Experiences Developing and Maintaining Scientific Applications on GPU-Accelerated Platforms

John E. Stone

Theoretical and Computational Biophysics Group

Beckman Institute for Advanced Science and Technology

University of Illinois at Urbana-Champaign

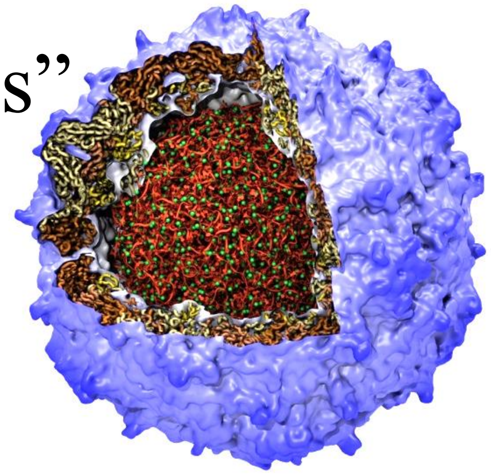
http://www.ks.uiuc.edu/Research/gpu/

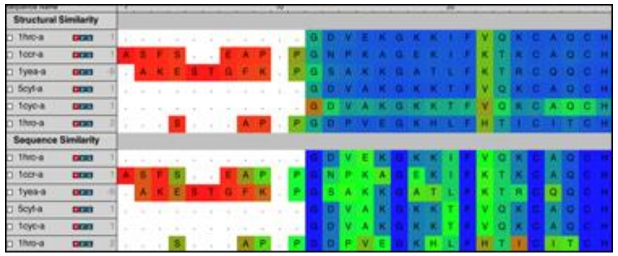
Big Red 2 Workshop

Indiana University, October 2, 2013

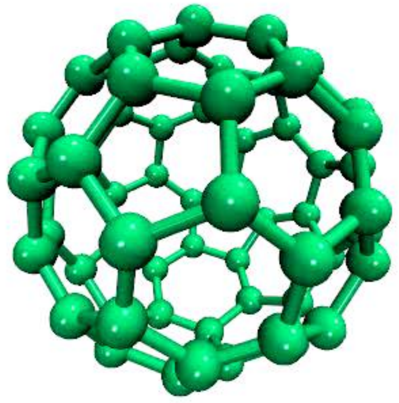
VMD – “Visual Molecular Dynamics”

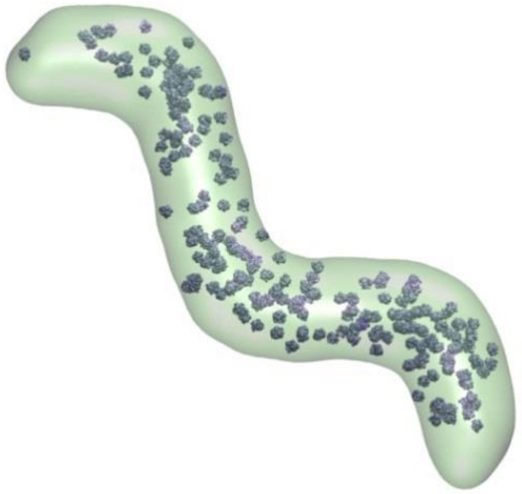
- Visualization and analysis of:
  - molecular dynamics simulations
  - quantum chemistry calculations
  - particle systems and whole cells
  - sequence data
- User extensible w/ scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/

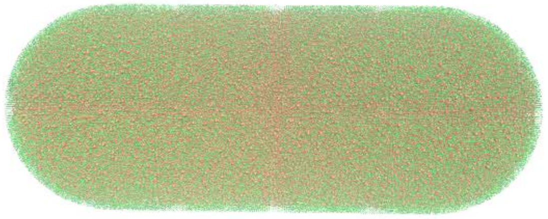

Poliovirus



Ribosome Sequences


Electrons in Vibrating Buckyball


Cellular Tomography, Cryo-electron Microscopy


Whole Cell Simulations

GPU Computing

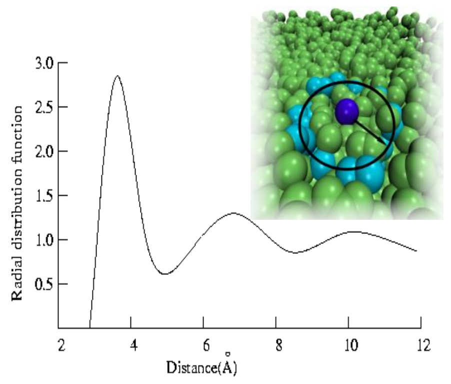
- Commodity devices, omnipresent in modern computers (over a **million** sold per **week**)
- Massively parallel hardware, hundreds of processing units, **throughput oriented architecture**
- Standard integer and floating point types supported
- Programming tools allow software to be written in dialects 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

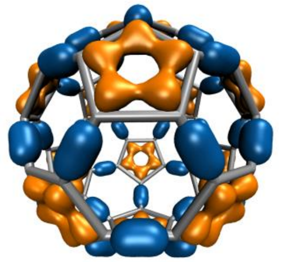
What Speedups Can GPUs Achieve?

- 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

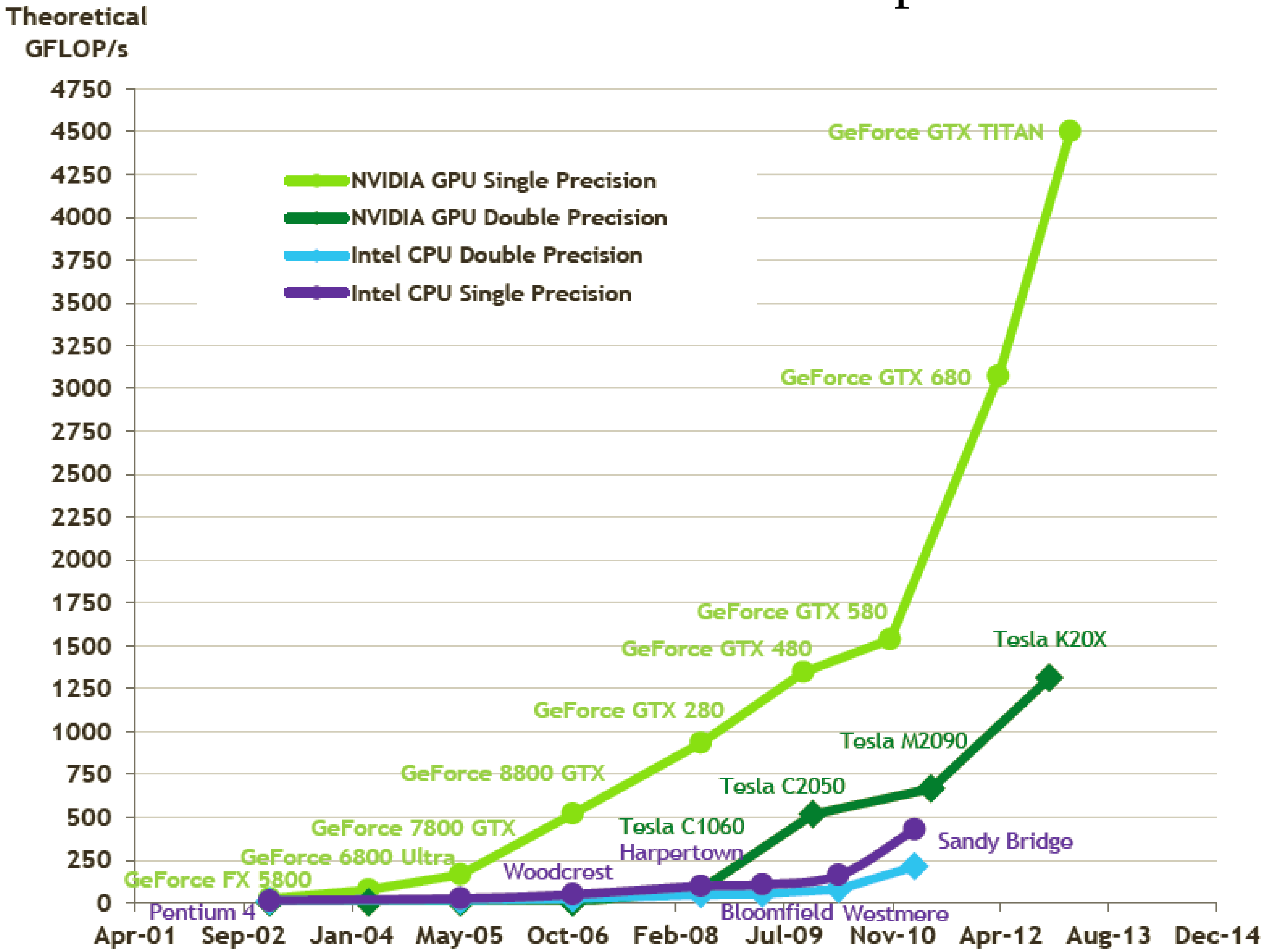
CUDA GPU-Accelerated Trajectory Analysis and Visualization in VMD

GPU-Accelerated Feature or Kernel	**Typical speedup vs. a single CPU core**
Molecular orbital display	**120x**
Radial distribution function	**92x**
Ray tracing w/ shadows	**46x**
Electrostatic field calculation	**44x**
Molecular surface display	**40x**
Ion placement	26x
MDFF density map synthesis	26x
Implicit ligand sampling	25x
Root mean squared fluctuation	25x
Radius of gyration	21x
Close contact determination	20x
Dipole moment calculation	15x

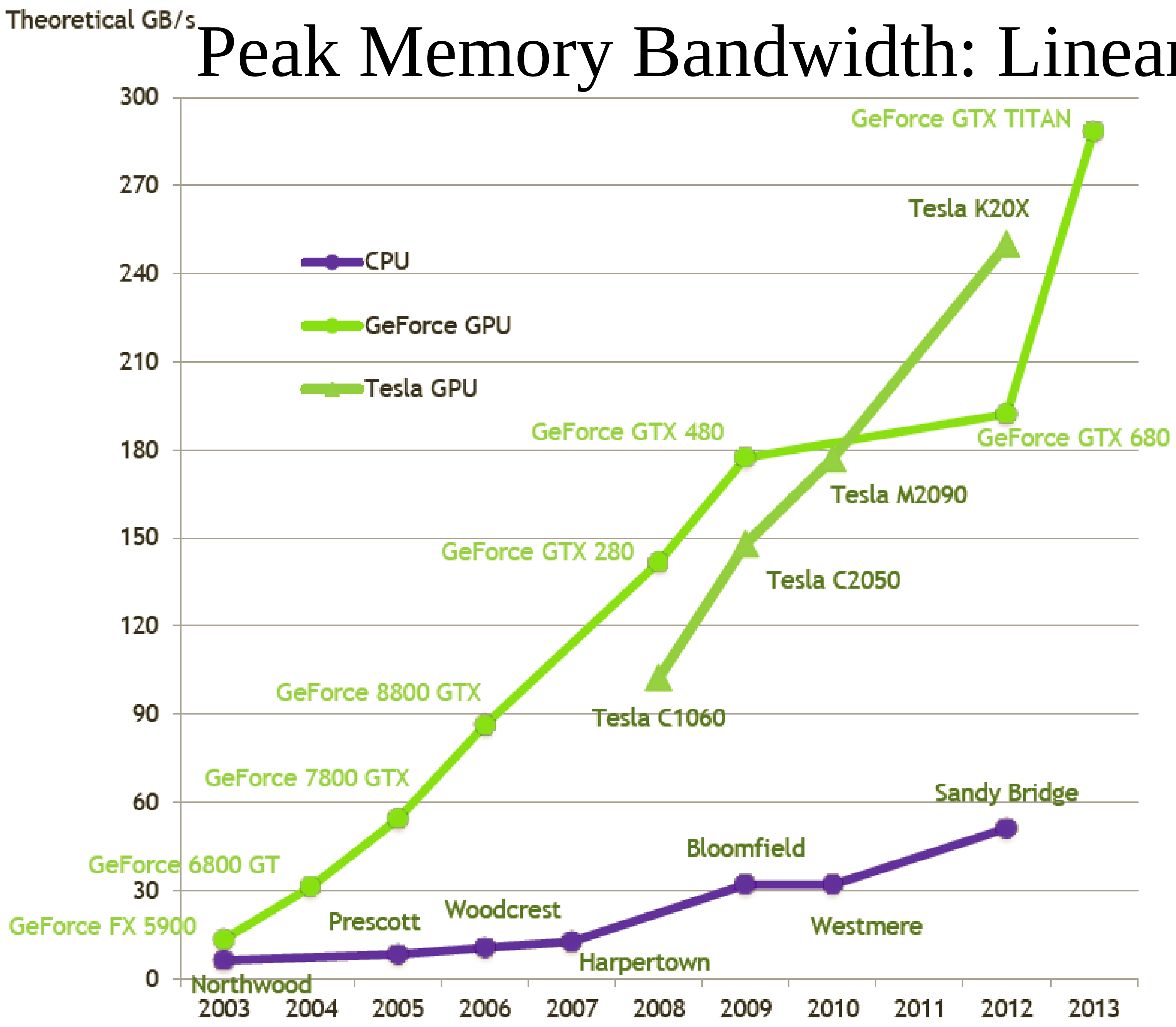




Peak Arithmetic Performance: Exponential Trend



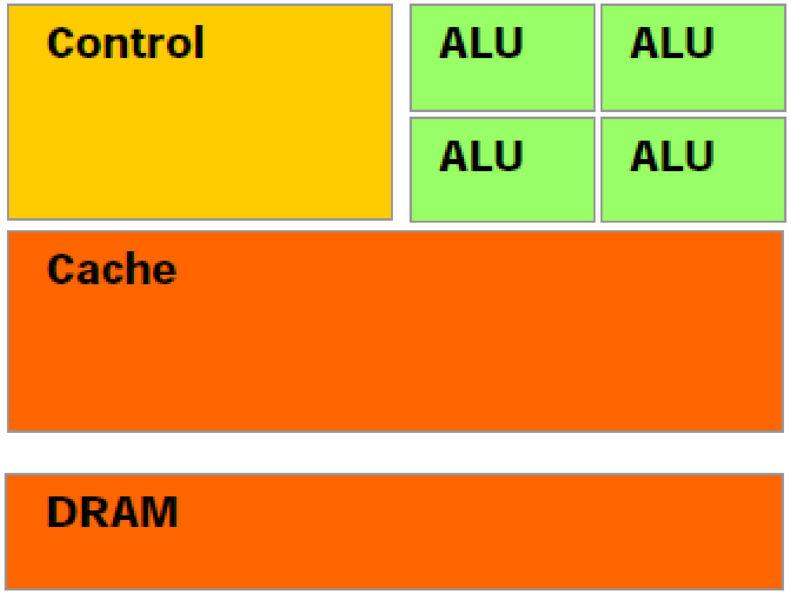
Peak Memory Bandwidth: Linear Trend

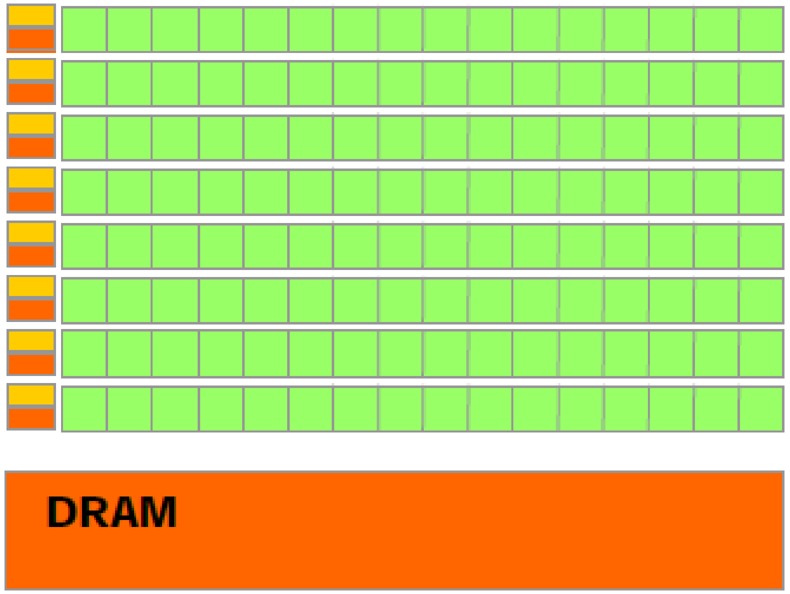


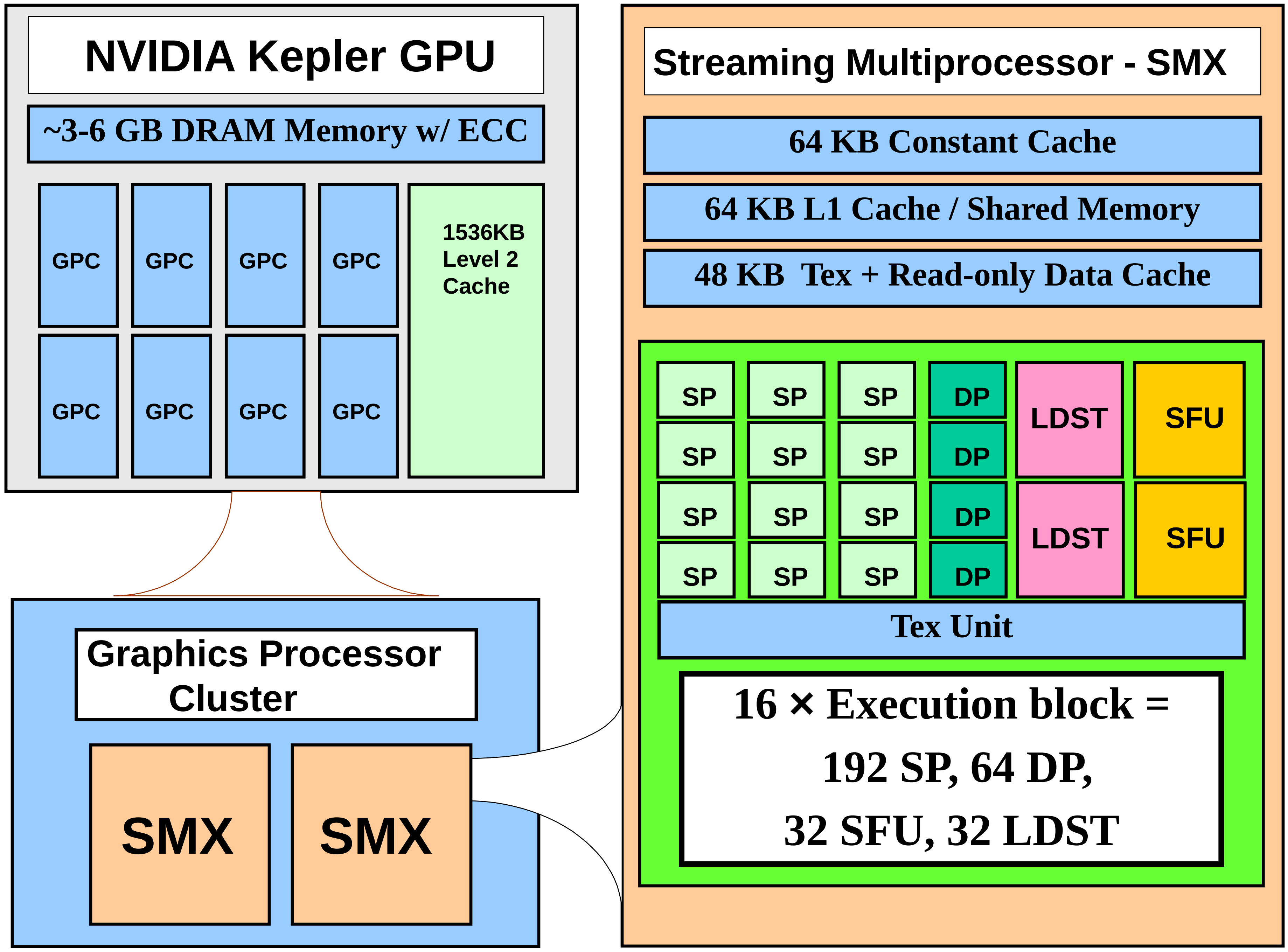
Comparison of CPU and GPU Hardware Architecture

CPU: Cache heavy, focused on individual thread performance

GPU: ALU heavy, massively parallel, throughput oriented





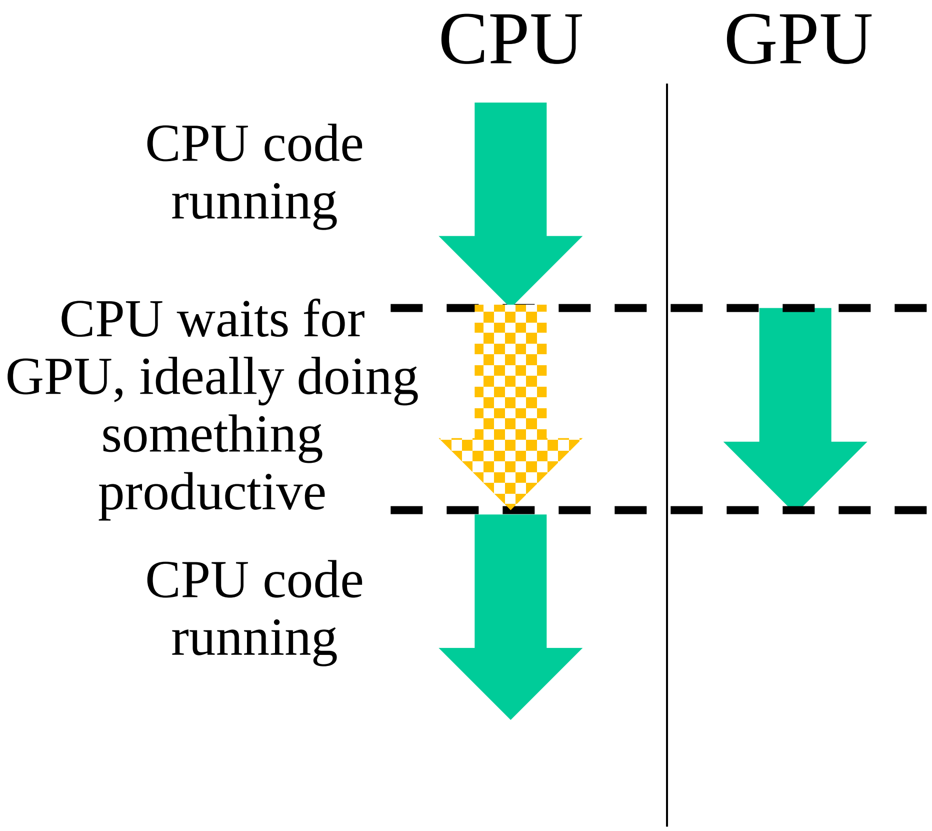

NVIDIA Kepler GPU
~3-6 GB DRAM Memory w/ ECC
GPC
GPC
GPC
GPC
GPC
GPC
GPC
GPC
1536KB Level 2 Cache
Graphics Processor Cluster
SMX
SMX
Streaming Multiprocessor - SMX
64 KB Constant Cache
64 KB L1 Cache / Shared Memory
48 KB Tex + Read-only Data Cache
SP
DP
LDST
SFU
Tex Unit
16 × Execution block = 192 SP, 64 DP, 32 SFU, 32 LDST

What Runs on a GPU?

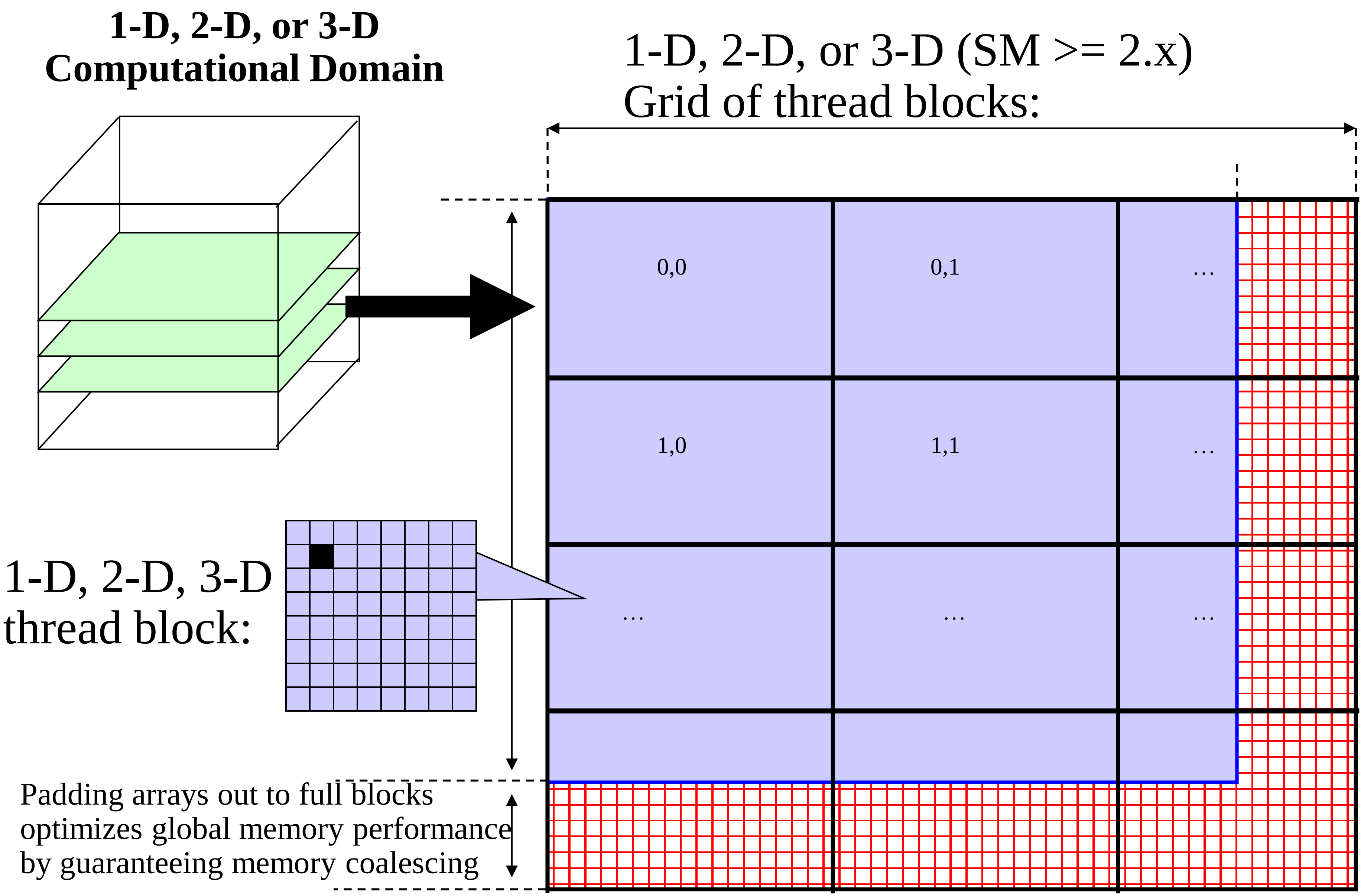
- GPUs run data-parallel programs called **"kernels"**
- GPUs are managed by a host CPU thread:
  - Create a CUDA context
  - Allocate/deallocate GPU memory
  - Copy data between host and GPU memory
  - Launch GPU kernels
  - Query GPU status
  - Handle runtime errors

CUDA Stream of Execution

- Host CPU thread launches a CUDA "kernel", a memory copy, etc. on the GPU
- GPU action runs to completion
- Host synchronizes with completed GPU action



CUDA Grid/Block/Thread Decomposition



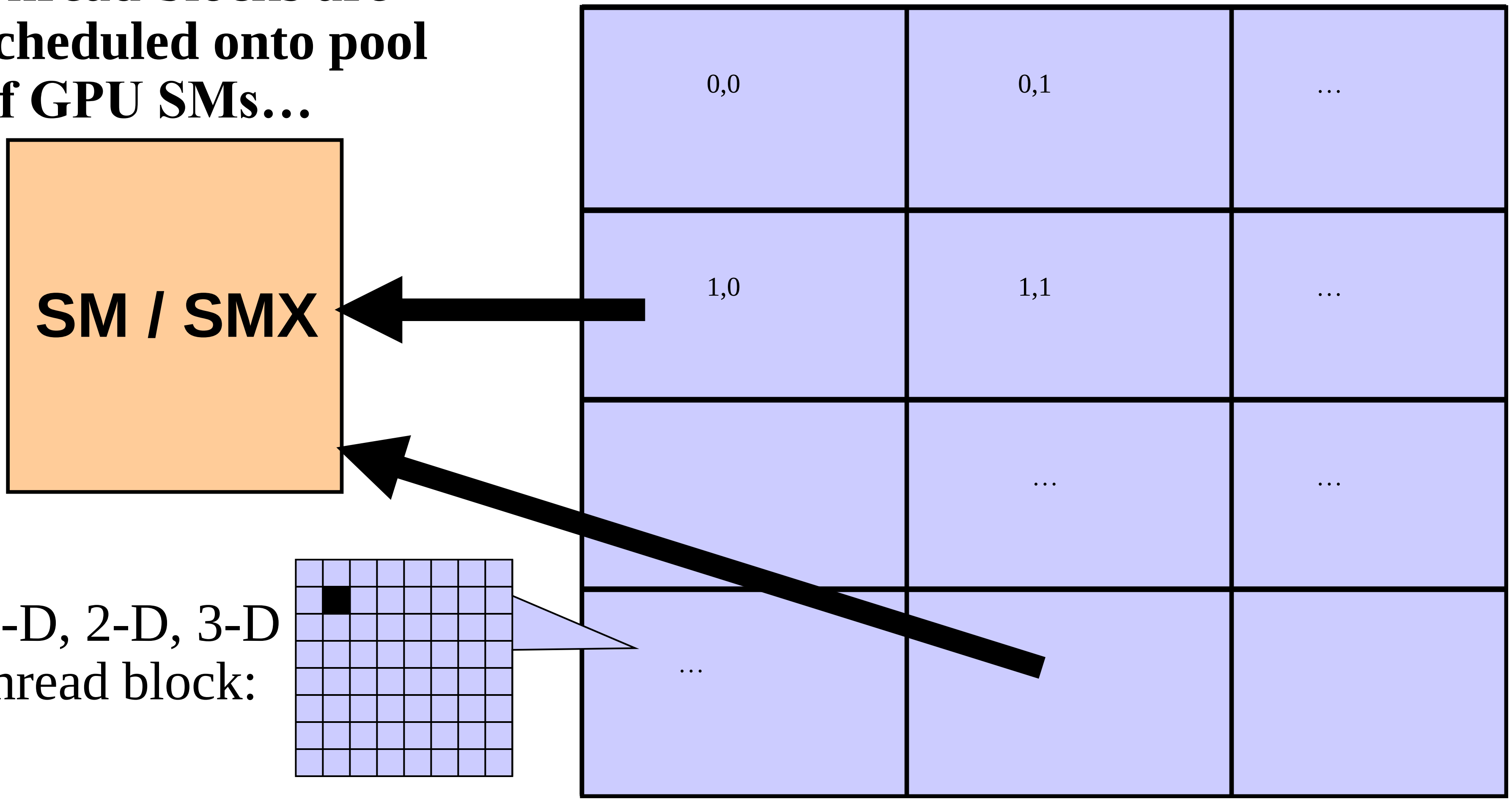
Padding arrays out to full blocks optimizes global memory performance by guaranteeing memory coalescing

CUDA Work Abstractions: Grids, Thread Blocks, Threads

1-D, 2-D, or 3-D (SM >= 2.x)
Grid of thread blocks:

Thread blocks are scheduled onto pool of GPU SMs…

1-D, 2-D, 3-D thread block:

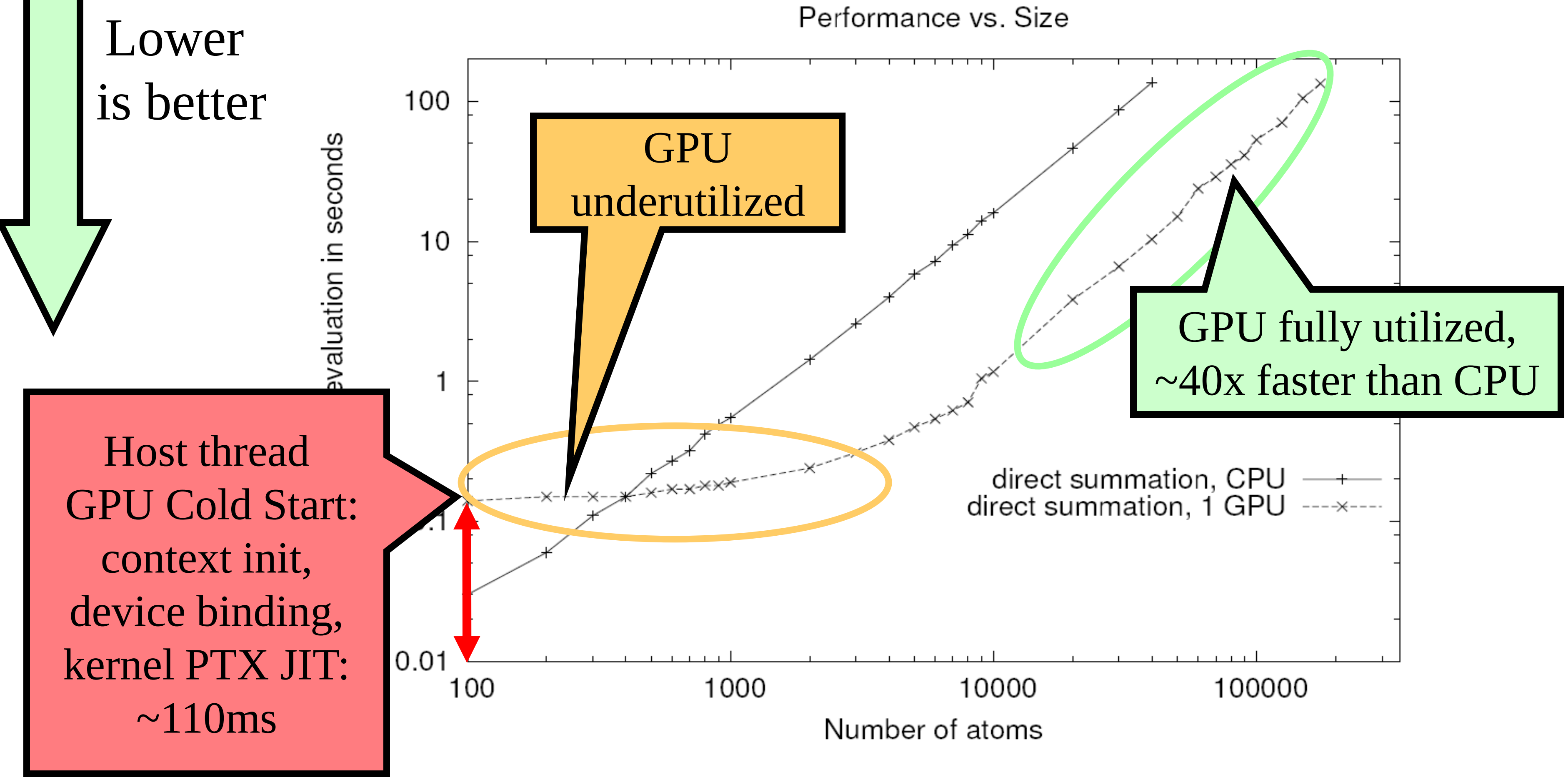


An Approach to Writing CUDA Kernels

- Find an algorithm that can expose **substantial parallelism**, we'll ultimately need thousands of independent threads…
- Identify **appropriate** GPU memory or texture subsystems used to store data used by kernel
- Are there trade-offs that can be made to exchange computation for **more parallelism**?
  - Though counterintuitive, past successes resulted from this strategy
  - "Brute force" methods that expose significant parallelism do surprisingly well on GPUs
- Analyze the real-world use case for the problem and select a specialized kernel for the problem sizes that will be heavily used

GPUs Require ~20,000 Independent Threads for Full Utilization, Latency Hidding

Lower is better



Host thread GPU Cold Start: context init, device binding, kernel PTX JIT: ~110ms

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.

Getting Performance From GPUs

- Don’t worry (much) about counting arithmetic operations…at least until you have nothing else left to do
- GPUs provide tremendous memory bandwidth, but even so, **memory bandwidth often ends up being the performance limiter**
- Keep/reuse data in **registers** as long as possible
- The main consideration when programming GPUs is **accessing memory efficiently**, and storing operands in the **most appropriate memory system** according to data size and access pattern

GPU Memory Systems

- GPU arithmetic rates dwarf memory bandwidth
- For Kepler K20 hardware:
  - ~2 TFLOPS vs. ~250 GB/sec
  - The ratio is roughly **40 FLOPS per memory reference** for single-precision floating point
- GPUs include multiple fast on-chip memories to help **narrow the gap**:
  - **Registers**
  - Constant memory (64KB)
  - **Shared memory (48KB / 16KB)**
  - Read-only data cache / Texture cache (48KB)

Loop Unrolling, Register Tiling

```
…for (atomid=0; atomid<numatoms; atomid++) {
    float dy = coory - atominfo[atomid].y;
    float dysqpdzsq = (dy * dy) + atominfo[atomid].z;
    float x = atominfo[atomid].x;
    float dx1 = coorx1 - x;
    float dx2 = coorx2 - x;
    float dx3 = coorx3 - x;
    float dx4 = coorx4 - x;
    float charge = atominfo[atomid].w;
    energyvalx1 += charge * rsqrtf(dx1*dx1 + dysqpdzsq);
    energyvalx2 += charge * rsqrtf(dx2*dx2 + dysqpdzsq);
    energyvalx3 += charge * rsqrtf(dx3*dx3 + dysqpdzsq);
    energyvalx4 += charge * rsqrtf(dx4*dx4 + dysqpdzsq);
  }
```

Compared to non-unrolled kernel: memory loads are decreased by 4x, and FLOPS per evaluation are reduced, but register use is increased…

Avoid Output Conflicts, Conversion of Scatter to Gather

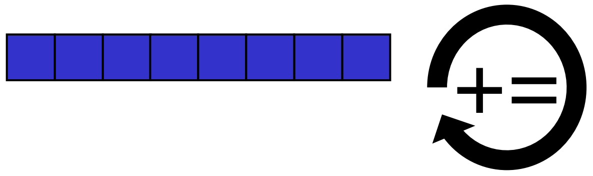
- Many CPU codes contain algorithms that “scatter” outputs to memory, to reduce arithmetic
- Scattered output can create bottlenecks for GPU performance due to bank conflicts
- On the GPU, it’s often better to do **more arithmetic**, in exchange for a **regularized output pattern**, or to **convert “scatter”** algorithms **to “gather”** approaches

Avoid Output Conflicts: Privatization Schemes

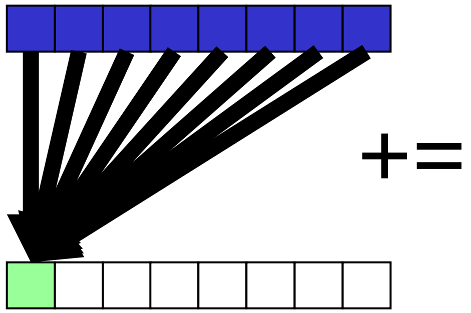
- ***Privatization***: use of private work areas for workers
  - Avoid/reduce the need for thread synchronization barriers
  - Avoid/reduce the need atomic increment/decrement operations during work, use **parallel reduction** at the end…
- By working in separate memory buffers, workers **avoid read/modify/write conflicts** of various kinds
- Huge GPU thread counts make it impractical to privatize data on a per-thread basis, so GPUs must use **coarser granularity: warps, thread-blocks**
- Use of the **on-chip shared memory** local to each SM can often be considered a form of privatization

Example: avoiding output conflicts when summing numbers among threads in a block

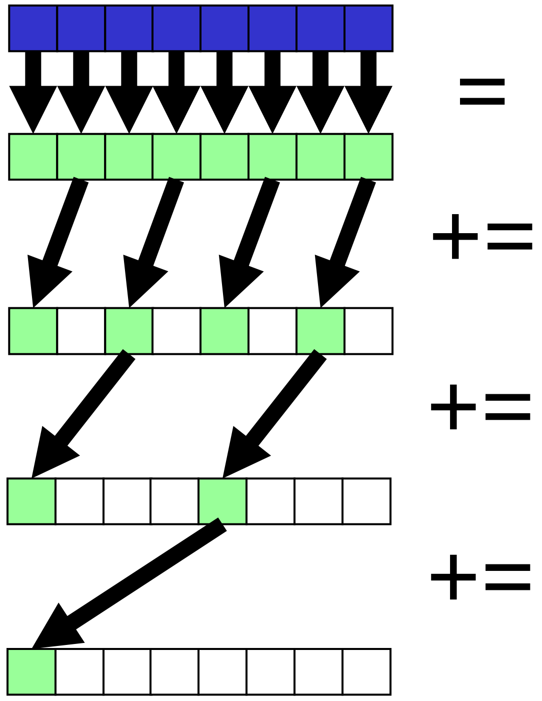
Accumulate sums in thread-local registers before doing any reduction among threads



N-way output conflict:
Correct results require **costly barrier synchronizations** or **atomic memory operations ON EVERY ADD** to prevent threads from overwriting each other…



Parallel reduction: no output conflicts, Log2(N) barriers

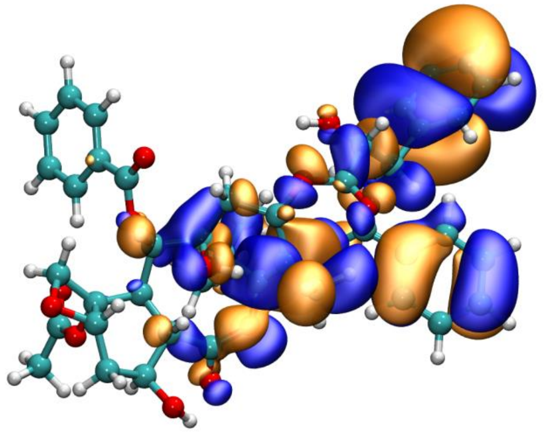


Using the CPU to Optimize GPU Performance

- 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; GPU processes the bulk of the work concurrently
  - On average, the GPU is kept highly occupied, attaining a high fraction of peak performance

Science 5: Quantum Chemistry Visualization

- Chemistry is the result of atoms sharing electrons
- Electrons occupy “clouds” in the space around atoms
- Calculations for visualizing these “clouds” are costly: **tens to hundreds of seconds** on CPUs – **non-interactive**
- GPUs enable the dynamics of electronic structures to be animated **interactively** for the first time

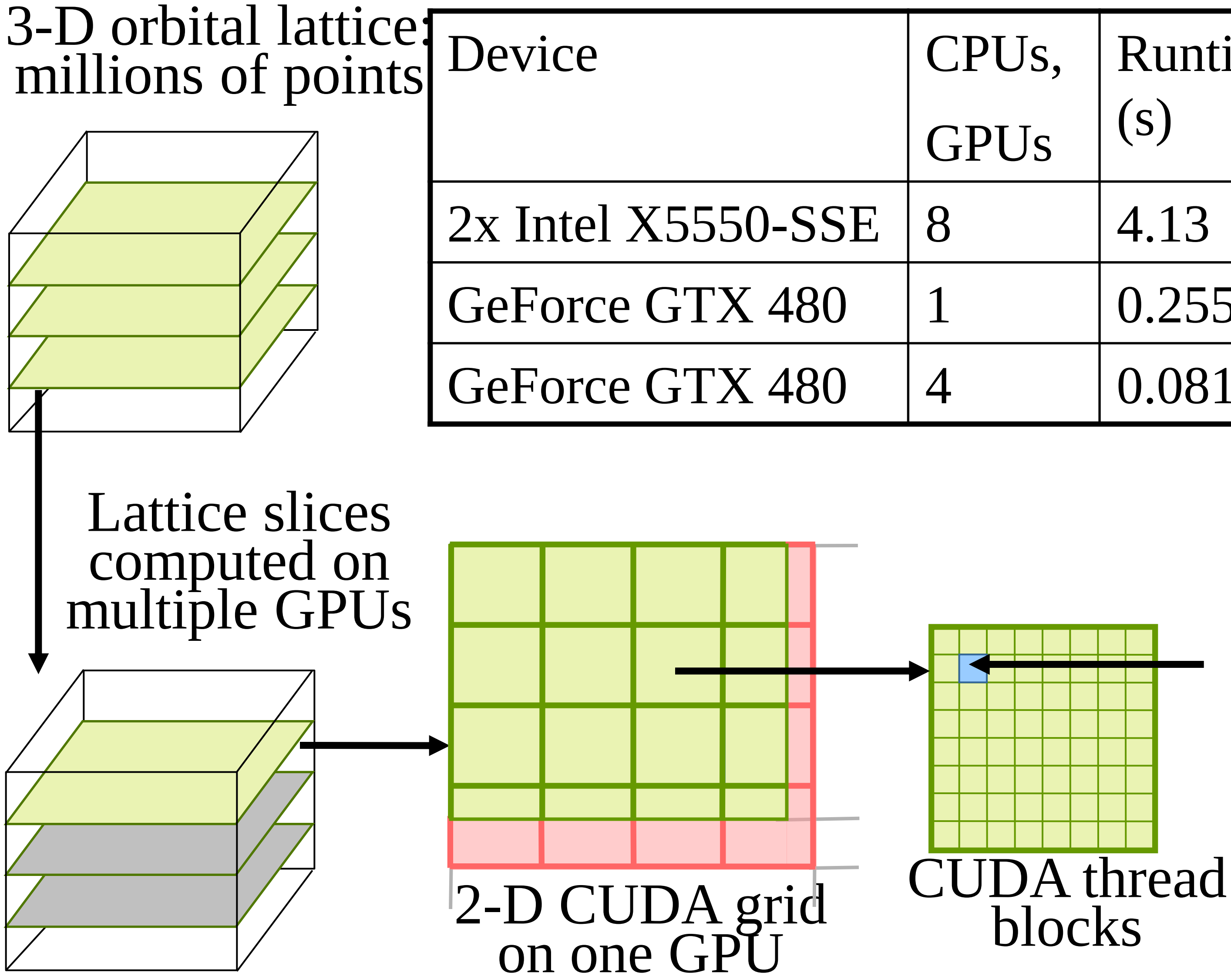


Taxol: cancer drug

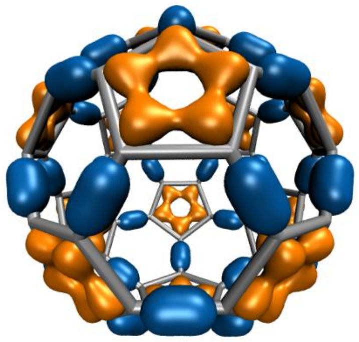
VMD enables interactive display of QM simulations, e.g. Terachem, GAMESS

GPU Solution: Computing C_{60} Molecular Orbitals

Device	CPUs, GPUs	Runtime (s)	Speedup
2x Intel X5550-SSE	8	4.13	1
GeForce GTX 480	1	0.255	16
GeForce GTX 480	4	0.081	51


3-D orbital lattice: millions of points
Lattice slices computed on multiple GPUs
2-D CUDA grid on one GPU
CUDA thread blocks
GPU threads each compute one point.

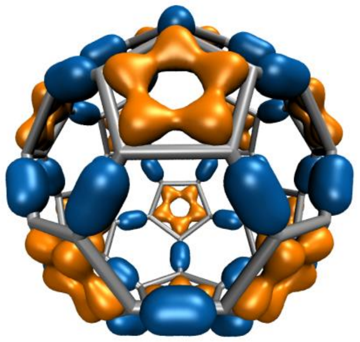
Molecular Orbital Inner Loop, Hand-Coded x86 SSE
Hard to Read, Isn't It? (And this is the "pretty" version!)



```
for (shell=0; shell < maxshell; shell++) {
    __m128 Cgto = _mm_setzero_ps();
    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];
        __m128 expval = _mm_mul_ps(_mm_load_ps1(&exponent), dist2);
        __m128 ctmp = _mm_mul_ps(_mm_load_ps1(&contract_coeff), exp_ps(expval));
        Cgto = _mm_add_ps(contracted_gto, ctmp);
        prim_counter += 2;
    }
    __m128 tshell = _mm_setzero_ps();
    switch (shell_types[shell_counter]) {
        case S_SHELL:
            value = _mm_add_ps(value, _mm_mul_ps(_mm_load_ps1(&wave_f[ifunc++]), Cgto));    break;
        case P_SHELL:
            tshell = _mm_add_ps(tshell, _mm_mul_ps(_mm_load_ps1(&wave_f[ifunc++]), xdist));
            tshell = _mm_add_ps(tshell, _mm_mul_ps(_mm_load_ps1(&wave_f[ifunc++]), ydist));
            tshell = _mm_add_ps(tshell, _mm_mul_ps(_mm_load_ps1(&wave_f[ifunc++]), zdist));
            value = _mm_add_ps(value, _mm_mul_ps(tshell, Cgto));      break;
```

Writing SSE kernels for CPUs requires assembly language, compiler intrinsics, various libraries, or a really smart autovectorizing compiler **and lots of luck...**

Molecular Orbital Inner Loop in CUDA

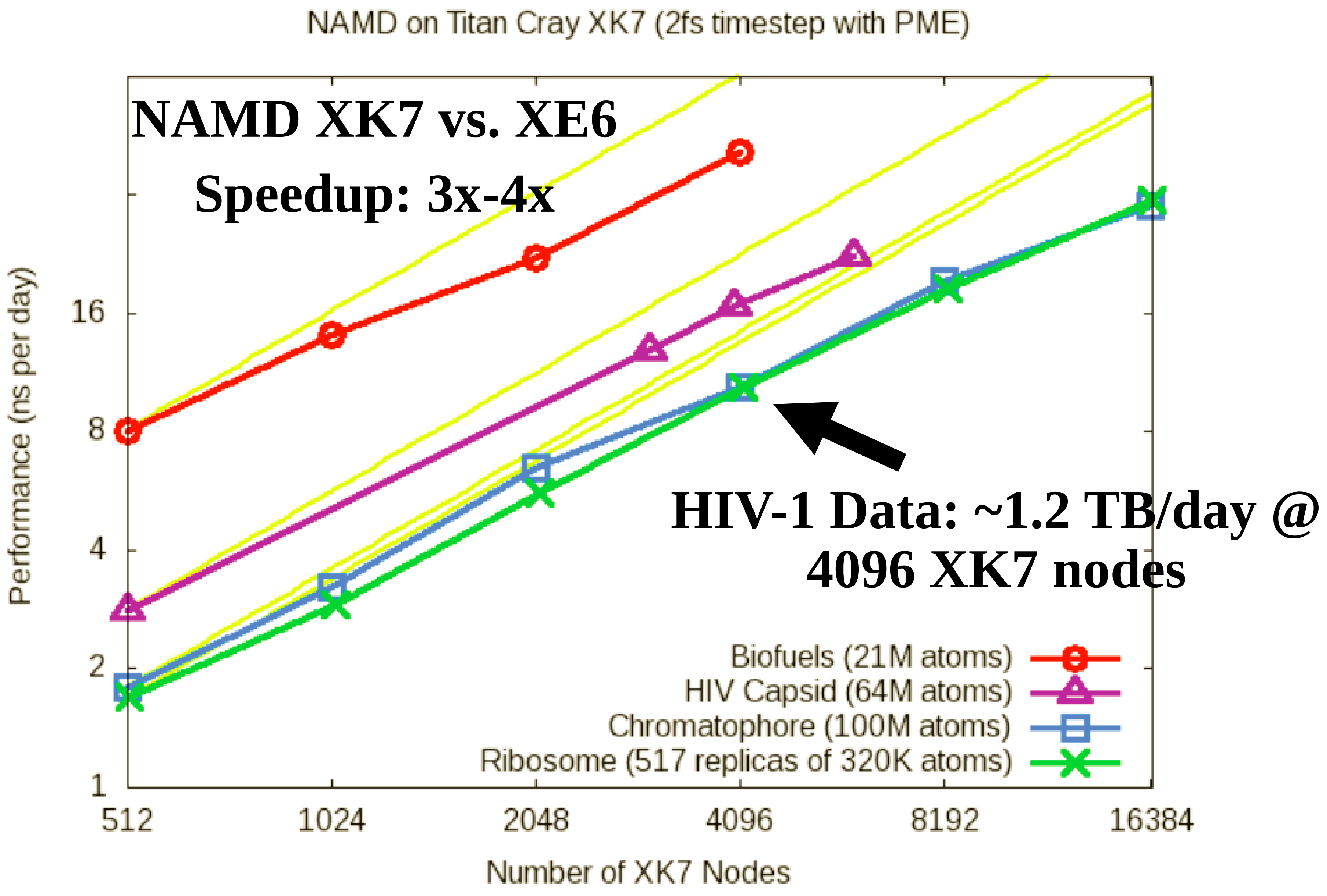


```
for (shell=0; shell < maxshell; shell++) {
    float contracted_gto = 0.0f;
    for (prim=0; prim<num_prim_per_shell[shell_counter]; prim++) {
      float exponent       = const_basis_array[prim_counter    ];
      float contract_coeff = const_basis_array[prim_counter + 1];
      contracted_gto += contract_coeff * exp2f(-exponent*dist2);
      prim_counter += 2;
    }
    float tmpshell=0;
    switch (const_shell_symmetry[shell_counter]) {
      case S_SHELL:
        value += const_wave_f[ifunc++] * contracted_gto;   break;
      case P_SHELL:
        tmpshell += const_wave_f[ifunc++] * xdist;
        tmpshell += const_wave_f[ifunc++] * ydist
        tmpshell += const_wave_f[ifunc++] * zdist;
        value += tmpshell * contracted_gto;   break;
```

Aaaaahhhh….

Data-parallel CUDA kernel looks like normal C code for the most part….

NAMD Titan XK7 Performance August 2013



NAMD XK7 vs. XE6

Speedup: 3x-4x

HIV-1 Data: ~1.2 TB/day @ 4096 XK7 nodes

VMD Petascale Visualization and Analysis

- Analyze/visualize large trajectories too large to transfer off-site:
  - Compute time-averaged electrostatic fields, MDFF quality-of-fit, etc.
  - User-defined parallel analysis operations, data types
  - Parallel rendering, movie making
- Parallel I/O rates up to **275 GB/sec** on 8192 Cray XE6 nodes – can read in **231 TB in 15 minutes!**
- Multi-level dynamic load balancing tested with up to 262,144 CPU cores
- **Supports GPU-accelerated Cray XK7 nodes for both visualization and analysis usage**

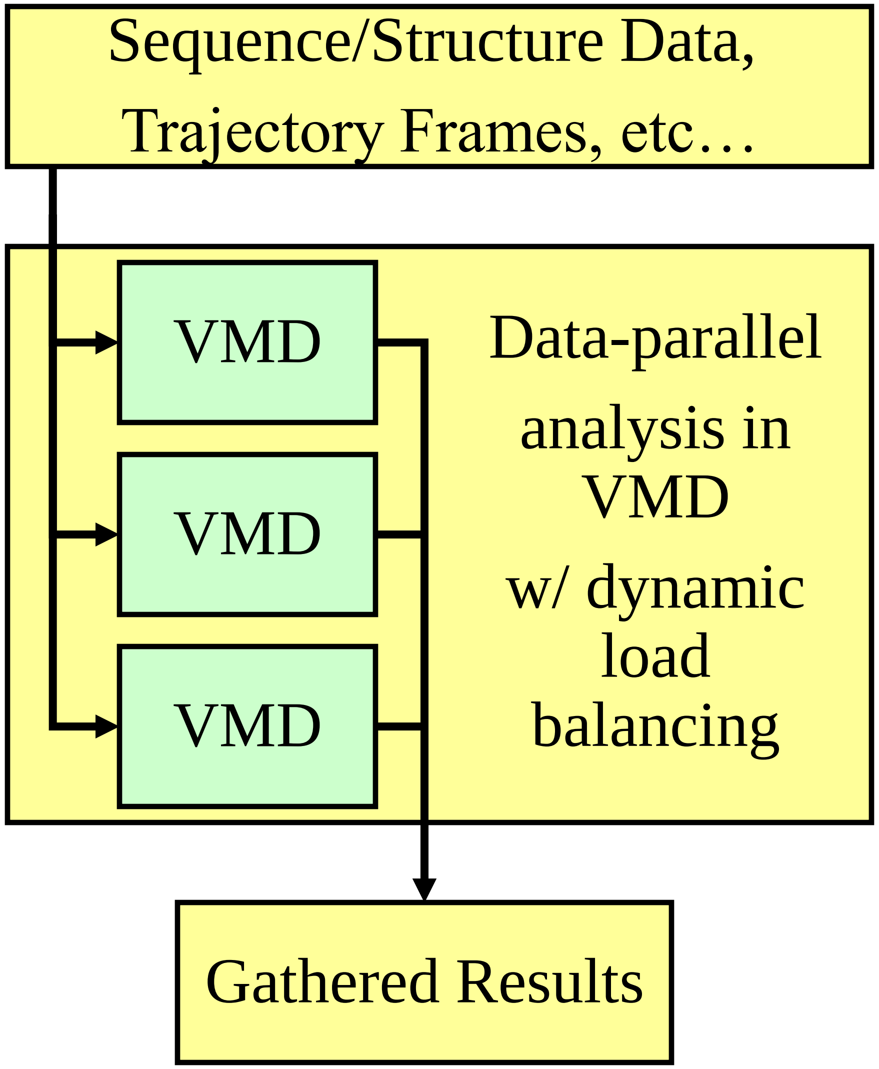
NCSA Blue Waters Hybrid Cray XE6 / XK7 Supercomputer

22,640 XE6 CPU nodes

4,224 XK7 nodes w/ GPUs support fast VMD OpenGL movie rendering and visualization

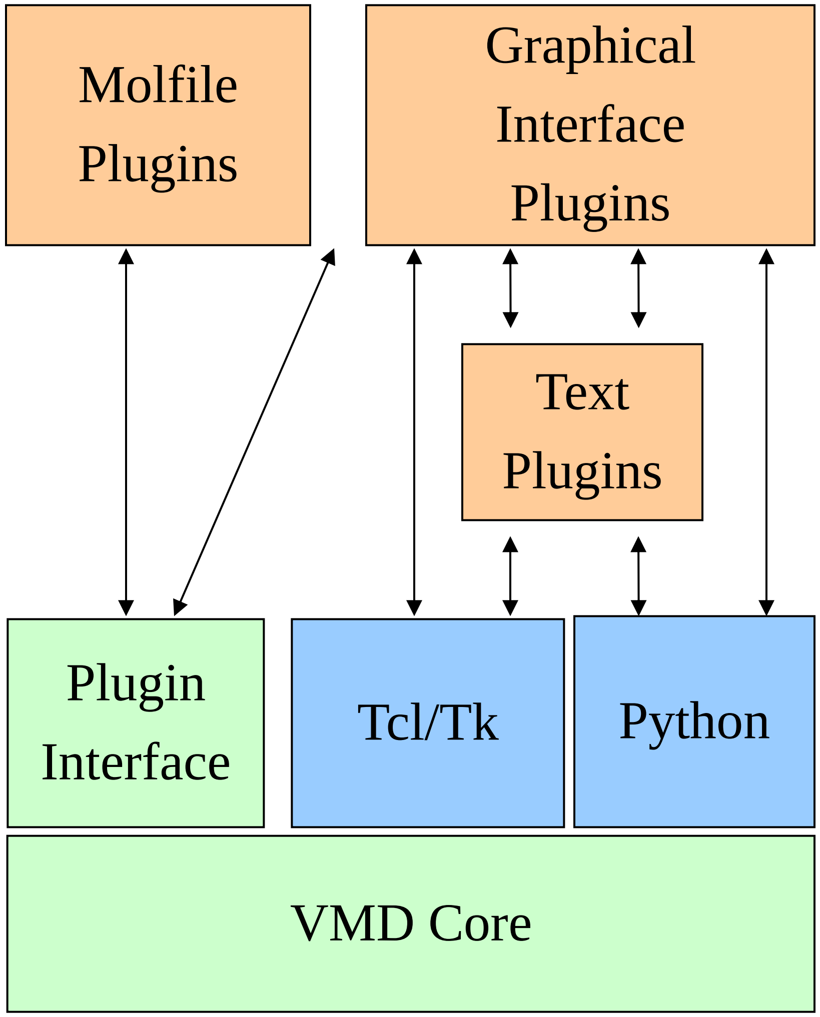
VMD for Demanding Analysis Tasks
Parallel VMD Analysis w/ MPI

- Compute time-averaged electrostatic fields, MDFF quality-of-fit, etc.
- Parallel rendering, movie making
- User-defined parallel reduction operations, data types
- **Parallel I/O on Blue Waters:**
  - **109 GB/sec on 512 nodes**
  - **275 GB/sec on 8,192 nodes**
- **Timeline per-residue SASA calc. achieves 800x speedup @ 1000 BW XE6 nodes**
- **Supports GPU-accelerated clusters and supercomputers**



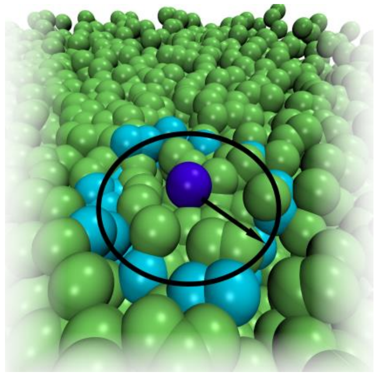
VMD as an Analysis Platform
Over 60 VMD Plugins Developed by Users

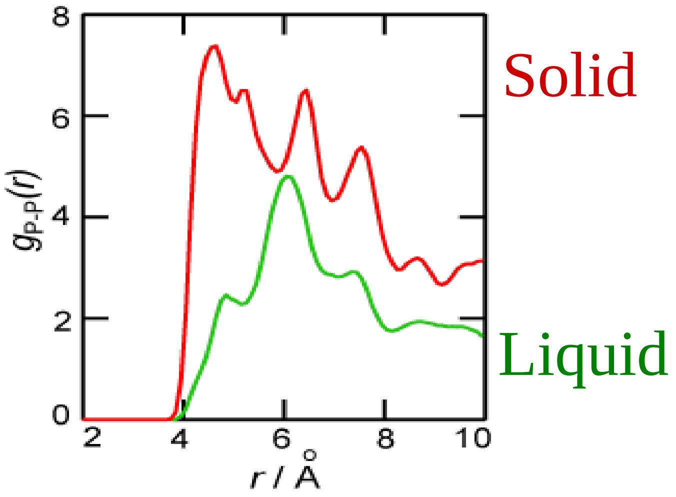
- VMD/NAMD sister programs, VMD is crucial for simulation analysis
- VMD user-extensible scripting w/ Tcl/Tk, Python
- Compiled C/C++ plugins loaded from shared libraries at runtime via **dlopen()**
- 70 molfile plugins provide access to molecular file formats
- **Built-in analysis commands exploit XE6 multi-core CPUs, XK7Tesla K20X GPUs**
- **New VMD collective ops and work scheduling interfaces enable existing code to be parallelized easily**



Radial Distribution Function

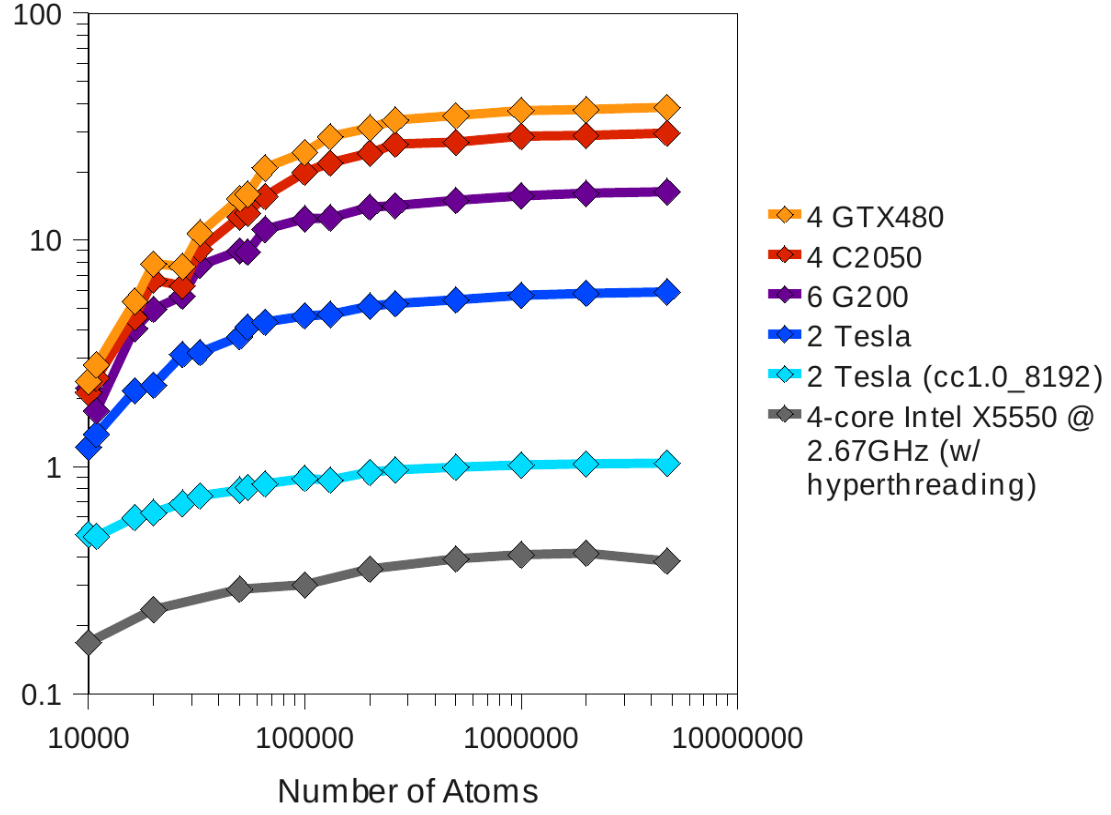
- RDFs describes how atom density varies with distance
- Can be compared with experiments
- Shape indicates phase of matter: sharp peaks appear for solids, smoother for liquids

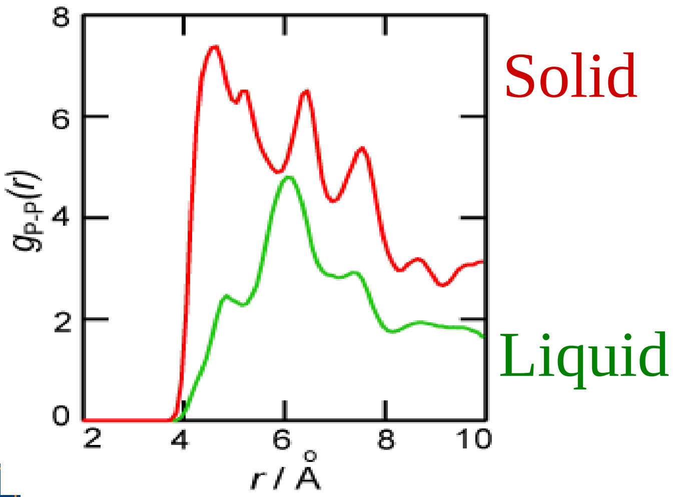




Multi-GPU RDF Performance

- 4 NVIDIA GTX480 GPUs 30 to 92x faster than 4-core Intel X5550 CPU
- Fermi GPUs ~3x faster than GT200 GPUs: larger on-chip shared memory

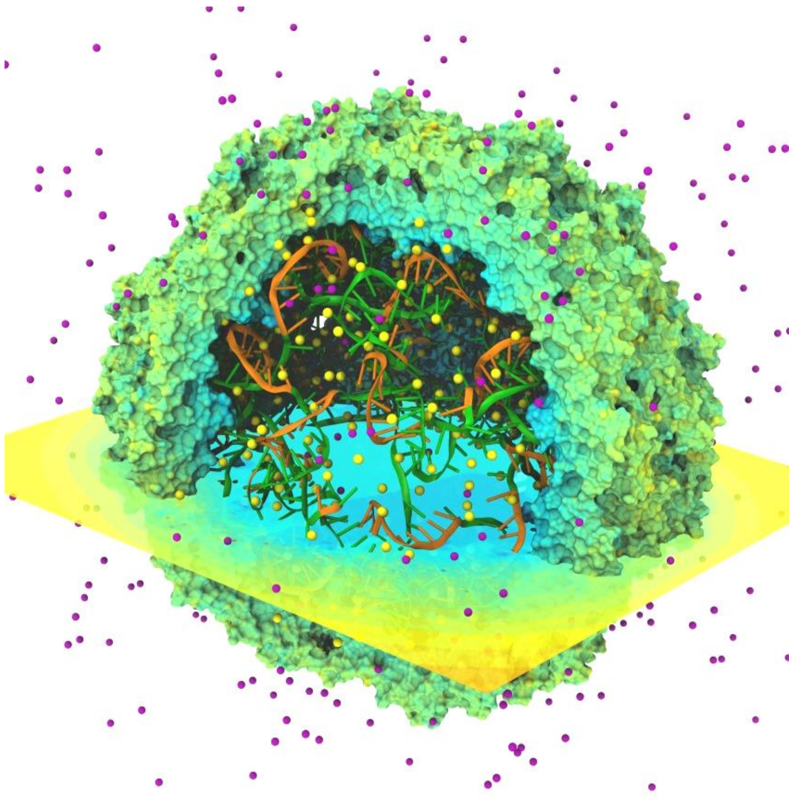




Fast Analysis of Molecular Dynamics Trajectories with Graphics Processing Units – Radial Distribution Functions. B. Levine, J. Stone, and A. Kohlmeyer. 2010. *J. Comp. Physics*, 230(9):3556-3569, 2011.

Time-Averaged Electrostatics Analysis on Energy-Efficient GPU Cluster

- **1.5 hour** job (CPUs) reduced to **3 min** (CPUs+GPU)
- Electrostatics of thousands of trajectory frames averaged
- Per-node power consumption on NCSA "AC" GPU cluster:
  - CPUs-only: 448 Watt-hours
  - CPUs+GPUs: 43 Watt-hours
- GPU Speedup: **25.5x**
- Power efficiency gain: **10.5x**



Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters. J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J. Stone, J. Phillips. *The Work in Progress in Green Computing*, pp. 317-324, 2010.

Time-Averaged Electrostatics Analysis on NCSA Blue Waters

NCSA Blue Waters Node Type	**Seconds per trajectory frame for one compute node**
Cray XE6 Compute Node: 32 CPU cores (2xAMD 6200 CPUs)	9.33
Cray XK6 GPU-accelerated Compute Node: 16 CPU cores + **NVIDIA X2090 (Fermi) GPU**	2.25
Speedup for GPU XK6 nodes vs. CPU XE6 nodes	**XK6 nodes are 4.15x faster overall**
Tests on XK7 nodes indicate MSM is CPU-bound with the Kepler K20X GPU. **Performance is not much faster (yet) than Fermi X2090** **Need to move spatial hashing, prolongation, interpolation onto the GPU…**	**In progress….** **XK7 nodes 4.3x faster overall**

Preliminary performance for VMD time-averaged electrostatics w/ Multilevel Summation Method on the NCSA Blue Waters Early Science System

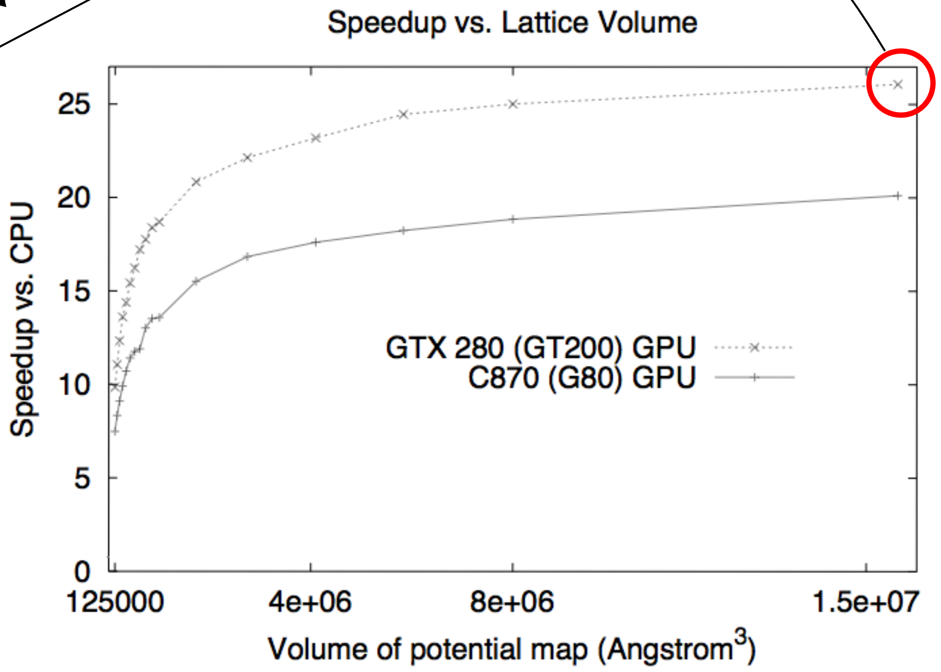
Multilevel Summation on the GPU

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.

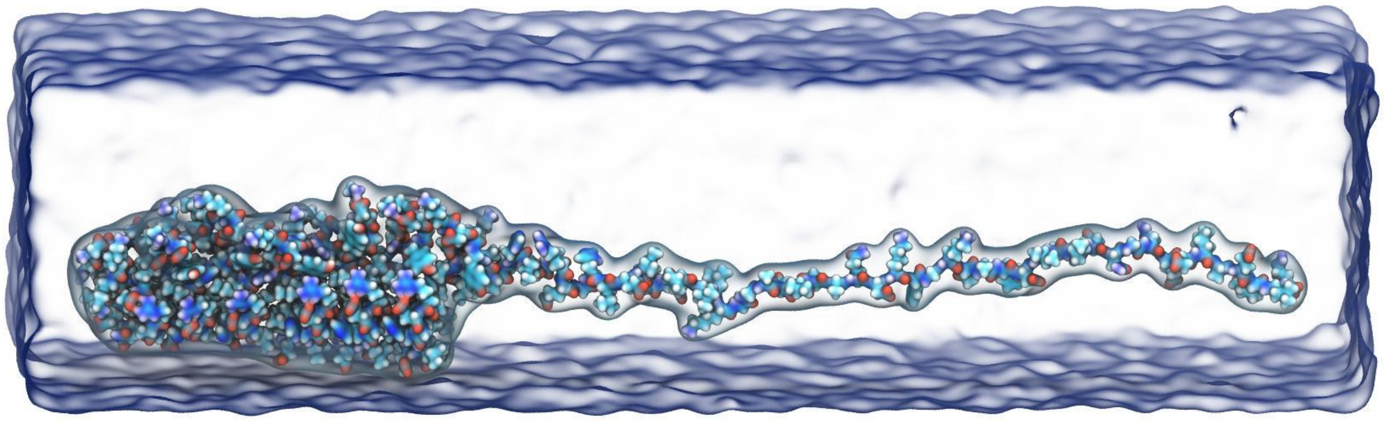
Computational steps	CPU (s)	w/ GPU (s)	Speedup
Short-range cutoff	480.07	14.87	32.3
Long-range anterpolation	0.18		
restriction	0.16		
lattice cutoff	49.47	1.36	36.4
prolongation	0.17		
interpolation	3.47		
Total	533.52	20.21	26.4



Multilevel summation of electrostatic potentials using graphics processing units. D. Hardy, J. Stone, K. Schulten. *J. Parallel Computing*, 35:164-177, 2009.

VMD "QuickSurf" Representation

- Displays continuum of structural detail:
  - All-atom models
  - Coarse-grained models
  - Cellular scale models
  - Multi-scale models: All-atom + CG, Brownian + Whole Cell
  - Smoothly variable between full detail, and reduced resolution representations of very large complexes

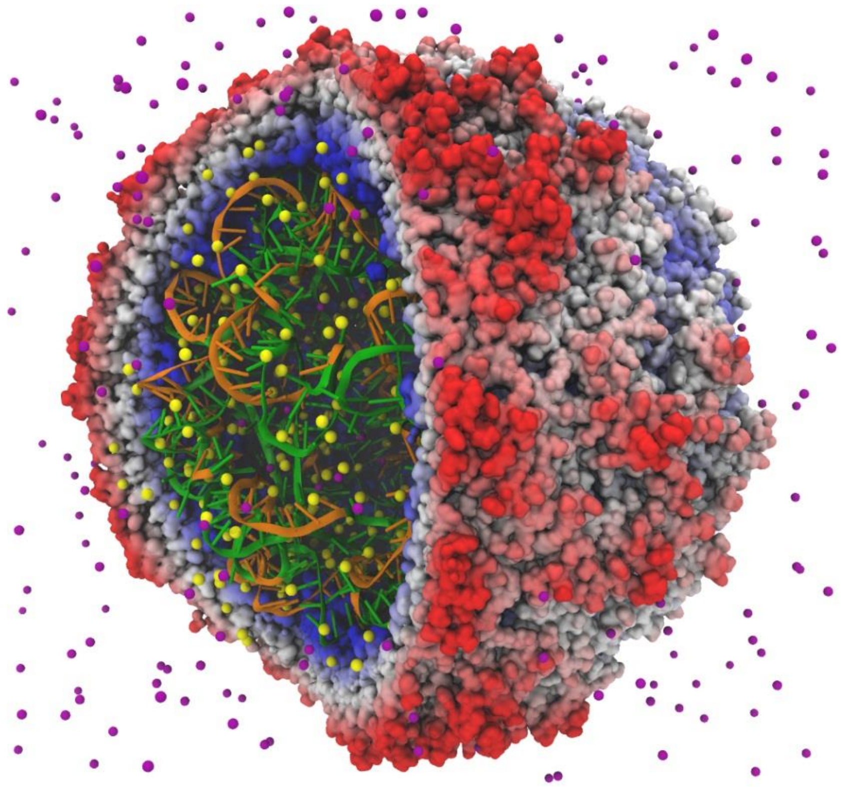


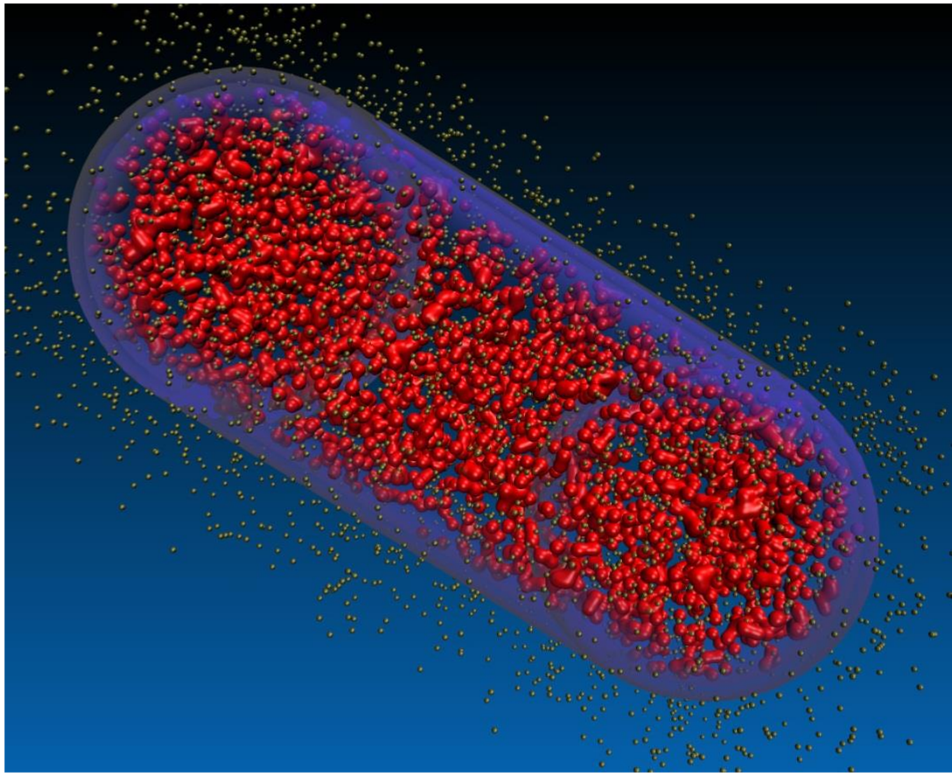
Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.

M. Krone, J. E. Stone, T. Ertl, K. Schulten. *EuroVis Short Papers*, pp. 67-71, 2012

VMD "QuickSurf" Representation

- Uses multi-core CPUs and GPU acceleration to enable **smooth real-time animation** of MD trajectories
- Linear-time algorithm, scales to millions of particles, as limited by memory capacity


Satellite Tobacco Mosaic Virus

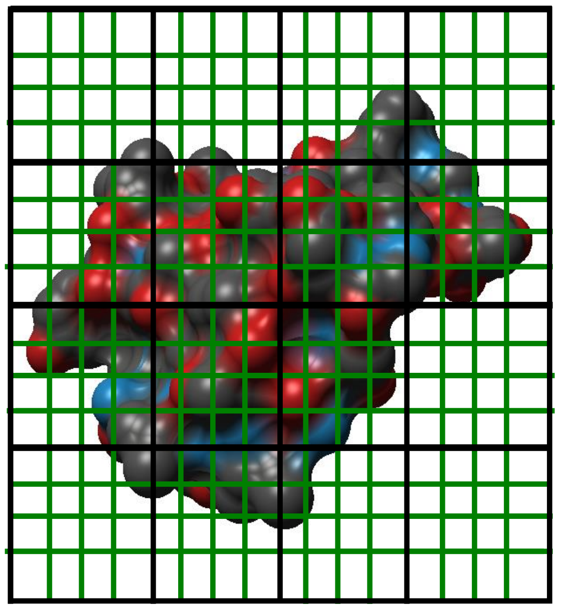

Lattice Cell Simulations

QuickSurf Algorithm Overview

- Build spatial acceleration data structures, optimize data for GPU
- Compute 3-D density map, 3-D volumetric texture map:

$$\rho(\vec{r};\vec{r}_1,\vec{r}_2,\ldots,\vec{r}_N)=\sum_{i=1}^{N} e^{\frac{-|\vec{r}-\vec{r}_i|^2}{2\alpha^2}}$$

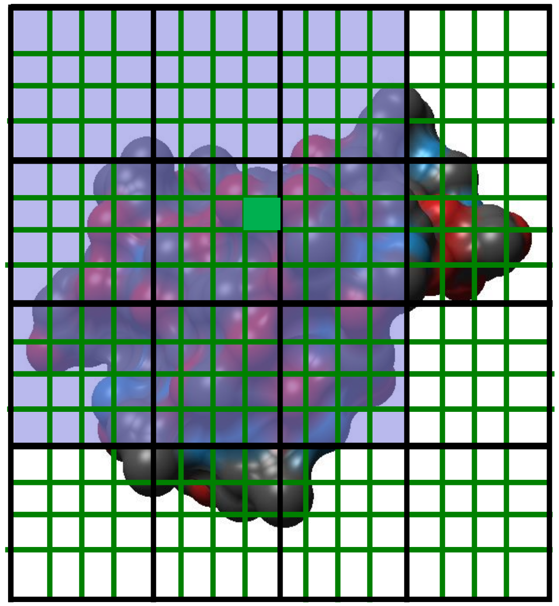
- Extract isosurface for a user-defined density value



3-D density map lattice, spatial acceleration grid, and extracted surface

QuickSurf Density Map Algorithm

- Spatial acceleration grid cells are sized to match the cutoff radius for the exponential, beyond which density contributions are negligible
- Density map lattice points computed by summing density contributions from particles in 3x3x3 grid of neighboring spatial acceleration cells
- Volumetric texture map is computed by summing particle colors normalized by their individual density contribution



3-D density map lattice point and the neighboring spatial acceleration cells it references

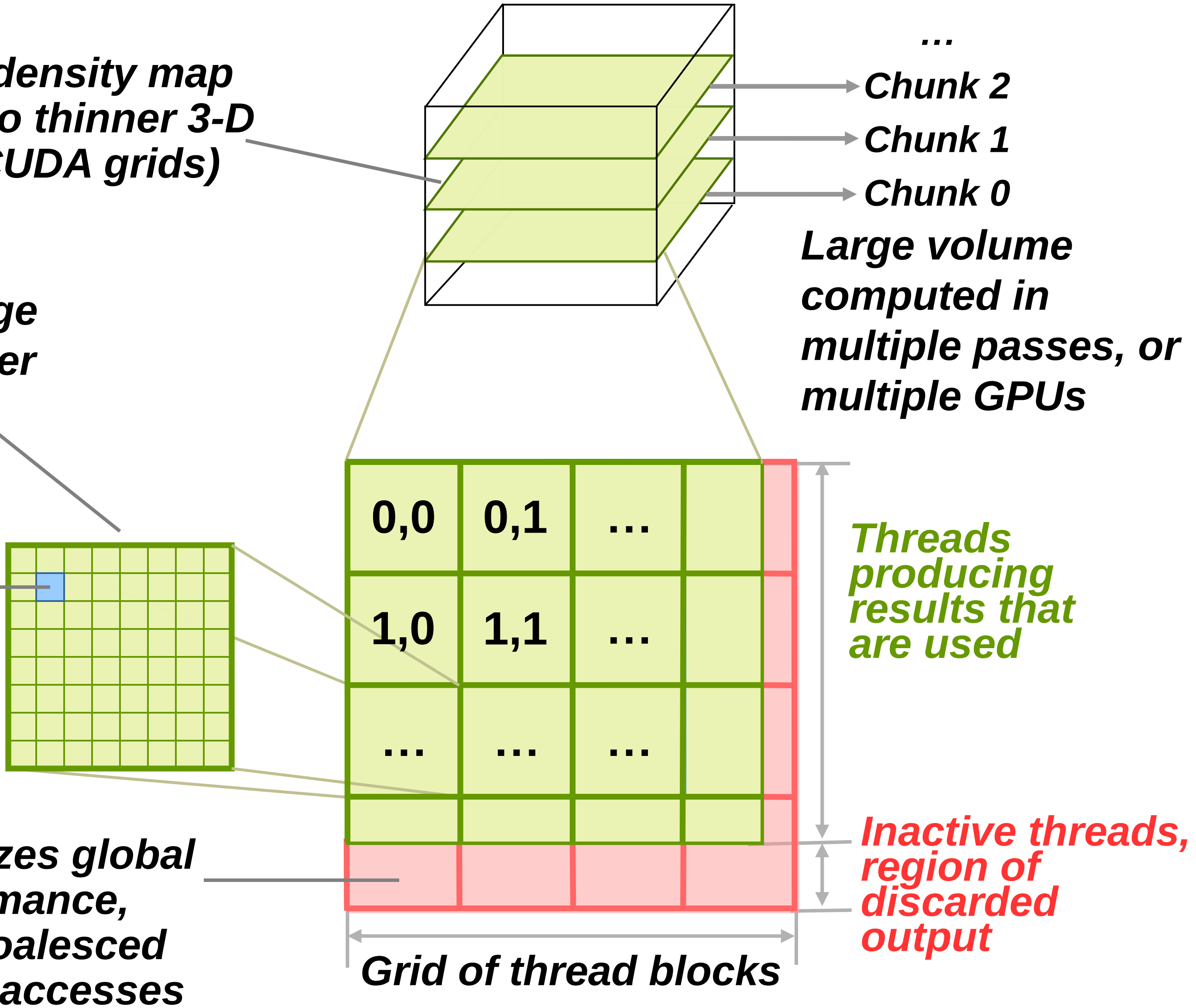
QuickSurf Density Parallel Decomposition

QuickSurf 3-D density map decomposes into thinner 3-D slabs/slices (CUDA grids)

Small 8x8 thread blocks afford large per-thread register count, shared memory

Each thread computes one or more density map lattice points

Padding optimizes global memory performance, guaranteeing coalesced global memory accesses

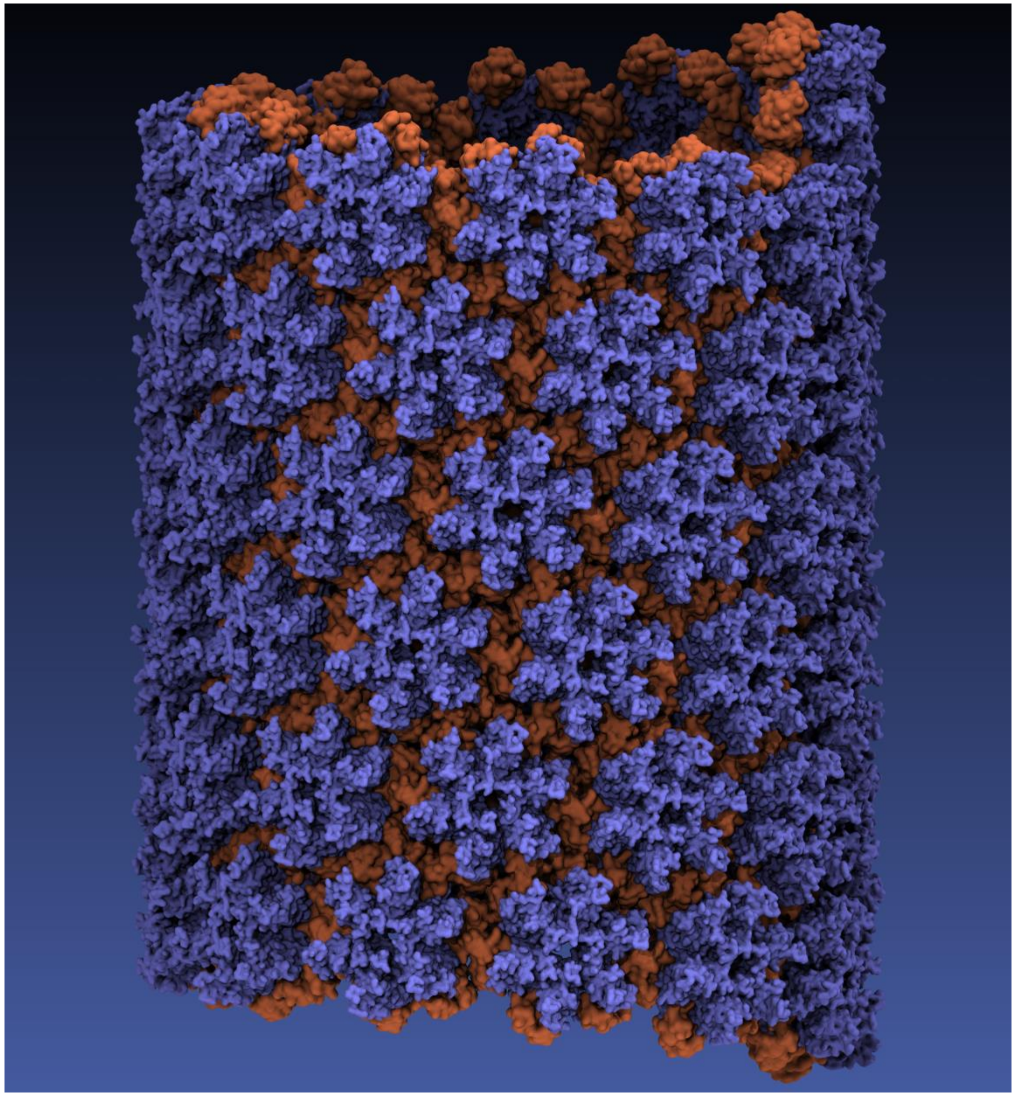


…
Chunk 2
Chunk 1
Chunk 0
0,0
0,1
…
1,0
1,1
…
…
…
…
Threads producing results that are used
Inactive threads, region of discarded output
Grid of thread blocks

Large volume computed in multiple passes, or multiple GPUs

Challenge: Support GPU-accelerated QuickSurf for **Large** Biomolecular Complexes

- Structures such as HIV initially needed all XK7 GPU memory to generate detailed surface renderings
- Goals and approach:
  - **Avoid slow CPU-fallback!**
  - Incrementally change algorithm phases to use more compact data types, while maintaining performance
  - Specialize code for different precision/performance/memory capacity cases



Supporting Multiple Data Types for QuickSurf Density Maps and Marching Cubes Vertex Arrays

- The major algorithm components of QuickSurf are now used for many other purposes:
  - Gaussian density map algorithm now used for MDFF Cryo EM density map fitting methods in addition to QuickSurf
  - Marching Cubes routines also used for Quantum Chemistry visualizations of molecular orbitals
- Rather than simply changing QuickSurf to use a particular internal numerical representation, it is desirable to instead use CUDA C++ templates to make type-generic versions of the key objects, kernels, and output vertex arrays
- Accuracy-sensitive algorithms use high-precision data types, performance and memory capacity sensitive cases use quantized or reduced precision approaches

Minimizing the Impact of Generality on QuickSurf Code Complexity

- A critical factor in the simplicity of supporting multiple QuickSurf data types arises from the so-called “gather” oriented algorithm we employ
  - Internally, **all in-register arithmetic is single-precision**
  - Compressed or reduced precision **data type conversions are performed on-the-fly as needed**
- Small inlined type conversion routines are defined for each of the cases we want to support
- Key QuickSurf kernels made type-generic using C++ template syntax, and the compiler **automatically** generates type-specific kernels as needed

Example Templated Density Map Kernel

```
template<class DENSITY, class VOLTEX>
__global__ static void
gaussdensity_fast_tex_norm(int natoms,
                           const float4 * RESTRICT sorted_xyzr,
                           const float4 * RESTRICT sorted_color,
                           int3 numvoxels,
                           int3 acncells,
                          float acgridspacing,
                          float invacgridspacing,
                          const uint2 * RESTRICT cellStartEnd,
                          float gridspacing, unsigned int z,
                          DENSITY * RESTRICT densitygrid,
                          VOLTEX * RESTRICT voltexmap,
                         float invisovalue) {
```

Example Templated Density Map Kernel

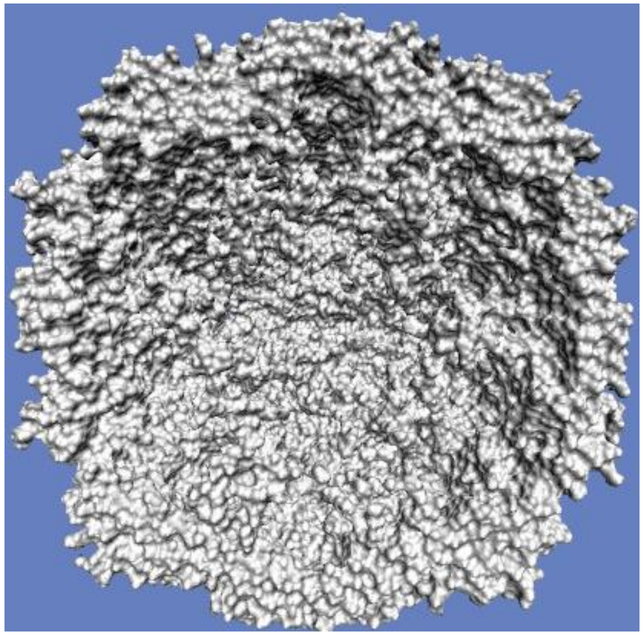
```
template<class DENSITY, class VOLTEX>
__global__ static void
gaussdensity_fast_tex_norm( …  ) {

  … Triple-nested and unrolled inner loops here …

  DENSITY densityout;
  VOLTEX texout;
  convert_density(densityout, densityval1);
  densitygrid[outaddr          ] = densityout;
  convert_color(texout, densitycol1);
  voltexmap[outaddr          ] = texout;
```

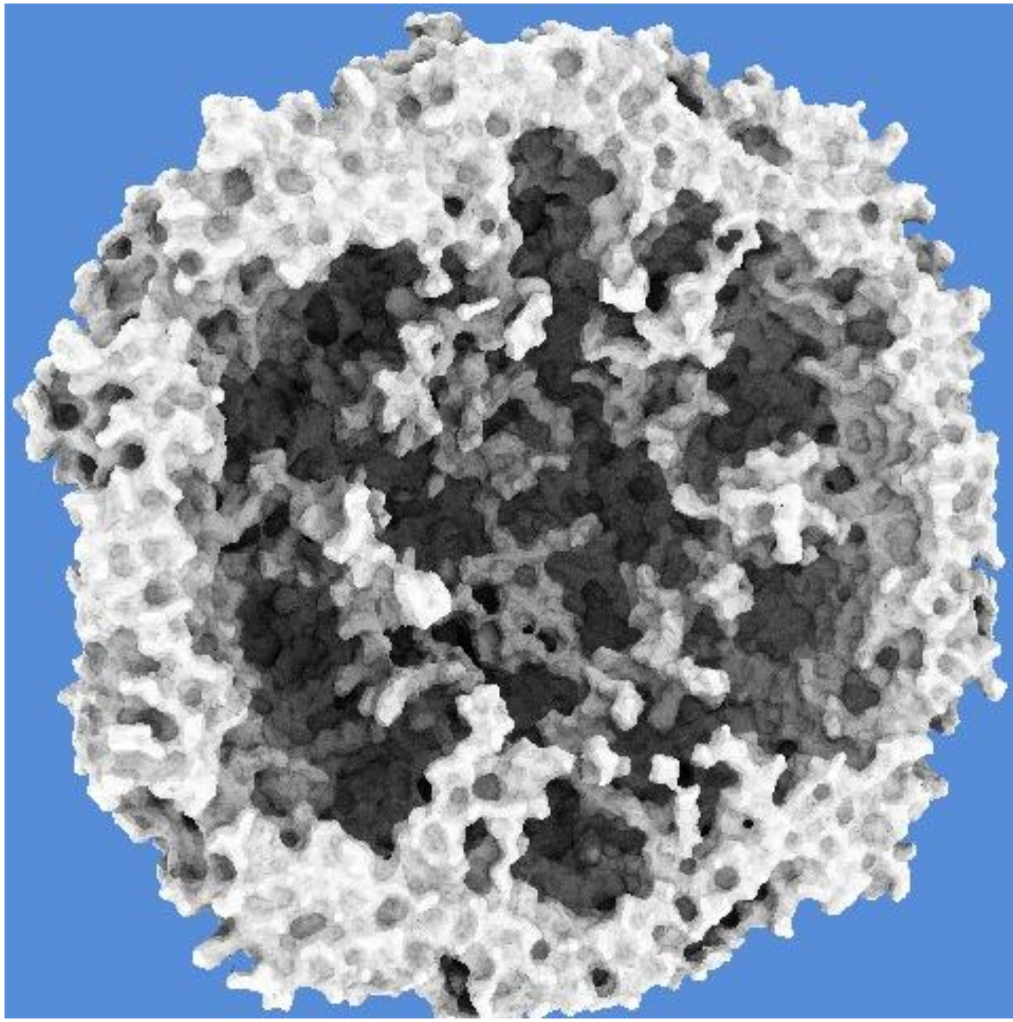
Net Result of QuickSurf Memory Efficiency Optimizations

- **Halved** overall GPU memory use
- Achieved **1.5x to 2x performance gain**:
  - The "gather" density map algorithm keeps type conversion operations out of the innermost loop
  - Density map global memory writes reduced to half
  - Multiple stages of Marching Cubes operate on smaller input and output data types
  - Same code path supports multiple precisions
- Users now get full GPU-accelerated QuickSurf in many cases that previously triggered CPU-fallback, all platforms (laptop/desk/super) benefit!

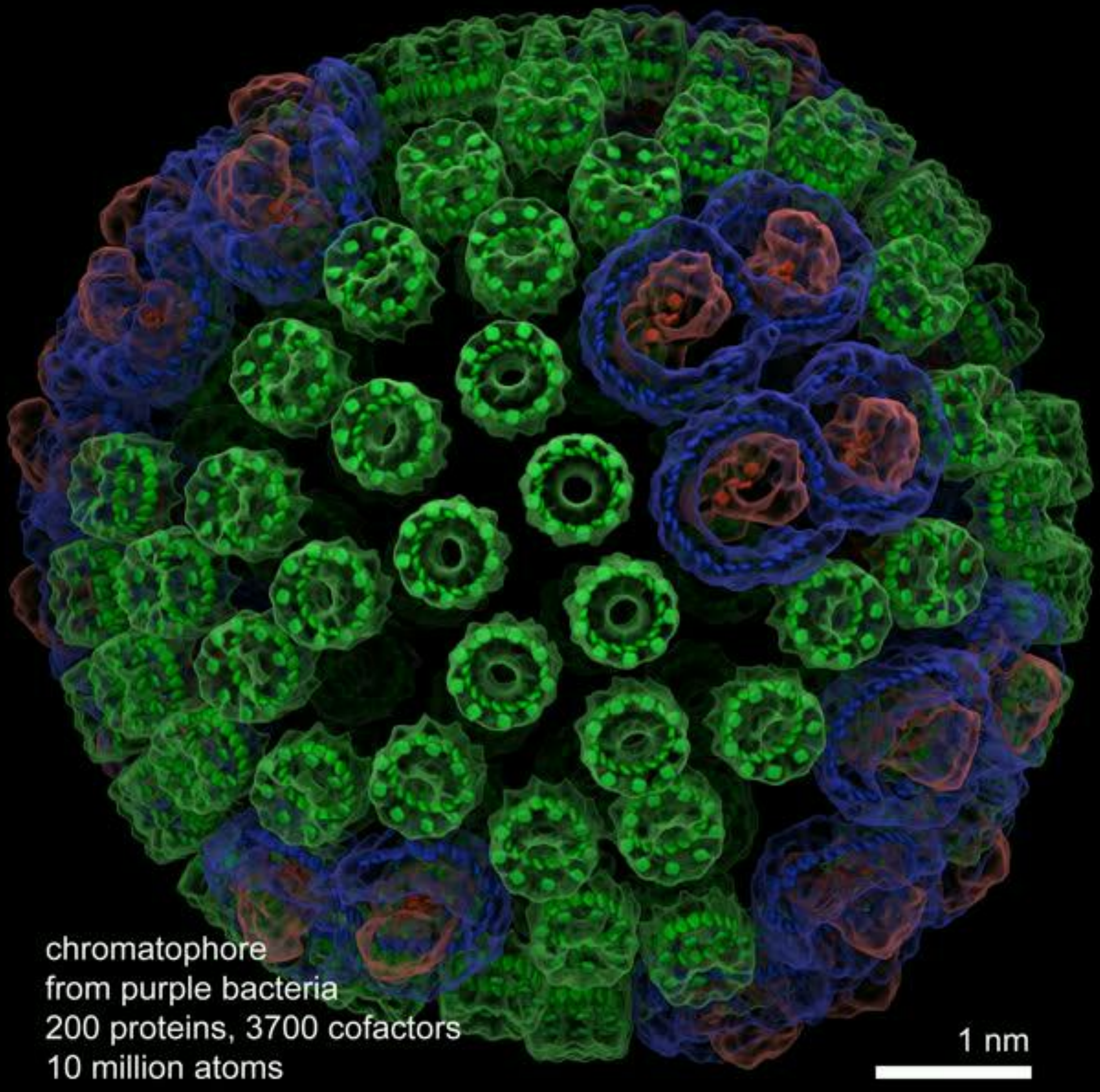
Ray Tracing Molecular Graphics

- Ambient occlusion lighting, shadows, reflections, transparency, and more…
- Satellite tobacco mosaic virus capsid w/ ~75K atoms


Standard OpenGL rasterization

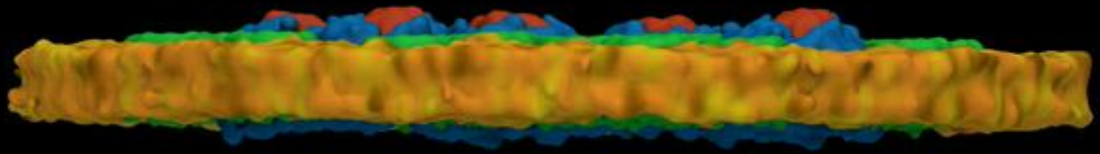

VMD/Tachyon/OptiX GPU ray tracing w/ ambient occlusion lighting

BW VMD/Tachyon Movie Generation



480 XE6 nodes for 85m @ 4096x2400

BW VMD/Tachyon Movie Generation



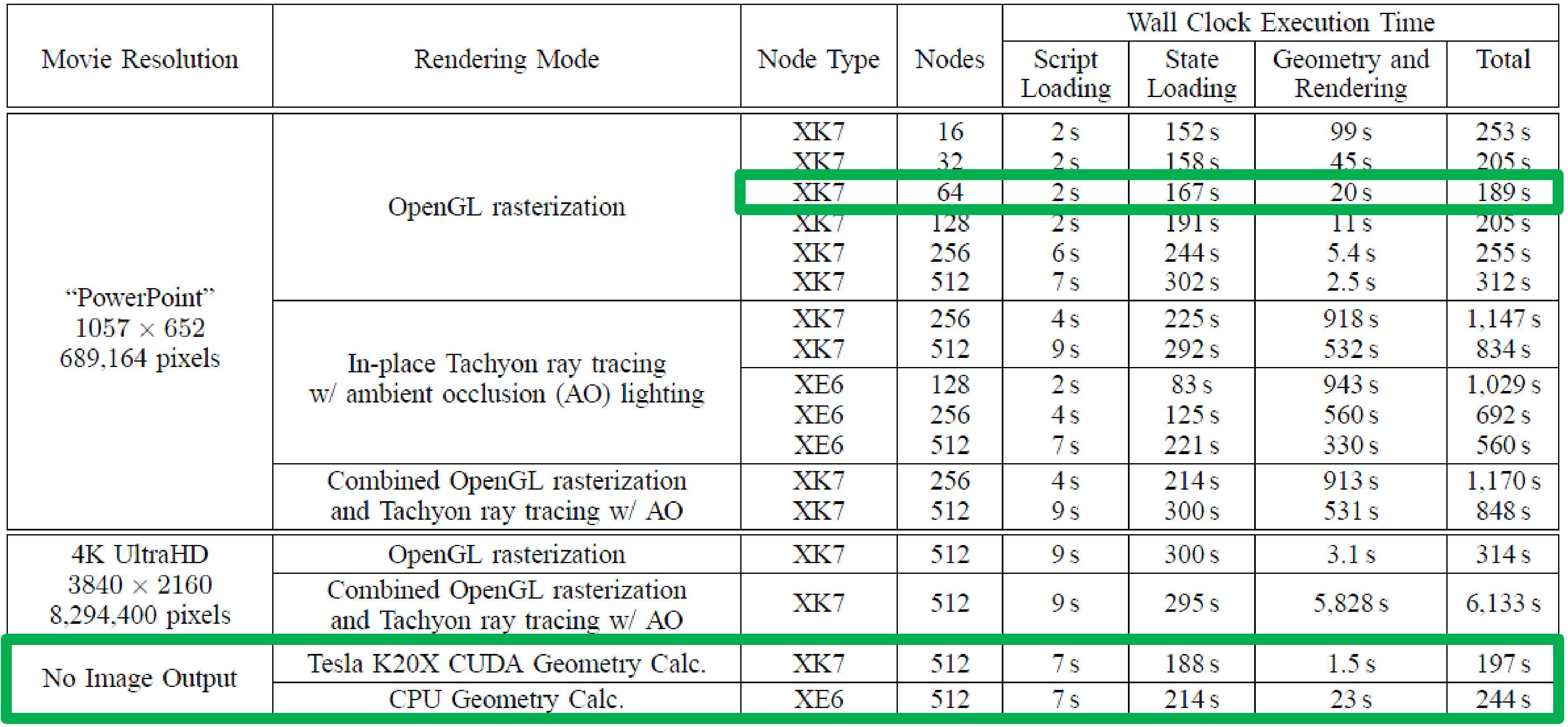
20 M atom chromatophore patch

360 XE6 nodes for 3h50m @ 4096x2400

Parallel Movie Rendering Results

- Unexpected I/O overhead from sourcing scripts!
- XK7 CUDA algorithms reduce per-frame surface and other geometry calculation times by a factor of ~15 vs. multithreaded SSE CPU code on XE6 nodes
- OpenGL rasterization is so fast it is essentially “free” – I/O time dominates OpenGL test cases currently... (XK7 partition had no I/O nodes)
- For CPU-only Tachyon, XE6 nodes render almost exactly 2x faster than XK7 nodes
- All test cases start to be penalized at >= 512 nodes due to increased I/O contention for common input files, reading of scripts, etc – need broadcast scheme for this data

VMD Movie Rendering on Blue Waters



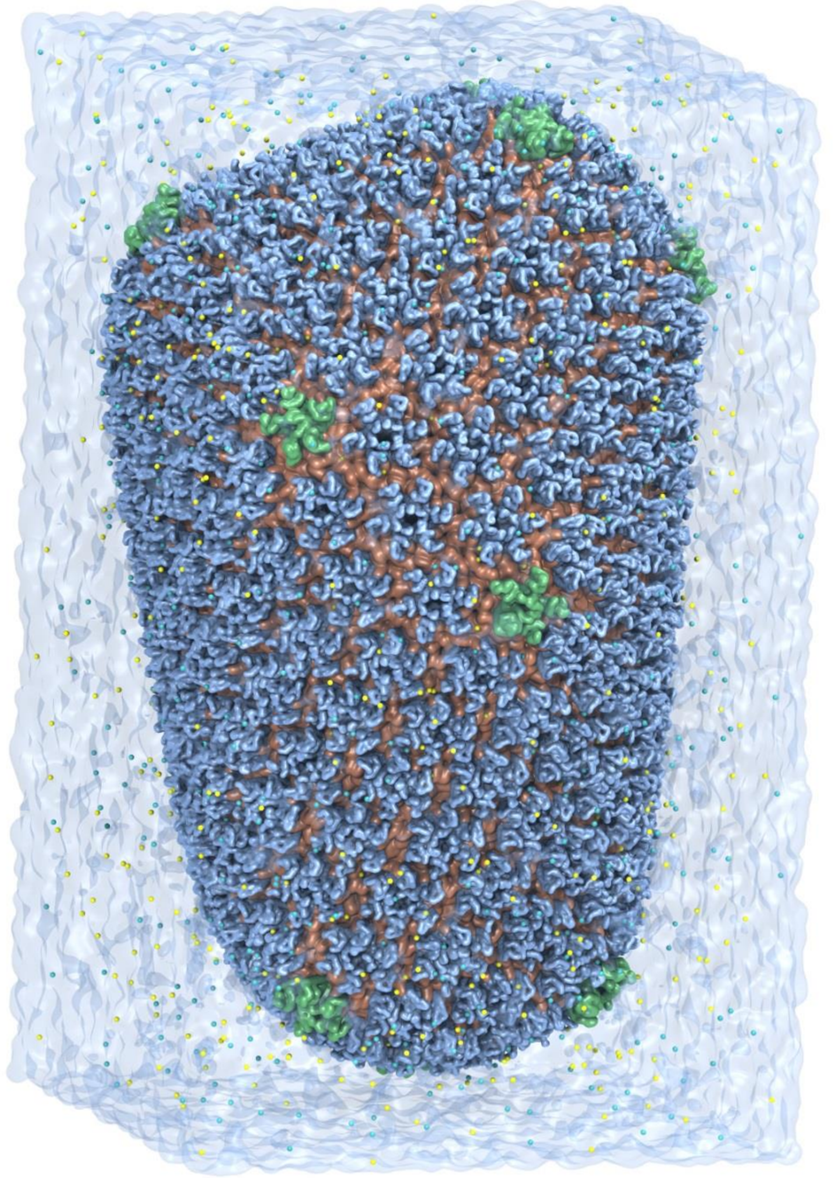
Movie Resolution	Rendering Mode	Node Type	Nodes	Wall Clock Execution Time			
				Script Loading	State Loading	Geometry and Rendering	Total
"PowerPoint" 1057 × 652 689,164 pixels	OpenGL rasterization	XK7	16	2 s	152 s	99 s	253 s
		XK7	32	2 s	158 s	45 s	205 s
		XK7	64	2 s	167 s	20 s	189 s
		XK7	128	2 s	191 s	11 s	205 s
		XK7	256	6 s	244 s	5.4 s	255 s
		XK7	512	7 s	302 s	2.5 s	312 s
	In-place Tachyon ray tracing w/ ambient occlusion (AO) lighting	XK7	256	4 s	225 s	918 s	1,147 s
		XK7	512	9 s	292 s	532 s	834 s
		XE6	128	2 s	83 s	943 s	1,029 s
		XE6	256	4 s	125 s	560 s	692 s
		XE6	512	7 s	221 s	330 s	560 s
	Combined OpenGL rasterization and Tachyon ray tracing w/ AO	XK7	256	4 s	214 s	913 s	1,170 s
		XK7	512	9 s	300 s	531 s	848 s
4K UltraHD 3840 × 2160 8,294,400 pixels	OpenGL rasterization	XK7	512	9 s	300 s	3.1 s	314 s
	Combined OpenGL rasterization and Tachyon ray tracing w/ AO	XK7	512	9 s	295 s	5,828 s	6,133 s
No Image Output	Tesla K20X CUDA Geometry Calc.	XK7	512	7 s	188 s	1.5 s	197 s
	CPU Geometry Calc.	XE6	512	7 s	214 s	23 s	244 s

TABLE II. VMD PARALLEL MOVIE RENDERING PERFORMANCE TESTS.

Early Experiences Scaling VMD Molecular Visualization and Analysis Jobs on Blue Waters. J. E. Stone, B. Isralewitz, and K. Schulten. In proceedings, Extreme Scaling Workshop, 2013. (In press)

GPU Ray Tracing of HIV-1 on Blue Waters

- Ambient occlusion lighting, shadows, transparency, antialiasing, depth cueing, 144 rays/pixel minimum
- 64 million atom virus simulation
- 1000+ movie frames
- Surface generation and ray tracing stages each use >= 75% of GPU memory



VMD HIV-1 Movie Ray Tracing on Blue Waters Cray XE6/XK7

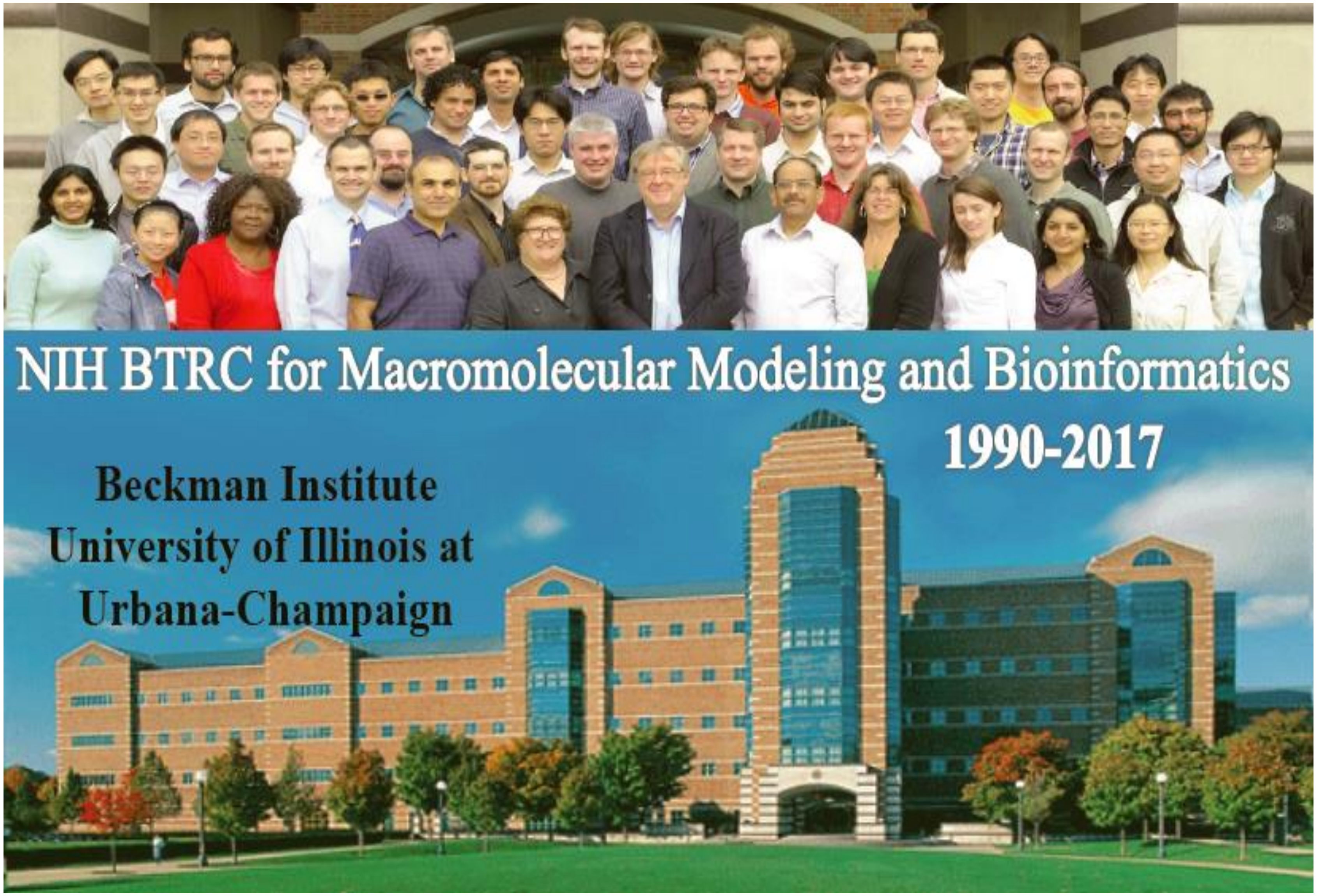
"HD" 1920x1080 rendering w/ Tachyon on XE6 vs. new "TachyonL-OptiX" on XK7 w/ K20 GPU:

Up to 8x geom+ray tracing speedup, 4x-5x overall speedup

Node Type and Count	**Script Load Time**	**State Load Time**	**Geometry + Ray Tracing**	**Total Time**
256 XE6	7 s	160 s	1374 s	1541 s
512 XE6	13 s	211 s	808 s	1032 s
64 XK7 Tesla K20X	2 s	38 s	655 s	695 s
128 XK7 Tesla K20X	4 s	74 s	331 s	410 s
256 XK7 Tesla K20X	7 s	110 s	171 s	288 s

Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- NCSA Blue Waters Team
- NVIDIA CUDA Center of Excellence, University of Illinois at Urbana-Champaign
- Many of the staff at NVIDIA and Cray
- Funding:
  - NSF OCI 07-25070
  - NSF PRAC “The Computational Microscope”
  - NIH support: 9P41GM104601, 5R01GM098243-02


NIH BTRC for Macromolecular Modeling and Bioinformatics
1990-2017
Beckman Institute
University of Illinois at
Urbana-Champaign

GPU Computing Publications

http://www.ks.uiuc.edu/Research/gpu/

- **Early Experiences Scaling VMD Molecular Visualization and Analysis Jobs on Blue Waters.** J. E. Stone, B. Isralewitz, and K. Schulten. In proceedings, Extreme Scaling Workshop, 2013. (In press)
- **Lattice Microbes: High-performance stochastic simulation method for the reaction-diffusion master equation.**
  E. Roberts, J. E. Stone, and Z. Luthey-Schulten.
  J. Computational Chemistry 34 (3), 245-255, 2013**.**
- **Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.** M. Krone, J. E. Stone, T. Ertl, and K. Schulten. *EuroVis Short Papers*, pp. 67-71, 2012.
- **Immersive Out-of-Core Visualization of Large-Size and Long-Timescale Molecular Dynamics Trajectories.** J. Stone, K. Vandivort, and K. Schulten. G. Bebis et al. (Eds.): *7th International Symposium on Visual Computing (ISVC 2011)*, LNCS 6939, pp. 1-12, 2011.
- **Fast Analysis of Molecular Dynamics Trajectories with Graphics Processing Units – Radial Distribution Functions.** B. Levine, J. Stone, and A. Kohlmeyer. *J. Comp. Physics*, 230(9):3556-3569, 2011.

GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/

- **Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters.** J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J. Stone, J Phillips. *International Conference on Green Computing*, pp. 317-324, 2010.
- **GPU-accelerated molecular modeling coming of age. J. Stone, D. Hardy, I. Ufimtsev, K. Schulten.** *J. Molecular Graphics and Modeling*, 29:116-125, 2010.
- **OpenCL: A Parallel Programming Standard for Heterogeneous Computing. J. Stone, D. Gohara, G. Shi.** *Computing in Science and Engineering*, 12(3):66-73, 2010.
- **An Asymmetric Distributed Shared Memory Model for Heterogeneous Computing Systems**. I. Gelado, J. Stone, J. Cabezas, S. Patel, N. Navarro, W. Hwu. *ASPLOS '10: Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 347-358, 2010.

GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/

- **GPU Clusters for High Performance Computing**. V. Kindratenko, J. Enos, G. Shi, M. Showerman, G. Arnold, J. Stone, J. Phillips, W. Hwu. *Workshop on Parallel Programming on Accelerator Clusters (PPAC),* In Proceedings IEEE Cluster 2009, pp. 1-8, Aug. 2009.
- **Long time-scale simulations of in vivo diffusion using GPU hardware**. E. Roberts, J. Stone, L. Sepulveda, W. Hwu, Z. Luthey-Schulten. In *IPDPS'09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Computing*, pp. 1-8, 2009.
- **High Performance Computation and Interactive Display of Molecular Orbitals on GPUs and Multi-core CPUs**. J. Stone, J. Saam, D. Hardy, K. Vandivort, W. Hwu, K. Schulten, *2nd Workshop on General-Purpose Computation on Graphics Pricessing Units (GPGPU-2), ACM International Conference Proceeding Series*, volume 383, pp. 9-18, 2009.
- **Probing Biomolecular Machines with Graphics Processors**. J. Phillips, J. Stone. *Communications of the ACM*, 52(10):34-41, 2009.
- **Multilevel summation of electrostatic potentials using graphics processing units**. D. Hardy, J. Stone, K. Schulten. *J. Parallel Computing*, 35:164-177, 2009.

GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/

- **Adapting a message-driven parallel application to GPU-accelerated clusters**. J. Phillips, J. Stone, K. Schulten. *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, IEEE Press, 2008.
- **GPU acceleration of cutoff pair potentials for molecular modeling applications**. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, and W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.
- **GPU computing**. J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips. *Proceedings of the IEEE*, 96:879-899, 2008.
- **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.
- **Continuous fluorescence microphotolysis and correlation spectroscopy**. A. Arkhipov, J. Hüve, M. Kahms, R. Peters, K. Schulten. *Biophysical Journal*, 93:4006-4017, 2007.