

# GPU Accelerated Visualization and Analysis in VMD

John Stone

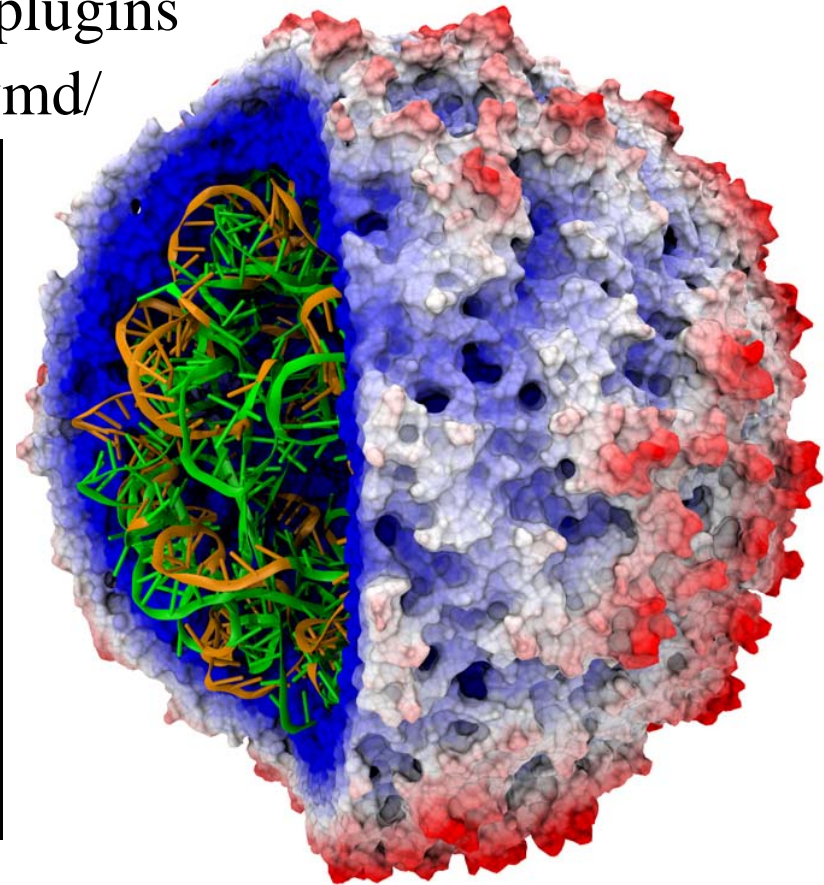
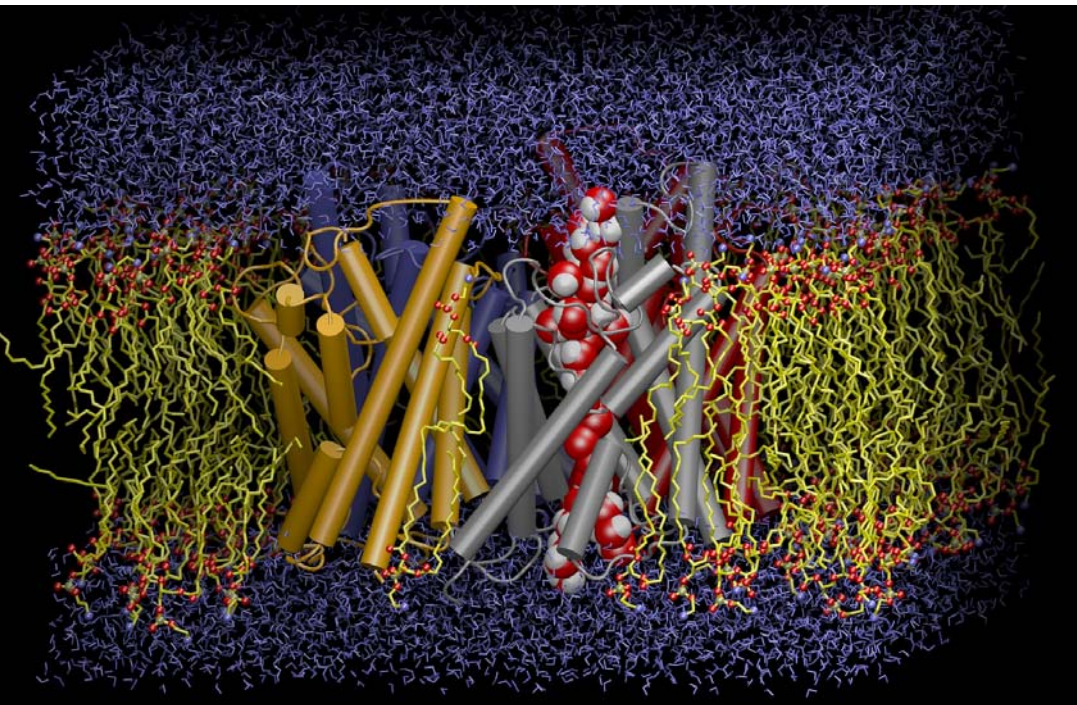
Theoretical and Computational Biophysics Group  
Beckman Institute for Advanced Science and Technology  
University of Illinois at Urbana-Champaign

**<http://www.ks.uiuc.edu/Research/vmd/>**

Center for Molecular Modeling  
University of Pennsylvania, June 9, 2009

# VMD – “Visual Molecular Dynamics”

- Visualization and analysis of molecular dynamics simulations, sequence data, volumetric data, quantum chemistry simulations, particle systems, ...
- User extensible with scripting and plugins
- <http://www.ks.uiuc.edu/Research/vmd/>



# Range of VMD Usage Scenarios

- Users run VMD on a diverse range of hardware: laptops, desktops, clusters, and supercomputers
- Typically used as a desktop application, for interactive 3D molecular graphics and analysis
- Can also be run in pure text mode for numerically intensive analysis tasks, batch mode movie rendering, etc...
- GPU acceleration provides an opportunity to make some **slow, or batch** calculations capable of being run **interactively, or on-demand...**

# Need for Multi-GPU Acceleration in VMD

- Ongoing increases in supercomputing resources at NSF centers such as NCSA enable increased simulation complexity, fidelity, and longer time scales...
- Drives need for more visualization and analysis capability at the desktop and on clusters running batch analysis jobs
- Desktop use is the most compute-resource-limited scenario, where **GPUs can make a big impact...**

# Programmable Graphics Hardware

Groundbreaking research systems:

AT&T Pixel Machine (1989):

82 x DSP32 processors

UNC PixelFlow (1992-98):

64 x (PA-8000 +

8,192 bit-serial SIMD)

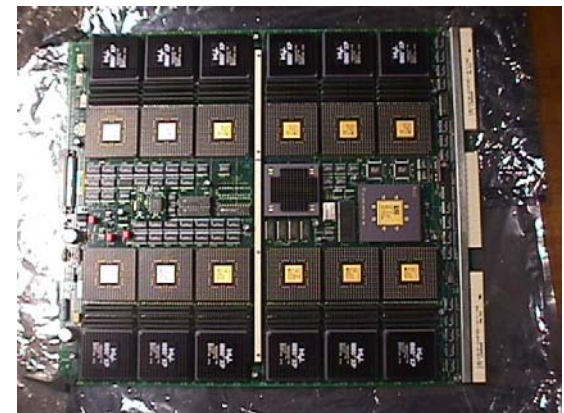
SGI RealityEngine (1990s):

Up to 12 i860-XP processors perform  
vertex operations (*u*code), fixed-  
func. fragment hardware

**All mainstream GPUs now incorporate  
fully programmable processors**

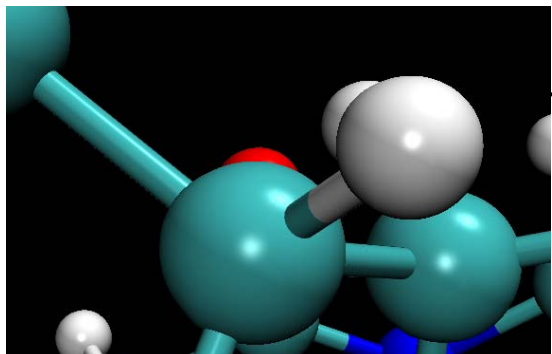


UNC PixelFlow Rack



SGI Reality Engine i860  
Vertex Processors

# GLSL Sphere Fragment Shader



- Written in OpenGL Shading Language
- High-level C-like language with vector types and operations
- Compiled dynamically by the graphics driver at *runtime*
- Compiled machine code executes on GPU

```
//  
// VMD Sphere Fragment Shader (not for normal geometry)  
//  
void main(void) {  
    vec3 raydir = normalize(V);  
    vec3 spheredir = spherepos - rayorigin;  
  
    // Perform ray-sphere intersection tests based on the code in Tachyon  
    float b = dot(raydir, spheredir);  
    float temp = dot(spheredir, spheredir);  
    float disc = b*b + sphereradsq - temp;  
  
    // only calculate the nearest intersection, for speed  
    if (disc <= 0.0)  
        discard; // ray missed sphere entirely, discard fragment  
  
    // calculate closest intersection  
    float tnear = b - sqrt(disc);  
  
    if (tnear < 0.0)  
        discard;  
  
    // calculate hit point and resulting surface normal  
    vec3 pnt = rayorigin + tnear * raydir;  
    vec3 N = normalize(pnt - spherepos);  
  
    // Output the ray-sphere intersection point as the fragment depth  
    // rather than the depth of the bounding box polygons.  
    // The eye coordinate Z value must be transformed to normalized device  
    // coordinates before being assigned as the final fragment depth.  
    if (vmdprojectionmode == 1) {  
        // perspective projection = 0.5 + (hfpn + (f * n / pnt.z)) / diff  
        gl_FragDepth = 0.5 + (vmdprojparms[2] + (vmdprojparms[1] * vmdprojparms[  
3]);  
    } else {  
        // orthographic projection = 0.5 + (-hfpn - pnt.z) / diff  
        gl_FragDepth = 0.5 + (-vmdprojparms[2] - pnt.z) / vmdprojparms[3];  
    }  
  
#ifdef TEXTURE  
    // perform texturing operations for volumetric data  
    // The only texturing mode that applies to the sphere shader
```

# GPU Computing

- Commodity devices, omnipresent in modern computers (over a **million** sold per **week**)
- Massively parallel hardware, hundreds of processing units, throughput oriented architecture
- Standard integer and floating point types supported
- Programming tools allow software to be written in dialects of familiar C/C++ and integrated into legacy software
- GPU algorithms are often multicore friendly due to attention paid to data locality and data-parallel work decomposition

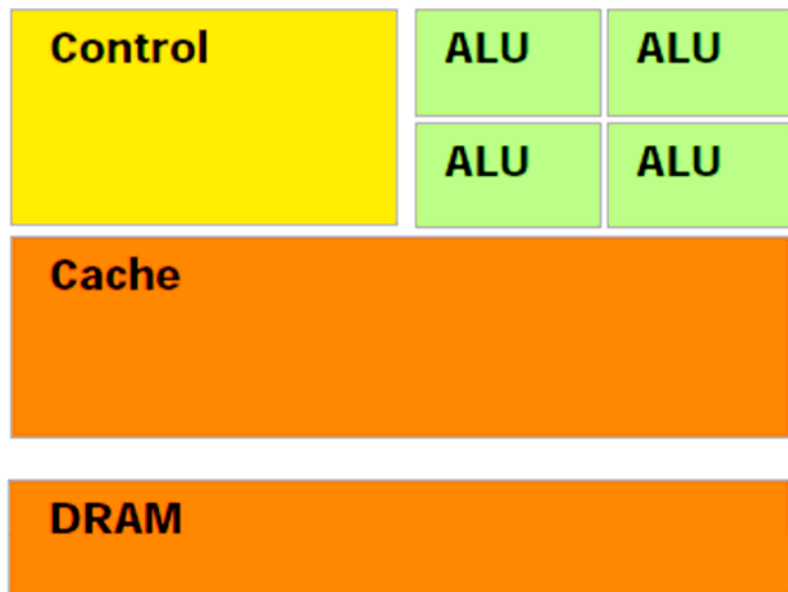
# What Speedups Can GPUs Achieve?

- Single-GPU speedups of **10x** to **30x** vs. one CPU core are common
- Best speedups can reach **100x** or more, attained on codes dominated by floating point arithmetic, especially native GPU machine instructions, e.g. **expf()**, **rsqrtf()**, ...
- Amdahl's Law can prevent legacy codes from achieving peak speedups with shallow GPU acceleration efforts

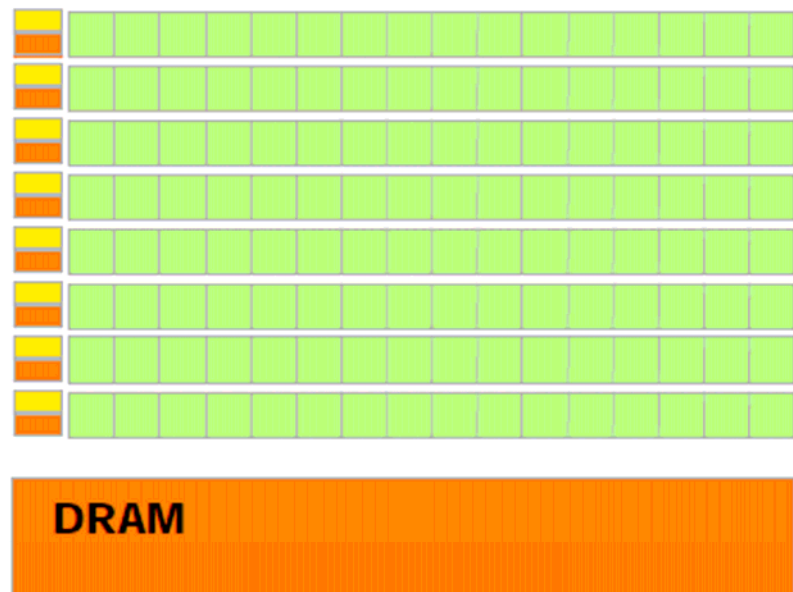


# Comparison of CPU and GPU Hardware Architecture

**CPU:** Cache heavy,  
focused on individual  
thread performance

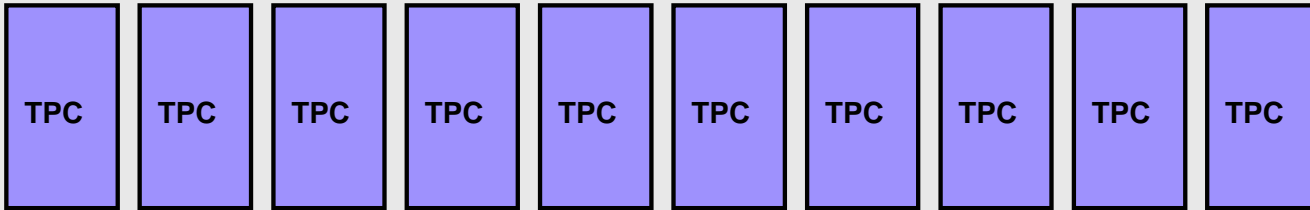


**GPU:** ALU heavy,  
massively parallel,  
throughput oriented



# NVIDIA GT200

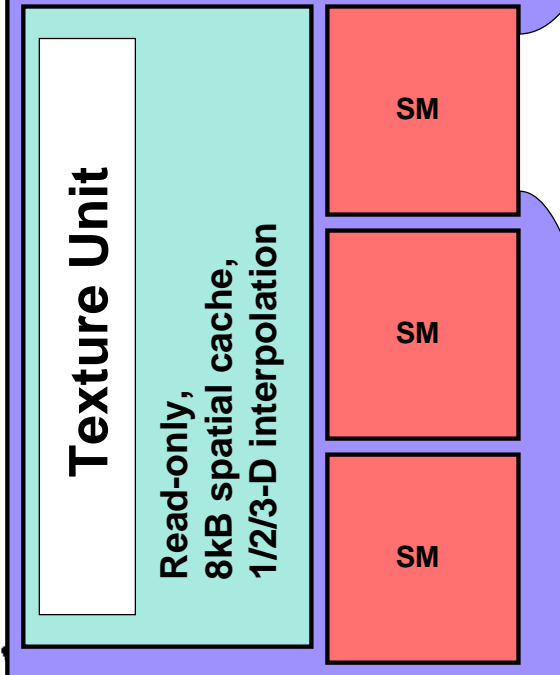
## Streaming Processor Array



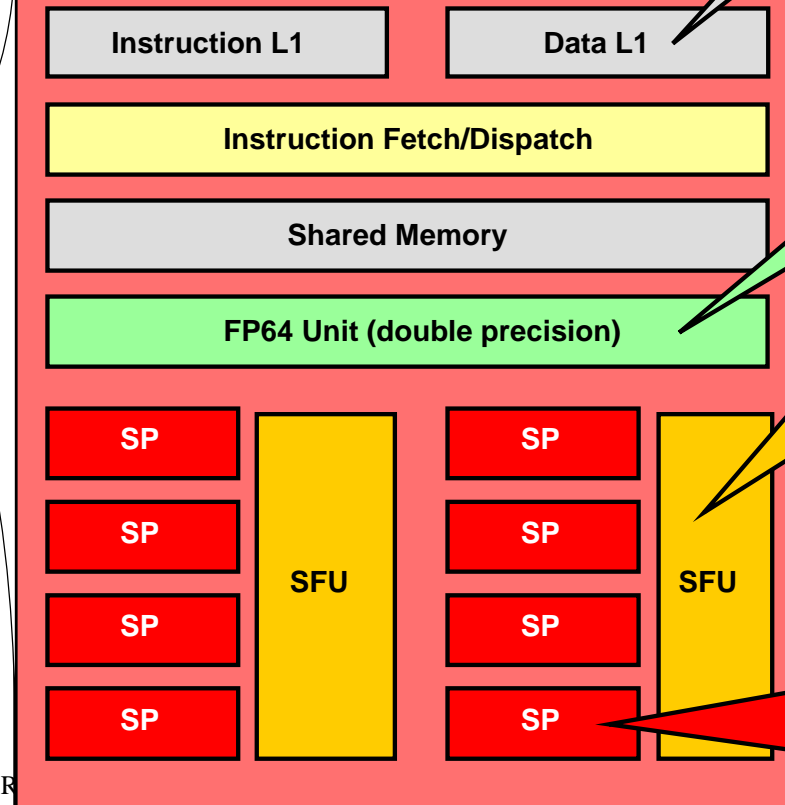
Constant Cache

64kB, read-only

## Texture Processor Cluster



## Streaming Multiprocessor



FP64 Unit

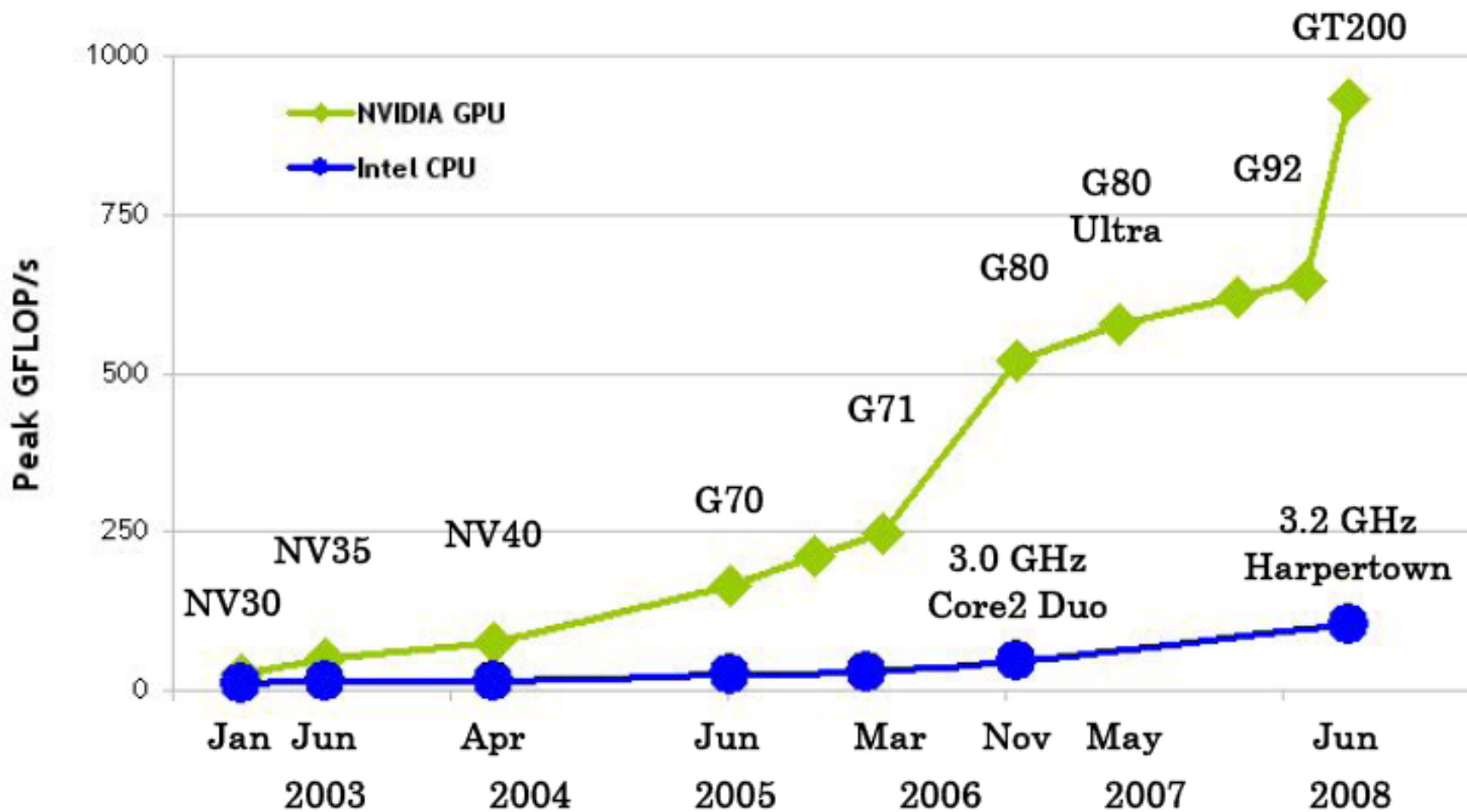
Special  
Function Unit

SIN, EXP,  
RSQRT, Etc...

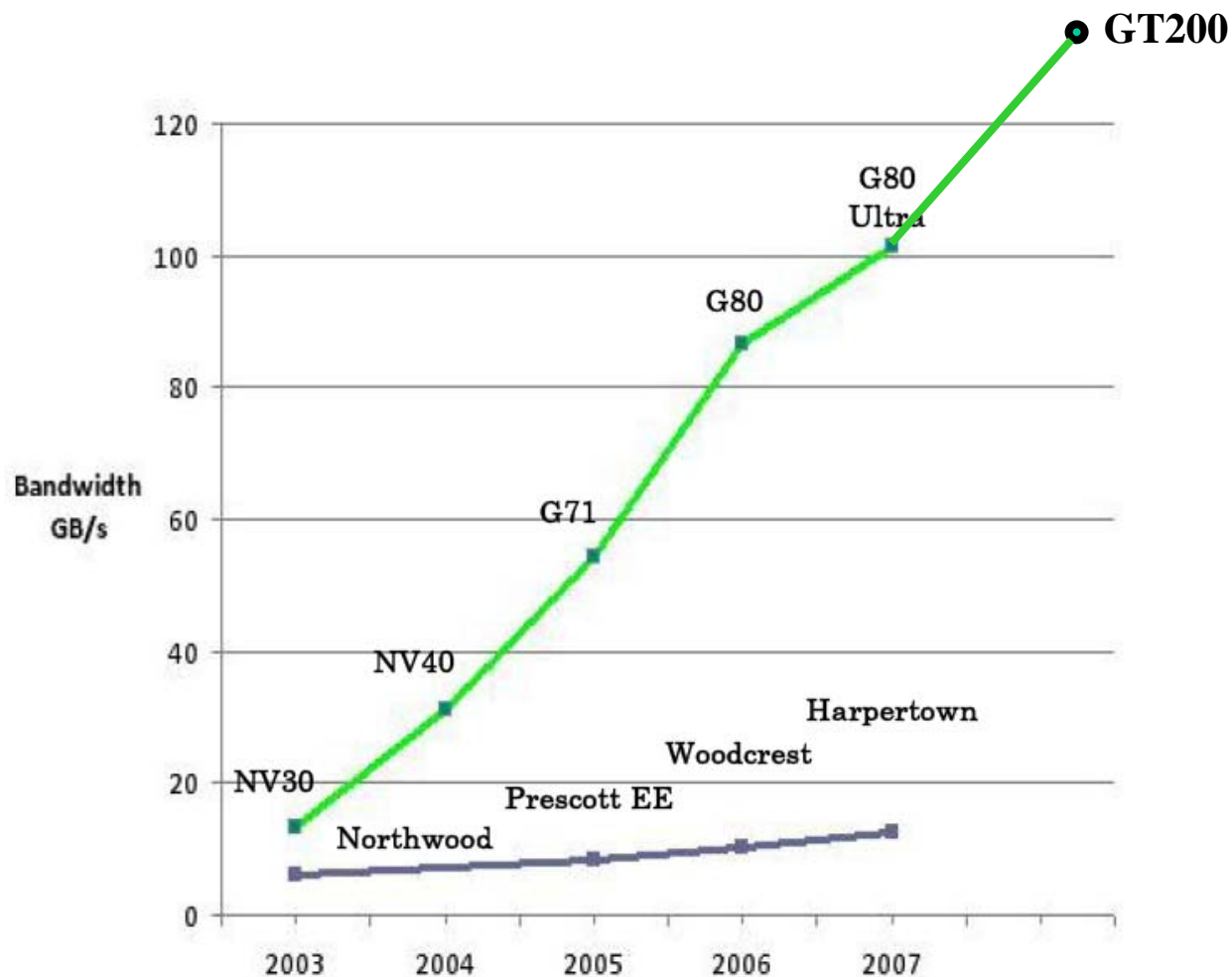
Streaming  
Processor

ADD, SUB  
MAD, Etc...

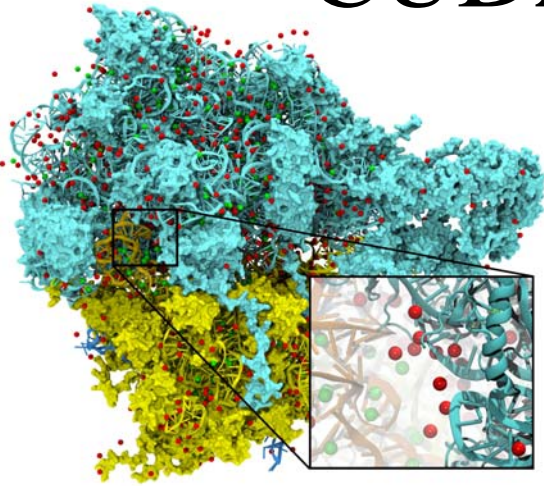
# GPU Peak Single-Precision Performance: Exponential Trend



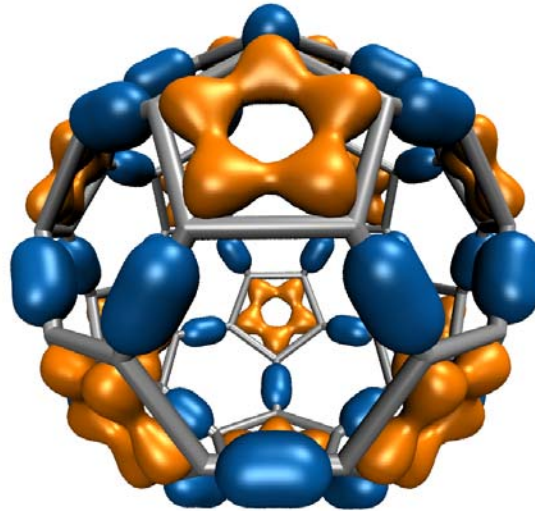
# GPU Peak Memory Bandwidth: Linear Trend



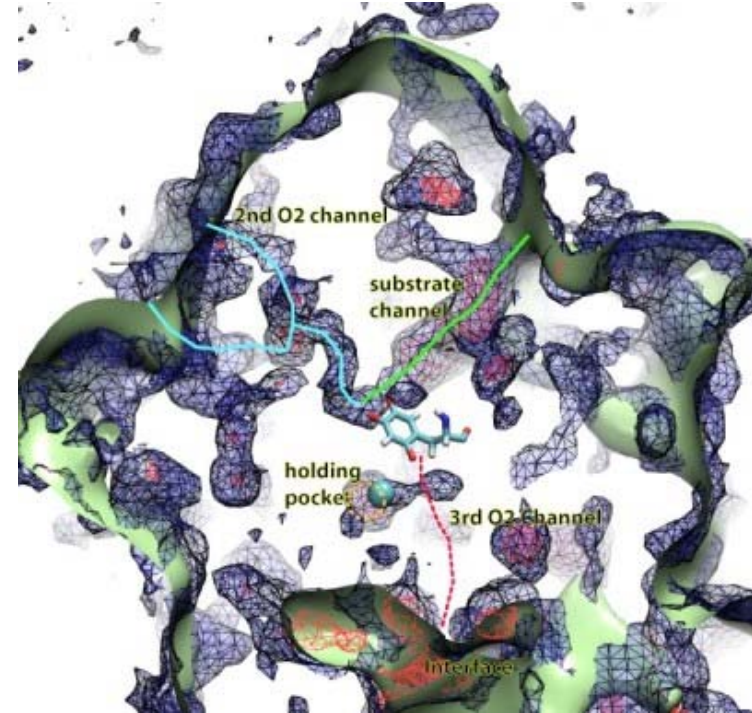
# CUDA Acceleration in VMD



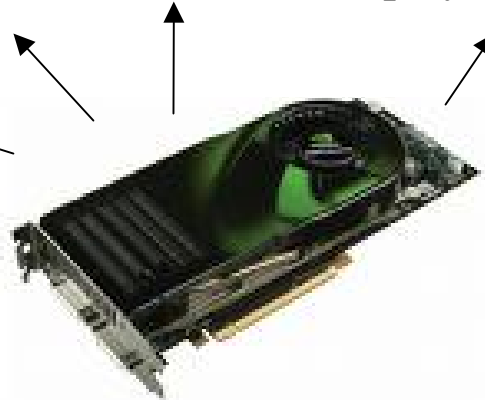
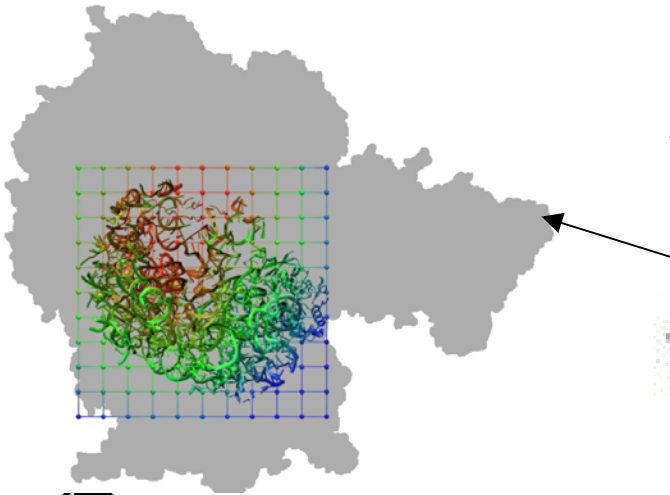
Electrostatic field  
calculation, ion placement



Molecular orbital  
calculation and display



Imaging of gas migration  
pathways in proteins with  
implicit ligand sampling

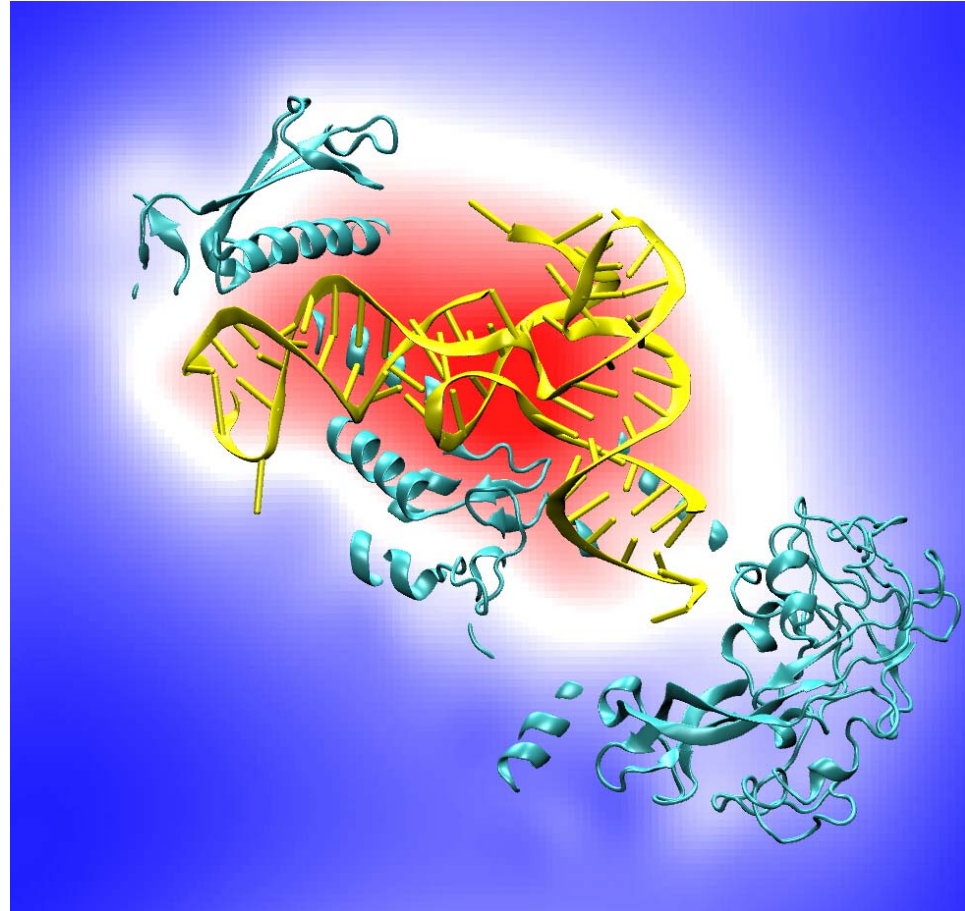


# Electrostatic Potential Maps

- Electrostatic potentials evaluated on 3-D lattice:

$$V_i = \sum_j \frac{q_j}{4\pi\epsilon_0|\mathbf{r}_j - \mathbf{r}_i|}$$

- Applications include:
  - Ion placement for structure building
  - Time-averaged potentials for simulation
  - Visualization and analysis

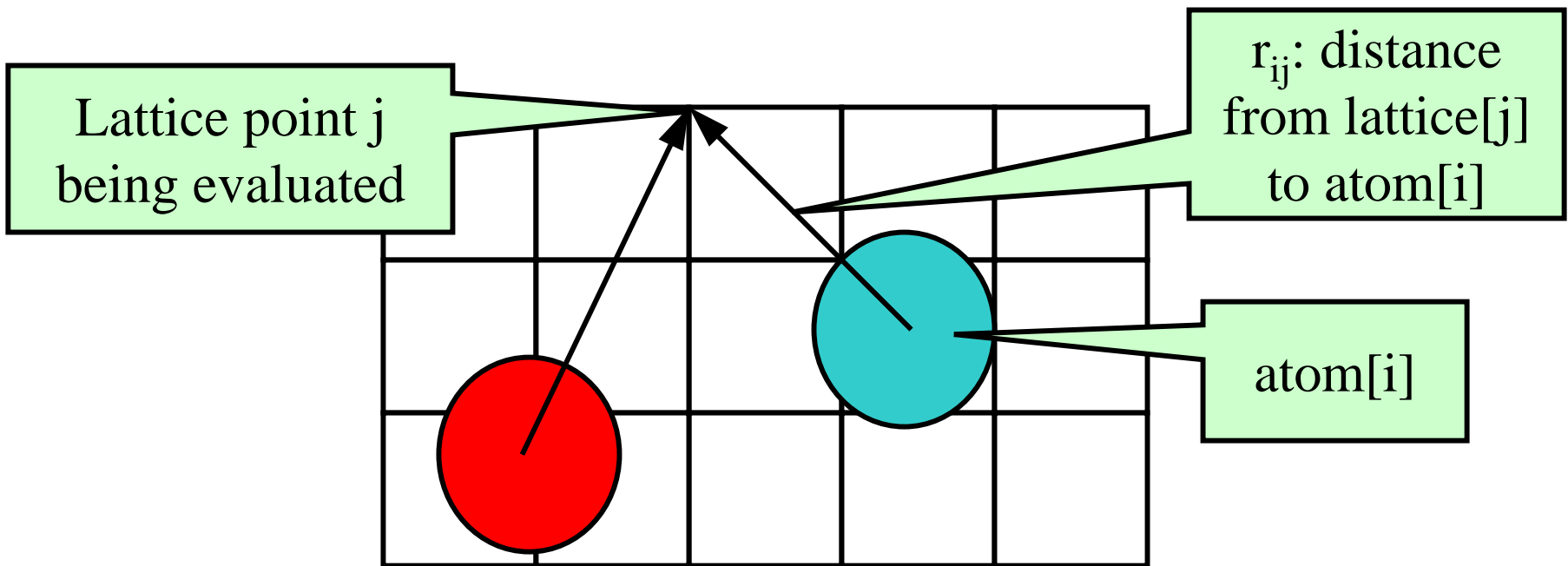


Isoleucine tRNA synthetase

# Direct Coulomb Summation

- Each lattice point accumulates electrostatic potential contribution from all atoms:

$$\text{potential}[j] += \text{charge}[i] / r_{ij}$$



# Direct Coulomb Summation on the GPU

- GPU outruns a CPU core by 44x
- Work is decomposed into tens of thousands of independent threads, multiplexed onto hundreds of GPU processing units
- Single-precision FP arithmetic is adequate for intended application
- Numerical accuracy can be improved by compensated summation, spatially ordered summation groupings, or accumulation of potential in double-precision
- Starting point for more sophisticated linear-time algorithms like multilevel summation



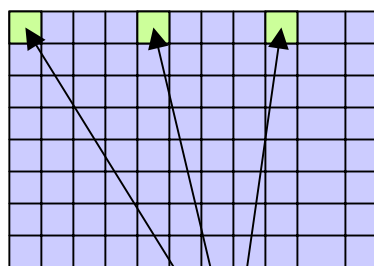
# DCS CUDA Block/Grid Decomposition

(unrolled, coalesced)

Grid of thread blocks:

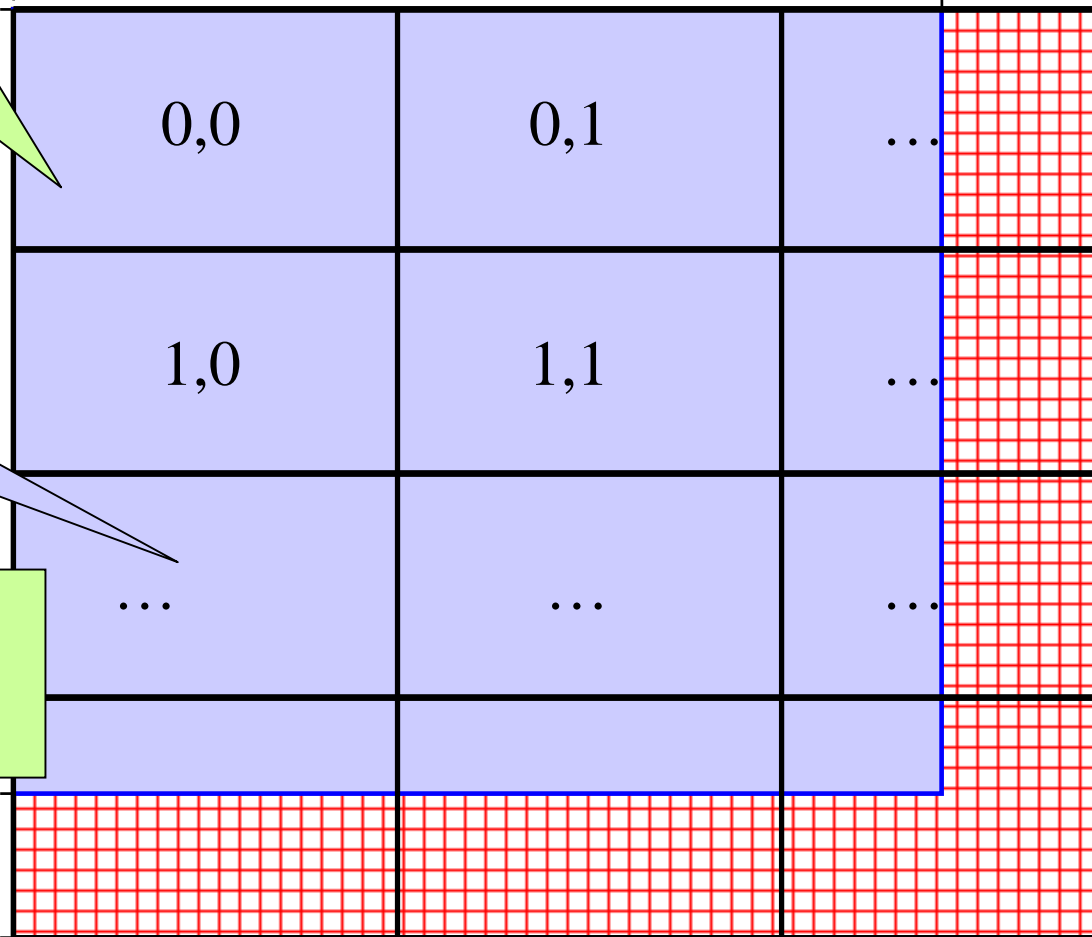
Unrolling increases computational tile size

Thread blocks:  
64-256 threads

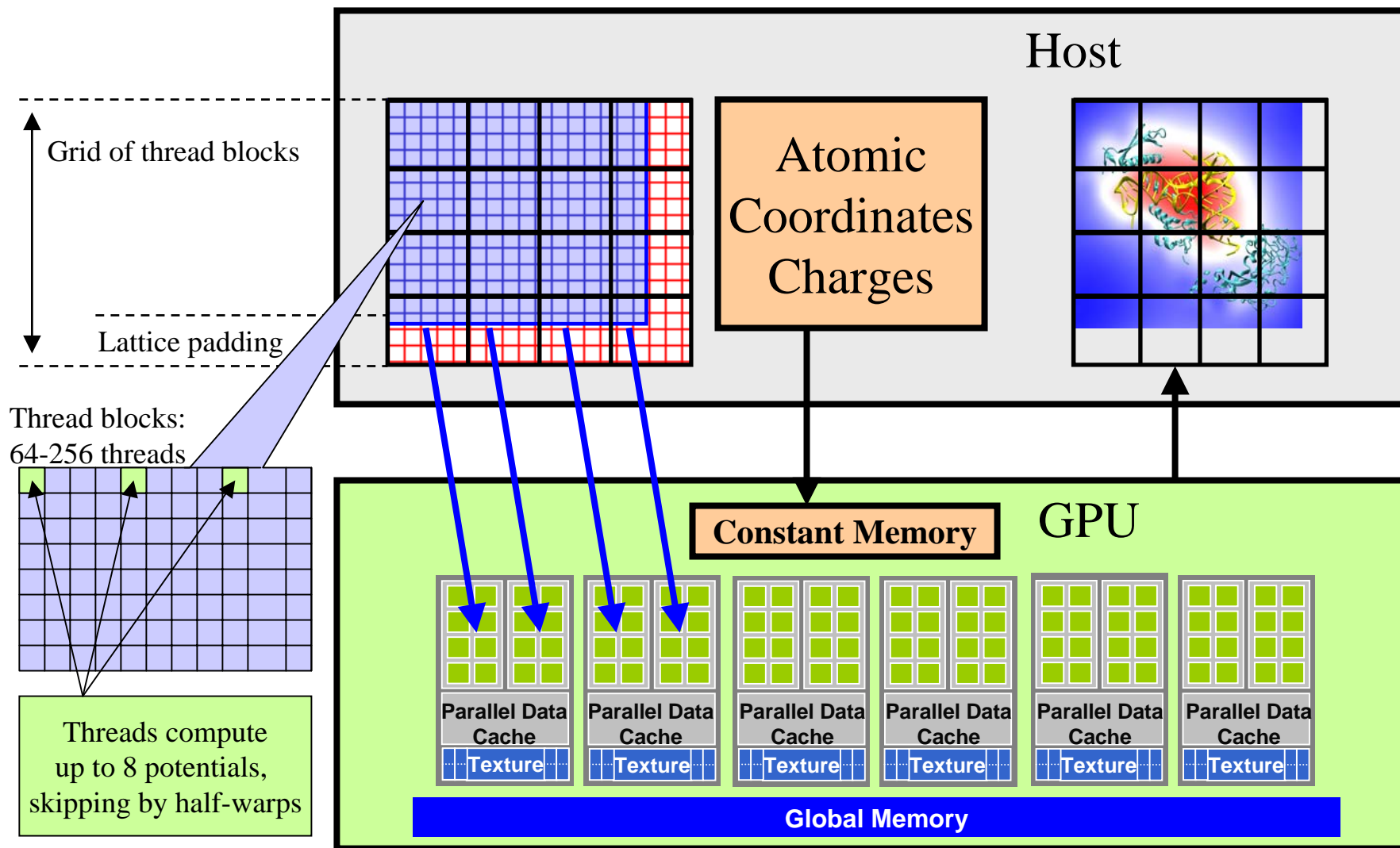


Threads compute up to 8 potentials, skipping by half-warps

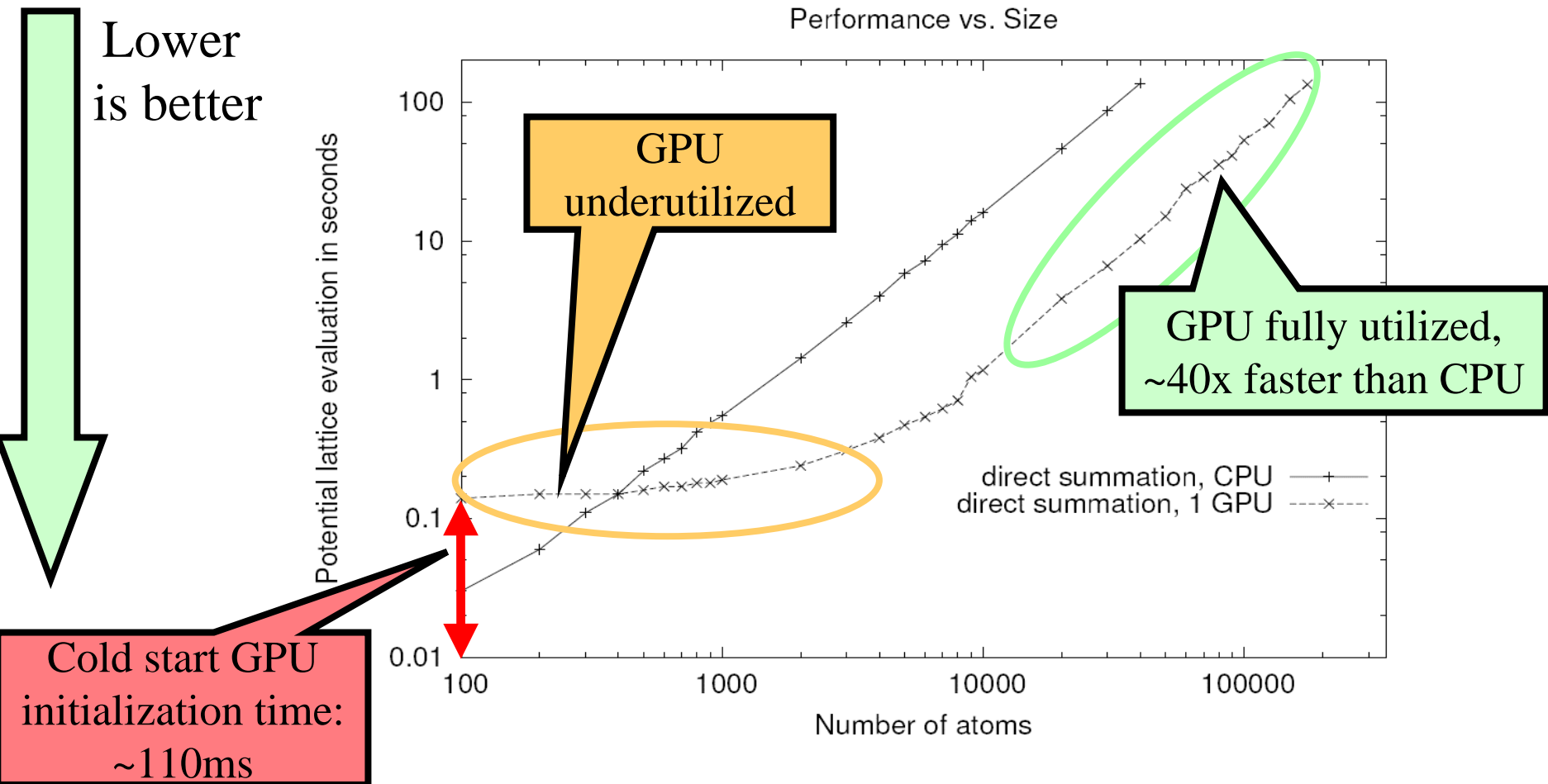
Padding waste



# Direct Coulomb Summation on the GPU



# Direct Coulomb Summation Runtime



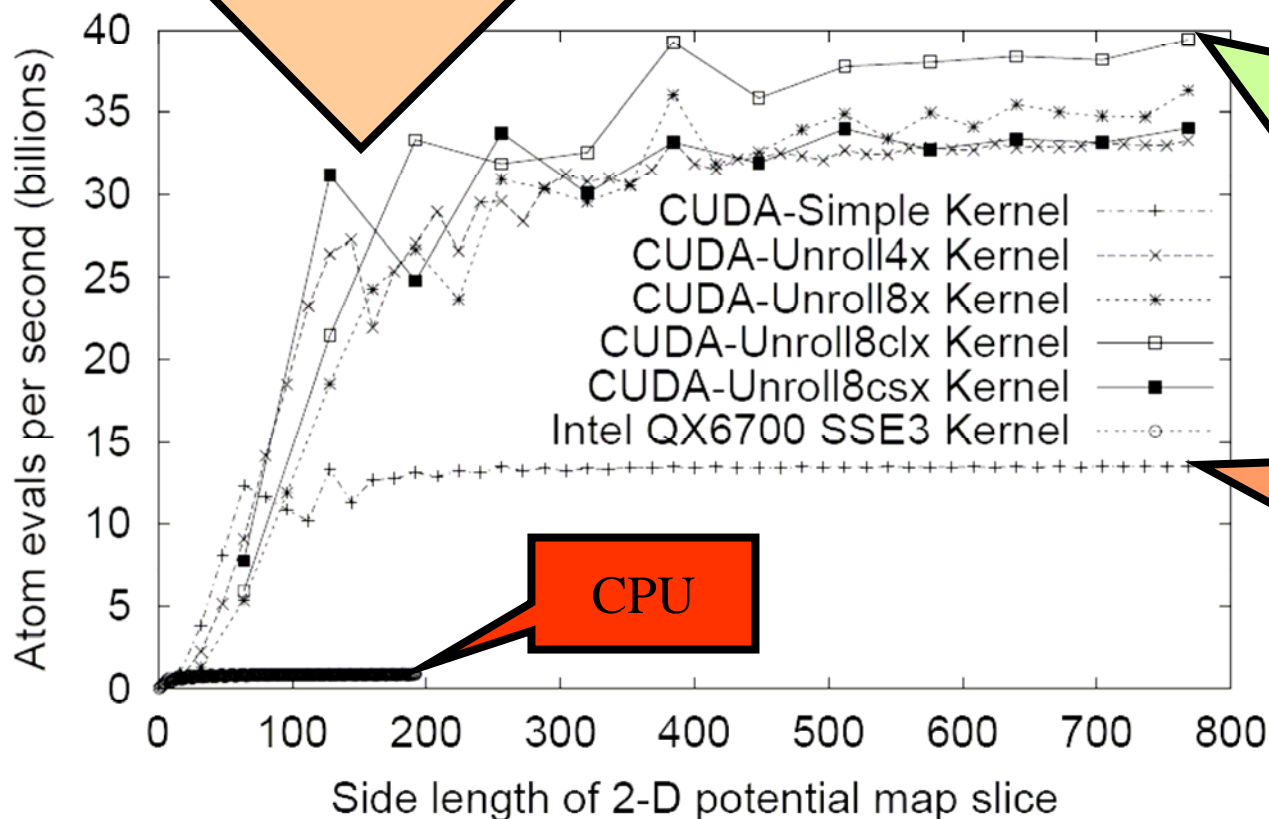
Accelerating molecular modeling applications with graphics processors.

J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, K. Schulten.

*J. Comp. Chem.*, 28:2618-2640, 2007.

# Direct Coulomb Summation Performance

Number of thread blocks modulo number of SMs results in significant performance variation for small workloads

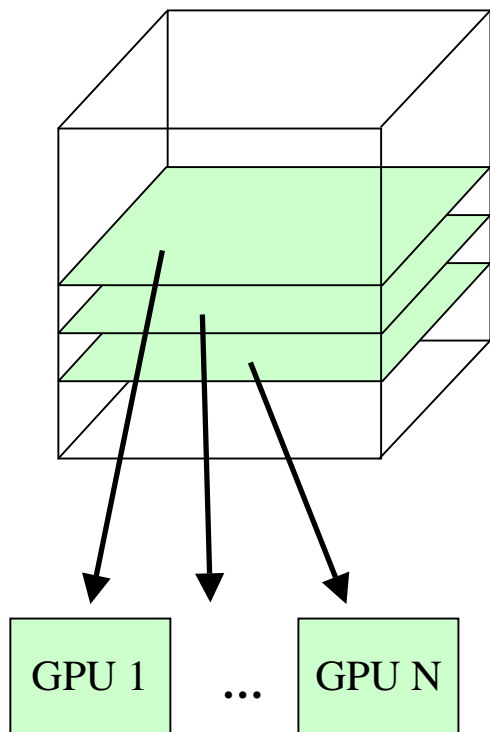


CUDA-Unroll8clx: fastest GPU kernel, 44x faster than CPU, 291 GFLOPS on GeForce 8800GTX

CUDA-Simple: 14.8x faster, 33% of fastest GPU kernel

GPU computing. J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips. *Proceedings of the IEEE*, 96:879-899, 2008.

# Multi-GPU Direct Coulomb Summation



NCSA GPU Cluster

<http://www.ncsa.uiuc.edu/Projects/GPUcluster/>

	Evals/sec	TFLOPS	Speedup*
4-GPU (2 Quadroplex) Opteron node at NCSA	157 billion	1.16	176
4-GPU GTX 280 (GT200)	241 billion	1.78	271

\*Speedups relative to Intel QX6700 CPU core w/ SSE

# Infinite vs. Cutoff Potentials

- Infinite range potential:
  - All atoms contribute to all lattice points
  - Summation algorithm has quadratic complexity
- Cutoff (range-limited) potential:
  - Atoms contribute within cutoff distance to lattice points
  - Summation algorithm has linear time complexity
  - Has many applications in molecular modeling:
    - Replace electrostatic potential with shifted form
    - Short-range part for fast methods of approximating full electrostatics
    - Used for fast decaying interactions (e.g. Lennard-Jones, Buckingham)

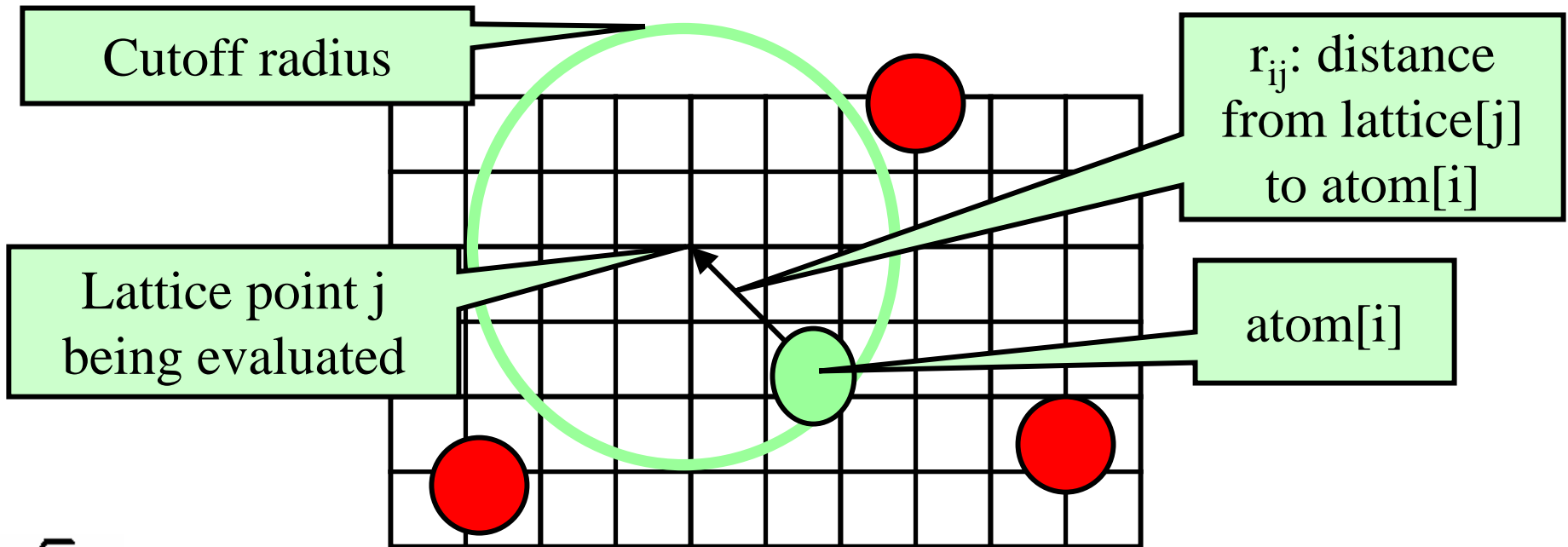
# Cutoff Summation

- Each lattice point accumulates electrostatic potential contribution from atoms within cutoff distance:

if ( $r_{ij} < \text{cutoff}$ )

potential[j] += (charge[i] /  $r_{ij}$ ) \* s( $r_{ij}$ )

- Smoothing function s(r) is algorithm dependent



# Cutoff Summation on the GPU

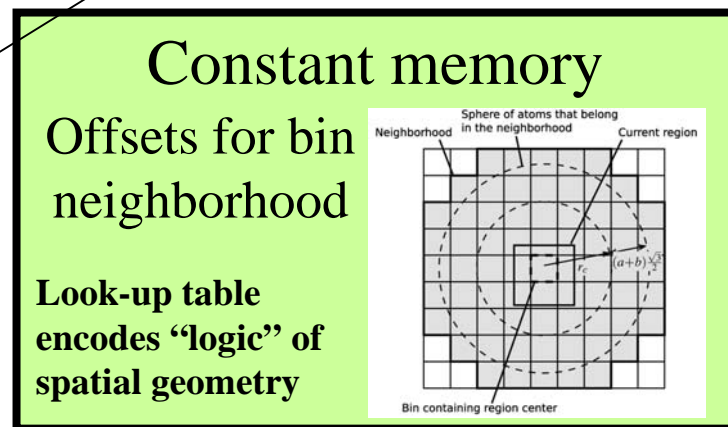
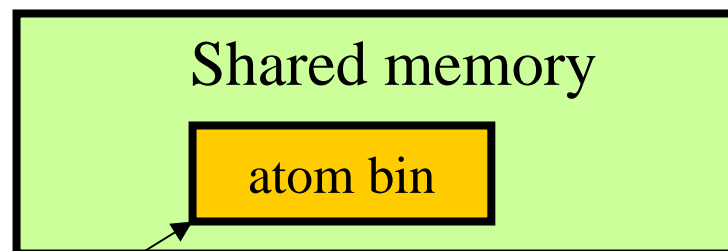
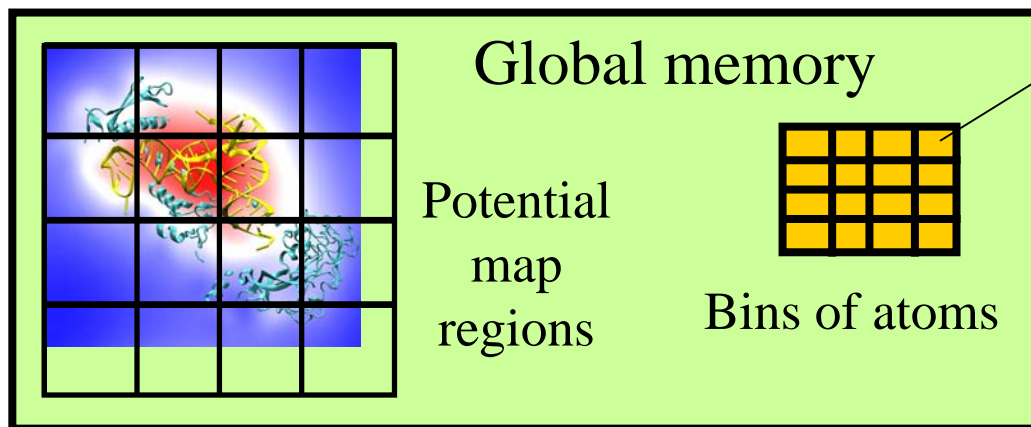
Atoms are spatially hashed into fixed-size bins

CPU handles overflowed bins (GPU kernel can be very aggressive)

GPU thread block calculates corresponding region of potential map,

Bin/region neighbor checks costly; solved with universal table look-up

Each thread block cooperatively loads atom bins from surrounding neighborhood into shared memory for evaluation

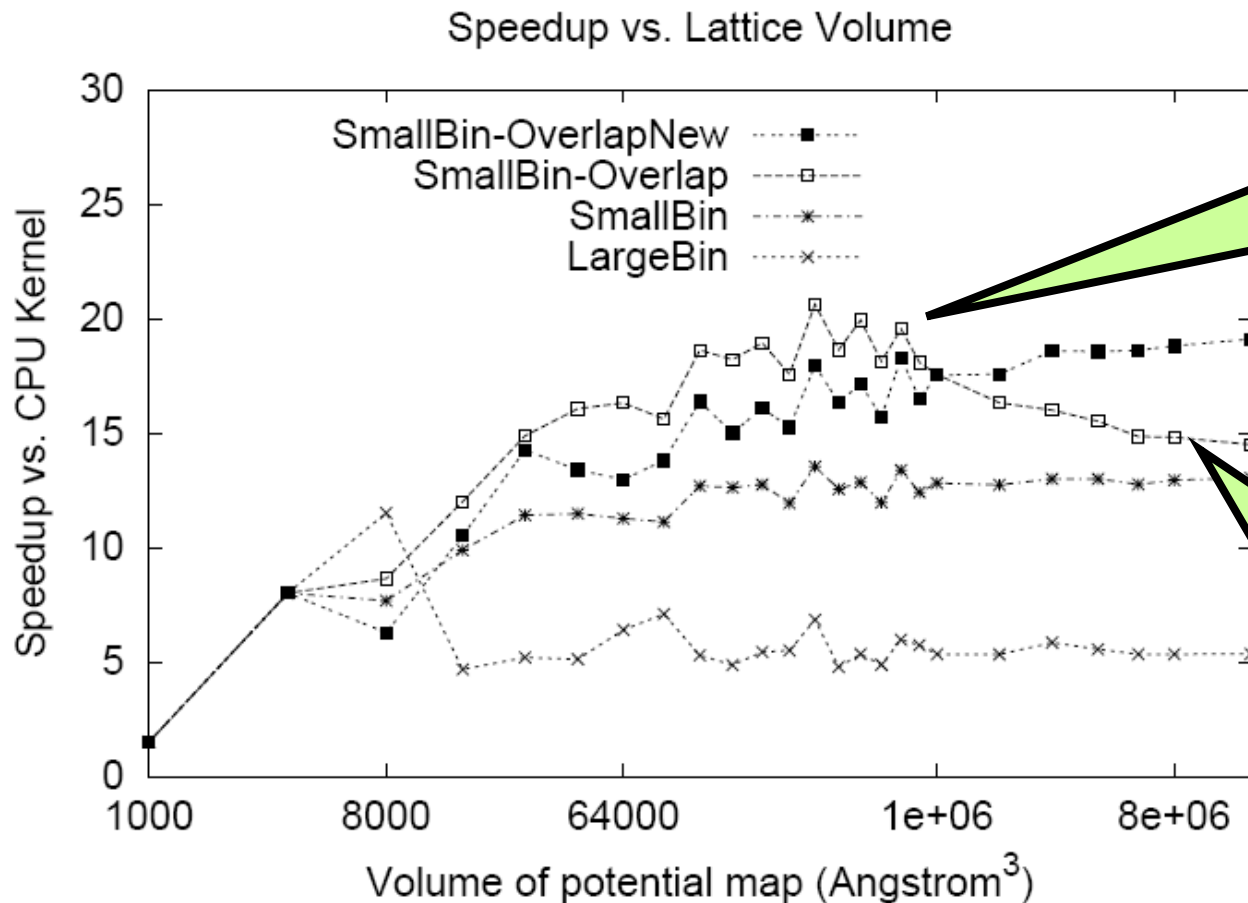




# Using the CPU to Improve GPU Performance

- GPU performs best when the work evenly divides into the number of threads/processing units
- Optimization strategy:
  - Use the CPU to “regularize” the GPU workload
  - Use fixed size bin data structures, with “empty” slots skipped or producing zeroed out results
  - Handle exceptional or irregular work units on the CPU while the GPU processes the bulk of the work
  - On average, the GPU is kept highly occupied, attaining a much higher fraction of peak performance

# Cutoff Summation Runtime



GPU cutoff with CPU overlap: 17x-21x faster than CPU core

If asynchronous stream blocks due to queue filling, performance will degrade from peak...

GPU acceleration of cutoff pair potentials for molecular modeling applications.  
C. Rodrigues, D. Hardy, J. Stone, K. Schulten, W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.

# Cutoff Summation Observations

- Use of CPU to handle overflowed bins is very effective, overlaps completely with GPU work
- Caveat: avoid overfilling the asynchronous stream queue with work, doing so can trigger blocking behavior (improved in current drivers)
- The use of compensated summation (all GPUs) or double-precision (GT200 only) for potential accumulation resulted in only a **~10%** performance penalty vs. pure single-precision arithmetic, while reducing the effects of floating point truncation

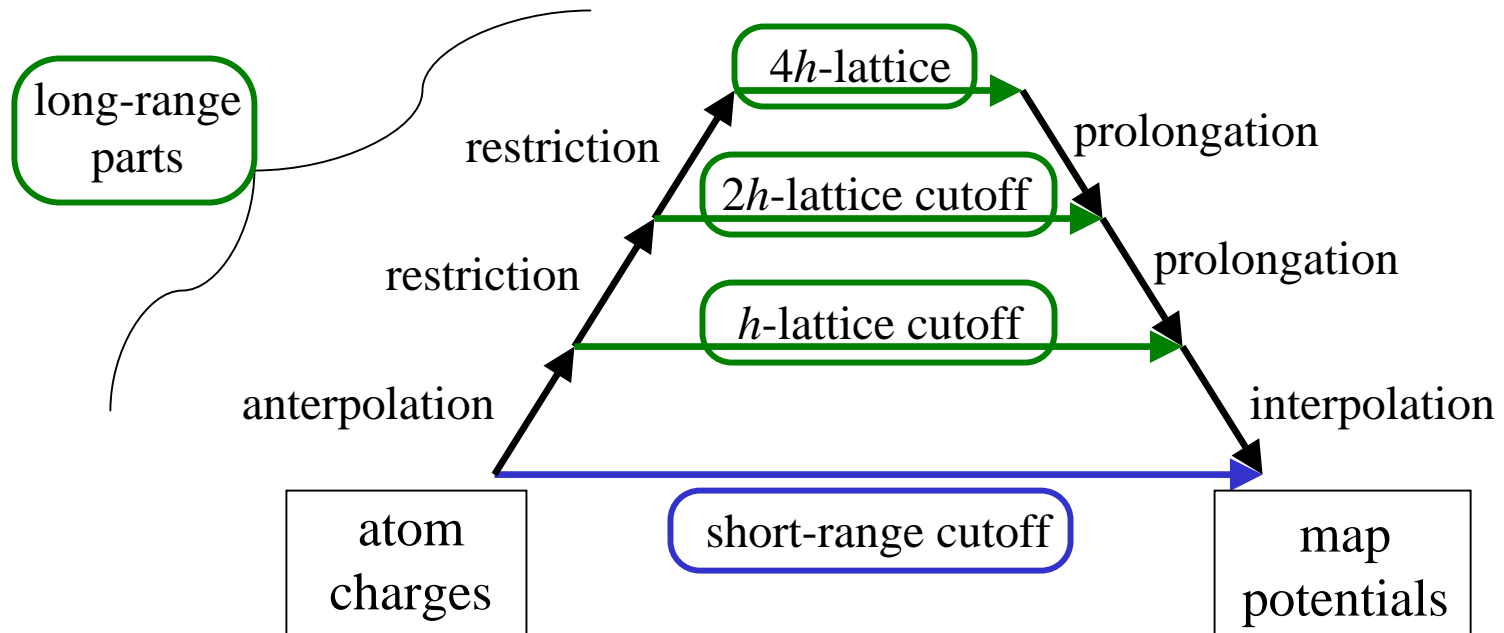
# Multilevel Summation

- Approximates full electrostatic potential
- Calculates sum of smoothed pairwise potentials interpolated from a hierarchy of lattices
- Advantages over PME and/or FMM:
  - Algorithm has **linear time complexity**
  - Permits non-periodic and periodic boundaries
  - Produces continuous forces for dynamics (advantage over FMM)
  - Avoids 3-D FFTs for better parallel scaling (advantage over PME)
  - Spatial separation allows use of multiple time steps
  - Can be extended to other pairwise interactions
- Skeel, Tezcan, Hardy, *J Comp Chem*, 2002 — Computing forces for molecular dynamics
- Hardy, Stone, Schulten, *J Paral Comp*, 2009 — GPU-acceleration of potential map calculation

# Multilevel Summation Calculation

$$\text{map potential} = \text{exact short-range interactions} + \text{interpolated long-range interactions}$$

## Computational Steps

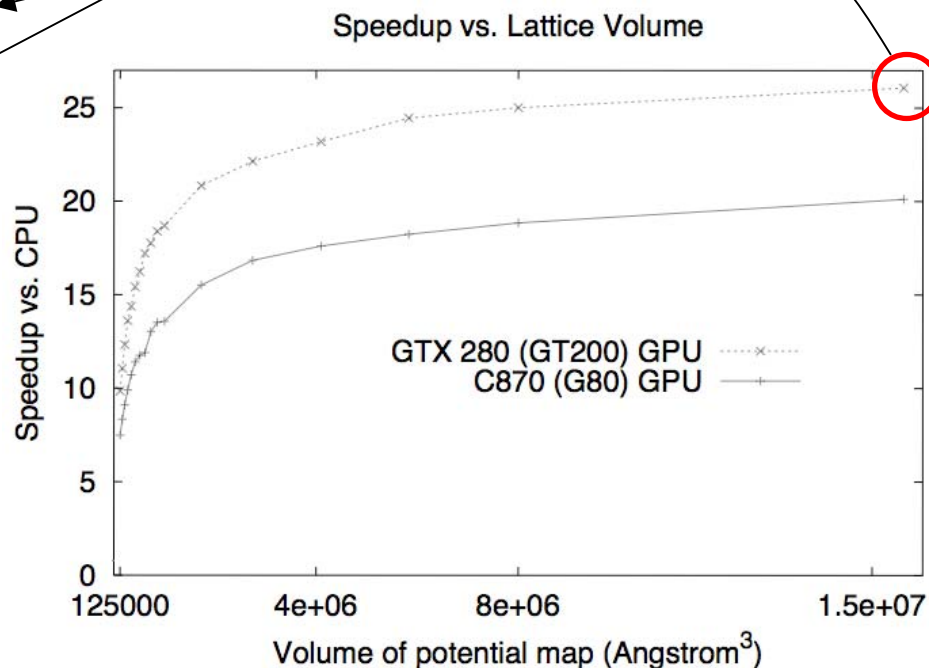


# Multilevel Summation on the GPU

Accelerate **short-range cutoff** and **lattice cutoff** parts

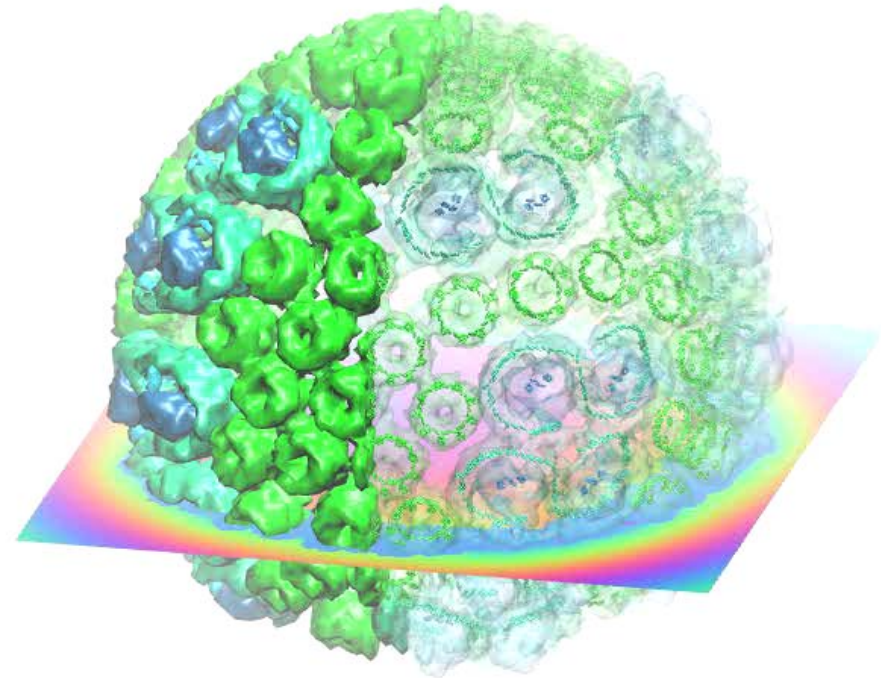
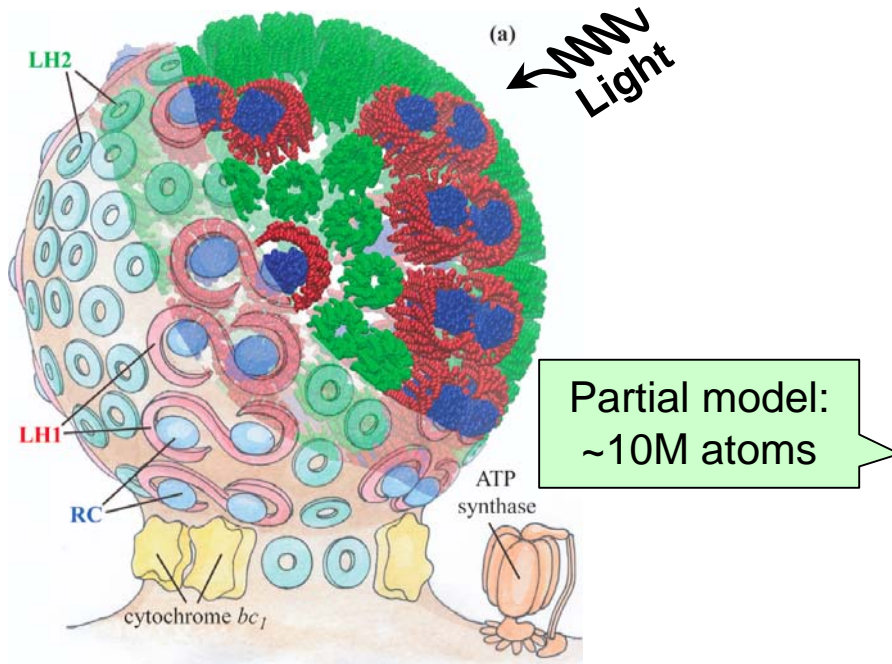
Performance profile for 0.5 Å map of potential for 1.5 M atoms.  
Hardware platform is Intel QX6700 CPU and NVIDIA GTX 280.

Computational steps	CPU (s)	w/ GPU (s)	Speedup
<b>Short-range cutoff</b>	480.07	14.87	32.3
Long-range anterpolation	0.18		
restriction	0.16		
<b>lattice cutoff</b>	49.47	1.36	36.4
prolongation	0.17		
interpolation	3.47		
Total	533.52	20.21	26.4



# Photobiology of Vision and Photosynthesis

## Investigations of the chromatophore, a photosynthetic organelle



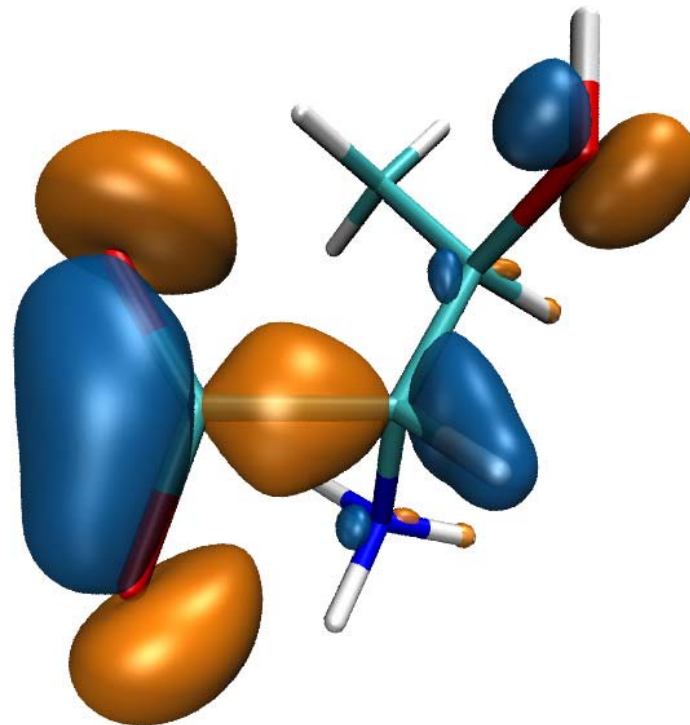
Electrostatics needed to build full structural model, place ions, study macroscopic properties

Electrostatic field of chromatophore model from multilevel summation method: computed with 3 GPUs (G80) in ~90 seconds, 46x faster than single CPU core

**Full chromatophore model will permit structural, chemical and kinetic investigations at a structural systems biology level**

# Molecular Orbitals

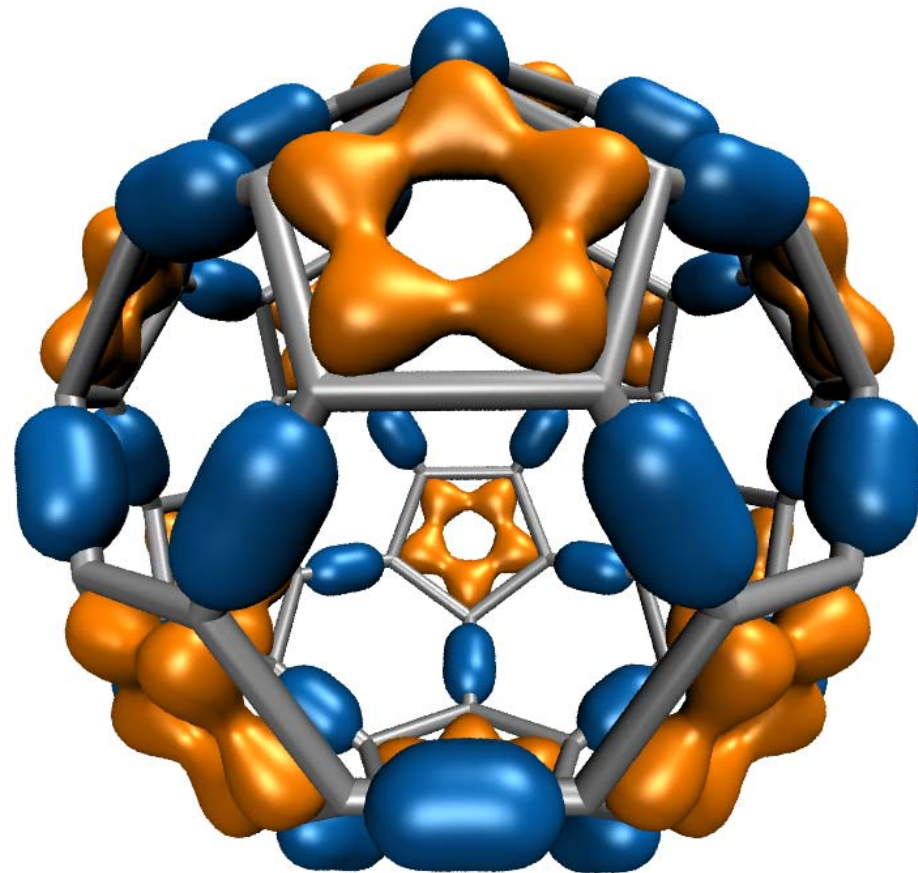
- Visualization of MOs aids in understanding the chemistry of molecular system
- MO spatial distribution is correlated with probability density for an electron(s)
- Algorithms for computing other interesting properties are similar, and can share code





# Computing Molecular Orbitals

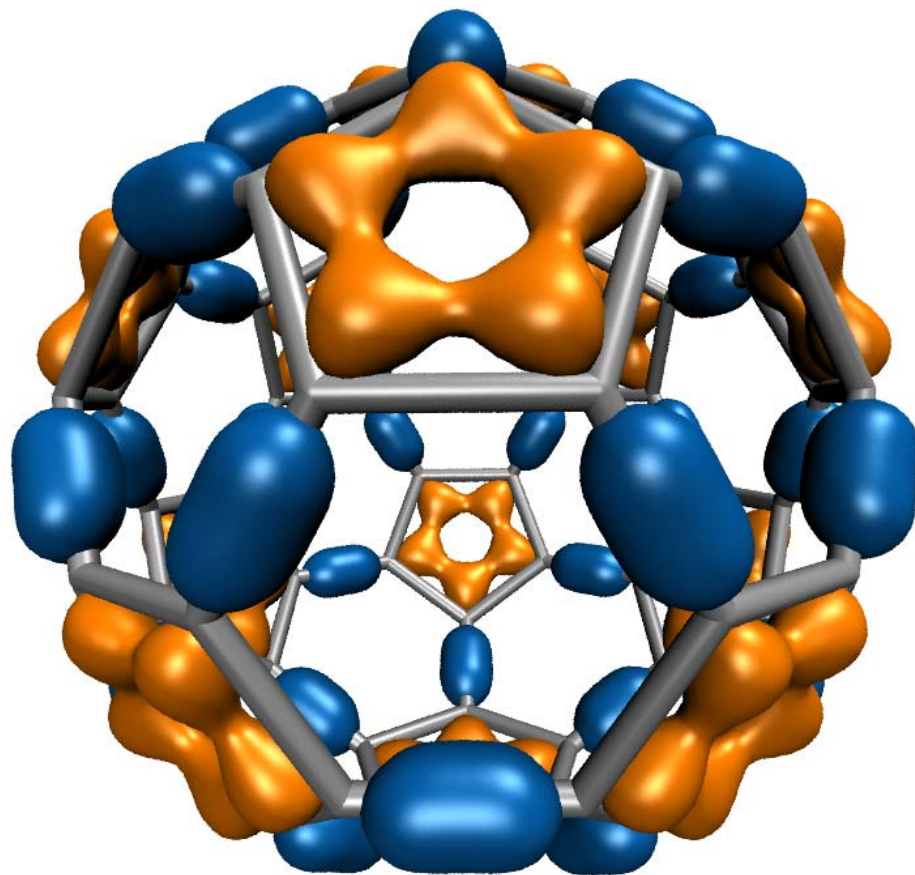
- Calculation of high resolution MO grids can require tens to hundreds of seconds in existing tools
- Existing tools cache MO grids as much as possible to avoid recomputation:
  - Doesn't eliminate the wait for initial calculation, hampers interactivity
  - Cached grids consume 100x-1000x more memory than MO coefficients



C<sub>60</sub>

# Animating Molecular Orbitals

- Animation of (classical mechanics) molecular dynamics trajectories provides insight into simulation results
- To do the same for QM or QM/MM simulations one must compute MOs at **~10 FPS** or more
- **>100x** speedup (GPU) over existing tools now makes this possible!



$C_{60}$

# Molecular Orbital Computation and Display Process

**One-time  
initialization**

**Initialize Pool of GPU  
Worker Threads**

Read QM simulation log file, trajectory

Preprocess MO coefficient data  
eliminate duplicates, sort by type, etc...

For current frame and MO index,  
retrieve MO wavefunction coefficients

**Compute 3-D grid of MO wavefunction amplitudes**  
Most performance-demanding step, run on **GPU...**

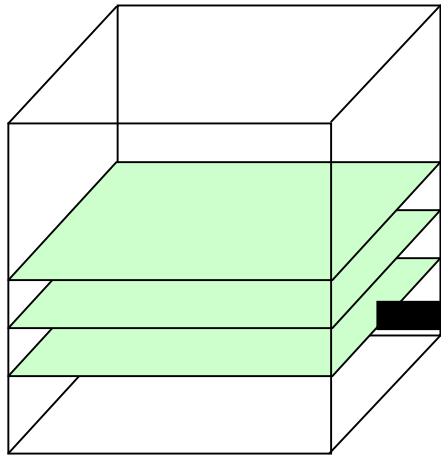
Extract isosurface mesh from 3-D MO grid

Apply user coloring/texturing  
and render the resulting surface

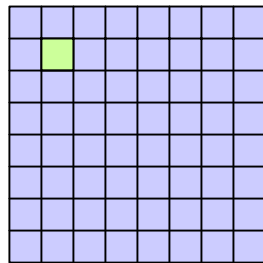
**For each trj frame, for  
each MO shown**

# CUDA Block/Grid Decomposition

MO 3-D lattice decomposes into  
2-D slices (CUDA grids)

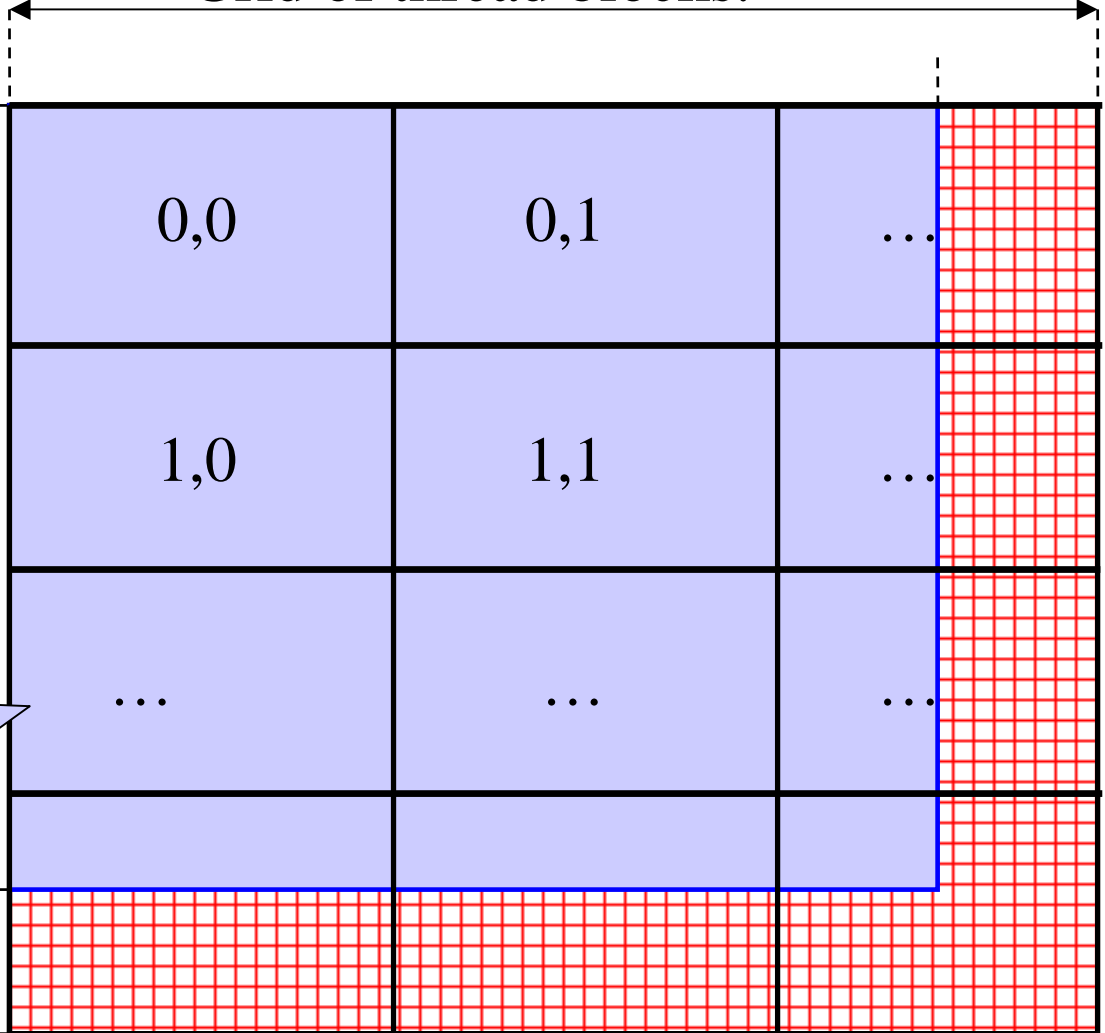


Small 8x8 thread  
blocks afford large  
per-thread register  
count, shared mem.  
Threads compute  
one MO lattice  
point each.



Padding optimizes glob. mem  
perf, guaranteeing coalescing

Grid of thread blocks:



# MO Kernel for One Grid Point (Naive C)

```
...
for (at=0; at<numatoms; at++) {
    int prim_counter = atom_basis[at];
    calc_distances_to_atom(&atompos[at], &xdist, &ydist, &zdist, &dist2, &xdiv);
    for (contracted_gto=0.0f, shell=0; shell < num_shells_per_atom[at]; shell++) {
        int shell_type = shell_symmetry[shell_counter];
        for (prim=0; prim < num_prim_per_shell[shell_counter]; prim++) {
            float exponent = basis_array[prim_counter];
            float contract_coeff = basis_array[prim_counter + 1];
            contracted_gto += contract_coeff * expf(-exponent*dist2);
            prim_counter += 2;
        }
        for (tmpshell=0.0f, j=0, zdp=1.0f; j<=shell_type; j++, zdp*=zdist) {
            int imax = shell_type - j;
            for (i=0, ydp=1.0f, xdp=pow(xdist, imax); i<=imax; i++, ydp*=ydist, xdp*=xdiv)
                tmpshell += wave_f[ifunc++] * xdp * ydp * zdp;
        }
        value += tmpshell * contracted_gto;
        shell_counter++;
    }
}
} .....
```

Loop over atoms

Loop over shells

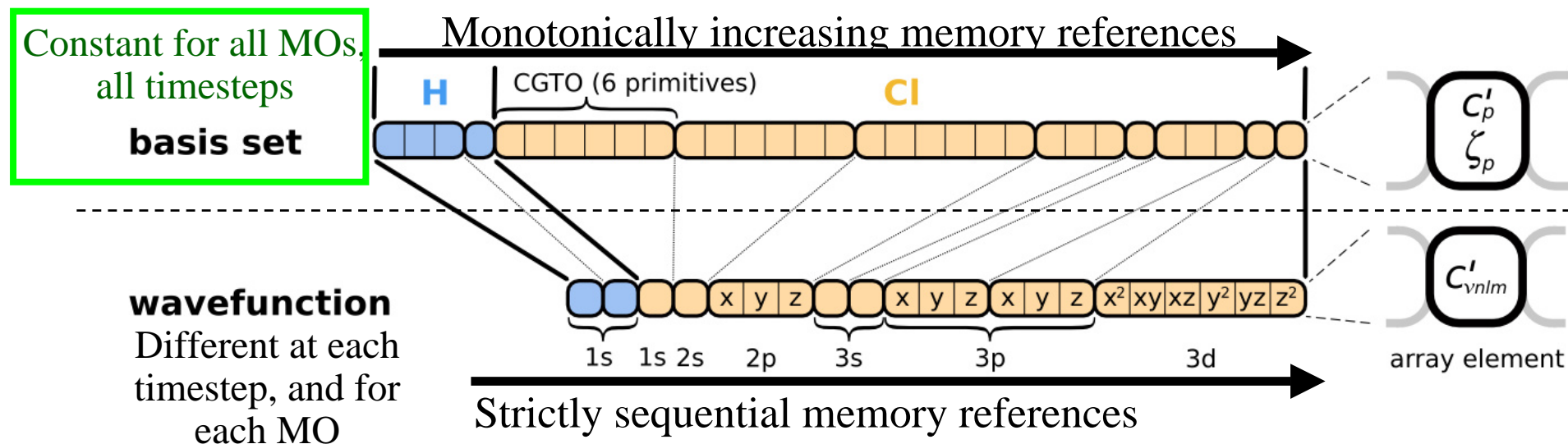
Loop over primitives:  
largest component of  
runtime, due to expf()

Loop over angular  
momenta  
(unrolled in real code)

# Preprocessing of Atoms, Basis Set, and Wavefunction Coefficients

- Must make effective use of high bandwidth, low-latency GPU on-chip memory, or CPU cache:
  - Overall storage requirement reduced by eliminating duplicate basis set coefficients
  - Sorting atoms by element type allows re-use of basis set coefficients for subsequent atoms of identical type
- Padding, alignment of arrays guarantees coalesced GPU global memory accesses, CPU SSE loads

# GPU Traversal of Atom Type, Basis Set, Shell Type, and Wavefunction Coefficients



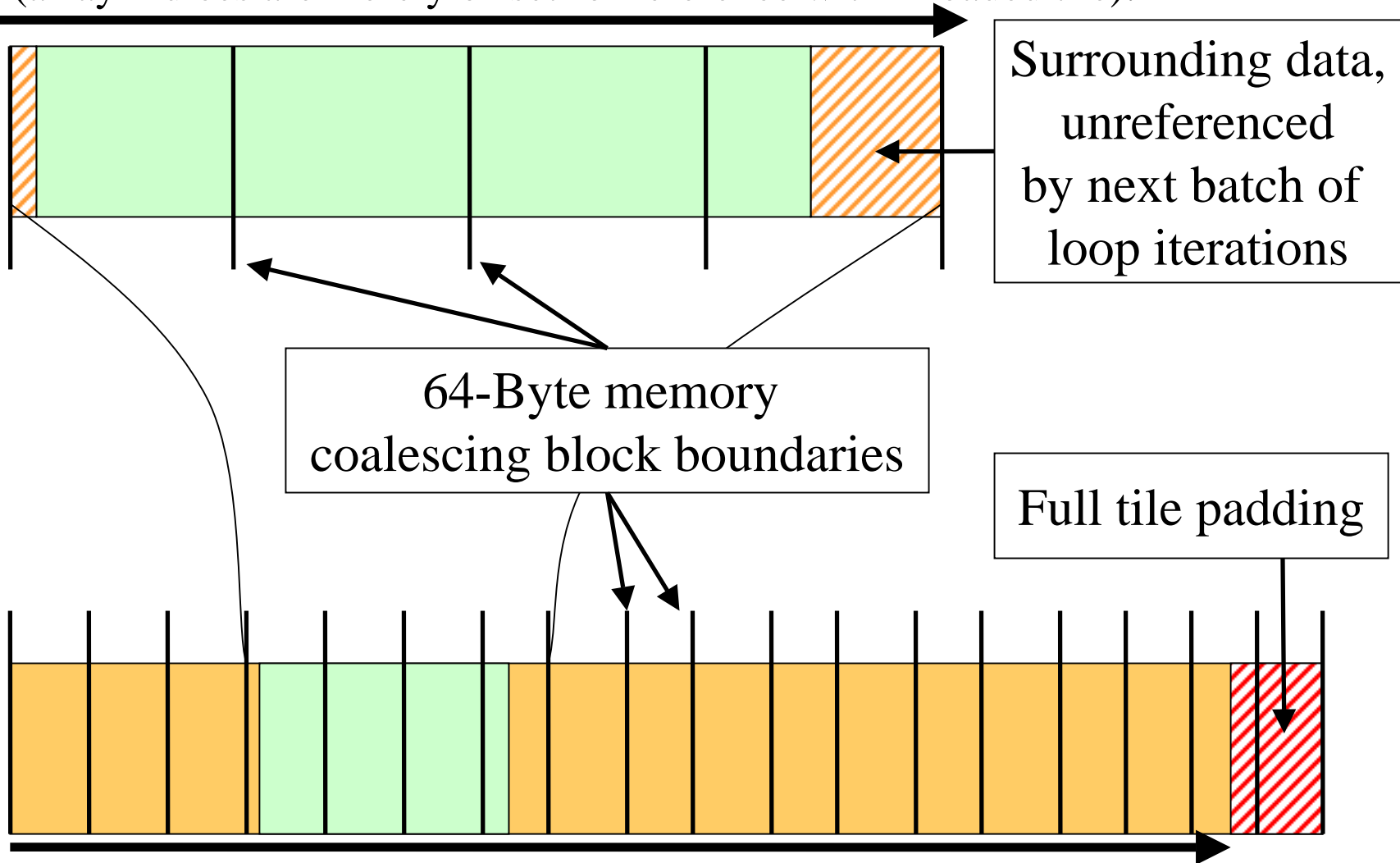
- Loop iterations always access same or consecutive array elements for all threads in a thread block:
  - Yields good constant memory cache performance
  - Increases shared memory tile reuse

# Use of GPU On-chip Memory

- If total data less than 64 kB, use only const mem:
  - Broadcasts data to all threads, no global memory accesses!
- For large data, shared memory used as a program-managed cache, coefficients loaded on-demand:
  - Tile data in shared mem is broadcast to 64 threads in a block
  - Nested loops traverse multiple coefficient arrays of varying length, complicates things significantly...
  - Key to performance is to locate tile loading checks outside of the two performance-critical inner loops
  - Tiles sized large enough to service entire inner loop runs
  - Only 27% slower than hardware caching provided by constant memory (GT200)



Array tile loaded in GPU shared memory. Tile size is a power-of-two, multiple of coalescing size, and allows simple indexing in inner loops (array indices are merely offset for reference within loaded tile).



Coefficient array in GPU global memory

# VMD MO Performance Results for C<sub>60</sub>

## Sun Ultra 24: Intel Q6600, NVIDIA GTX 280

Kernel	Cores/GPUs	Runtime (s)	Speedup
CPU ICC-SSE	1	46.58	1.00
CPU ICC-SSE	4	11.74	3.97
CPU ICC-SSE-approx**	4	3.76	12.4
CUDA-tiled-shared	1	0.46	100.
CUDA-const-cache	1	0.37	126.
<b>CUDA-const-cache-JIT*</b>	<b>1</b>	<b>0.27</b>	<b>173.</b> <b>(JIT 40% faster)</b>

C<sub>60</sub> basis set 6-31Gd. We used an unusually-high resolution MO grid for accurate timings. A more typical calculation has 1/8<sup>th</sup> the grid points.

\* Runtime-generated JIT kernel compiled using batch mode CUDA tools

\*\*Reduced-accuracy approximation of expf(),  
cannot be used for zero-valued MO isosurfaces



# Performance Evaluation: Molekel, MacMolPlt, and VMD

## Sun Ultra 24: Intel Q6600, NVIDIA GTX 280

	<b>C<sub>60</sub>-A</b>	<b>C<sub>60</sub>-B</b>	<b>Thr-A</b>	<b>Thr-B</b>	<b>Kr-A</b>	<b>Kr-B</b>
Atoms	60	60	17	17	1	1
Basis funcs (unique)	<b>300 (5)</b>	<b>900 (15)</b>	<b>49 (16)</b>	<b>170 (59)</b>	<b>19 (19)</b>	<b>84 (84)</b>

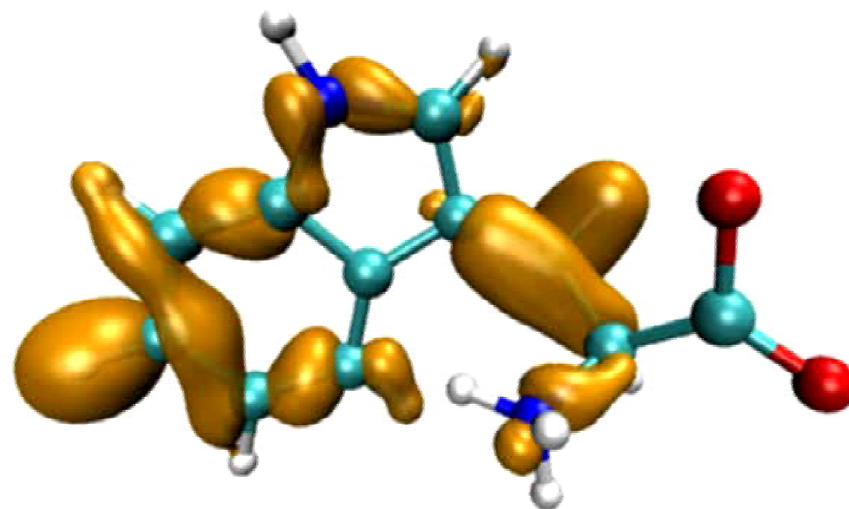
Kernel	Cores GPUs	Speedup vs. Molekel on 1 CPU core					
Molekel	1*	1.0	1.0	1.0	1.0	1.0	1.0
MacMolPlt	4	2.4	2.6	2.1	2.4	4.3	4.5
VMD GCC-cephes	4	3.2	4.0	3.0	3.5	4.3	6.5
VMD ICC-SSE-cephes	4	16.8	17.2	13.9	12.6	17.3	21.5
VMD ICC-SSE-approx**	4	59.3	53.4	50.4	49.2	54.8	69.8
VMD CUDA-const-cache	1	552.3	533.5	355.9	421.3	193.1	571.6

# VMD Orbital Dynamics Proof of Concept

One GPU can compute and animate this movie on-the-fly!

CUDA const-cache kernel,  
Sun Ultra 24, GeForce GTX 285

GPU MO grid calc.	<b>0.016 s</b>
CPU surface gen, volume gradient, and GPU rendering	<b>0.033 s</b>
<b>Total runtime</b>	<b>0.049 s</b>
<b>Frame rate</b>	<b>20 FPS</b>

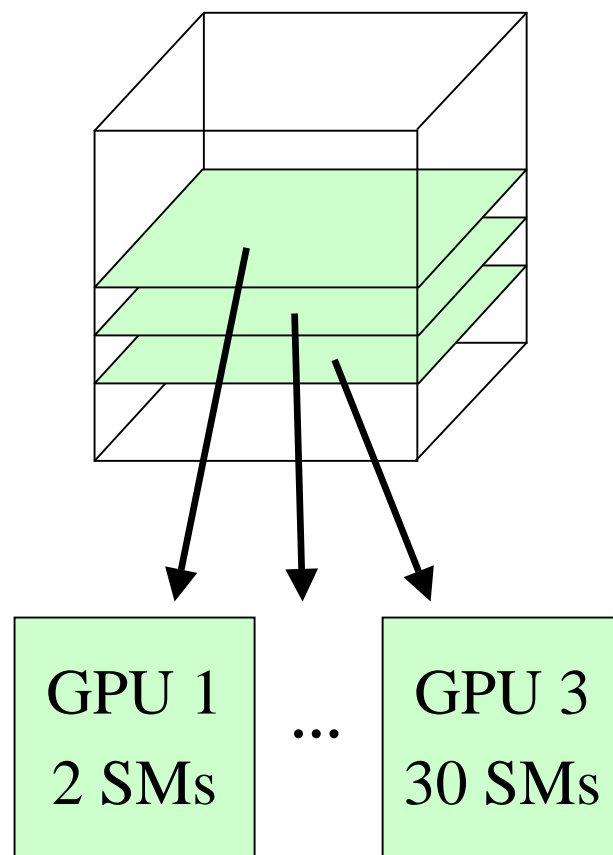


threonine

With GPU speedups over **100x**, previously insignificant CPU surface gen, gradient calc, and rendering are now **66%** of runtime. Need GPU-accelerated surface gen next...

# Multi-GPU Load Balance

- Many early CUDA codes assumed all GPUs were identical
- All new NVIDIA cards support CUDA, so a typical machine may have a diversity of GPUs of varying capability
- Static decomposition works poorly for non-uniform workload, or diverse GPUs, e.g. 2 SM, 16 SM, 30 SM



# VMD Multi-GPU Molecular Orbital Performance Results for C<sub>60</sub>

Kernel	Cores/GPUs	Runtime (s)	Speedup	Parallel Efficiency
CPU-ICC-SSE	1	46.580	1.00	100%
CPU-ICC-SSE	4	11.740	3.97	99%
CUDA-const-cache	1	0.417	112	100%
CUDA-const-cache	2	0.220	212	94%
CUDA-const-cache	3	0.151	308	92%
CUDA-const-cache	4	0.113	412	92%

Intel Q6600 CPU, 4x Tesla C1060 GPUs,

Uses persistent thread pool to avoid GPU init overhead,  
dynamic scheduler distributes work to GPUs

# Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- Wen-mei Hwu and the IMPACT group at University of Illinois at Urbana-Champaign
- NVIDIA Center of Excellence, University of Illinois at Urbana-Champaign
- NCSA Innovative Systems Lab
- David Kirk and the CUDA team at NVIDIA
- NIH support: P41-RR05969