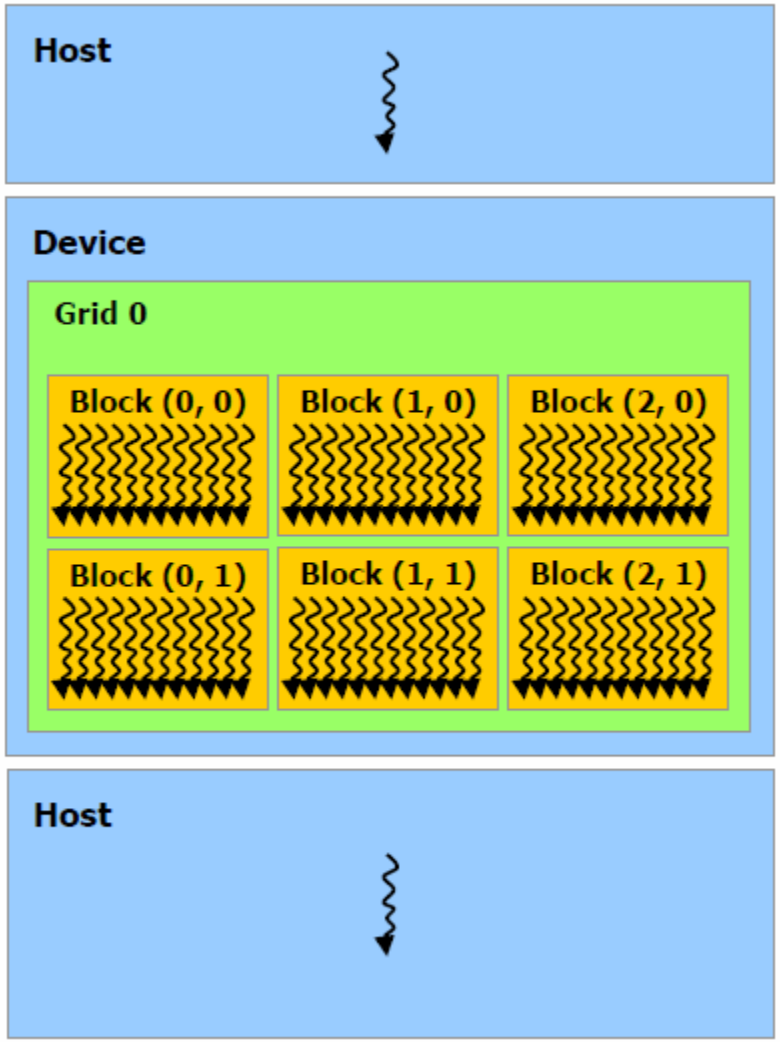
GAMESS

http://www.msg.chem.iastate.edu/gamess/

GPU Raw Specs

	Tesla C1060	GTX 580	Tesla C2070
DP Units	60	64	224
SP Units	240	512	448
Core Clock	1296 MHz	1544 MHz	1150 MHz
DP Performance	78 GFLOPS	198 GFLOPS	515 GFLOPS
SP Performance	622 GFLOPS	1.58 TFLOPS	1.03 TFLOPS
Memory Bandwidth	102 GB/sec	192.4 GB/sec	144 GB/sec

CUDA Programming Model



GPUs excel for workloads that

- are massively parallel
- are data independent
- use fine-grained parallelism
- rely on regular memory access patterns

Image courtesy Nvidia Programming Guide 4.0

DFT Quadrature Grids

$$E[\rho] = T_s[\rho] + E_{ext}[\rho] + J[\rho] + E_{xc}[\rho]$$

$$E_{xc} = \int dr\, f\left(\rho_\alpha, \rho_\beta, \gamma_{\alpha\alpha}, \gamma_{\alpha\beta}, \gamma_{\beta\beta}\right)$$

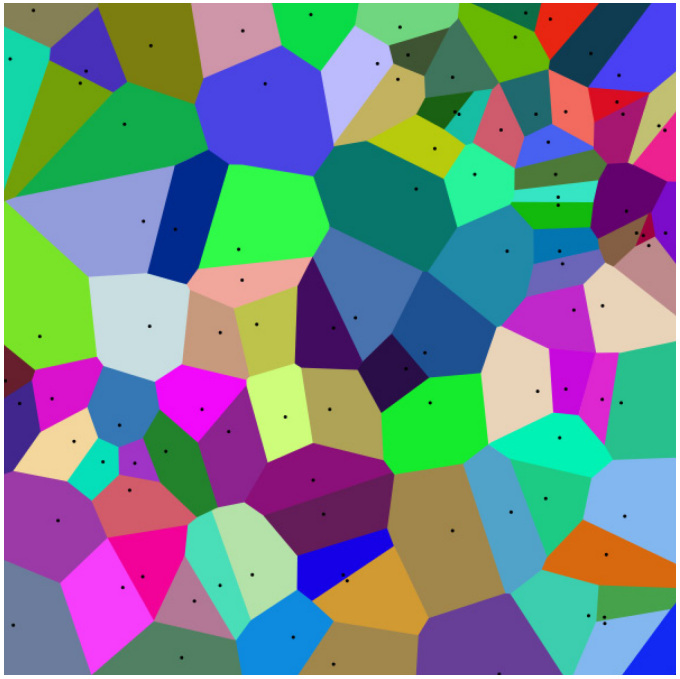
$$\rho_\alpha(r) = \sum_i^N \left|\varphi_i(r,\alpha)\right|^2 \quad \gamma_{\alpha\beta} = \nabla\rho_\alpha \cdot \nabla\rho_\beta$$

$$\left(-\frac{\nabla^2}{2} + \upsilon_{ext}(r_1) + \int \frac{\rho(r_2)}{r_{12}} dr_2 + F^{xc\alpha}\right)\varphi_i = \varepsilon_i \varphi_i$$

$$F_{\mu\nu}^{xc\alpha} = \int dr \left\{ \frac{\partial f}{\partial \rho_\alpha} \varphi_\mu \varphi_\nu + \left[2\frac{\partial f}{\partial \gamma_{\alpha\alpha}} \nabla\rho_\alpha + \frac{\partial f}{\partial \gamma_{\alpha\beta}} \nabla\rho_\beta \right] \cdot \nabla\left(\varphi_\mu \varphi_\nu\right) \right\}$$

Becke Grid

- Use spherical quadrature around each atom
- Harmonize quadratures from different atoms using Voronoi partitions

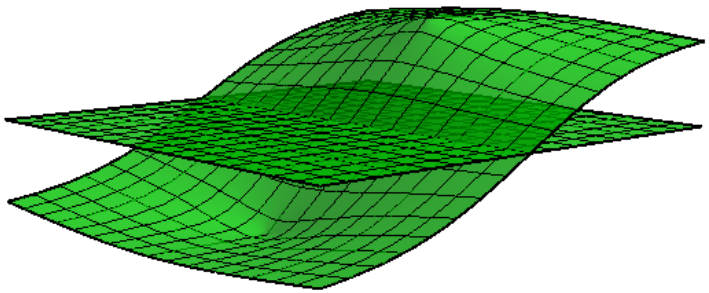


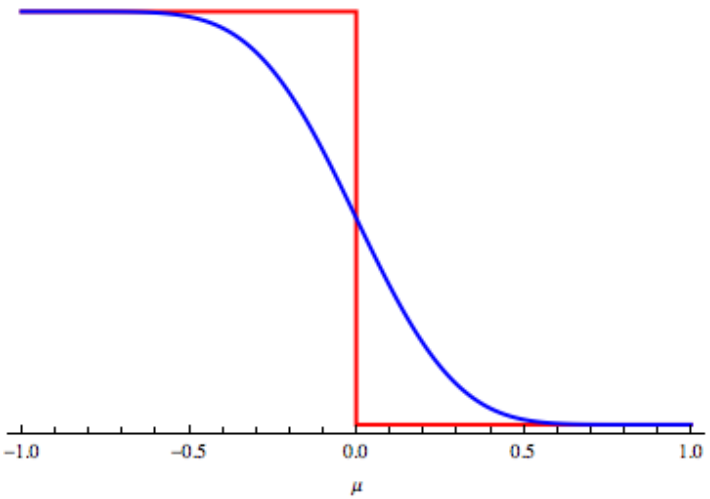
$$E_{xc} = \int dr\, f(r) \rightarrow \sum_{a}^{nAtoms} \sum_{i}^{nPts} \omega_{ai} l_i f(r_{ia})$$

Image courtesy Wikipedia Commons

Weighting the Points

$$\mu_{ab}(r) = \frac{r_a - r_b}{R_{ab}}$$





$$s(\mu_{ab}(r)) = \begin{cases} 1 & \mu_{ab}(r) \le 0 \\ 0 & \mu_{ab}(r) > 0 \end{cases}$$

$$P_a(r_{ai}) = \prod_{b \neq a} s(\mu_{ab}(r))$$

$$\omega_{ai} = \frac{P_a(r_{ai})}{\sum_b P_b(r_{ai})}$$

CPU Code

```
float[numAtoms][numAtoms] atomic_dist
...
for(ai=0; ai<numPoints; ++ai)
    for(b=0; b<numAtoms; ++b)
        pt_atom_dist[b] = dist(point[ai], atom[b])
    p_a = 0.0
    p_sum = 0.0
    for(c=0; c<numAtoms; ++c)
        p = 1.0
        rc  = pt_atom_dist[ c ]
        for(d=0; d<numAtoms; ++d)
            if( c == d )    continue
            rd  = pt_atom_dist[ d ]
            Rcd = atomic_dist[a][b]
            mu = (rc - rd) / Rcd
            p *= s( mu )
      p_sum += p
      if( c == center[ai] )
            p_a = p
    w[ai] = p_a / p_sumll
```

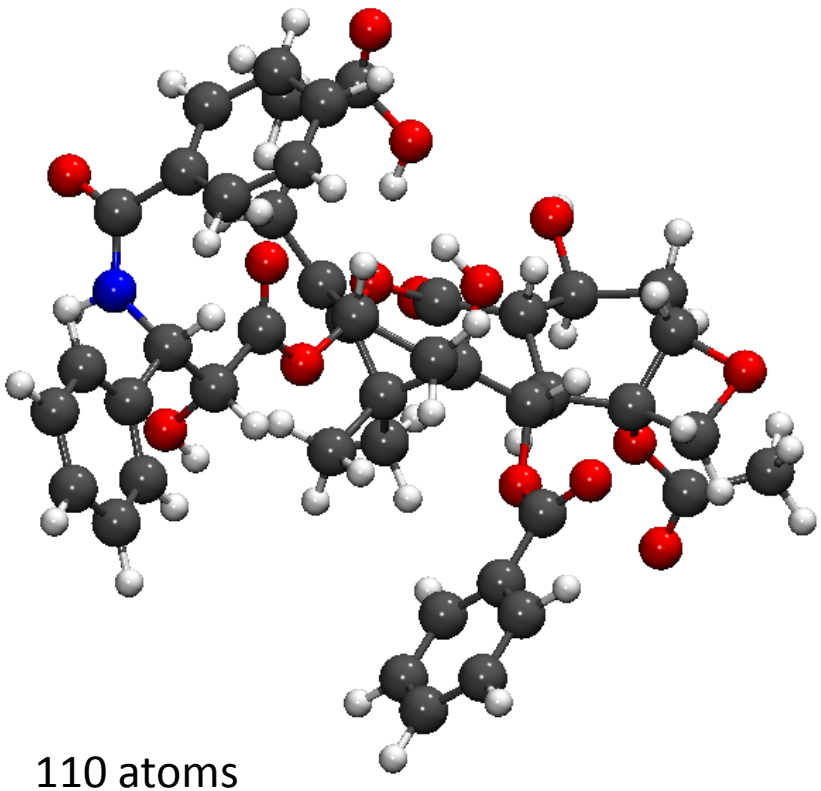
$$\omega_{ai} = \frac{\prod_{b \neq a} s(\mu_{ab}(r_{ai}))}{\sum_{c} \prod_{d \neq c} s(\mu_{cd}(r_{ai}))}$$

GPU Device Code

```
__global__ void kernel(...)
    ai = threadIdx.x + blockIdx.x*blockDim.x
    if( ai >= numPoints ) return
    p_a = 0.0
    p_sum = 0.0
    for(c=0; c<numAtoms; ++c)
        p = 1.0
        rc  = dist( point[ai], atom[c] )
        for(d=0; d<numAtoms; ++d)
            if( c == d )    continue
            rd  = dist( point[ai], atom[d] )
            Rcd = dist( atom[c],   atom[d] )
            mu = (rc - rd) / Rcd
            p *= s( mu )
      p_sum += p
      if( c == center[ai] )
            p_a = p
    w[ai] = p_a / p_sum
```

$$\omega_{ai} = \frac{\prod_{b \neq a} s(\mu_{ab}(r_{ai}))}{\sum_{c} \prod_{d \neq c} s(\mu_{cd}(r_{ai}))}$$

Performance



110 atoms
~15,000 grid points per

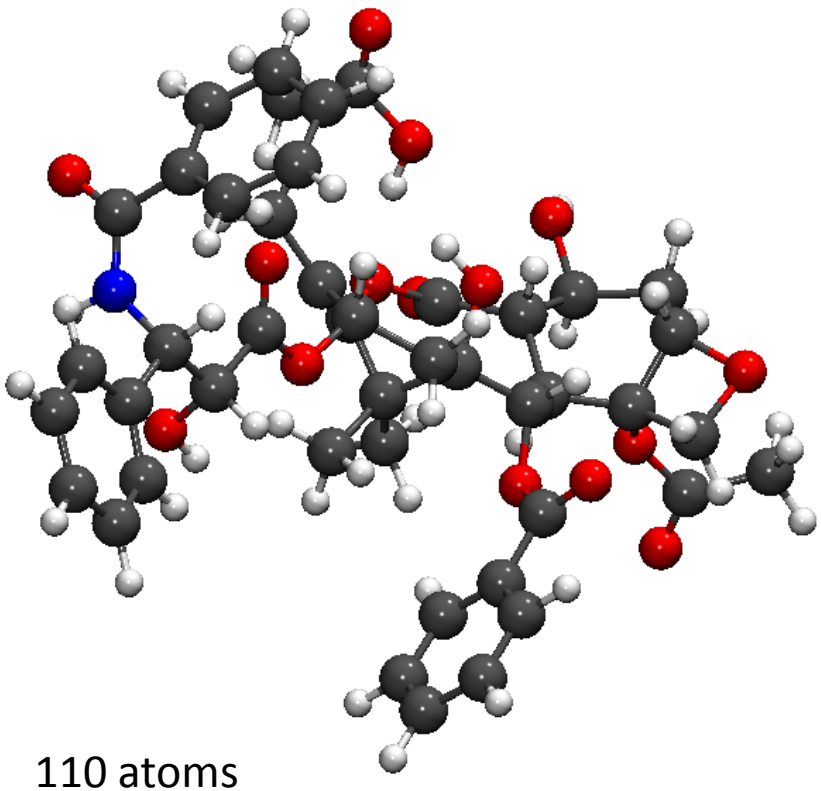
Processor	Runtime	GFLOPS
Xeon X5690	36.0 sec	1.67
Naïve Kernel		
Tesla C1060	1.89 sec	63.9
GTX 580	1.37 sec	88.4

GPU Device Code

```
__constant__ c_atoms[nAtoms]
__global__ void kernel(...)
    ai = threadIdx.x + blockIdx.x*blockDim.x
    if( ai >= numPoints ) return
    p_a = 0.0
    p_sum = 0.0
    for(c=0; c<numAtoms; ++c)
        p = 1.0
        rc  = dist( point[ai], atom[c] )
        for(d=0; d<numAtoms; ++d)
            if( c == d )    continue
            rd  = dist( point[ai], c_atom[d] )
            Rcd = dist_cache(c, d)
            mu = (rc - rd) / Rcd
            p *= s( mu )
      p_sum += p
      if( c == center[ai] )
            p_a = p
    w[ai] = p_a / p_sum
```

$$\omega_{ai} = \frac{\prod_{b \neq a} s(\mu_{ab}(r_{ai}))}{\sum_{c} \prod_{d \neq c} s(\mu_{cd}(r_{ai}))}$$

Performance



110 atoms
~15,000 grid points per

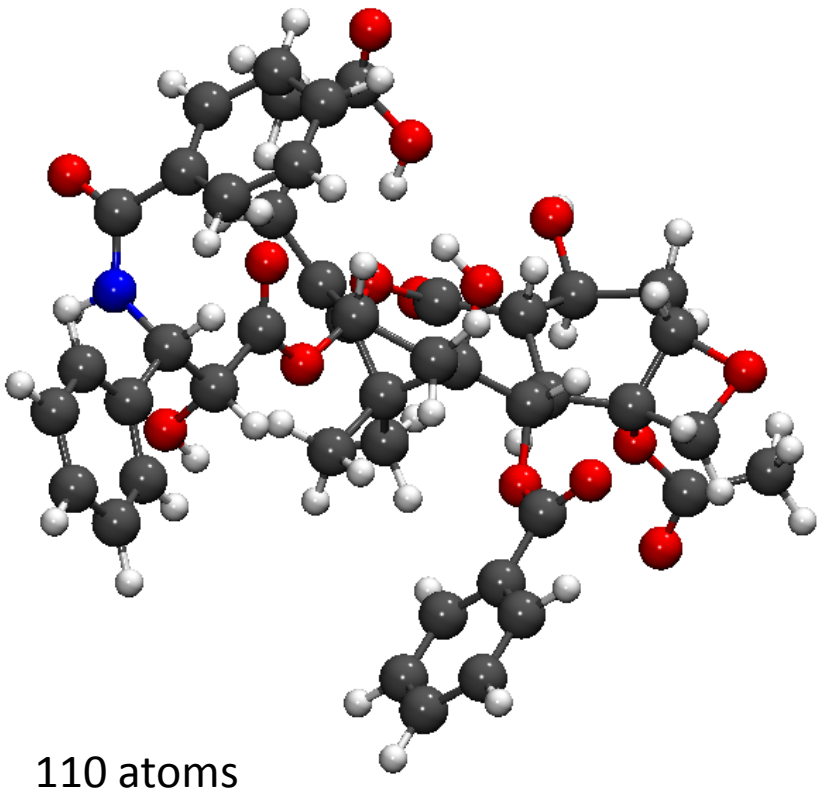
Processor	Runtime	GFLOPS
Xeon X5690	36.0 sec	1.67
Naïve Kernel		
Tesla C1060	1.89 sec	63.9
GTX 580	1.37 sec	88.4
Optimized Kernel		
Tesla C1060	0.75 sec	109
GTX 580	0.47	175

GPU Device Code

```
__global__ void kernel(...)
    ai = threadIdx.x + blockIdx.x*blockDim.x
    if( ai >= numPoints ) return
    p_a = 0.0
    p_sum = 0.0
    for(c=0; c<numAtoms; ++c)
        p = 1.0
        Rcd_next = dist_cache(c, 0)
        rc  = dist( point[ai], atom[c] )
        for(d=0; d<numAtoms; ++d)
            Rcd = Rcd_next
            Rcd_next = dist_cache(c, d+1)
            if( c == d )    continue
            rd  = dist( point[ai], c_atom[d] )
            mu = (rc - rd) / Rcd
            p *= s( mu )
      p_sum += p
      if( c == center[ai] )
            p_a = p
    w[ai] = p_a / p_sum
```

$$\omega_{ai} = \frac{\prod_{b \neq a} s(\mu_{ab}(r_{ai}))}{\sum_{c} \prod_{d \neq c} s(\mu_{cd}(r_{ai}))}$$

Performance



110 atoms
~15,000 grid points per

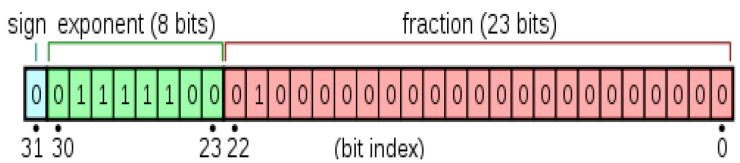
Processor	Runtime	GFLOPS
Xeon X5690	36.0 sec	1.67
Naïve Kernel		
Tesla C1060	1.89 sec	63.9
GTX 580	1.37 sec	88.4
Optimized Kernel		
Tesla C1060	0.75 sec	109
GTX 580	0.47	175
Double Buffered		
Tesla C1060	0.65	126
GTX 580	0.51	161

Mixed Precision Integral Evaluation

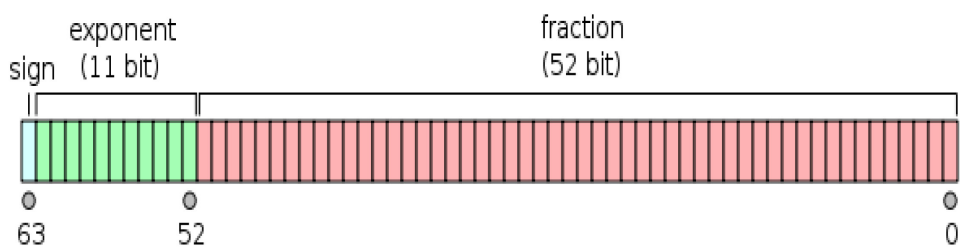
$$(ij|kl) = \frac{\varphi_i(r_1)\varphi_j^*(r_1)\varphi_k(r_2)\varphi_l^*(r_2)}{|r_1 - r_2|}$$

$$F_{ij} = \sum_{kl} P_{kl}[2(ij|kl) - (ik|jl)]$$

IEEE 754 Single Precision ~7 decimal digits



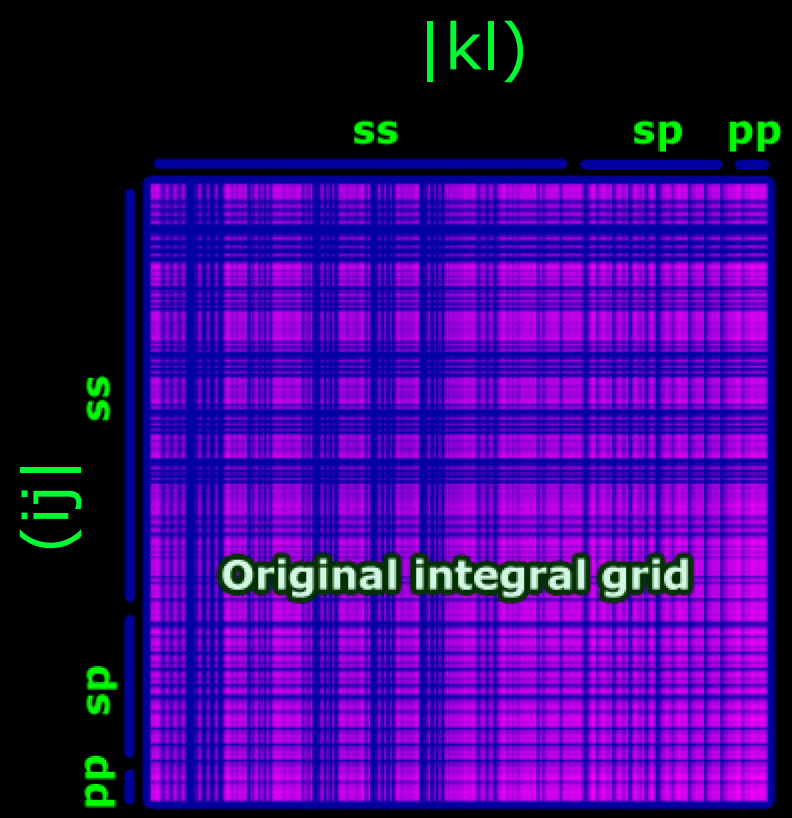
IEEE 754 Double Precision ~15 decimal digits



Images courtesy Wikipedia Commons

Arrangement of Integrals

$$(ij|kl) \leq \sqrt{(ij|ij)}\sqrt{(kl|kl)} > 10^{-11}$$



Note: Pre-computation puts problem in form suitable for GPU – this is (in our experience) the key to good performance

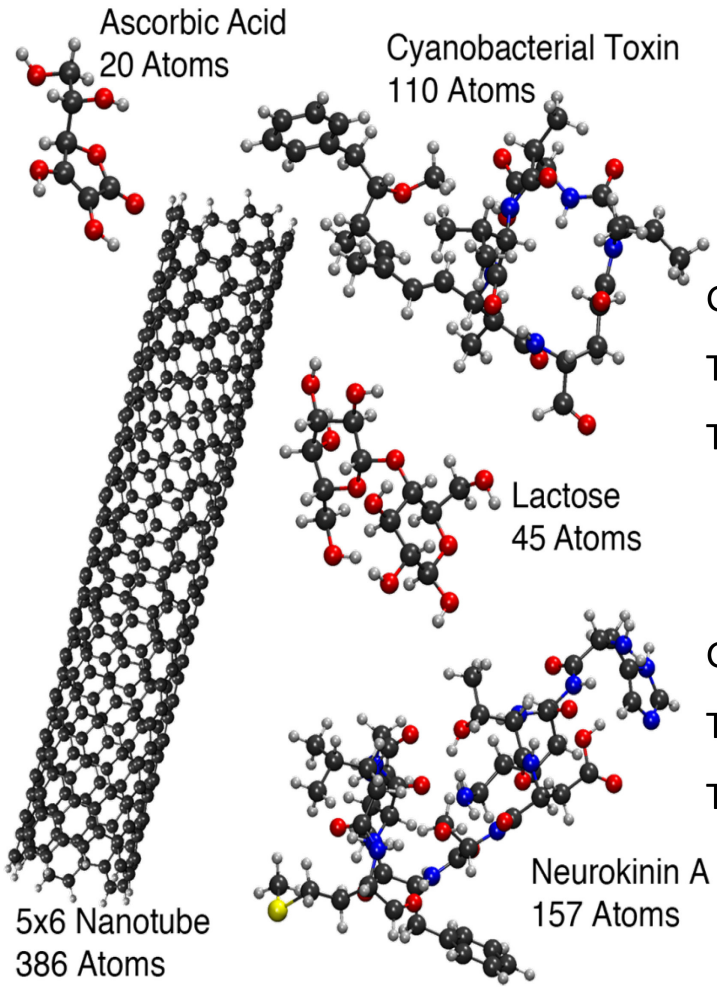
Mixed Precision Integral Evaluation

	5	8	5	1	8	.	1	2	NOISE		
+				1	4	.	8	3	7	2	6
	5	8	5	3	2	.	9	5	?	?	?

- Effective precision is often truncated by intermediate operations.

$$F_{ij}^{64} \mathrel{+}= \left\{ \sum_{kl} P_{kl}^{32} [2(ij|kl)^{32} - (ik|jl)^{32}] \right\}^{64}$$

“Single” vs. Double Precision



	Ascorbic Acid	Lactose	Cyano Toxin
GAMESS	-680.5828947	-1289.6666250	-2491.2058893
TeraChem DP	-680.5828947	-1289.6666250	-2491.2058890
TeraChem SP	-680.5828071	-1289.6664266	-2491.2053660

	Neurokinin A	5x6 Nanotube
GAMESS	-4089.6883770	-13790.1415171
TeraChem DP	-4089.6883762	-13790.1415176
TeraChem SP	-4089.6879824	-13790.1389987

Mixed Precision Integral Evaluation

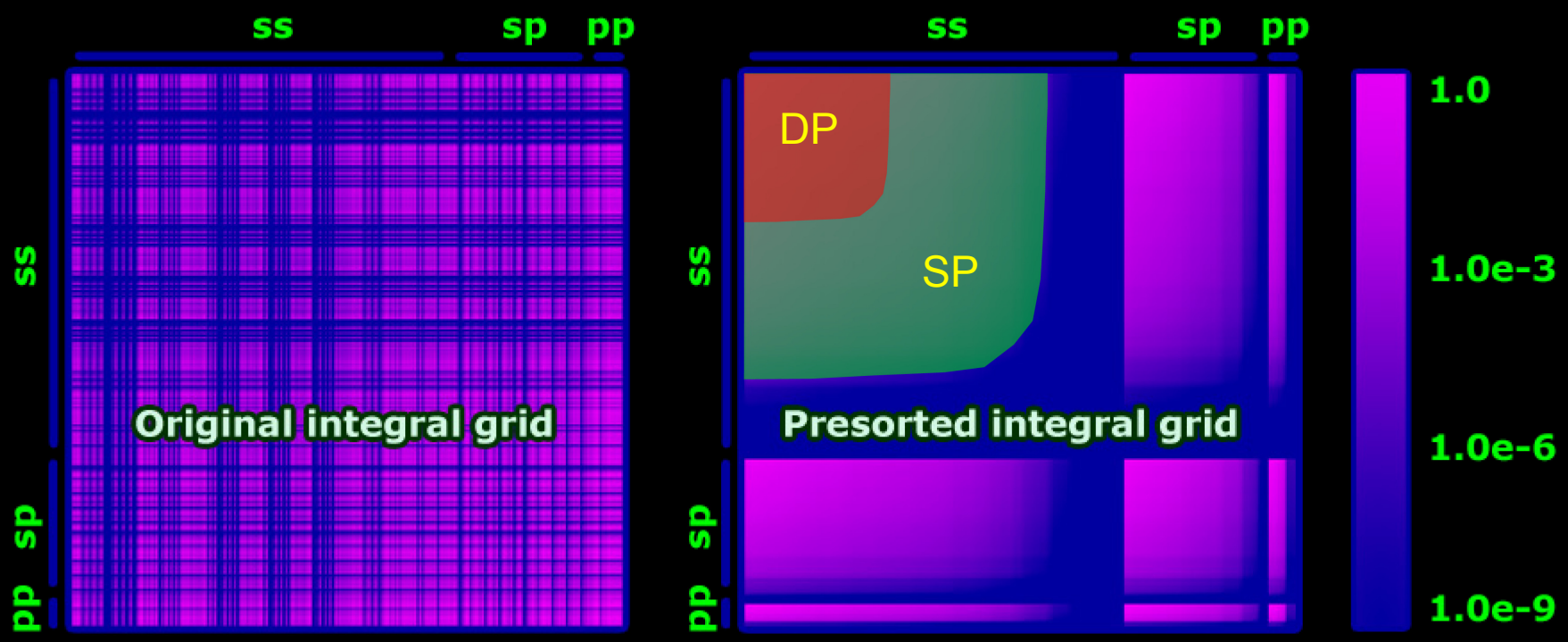
	5	8	5	1	8	.	1	2	NOISE		
+				1	4	.	8	3	7	2	6
	5	8	5	3	2	.	9	5	?	?	?

- Effective precision is often truncated by intermediate operations.

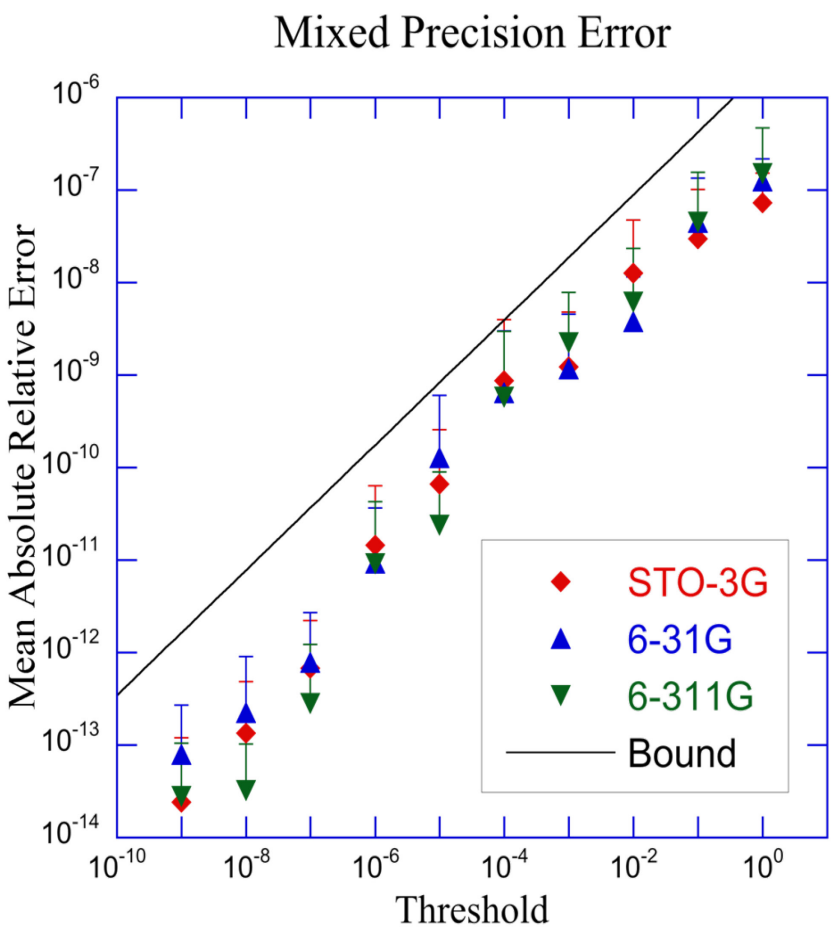
$$J_{ij}^{64} \mathrel{+}= \left\{ \sum_{kl} 2P_{kl}^{32}(ij|kl)^{32} \right\}^{64} \quad \text{If } |(ij|kl)P_{kl}| \leq PrecThre$$

$$J_{ij}^{64} \mathrel{+}= \sum_{kl} 2P_{kl}^{64}(ij|kl)^{64} \quad \text{If } |(ij|kl)P_{kl}| > PrecThre$$

Single and double precision GPU kernels



Precision Thresholds

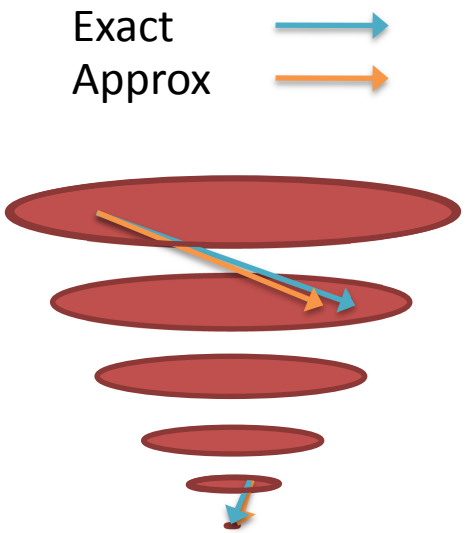


$$\text{Error} < 2 \times 10^{-6} \left(\text{PrecThres}\right)^{0.7}$$

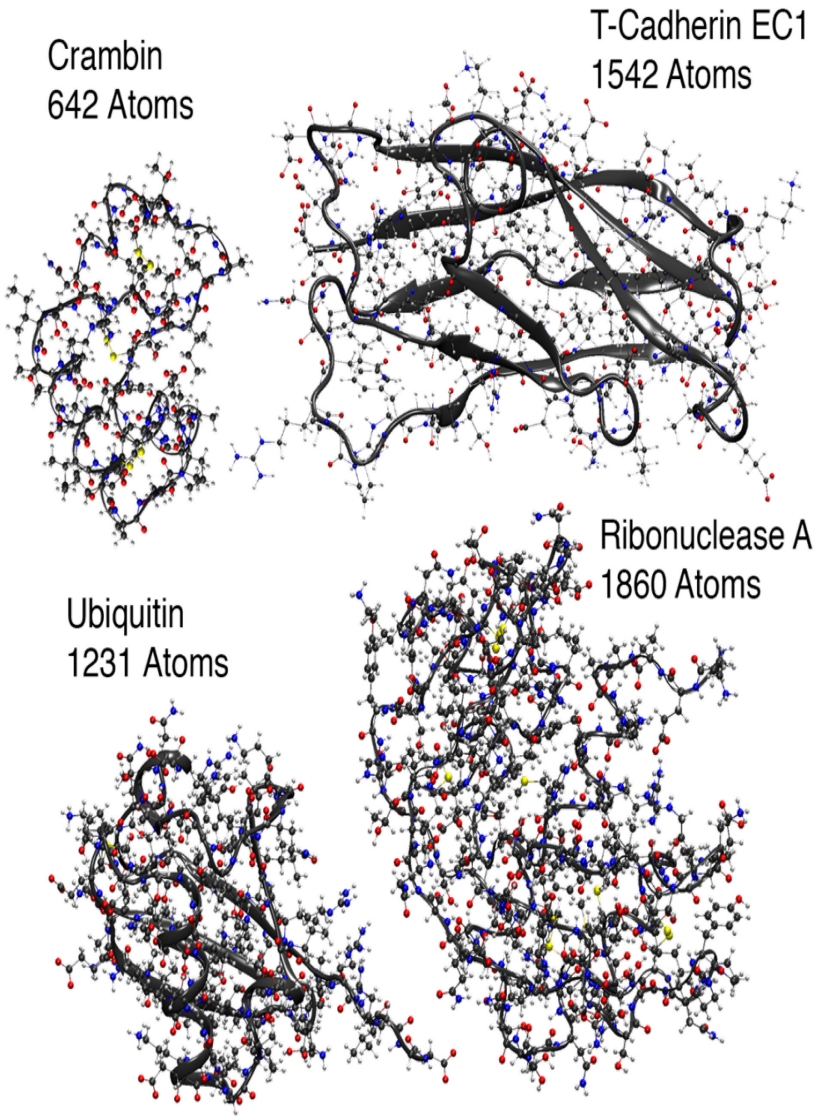
$$\text{PrecThres} = \left(5 \times 10^{5}\,\text{Error}\right)^{\frac{1}{0.7}}$$

Dynamic Precision

- The density matrix in early SCF iterations is approximate.
- Computing the Fock matrix precisely from an imprecise density matrix is essentially computing error.
- By varying the PrecThres between iterations we can avoid over-computing F without hampering convergence.



Dynamic Precision Performance



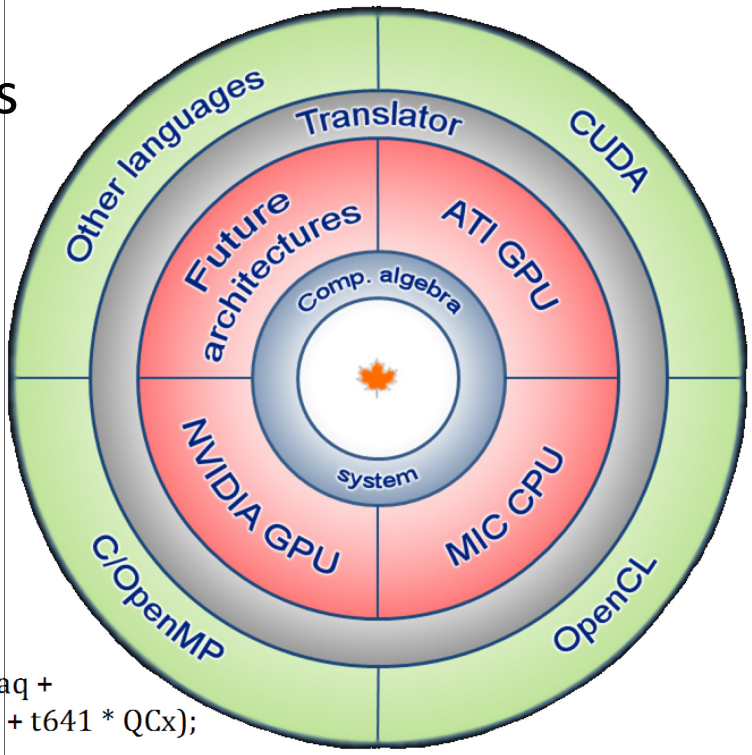
	Nvidia Tesla C2050		
	Double Runtime	Dynamic Runtime	Speedup
Ascorbic Acid	4.43	2.93	1.5
Lactose	15.79	8.41	1.9
Cyano Toxin	156.79	68.44	2.3
Neurokinin A	337.68	149.76	2.3
Nanotube	21276.05	7613.76	2.8
Crambin	8862.12	4312.67	2.1
Ubiquitin	102562.54	47152.51	2.2
T-Cadherin EC1	139691.26	64932.58	2.2

Dynamic Precision Reliability

	Double Precision		Dynamic Precision		Precision	Conv
	Final Energy	Iter	Final Energy	Iter	Error	Thres
Nanotube (Minimum)	-13793.7293925221	24	-13793.7293925**323**	23	1.02E-08	10^{-7}
	-13793.7293924922	15	-13793.729392**8287**	15	3.37E-07	10^{-5}
Nanotube (2000K)	-13790.1415175662	29	-13790.141517558**4**	27	7.80E-09	10^{-7}
	-13790.1415175332	18	-13790.1415**91026**	18	1.57E-06	10^{-5}
Crambin	-17996.6562925538	18	-17996.6562926**036**	18	4.98E-08	10^{-7}
	-17996.6562925535	12	-17996.656292**7894**	12	2.36E-07	10^{-5}
Ubiquitin	-29616.4426376594	24	-29616.442637659**6**	24	2.00E-10	10^{-7}
	-29616.4426376302	18	-29616.4426376**655**	18	3.53E-08	10^{-5}
T-Cadherin EC1	-36975.6726049407	21	-36975.672604939**4**	21	1.30E-09	10^{-7}
	-36975.6726049265	16	-36975.67260**8777**	16	4.88E-08	10^{-5}
Ribonuclease A	-50813.1471248227	19	-50813.147124**8179**	19	4.80E-09	10^{-7}
	-50813.1471247051	12	-50813.14712**50247**	12	3.20E-07	10^{-5}

Managing D-Function Kernels

- Automated architecture tuning
  - Computer algebra → Pseudocode → Translation to CUDA/OpenMP/etc.
  - Empirically test different pathways
  - E.g. (pp|pp) integrals:




```
float t650 = Pxy * R2100 + Pxz * R2010 + Pxx * R3000;
float t652 = Pxy * R1100 + Pxz * R1010 + Pxx * R2000;
float t639 = t650 * rtap + t652 * PAx;
float t658 = -rtap * R2000 - PAx * R1000;
float t660 = Pxx * QDx + QDy * Pxy + QDz * Pxz;
float t659 = -Pxx * R1000 - Pxy * R0100 - Pxz * R0010;
float t656 = rtap * R1000 + PAx * R0000;
float t641 = t656 * PBx + (R0000 - t658) * rtap;
float t640 = -t652 * rtap + t659 * PAx;
float t624 = t640 * PBx + (t659 - t639) * rtap;
tmp_0 += ((t639 * PBx + ((Pxy * R3100 + Pxz * R3010 + Pxx * R4000) *
      rtap + t650 * PAx + t652) * rtap) * rtaq + t624 * QCx + t641 * Pxx) * rtaq +
t660 * ((t658 * PBx + (-rtap * R3000 - PAx * R2000 - R1000) * rtap) * rtaq + t641 * QCx);
```

vs.

```
float R1000 = a * R0001;
tmp_0 += R1000 * ((-PBx * (Pxz * PAx + PAz * Pzz + PAy * Pyz) * QDz - PBx * (PAy * Pyx + PAz * Pzx + Pxx * PAx) * QCx
- PBx * (PAy * Pyx + PAz * Pzx + Pxx * PAx) * QDx - PBx * (PAz * Pzy + PAy * Pyy + Pxy * PAx) * QDy) * rtaq +
((-Pxz * QDz + PAz * Pzx - Pxy * QDy - Pxx * QDx - Pxx * QCx + Pxx * PBx + Pxx * PAx + PAy * Pyx) * rtaq +
((PAy * Pyx + PAz * Pzx + Pxx * PBx + Pxx * PAx) * QDx +  (Pxy * PAx + PAz * Pzy + PAy * Pyy + Pxy * PBx) * QDy +
(PAy * Pyz + Pxz * PAx + PAz * Pzz + Pxz * PBx) * QDz) * QCx) * rtap);
```

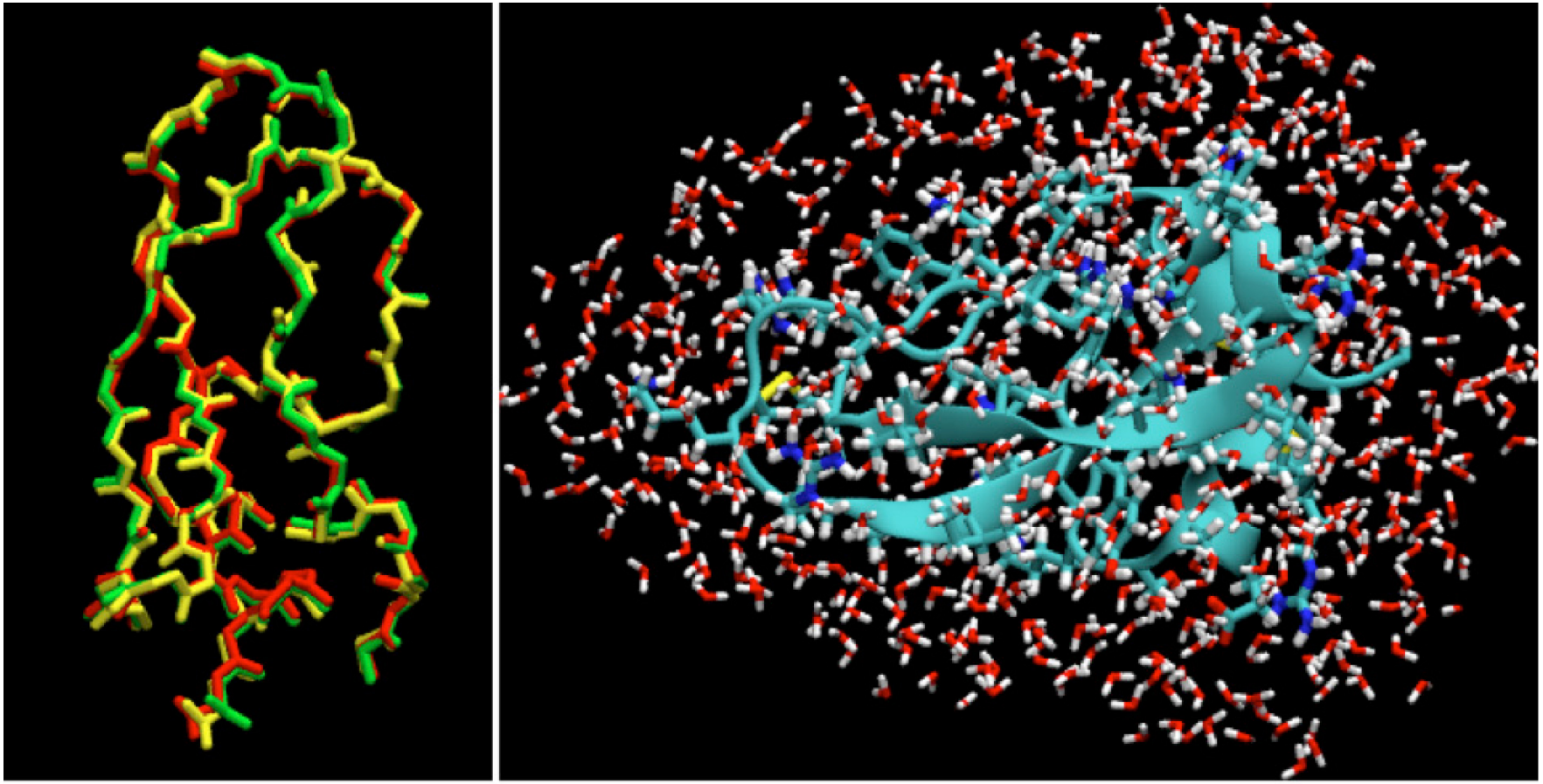
TeraChem Capabilities

- Restricted and unrestricted HF and DFT, various DFT functionals
- MM water, DFT-D, HF-D (w/ Stefan Grimme, U. Münster)
- Excited states: CIS and TDDFT
- Ab initio molecular dynamics (ground & excited states)
- Geometry optimization including constraints
- Reaction path and transition state search (through DL-FIND, Kastner)
- Electrostatic potential maps
- Single / Dynamical / Double precision
- Up to d angular momentum basis functions
- QM/MM with TeraChem/Amber – (w/ Ross Walker, UCSD/SDSC)
- Global optimization (w/ Bernd Engels, U Wuerzburg)

Is RHF/DFT Good Enough for Protein Structure?

- Full answer will require free energy calculations
- At least a preliminary answer can be obtained by
  - optimizing structures from the PDB (local minimization)
  - comparison to experimental structures (RMSD)
  - assessment of protein structure health (MolProbity)

Optimized Geometry for BPTI

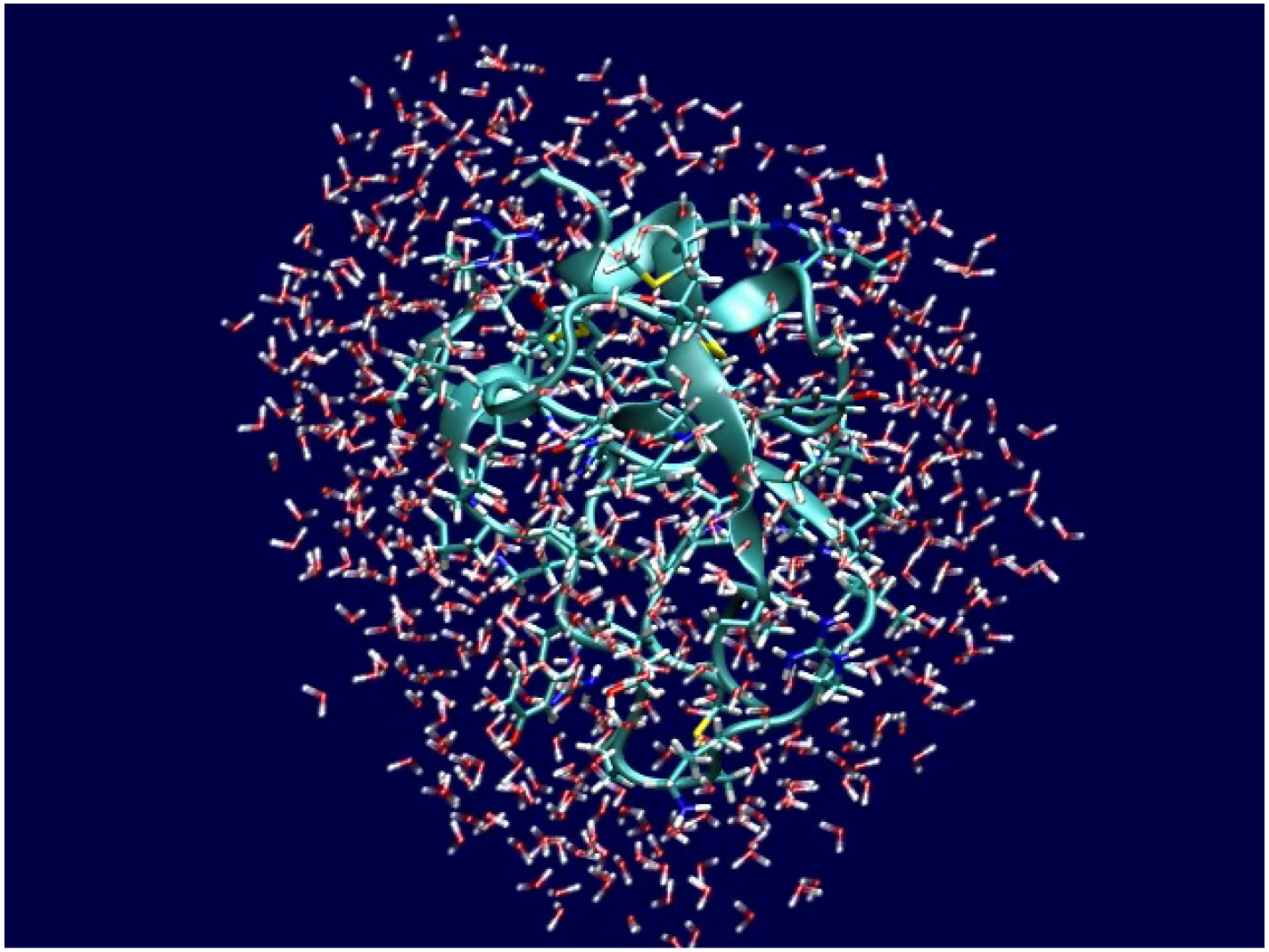


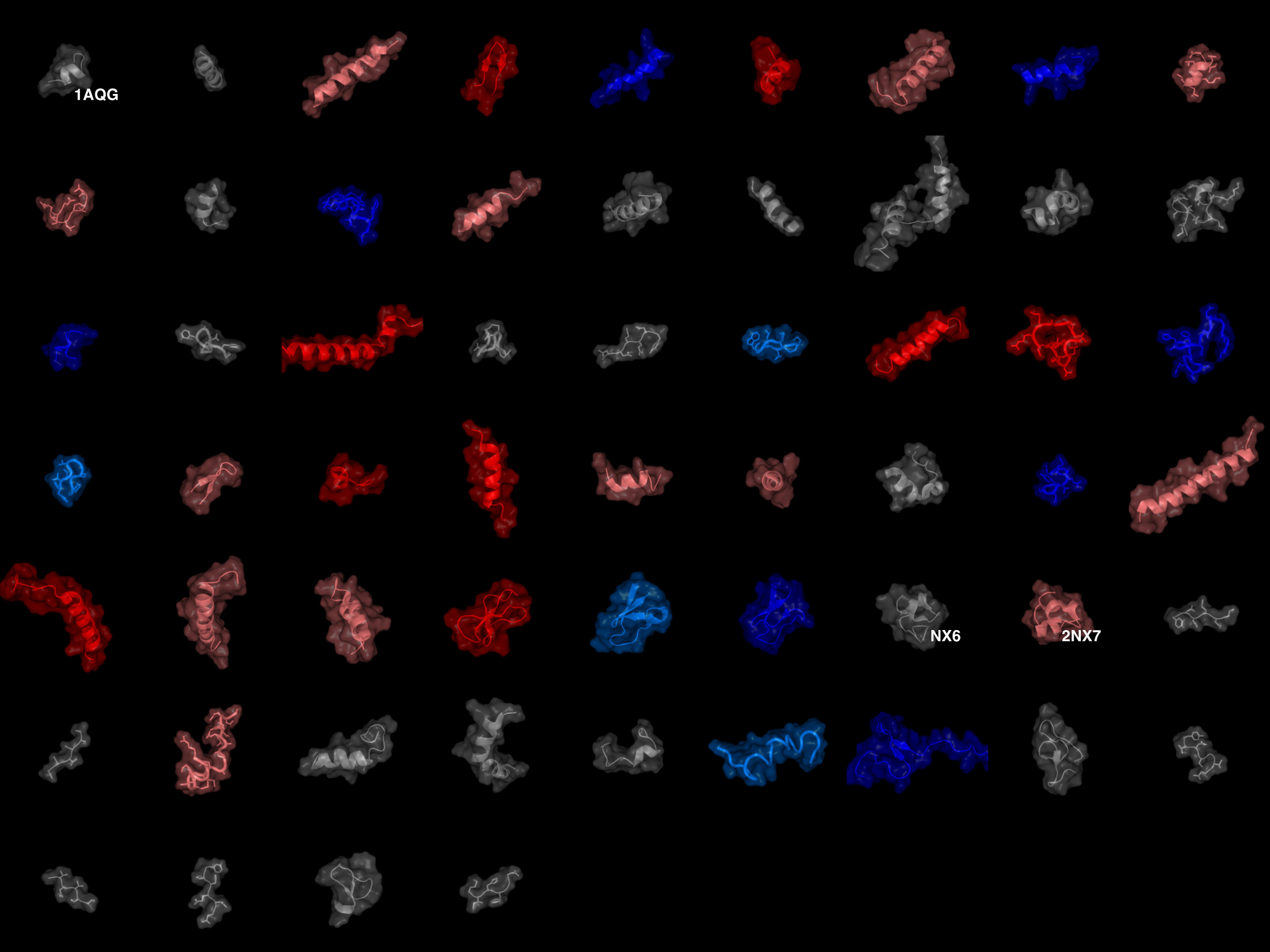
Red = X-Ray; Yellow = RHF/STO-3G; Green = RHF/6-31G

RMSD = 0.42A/0.31A for STO-3G/6-31G

1ps QM/MM simulation of microsolvated BPTI

882 QM atoms (BPTI), 1752 MM atoms (water), sto-3G




1AQG
NX6
2NX7

Health Checks

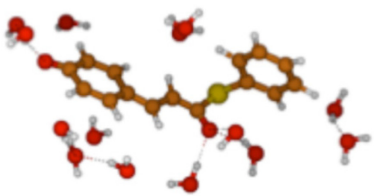
Method	Ca-RMSD	Clash/1000	Poor Rot	Good Rama	
AMBER	0.61	3	9%	80%	
RHF/MINI	0.69	45	18%	80%	
RHF/MINI-D	0.67	44	18%	78%	
RHF/STO-3G	0.70	40	15%	75%	
RHF/3-21G	0.68	14	11%	86%	
RHF/6-31G	0.72	8	9%	86%	
wPBEh/MINI	0.71	75	24%	69%	
wPBEh/STO-3G	0.77	72	19%	71%	
wPBEh/3-21G	0.69	21	15%	81%	
wPBEh/6-31G	0.63	9	13%	85%	
Experiment	--	9	19%	80%	

Except for minimal basis sets, no major problems
"Health" improves as basis set increases
Why no BLYP or B3LYP? Convergence difficulties…

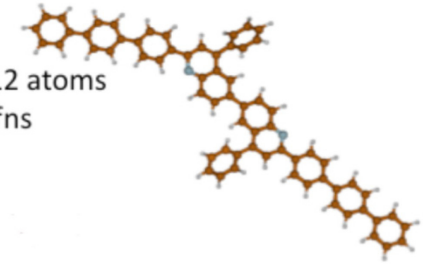
Excited States

molecule (atoms; basis functions)	CIS timings (s)		speedup		$\Delta E_{S0/S1}$ (eV)	
	GPU	GAMESS	SCF	CIS	GPU	GAMESS
B3PPQ oligoquinoline (112; 700)	41.9	371.5	11	9	4.7056398	4.7056482
PYP chromophore + 101 waters (332; 1501)	133.2	3032.7	48	23	3.6409681	3.6409411
PYP chromophore + 146 waters (467; 2086)	217.5	8654.9	84	40	3.6394478	3.6394222
PYP chromophore + 192 waters (605; 2684)	318.1	20546.8	131	65	3.6425942	3.6425632
PYP chromophore + 261 waters (812; 3581)	493.2	57800.5	218	117	3.6454079	3.6453773
PYP chromophore + 397 waters (1220; 5349)	894.0	243975.7	426	273	3.6496150	3.6495829
PYP chromophore + 487 waters (1490; 6519)	1221.2	562606.6	547	461	3.6549966	3.6549636

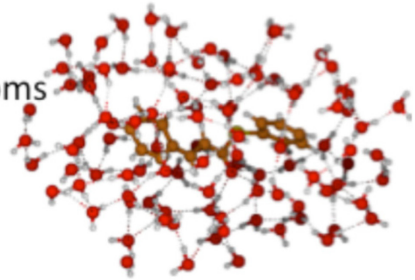
chromophore of PYP
+ 14 waters, 71 atoms
370 basis fns



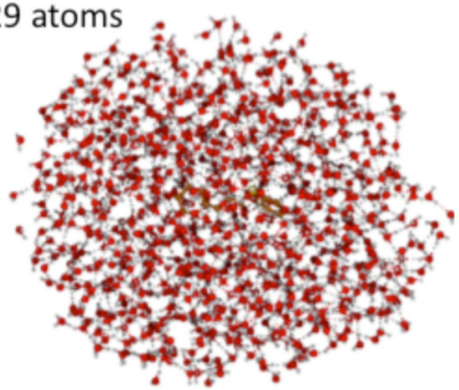
B3PPQ, 112 atoms
700 basis fns



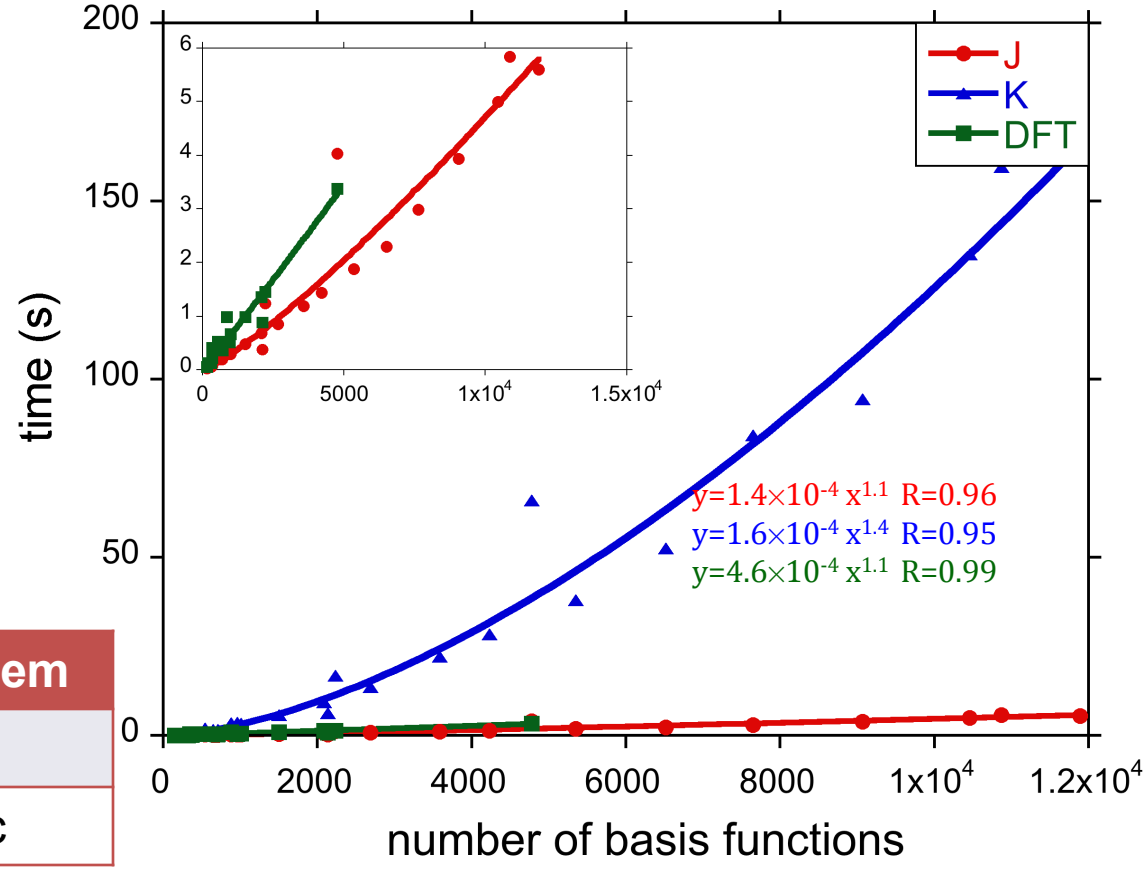
chromophore of PYP
+ 101 waters, 332 atoms
1,501 basis fns



chromophore of PYP
+ 900 waters, 2729 atoms
11,888 basis fns



TD-B3LYP AX Build Timings



Timings vs. GAMESS

	GAMESS	TeraChem
S1	67 sec	18 sec
S2	1191 sec	131 sec

TeraChem: 8 GPUs (Tesla C1060)
GAMESS: Dual Quad-Core Nehalem

Errors < 1 microhartree

Conclusions

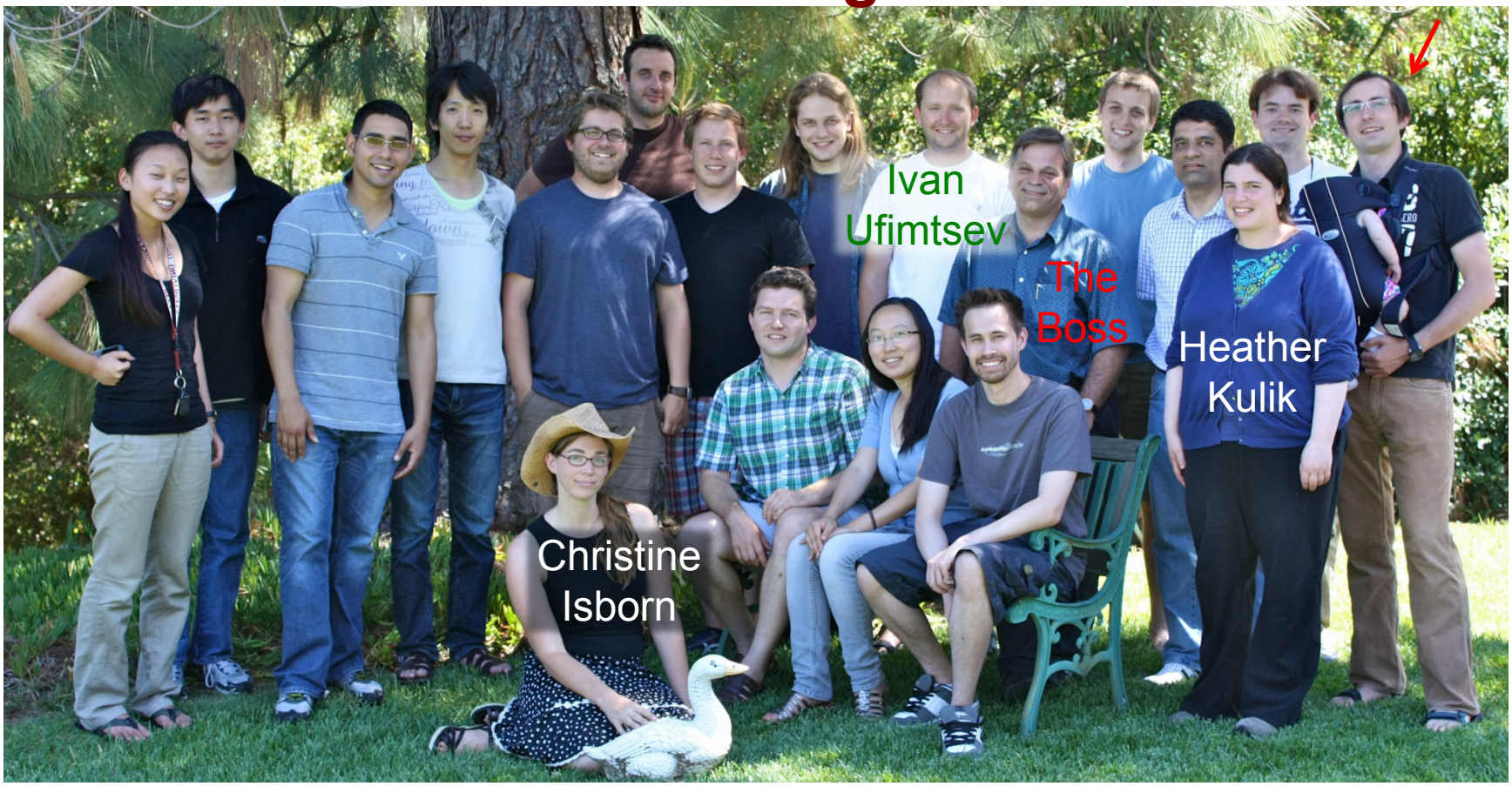
- Understanding an architecture’s constraints and corresponding advantages is critical for developing efficient software.

- It is possible to quickly achieve very good performance on GPUs if your problem maps nicely onto the architecture's strengths.

- Mixed precision schemes are fast and provide acceptable accuracy. More aggressive precision schemes are likely warranted as molecules get larger.

- GPUs bring QC calculations to a new level of computational performance and can qualitatively extend the domain of ab initio methods to large biologically relevant molecules.

Acknowledgements



Funding

STTR - AFOSR

NVIDIA

PetaChem, LLC

Not shown: Jeff Gour, Ed Hohenstein

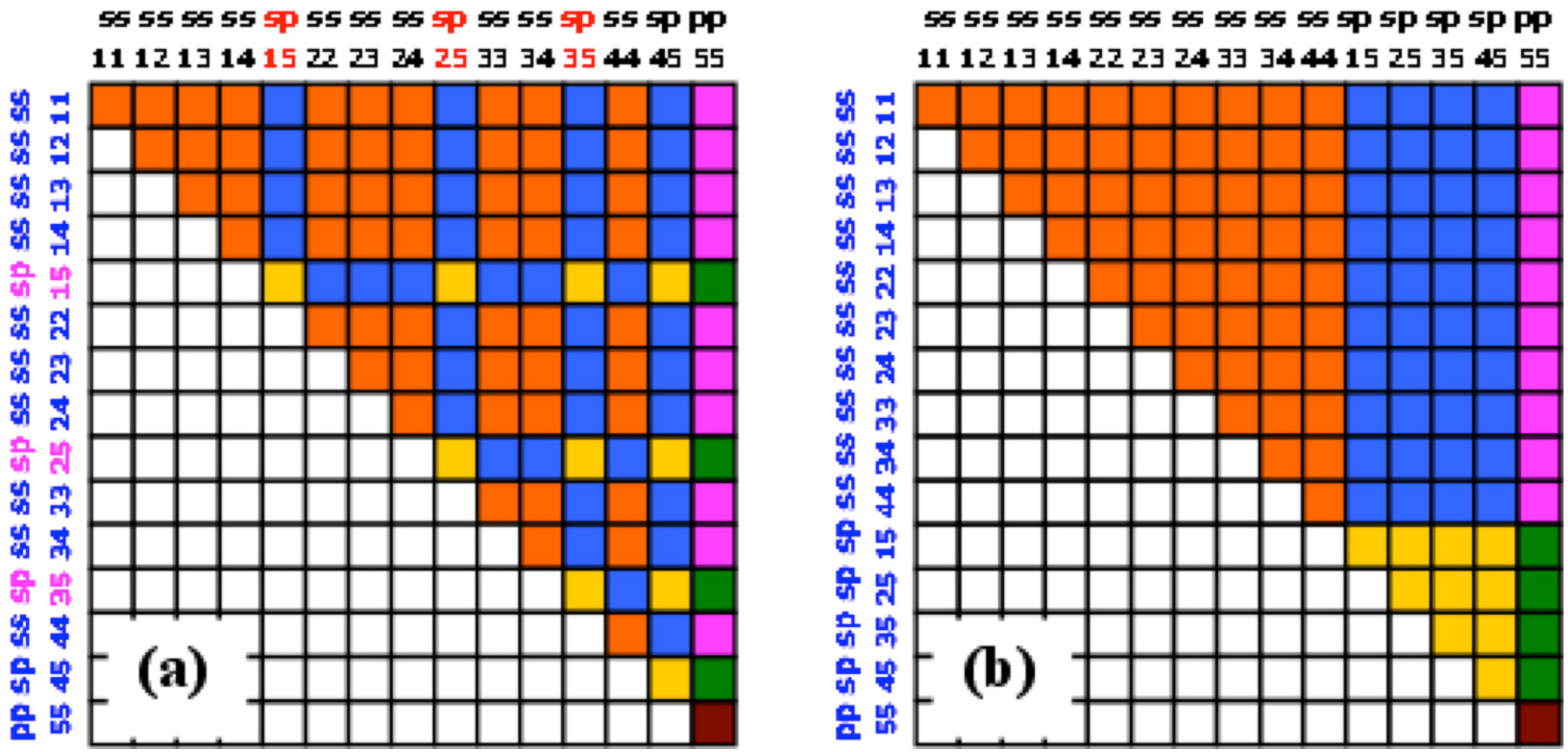
Jason Quenneville, Spectral Sciences

NCSA

Vlad Kindratenko

Guochun Shi

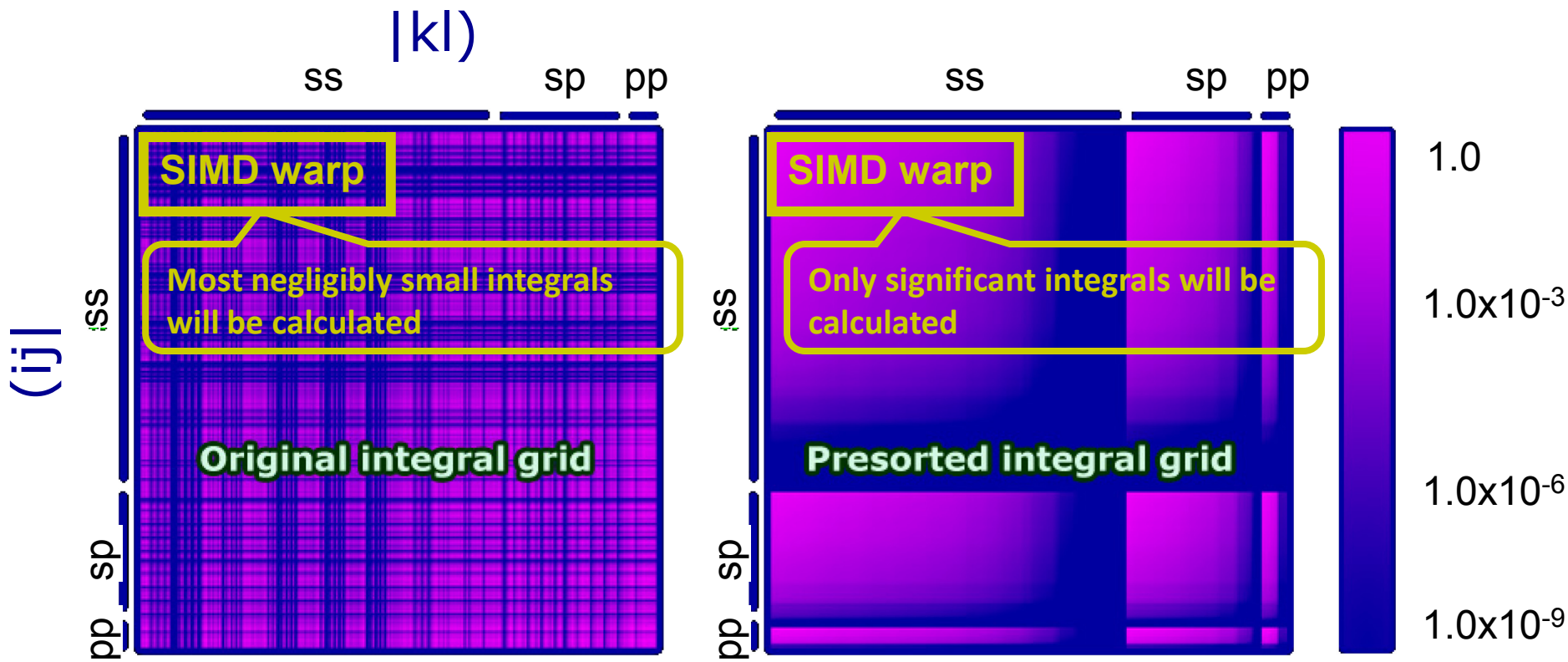
Parallel Computation of Two-Electron Integrals



Different code for each integral type, e.g. [ss|ss] vs [sp|ss]
Rearrange computation to optimize kernel execution

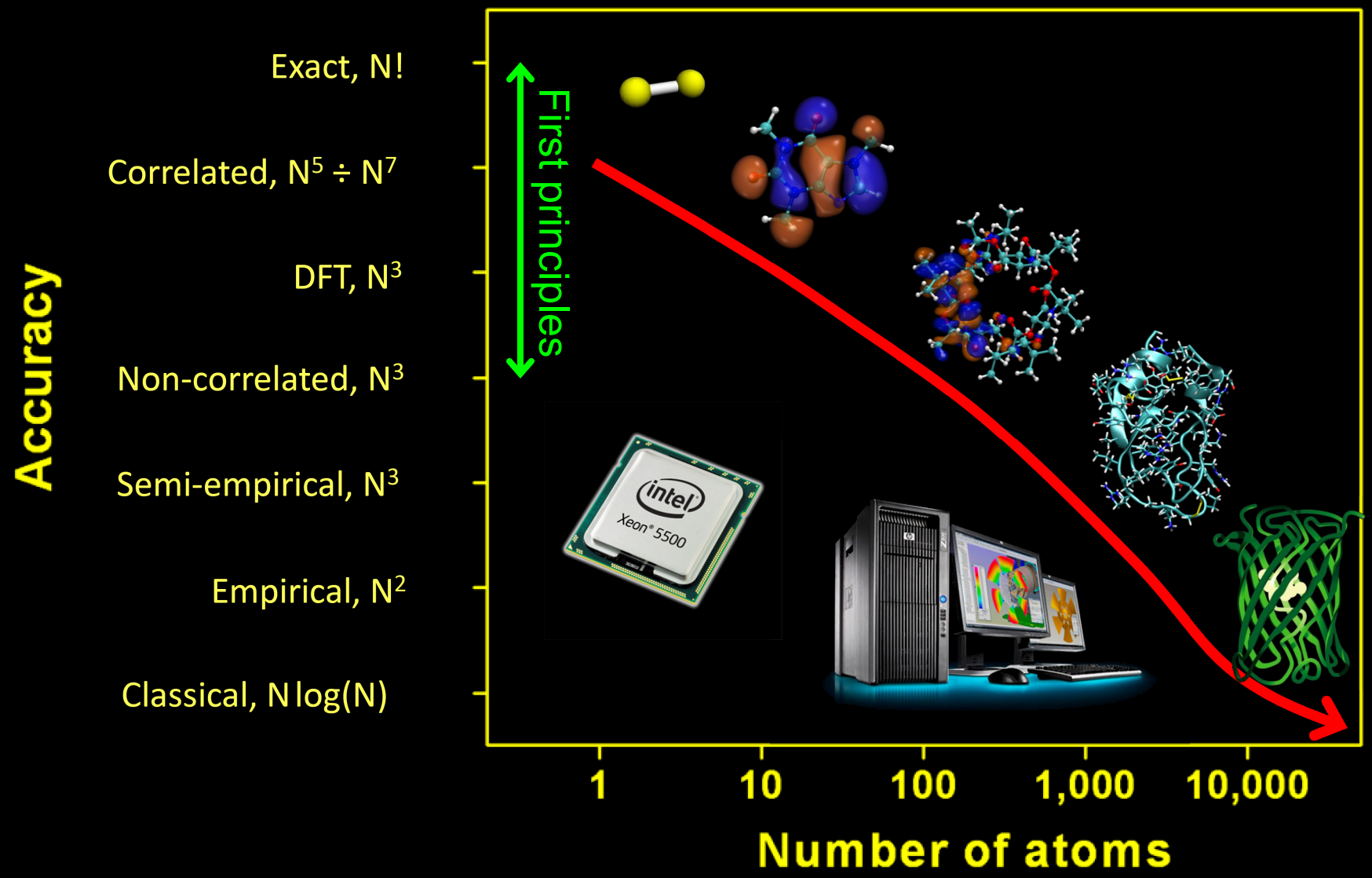
Arrangement of Integrals

$(ij|kl) \leq \sqrt{(ij|ij)}\sqrt{(kl|kl)} > 10^{-11}$ leaves only N^2 out of N^4 integrals



Note: Pre-computation puts problem in form suitable for GPU – this is (in our experience) the key to good performance

Molecular modeling on traditional processors



Excited States in TeraChem

CI-Singles and TDA-TDDFT

$$\mathbf{A}\mathbf{x} = \omega \mathbf{x}$$

$$A_{ia,jb} = \delta_{ij}\delta_{ab}\left(\varepsilon_a - \varepsilon_i\right) + (ia\,|\,jb) + (ia\,|\,f_{xc}\,|\,jb)$$

f_{xc} = HF exchange for CIS; 2nd Derivative of K_{xc} for TDDFT

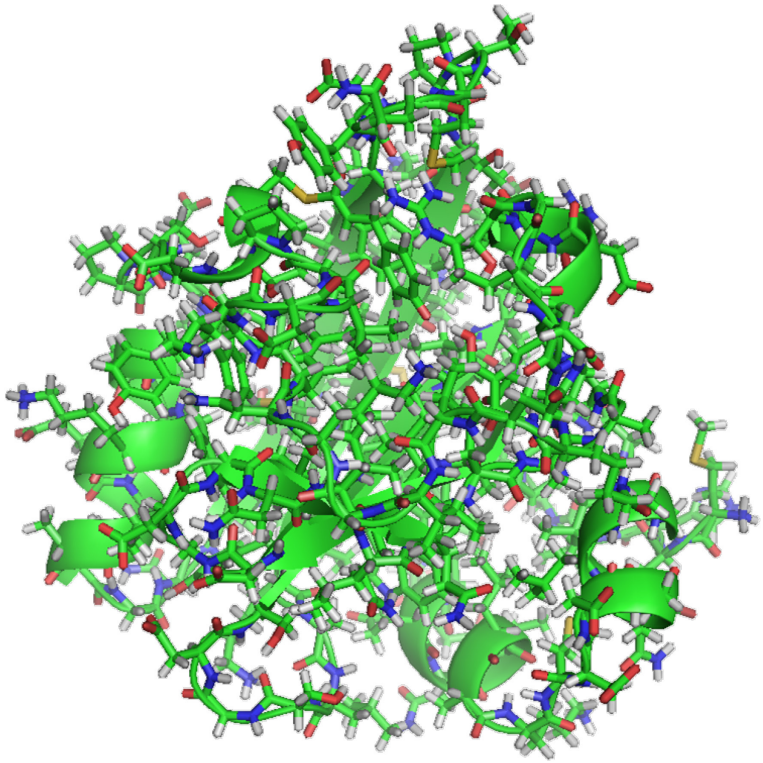
$$\left(\mathbf{A}\mathbf{x}\right)_{jb} += \sum_{\mu\nu} C_{\mu j} C_{\nu b} F_{\mu\nu}$$

$$F_{\mu\nu} = \sum_{\lambda\theta} T_{\lambda\theta}\left\{(\mu\lambda\,|\,\nu\theta) - (\mu\nu\,|\,\lambda\theta)\right\} \qquad T_{\lambda\theta} = \sum_{ia} X_{ia} C_{\lambda i} C_{\theta a}$$

Reuse SCF J and K kernels with transition density matrices

Isborn, Luehr, Ufimtsev and Martinez, JCTC **7** 1814 (2011)

CIS on Large Systems



Photoactive Yellow Protein

1935 QM atoms, 5955 basis fns,
6319 MM waters (STO-3G)

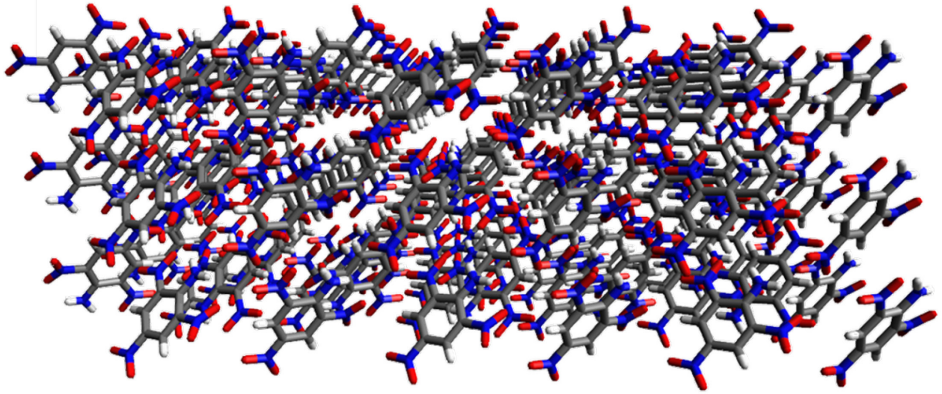
$$\Delta E^{CIS}_{S0-S1} = 4.2eV$$

Time to solution: 132 sec/iteration
(3 x Tesla C1060)

2,4,6-trinitroaniline energetic material

9 unit cells : 2960 atoms,
9072 basis fns (STO-3G)

$$\Delta E^{CIS}_{S0-S1} = 4.0eV$$

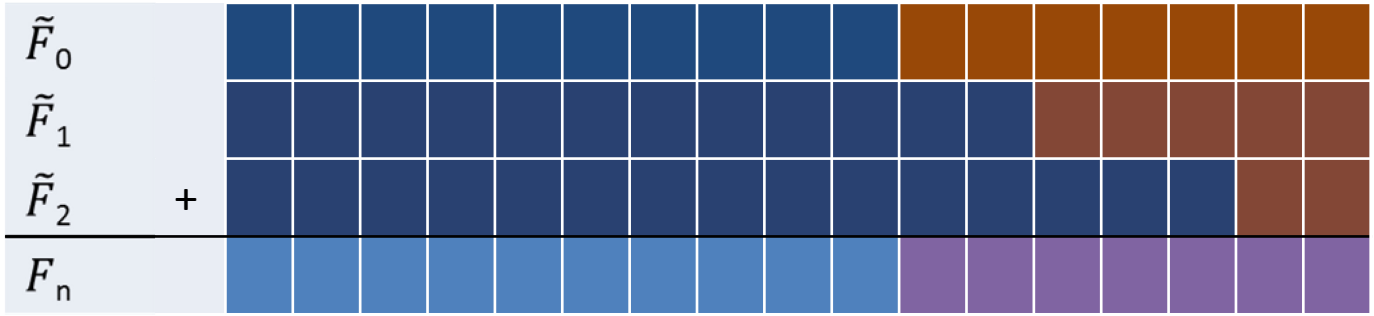


Iterative Fock Construction

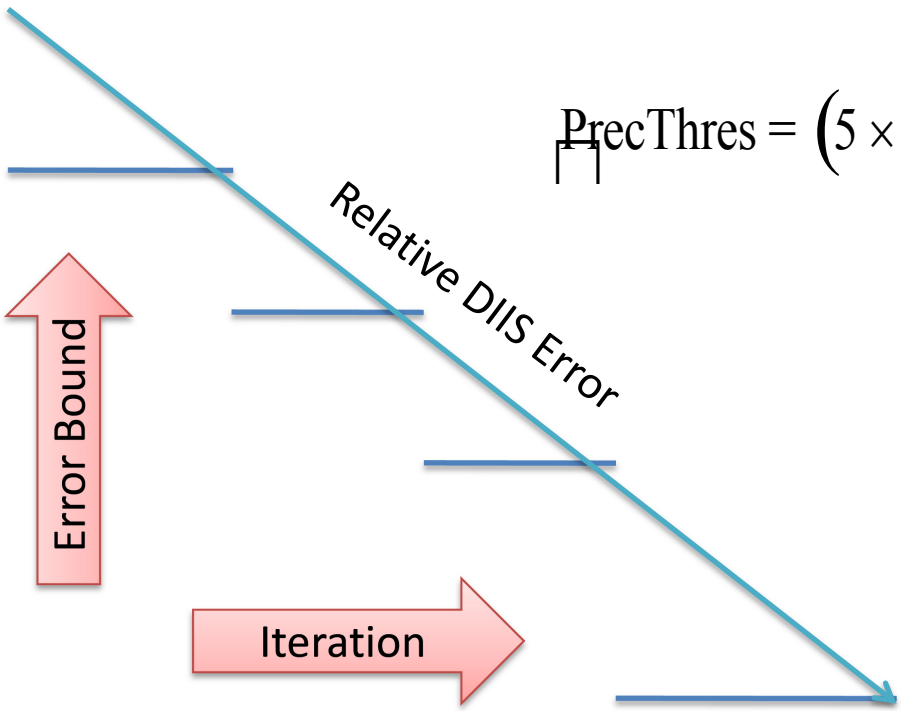
$$\hat{F}(P_i) = \hat{F}(P_{i-1}) + \hat{F}(P_i - P_{i-1})$$

$$F_i = \hat{F}(P_i - P_{i-1})$$

$$F_n = \tilde{F}_0 + \cdots + \tilde{F}_n$$



Dynamic Algorithm



$$PrecThres = \left(5 \times 10^{5} \cdot Error\right)^{\frac{1}{0.7}}$$

Weight Gradients

$$\nabla_g \omega_{ai} = \frac{\nabla_g P_a(r_{ai})}{\sum_b P_b(r_{ai})} - P_a(r_{ai}) \frac{\sum_b \nabla_g P_b(r_{ai})}{\left(\sum_b P_b(r_{ai})\right)^2}$$

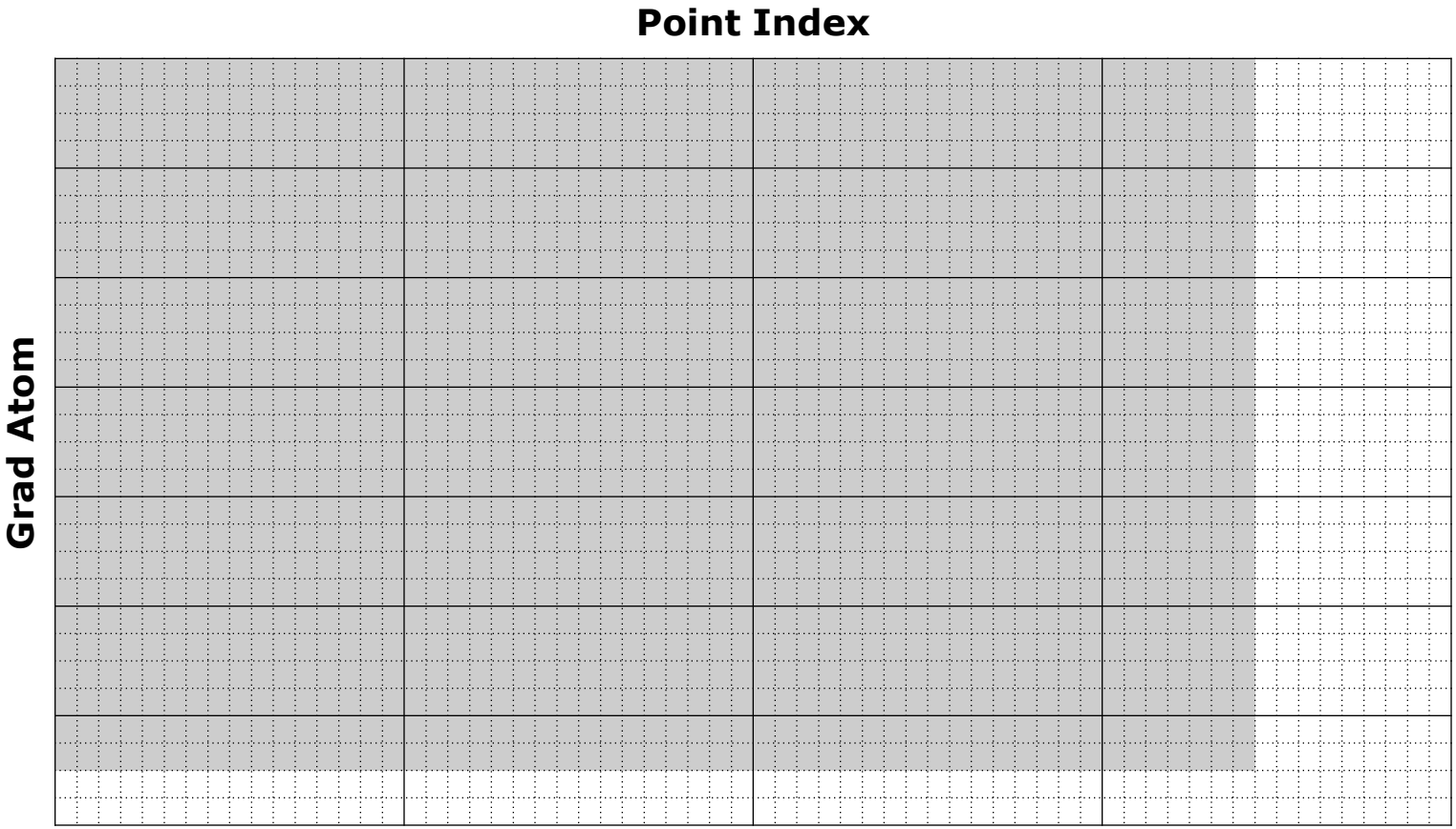
$$\nabla_g P_g = P_g \sum_{a \neq g} t(\mu_{ga}) \nabla_g \mu_{ga}$$

$$\nabla_g P_a = -P_a t(\mu_{ag}) \nabla_g \mu_{ga}$$

Kernel Structure

- Cache Kernel generates P_a values for each point that is about to run.
- Gradient Kernel generates x, y and z gradients for all atoms except its own Lebedev center and accumulates these into global memory.
- Translational Invariance Kernel accumulates the negative sum of all gradients except its Lebedev centers into the Lebedev center gradient.
- Iterate until all points have been processed.
- Summation Kernel accumulates all of the gradients into a final sum for each atom.

Thread Layout



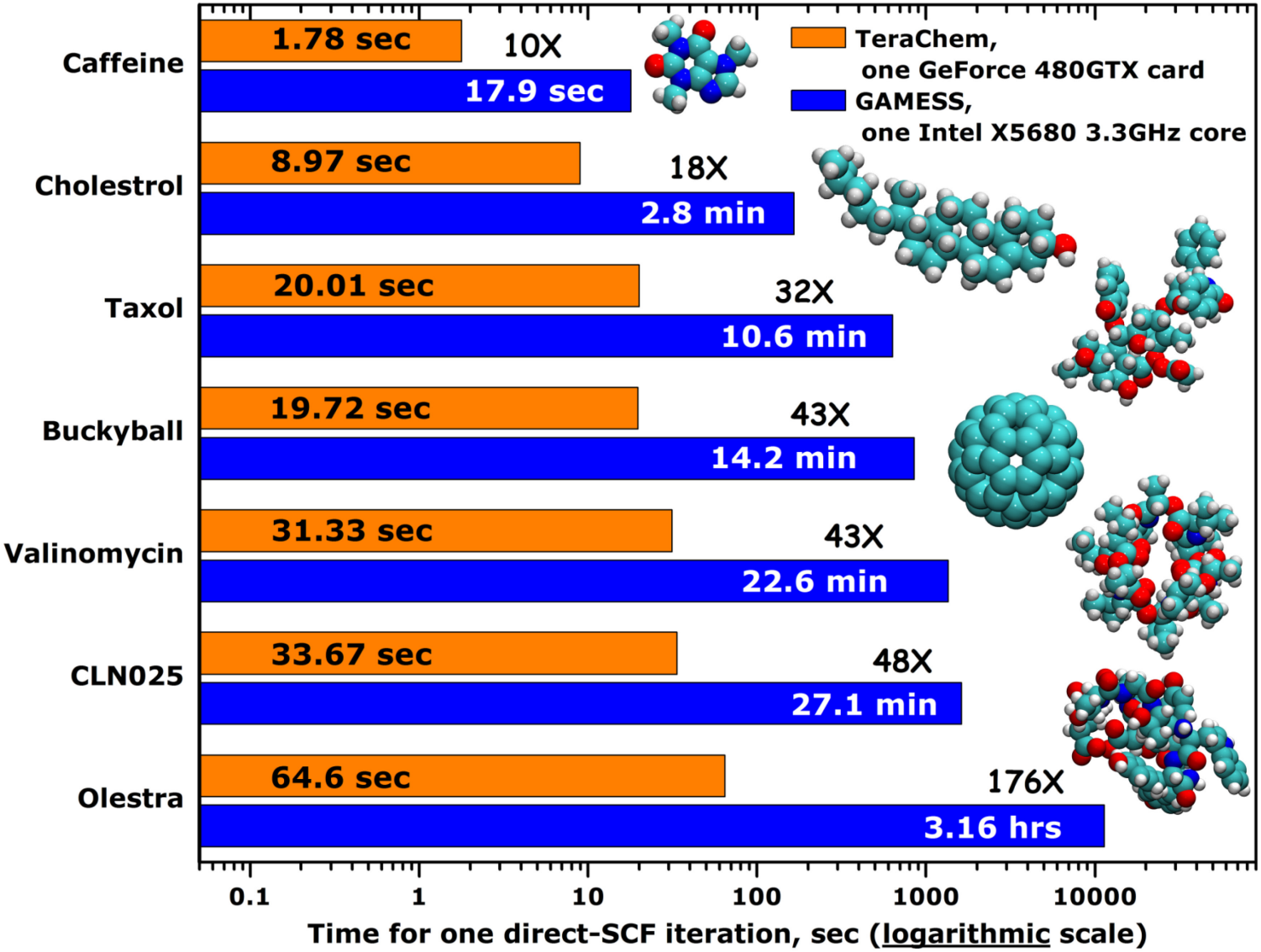
- **Point Index = threadIdx.x + blockIdx.x*blockDim.x**
- **Grad Atom = threadIdx.y + blockIdx.y*blockDim.y**

What about other QC Codes?

CIS/6-31G	Bfns	TeraChem	GAMESS	NWChem
S1 Dendrimer	322	1.1	10	28
B3PPQ	700	5.4	23	146
S2 Dendrimer	958	2.6	42	360
PYP + 487 H_2O	6519	122	56260	?

Times in sec per iteration
All calculations on 8 GPUs or 8 CPU cores (Intel Xeon X5550 3GHz)

Higher Angular Momentum



DFT(BLYP)/
6-31G*

Higher angular momentum means denser kernels
Increased register pressure, but still can be done efficiently