Broadening the Use of Scalable Kernels in NAMD/VMD

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/

VSCSE 2012

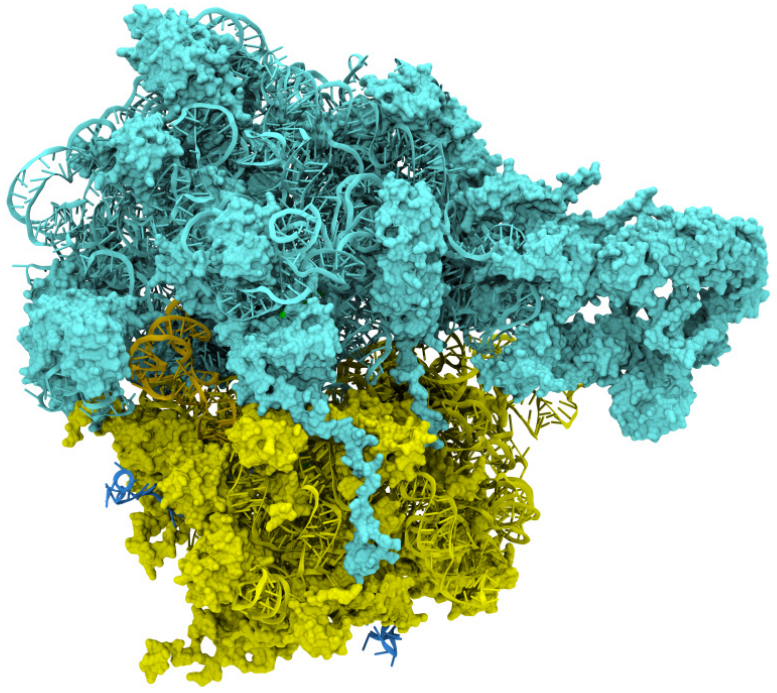
National Center for Supercomputing Applications,

Urbana, IL, August 16, 2012

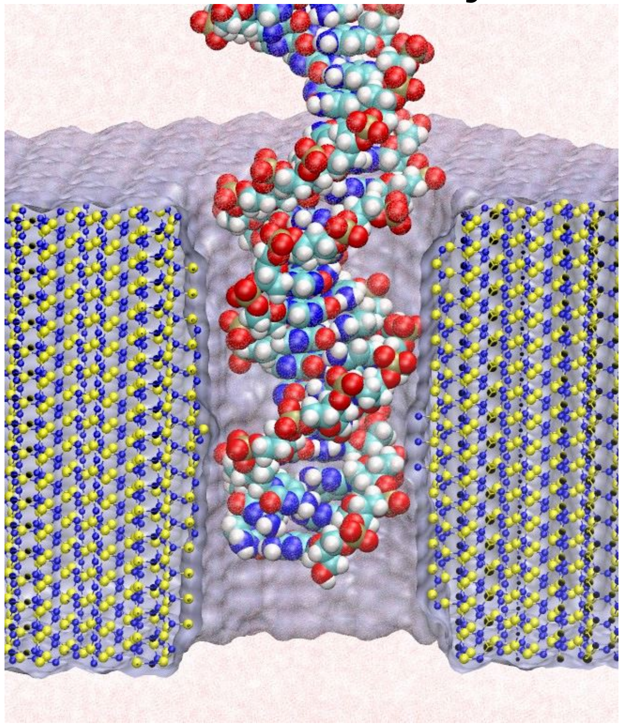
Goal: A Computational Microscope

- Study the molecular machines in living cells

Ribosome: synthesizes proteins from genetic information, target for antibiotics

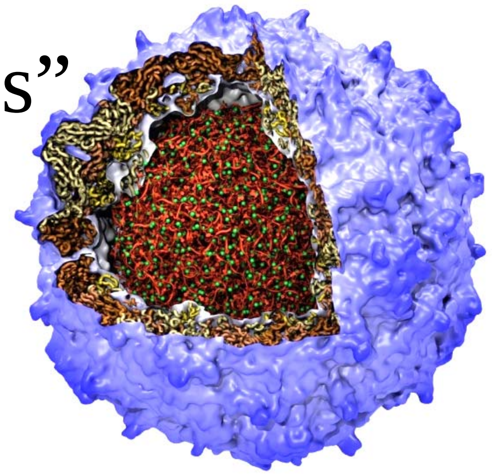


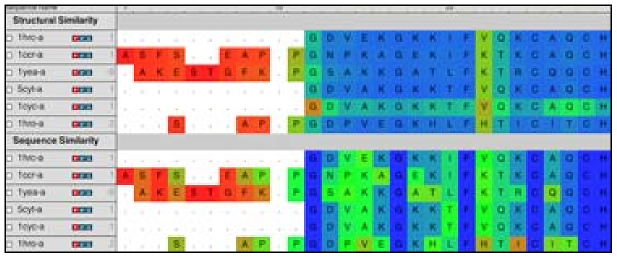
Silicon nanopore: bionanodevice for sequencing DNA efficiently



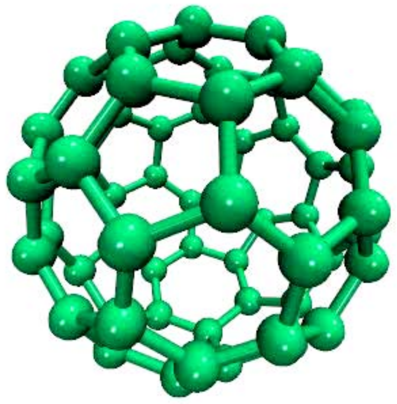
VMD – “Visual Molecular Dynamics”

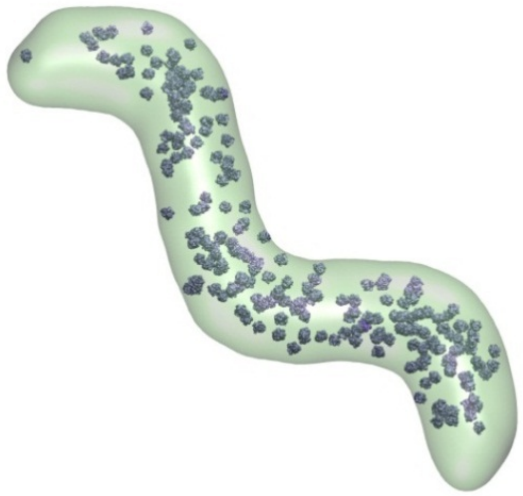
- Visualization and analysis of:
  - molecular dynamics simulations
  - quantum chemistry calculations
  - particle systems and whole cells
  - sequence data
- User extensible w/ scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/

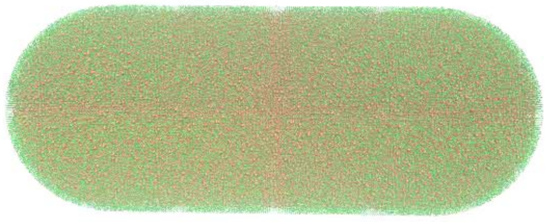

Poliovirus



Ribosome Sequences


Electrons in
Vibrating Buckyball

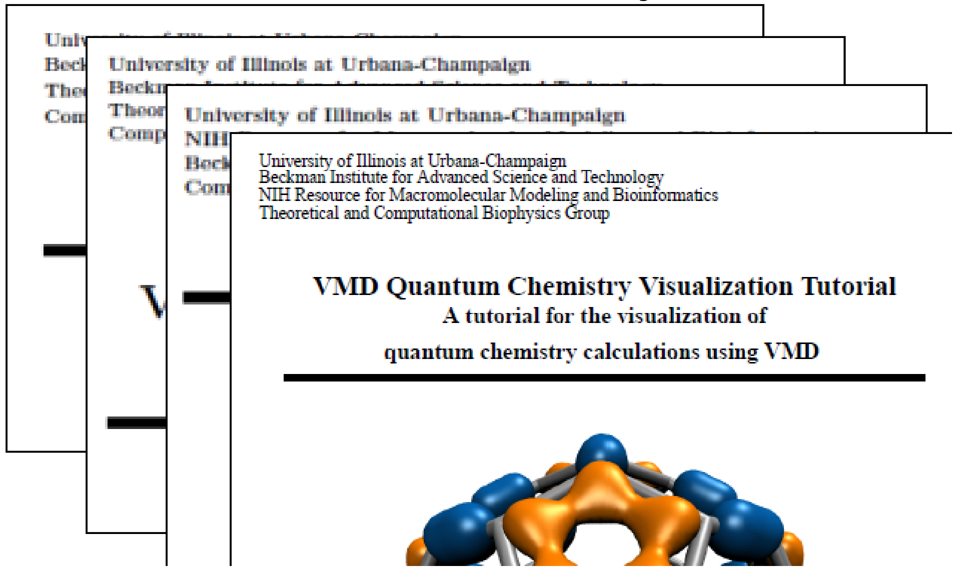

Cellular Tomography,
Cryo-electron Microscopy


Whole Cell Simulations

Meeting the Diverse Needs of the Molecular Modeling Community

- Over 212,000 registered users
  - 18% (39,000) are NIH-funded
  - Over 49,000 have downloaded multiple VMD releases
- Over 6,600 citations
- User community runs VMD on:
  - MacOS X, Unix, Windows operating systems
  - Laptops, desktop workstations
  - Clusters, supercomputers

- VMD user support and service efforts:
  - 20,000 emails, 2007-2011
  - Develop and maintain VMD tutorials and topical mini-tutorials; 11 in total
  - Periodic user surveys



University of Illinois at Urbana-Champaign

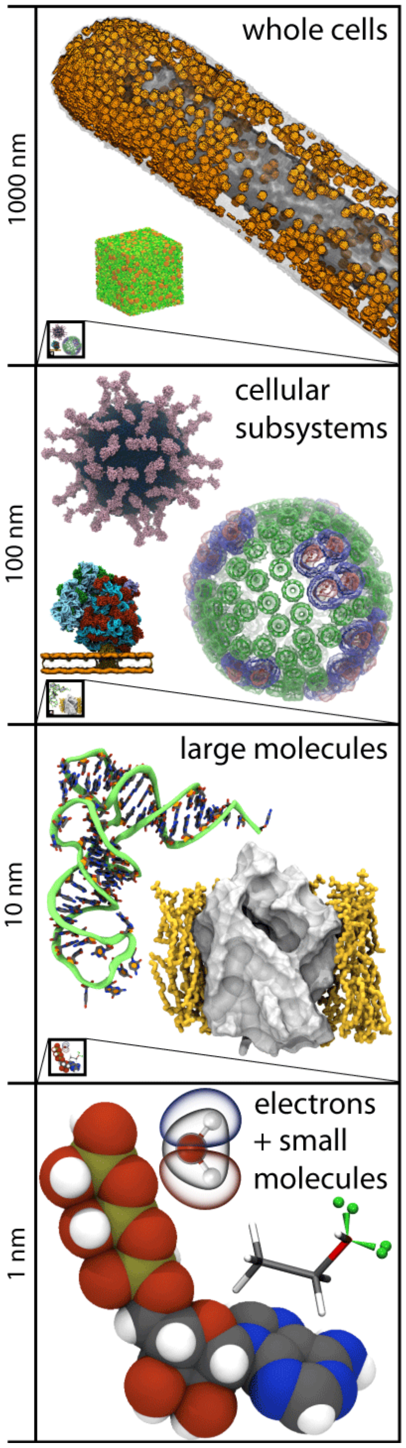
University of Illinois at Urbana-Champaign

University of Illinois at Urbana-Champaign
Beckman Institute for Advanced Science and Technology
NIH Resource for Macromolecular Modeling and Bioinformatics
Theoretical and Computational Biophysics Group

VMD Quantum Chemistry Visualization Tutorial
A tutorial for the visualization of
quantum chemistry calculations using VMD

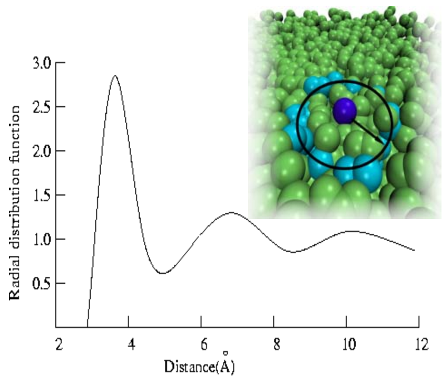
VMD Interoperability – Linked to Today's Key Research Areas

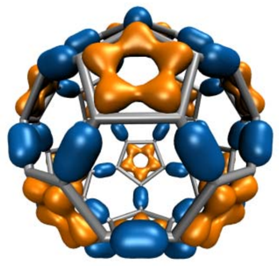
- Unique in its interoperability with a broad range of modeling tools: AMBER, CHARMM, CPMD, DL_POLY, GAMESS, GROMACS, HOOMD, LAMMPS, NAMD, and many more …
- Supports key data types, file formats, and databases, e.g. electron microscopy, quantum chemistry, MD trajectories, sequence alignments, super resolution light microscopy
- Incorporates tools for simulation preparation, visualization, and analysis



GPU Accelerated Trajectory Analysis and Visualization in VMD

GPU-Accelerated Feature	**Speedup vs. single CPU core**
Molecular orbital display	120x
Radial distribution function	92x
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





Ongoing VMD GPU Development

- Development of new CUDA kernels for common molecular dynamics trajectory analysis tasks, faster surface renderings, and more…
- Support for CUDA in MPI-enabled builds of VMD for analysis runs
  - GPU accelerated commodity clusters
  - **GPU-accelerated Cray XK6/XK7 supercomputers: NCSA Blue Waters, ORNL Titan**
- Updating existing CUDA kernels to take advantage of new hardware features on the new NVIDIA "Kepler" GPUs

Molecular Visualization and Analysis Challenges for Petascale Simulations

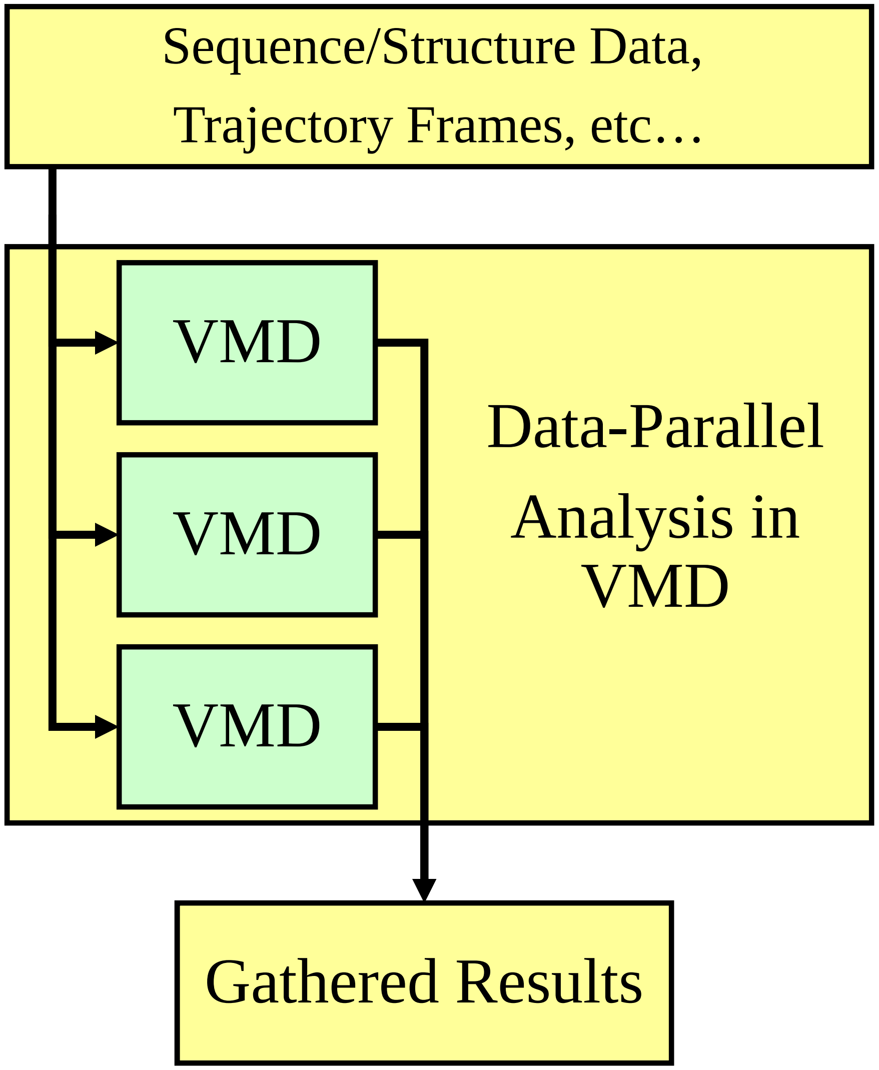
- Very large structures (10M to over 100M atoms)
  - 12-bytes per atom per trajectory frame
  - One 100M atom trajectory frame: 1200MB!
- Long-timescale simulations produce huge trajectories
  - MD integration timesteps are on the femtosecond timescale (10^{-15} sec) but many important biological processes occur on microsecond to millisecond timescales
  - Even storing trajectory timesteps infrequently, resulting trajectories frequently contain millions of frames
- **Terabytes to petabytes of data, often too large to move**
- **Viz and analysis must be done primarily on the supercomputer where the data already resides**

Approaches for Visualization and Analysis of Petascale Molecular Simulations with VMD

- Abandon conventional approaches, e.g. bulk download of trajectory data to remote viz/analysis machines
  - In-place processing of trajectories on the machine running the simulations
  - Use remote visualization techniques: Split-mode VMD with remote front-end instance, and back-end viz/analysis engine running in parallel on supercomputer
- Large-scale parallel analysis and visualization via distributed memory MPI version of VMD
- Exploit GPUs to increase per-node analytical capabilities , e.g. NCSA Blue Waters Cray XK6/XK7

Parallel VMD Analysis w/ MPI

- Analyze trajectory frames, structures, or sequences in parallel supercomputers:
  - Parallelize user-written analysis scripts with minimum difficulty
  - Parallel analysis of independent trajectory frames
  - Parallel structural analysis using custom parallel reductions
  - Parallel rendering, movie making
- Dynamic load balancing:
  - Tested with 15,360 CPU cores on Blue Waters Early Science System
- **Supports GPU-accelerated clusters and supercomputers**



Quantifying GPU Performance and Energy Efficiency in HPC Clusters

- NCSA "AC" Cluster
- Power monitoring hardware on one node and its attached Tesla S1070 (4 GPUs)
- Power monitoring logs recorded separately for host node and attached GPUs
- Logs associated with batch job IDs

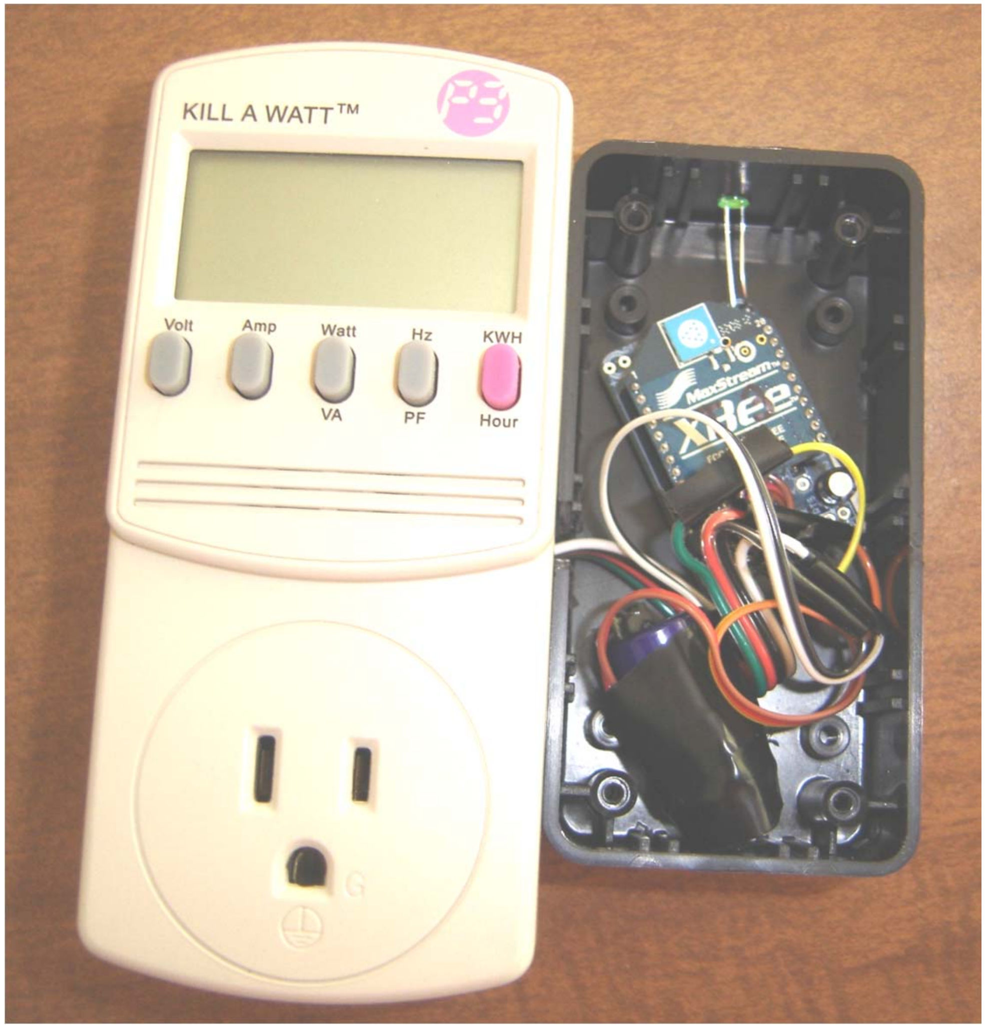
•32 HP XW9400 nodes

•128 cores, 128 Tesla C1060 GPUs

•QDR Infiniband

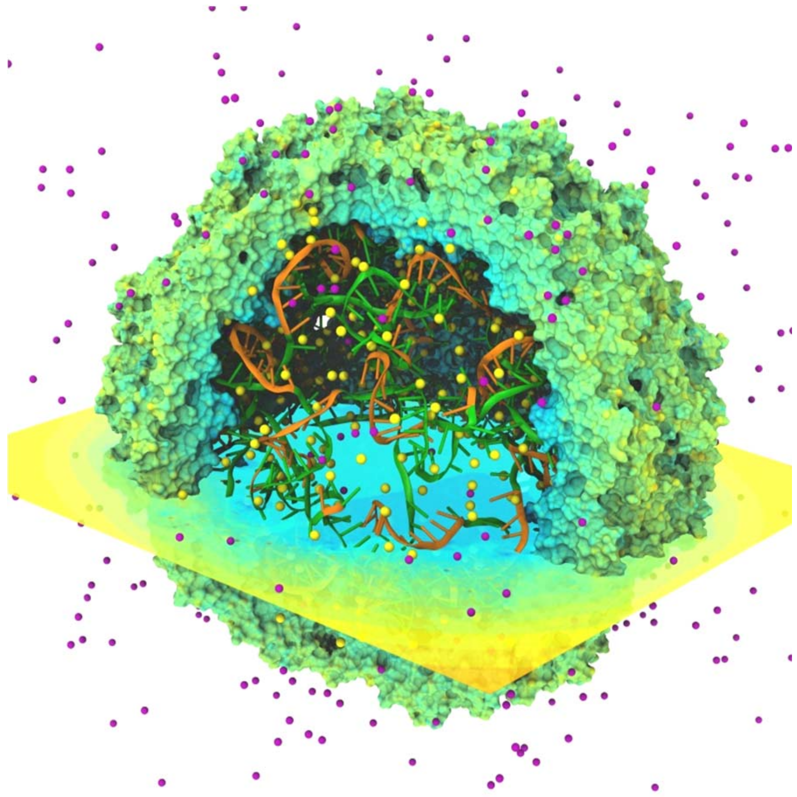
Tweet-a-Watt

- Kill-a-watt power meter
- Xbee wireless transmitter
- Power, voltage, shunt sensing tapped from op amp
- Lower transmit rate to smooth power through large capacitor
- Readout software upload samples to local database
- We built 3 transmitter units and one Xbee receiver
- **Currently integrated into AC cluster as power monitor**



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: 299 watts
  - CPUs+GPUs: 742 watts
- 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.

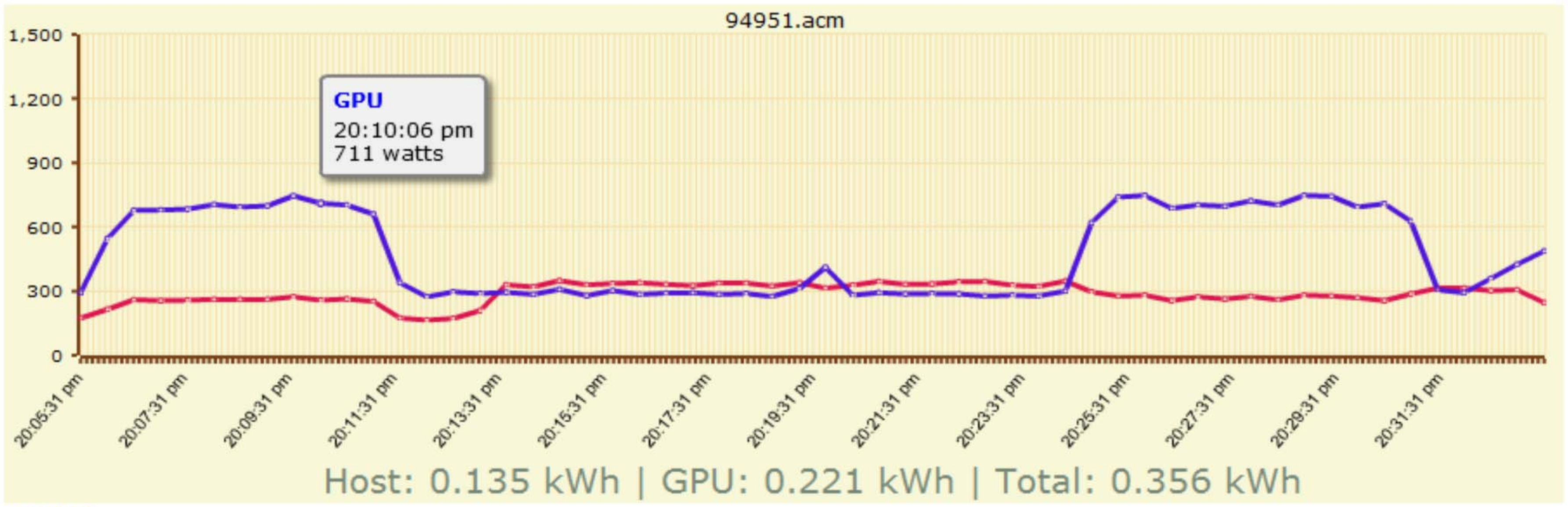
AC Cluster GPU Performance and Power Efficiency Results

Application	GPU speedup	Host watts	Host+GPU watts	Perf/watt gain
NAMD	6	316	681	2.8
VMD	25	299	742	10.5
MILC	20	225	555	8.1
QMCPACK	61	314	853	22.6

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.

Power Profiling: Example Log

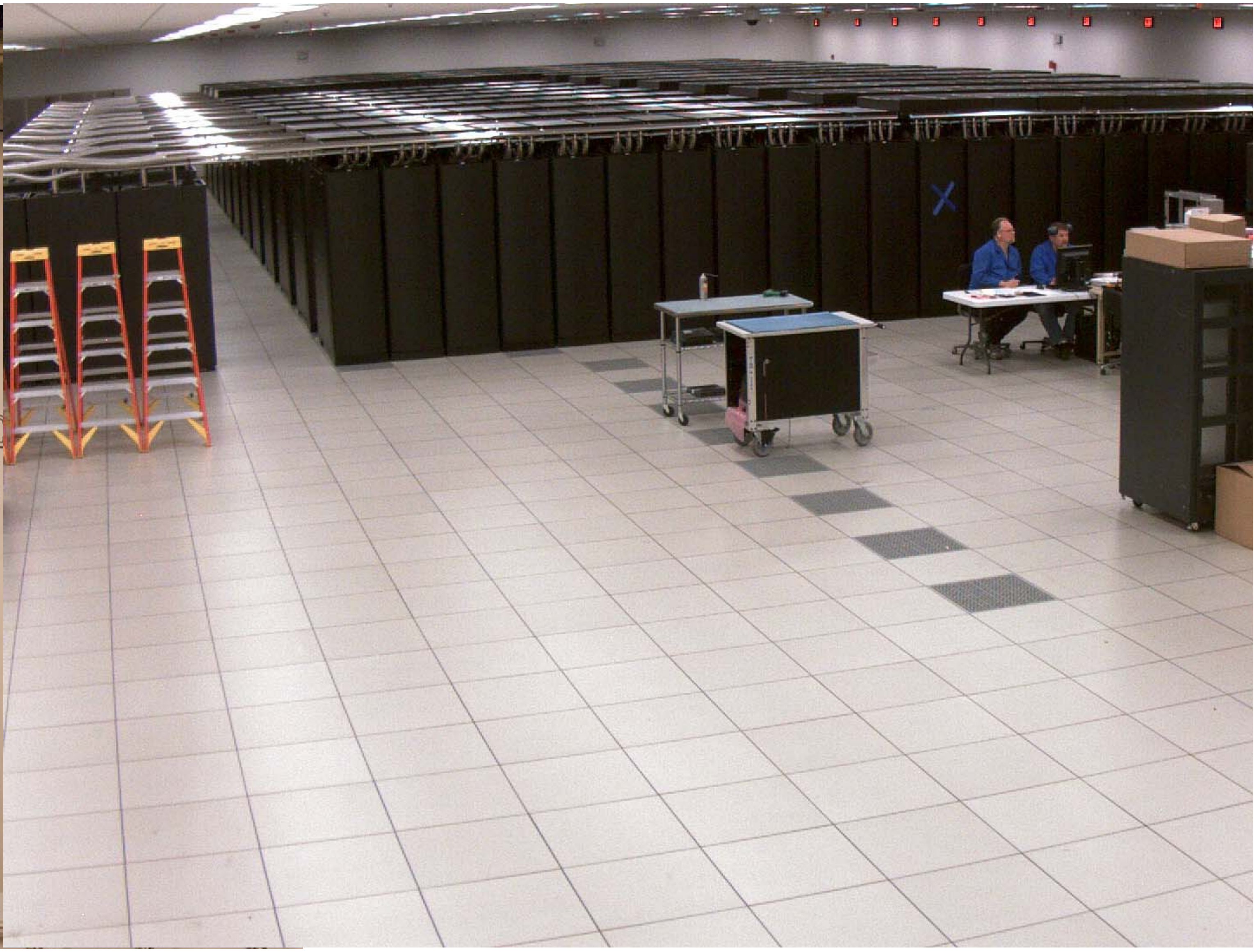
AC Power Utilization



JSON Data

- Mouse-over value displays
- Under curve totals displayed
- If there is user interest, we may support calls to add custom tags from application

NCSA Blue Waters Early Science System
Cray XK6 nodes w/ NVIDIA Tesla X2090 GPUs



Time-Averaged Electrostatics Analysis on NCSA Blue Waters Early Science System

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	**GPU nodes are 4.15x faster overall**

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

Early Experiences with Kepler

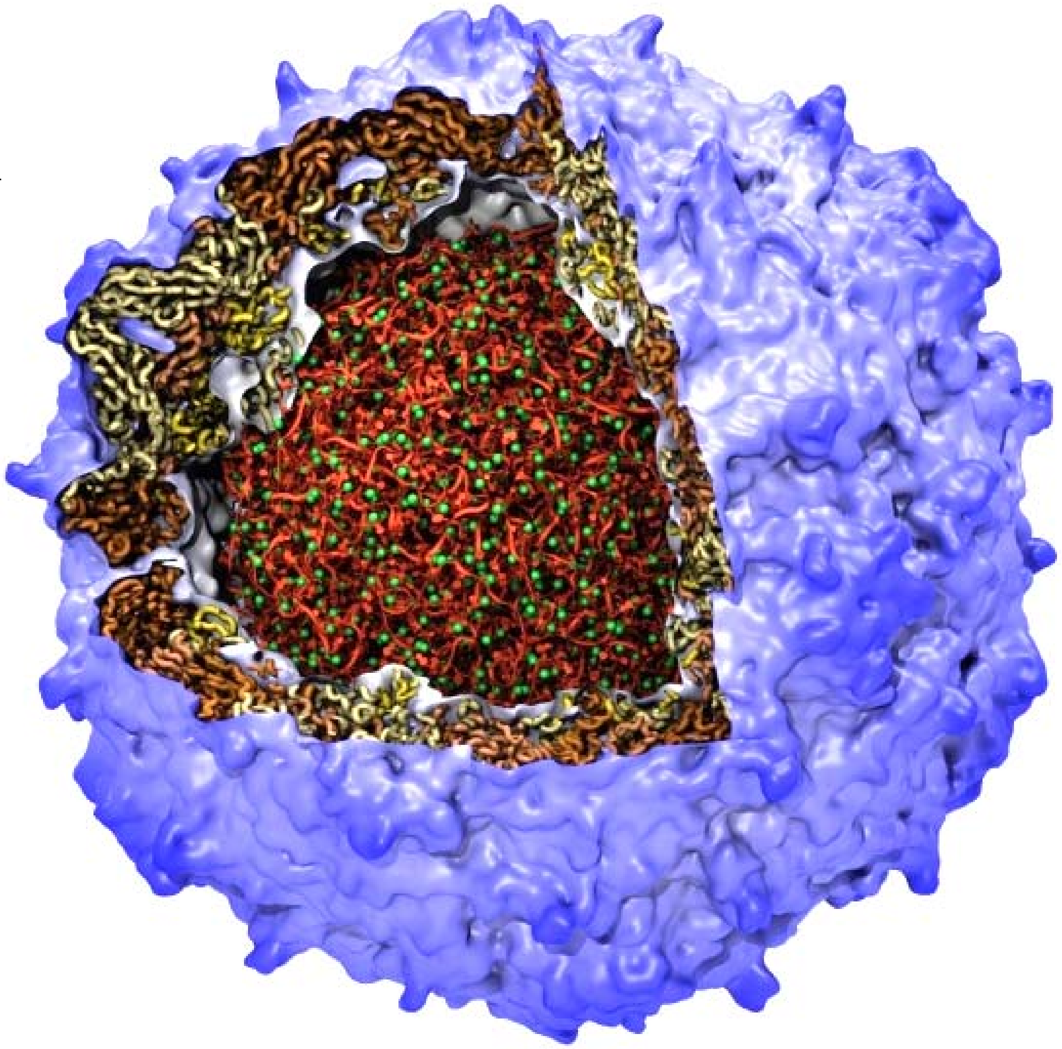
Preliminary Observations

- **Arithmetic is cheap, memory references are costly** (trend is certain to continue & intensify…)
- Different performance ratios for registers, shared mem, and various floating point operations vs. Fermi
- Kepler GK104 (e.g. GeForce 680) brings improved performance for some special functions vs. Fermi:

CUDA Kernel	**Dominant Arithmetic Operations**	**Kepler (GeForce 680) Speedup vs. Fermi (Quadro 7000)**
Direct Coulomb summation	rsqrtf()	2.4x
Molecular orbital grid evaluation	expf(), exp2f(), Multiply-Add	1.7x

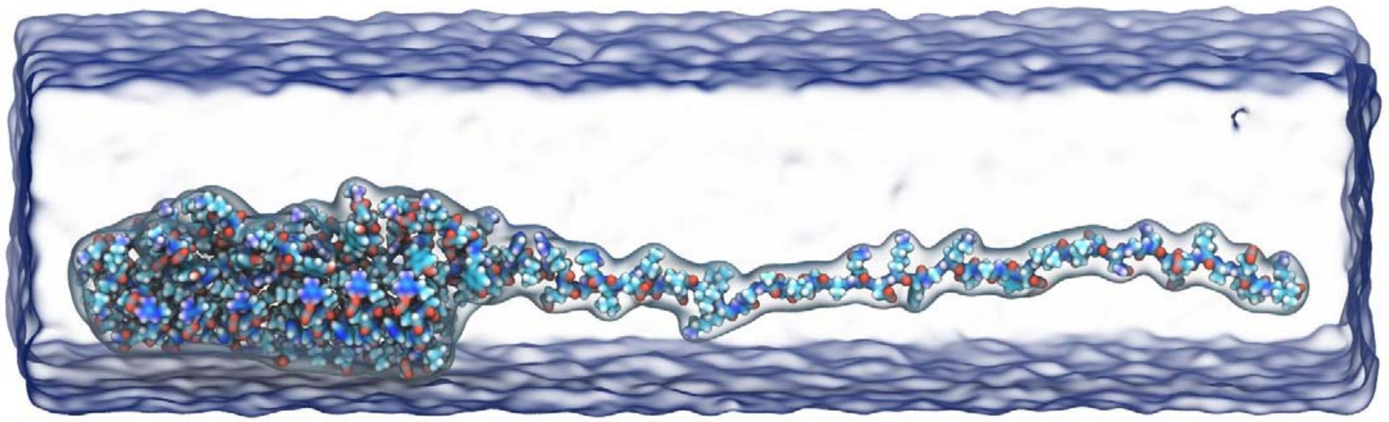
Molecular Surface Visualization

- Large biomolecular complexes are difficult to interpret with atomic detail graphical representations
- Even secondary structure representations become cluttered
- Surface representations are easier to use when greater abstraction is desired, but are computationally costly
- Existing surface display methods incapable of animating dynamics of large structures


Poliovirus

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

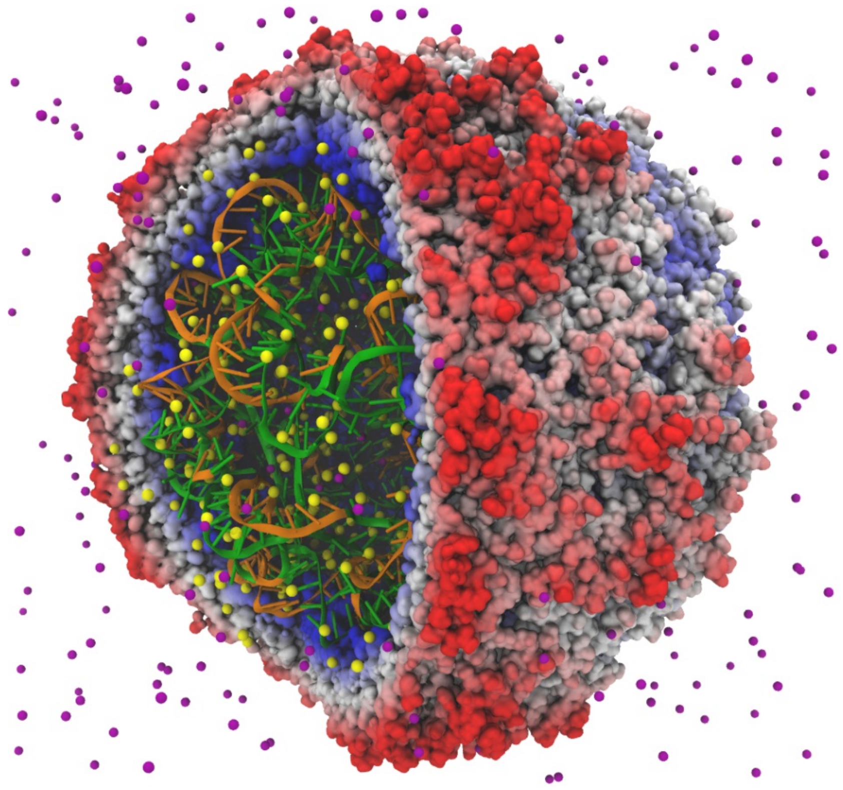


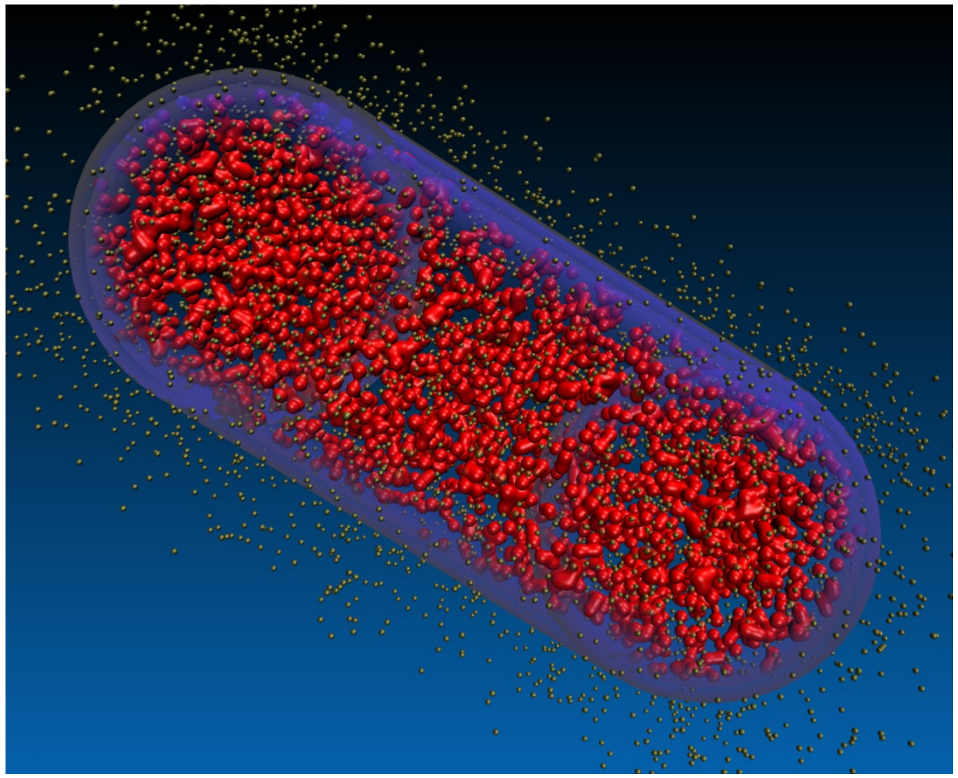
Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.

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

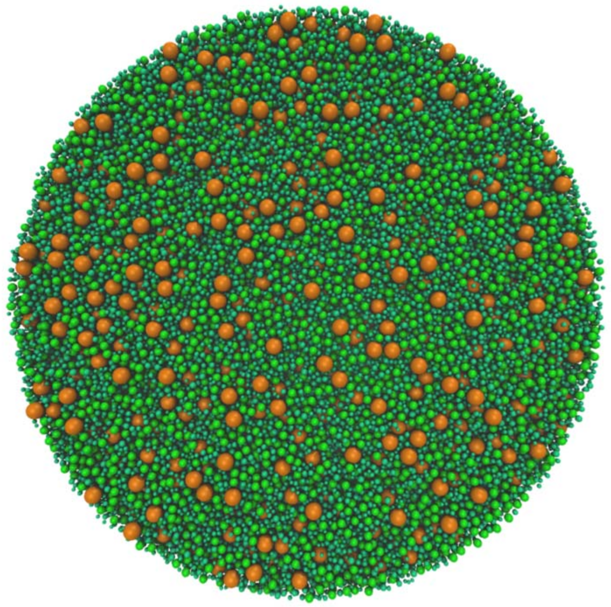
VMD “QuickSurf” Representation

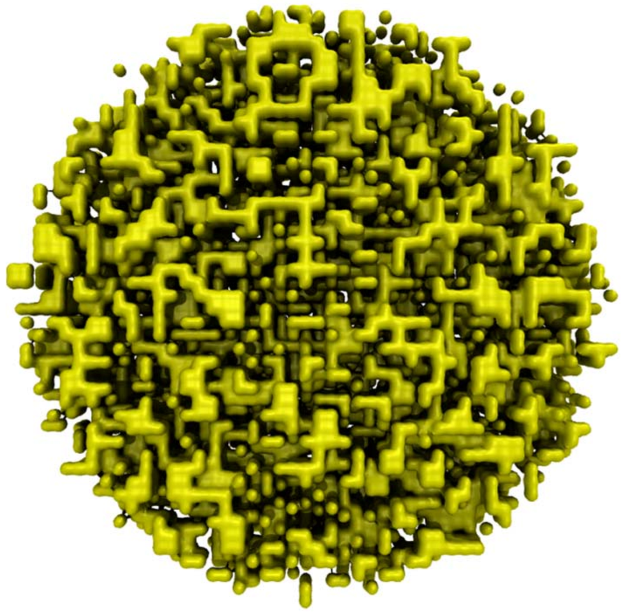
- Uses multi-core CPUs and GPU acceleration to enable **smooth real-time animation** of MD trajectories
- Linear-time algorithm, scales to millions of particles, as limited by memory capacity

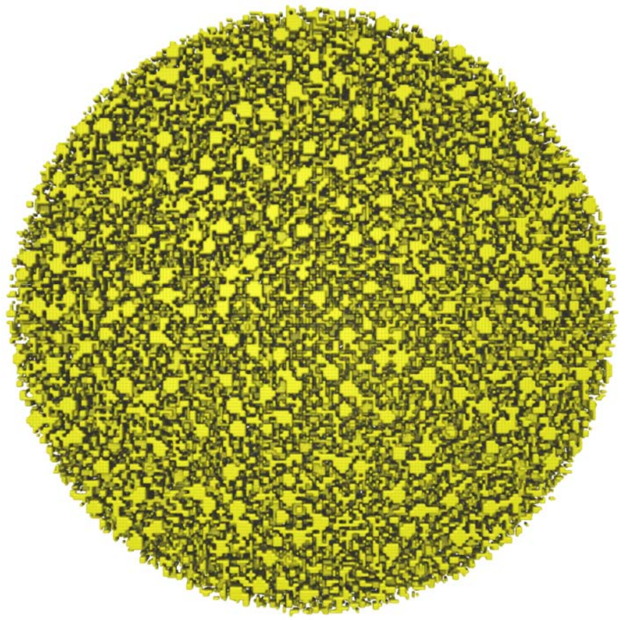

Satellite Tobacco Mosaic Virus


Lattice Cell Simulations

QuickSurf Representation of Lattice Cell Models







Continuous particle based model – often 70 to 300 million particles

Discretized lattice models derived from continuous model shown in a surface representation

Lattice Microbes: High-Performance Stochastic Simulation Method for the Reaction-Diffusion Master Equation.

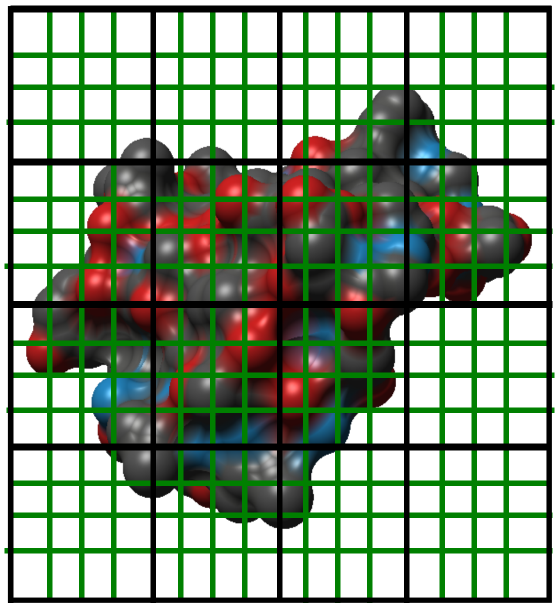
Elijah Roberts, John E. Stone, Zaida Luthey-Schulten. 2012. (Submitted)

QuickSurf Algorithm Overview

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

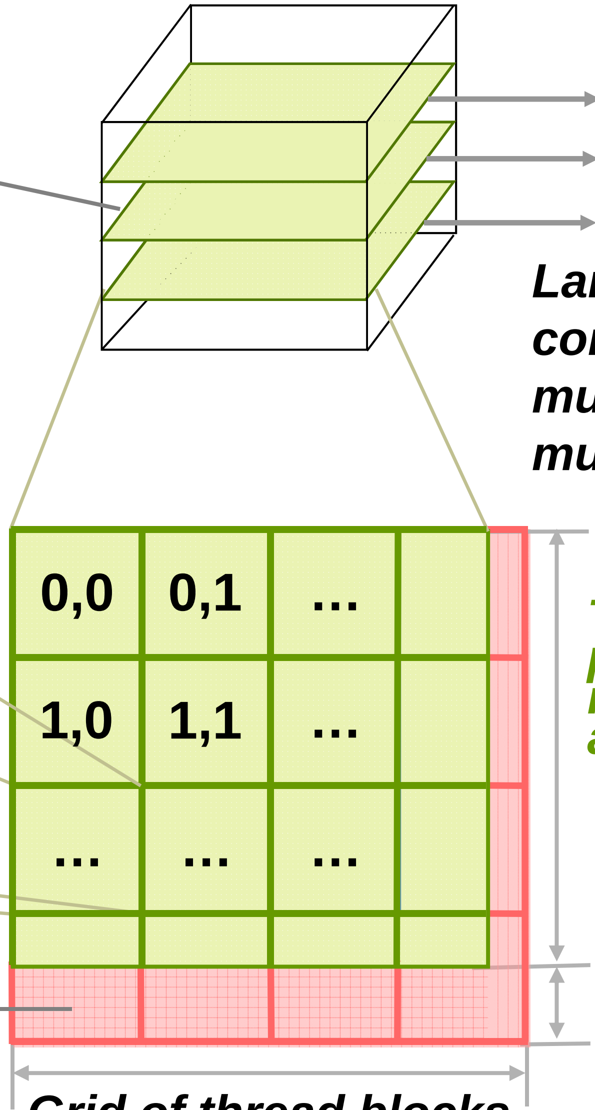
$$\rho(\vec{r};\vec{r}_1,\vec{r}_2,\ldots,\vec{r}_N)=\sum_{i=1}^{N} e^{\frac{-|\vec{r}-\vec{r}_i|^2}{2\alpha^2}}$$

- Extract isosurface for a user-defined density value



3-D density map lattice and extracted surface

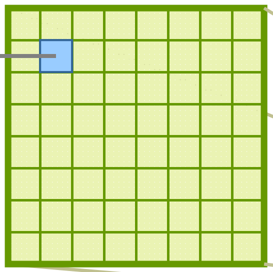
QuickSurf GPU Parallel Decomposition



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

Large volume computed in multiple passes, or multiple GPUs

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



Each thread computes one density map lattice point.

Threads producing results that are used

Inactive threads, region of discarded output

Padding optimizes global memory performance, guaranteeing coalesced global memory accesses

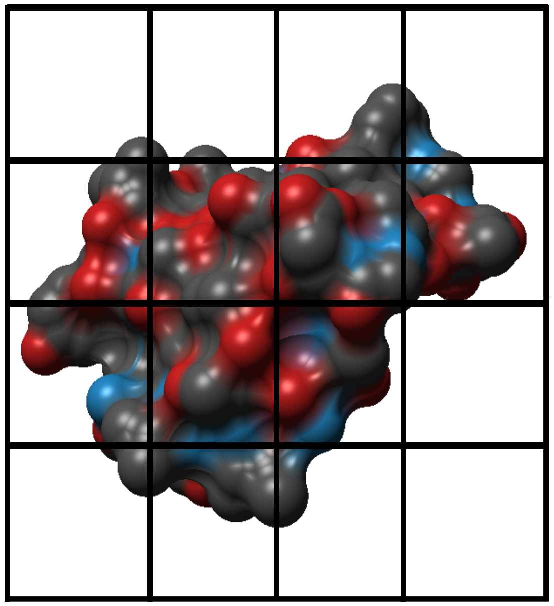
Grid of thread blocks

QuickSurf and Limited GPU Global Memory

- High resolution molecular surfaces require a fine lattice spacing
- Memory use grows cubically with decreased lattice spacing
- Not typically possible to compute a surface in a single pass, so we loop over sub-volume “chunks” until done…
- Chunks pre-allocated and sized to GPU global mem capacity to prevent unexpected memory allocation failure while animating…
- Complication:
  - Thrust allocates GPU mem. on-demand, no recourse if insufficient memory, have to re-gen QuickSurf data structures if caught by surprise!
- Workaround:
  - Pre-allocate guesstimate workspace for Thrust
  - Free the Thrust workspace right before use
  - Newest Thrust allows user-defined allocator code…

QuickSurf Particle Sorting, Bead Generation, Spatial Hashing

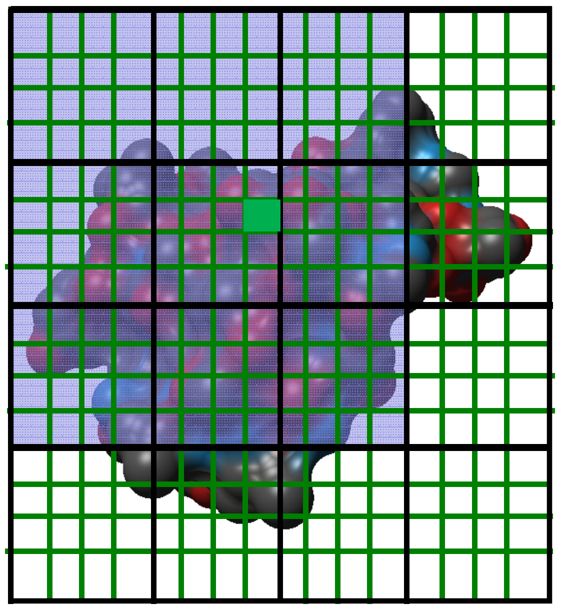
- Particles sorted into spatial acceleration grid:
  - Selected atoms or residue “beads” converted lattice coordinate system
  - Each particle/bead assigned cell index, sorted w/NVIDIA Thrust template library
  - Once particles are assigned cell indices and are sorted, a second kernel generates a cell lookup table to translate cell indices into a starting and ending indices in the sorted particle array
  - The cell lookup table is used by the density map algorithm to loop over all of the atoms contained within a given cell



Coarse resolution spatial acceleration grid

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 Map Kernel Optimizations

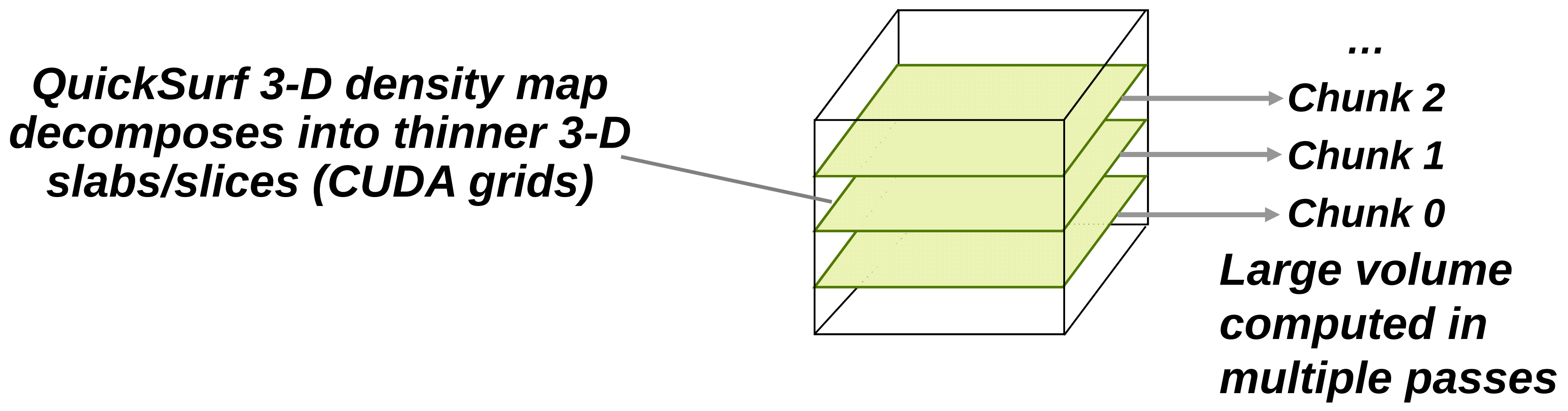
- Compute reciprocals, prefactors, other math prior to kernel launch
- Use of **intN** and **floatN** vector types for improved global memory bandwidth
- Thread coarsening: one thread computes multiple output densities and colors
- Input data and register tiling: share blocks of input, partial distances in regs shared among multiple outputs
- Global memory (L1 cache) broadcasts: all threads in the block traverse the same atom/particle at the same time

QuickSurf Density Map Kernel Snippet…

```
for (zab=zabmin; zab<=zabmax; zab++) {
  for (yab=yabmin; yab<=yabmax; yab++) {
   for (xab=xabmin; xab<=xabmax; xab++) {
    int abcellidx = zab * acplanesz + yab * acncells.x + xab;
    uint2 atomstartend = cellStartEnd[abcellidx];
    if (atomstartend.x != GRID_CELL_EMPTY) {
     for (unsigned int atomid=atomstartend.x; atomid<atomstartend.y; atomid++) {
      float4 atom = sorted_xyzr[atomid];
      float dx = coorx - atom.x;      float dy = coory - atom.y;      float dz = coorz - atom.z;
      float dxy2 = dx*dx + dy*dy;
      float r21 = (dxy2 + dz*dz) * atom.w;
      densityval1 += exp2f(r21);
       /// Loop unrolling and register tiling benefits begin here……
      float dz2 = dz + gridspacing;
      float r22 = (dxy2 + dz2*dz2) * atom.w;
      densityval2 += exp2f(r22);
      /// More loop unrolling ….
```

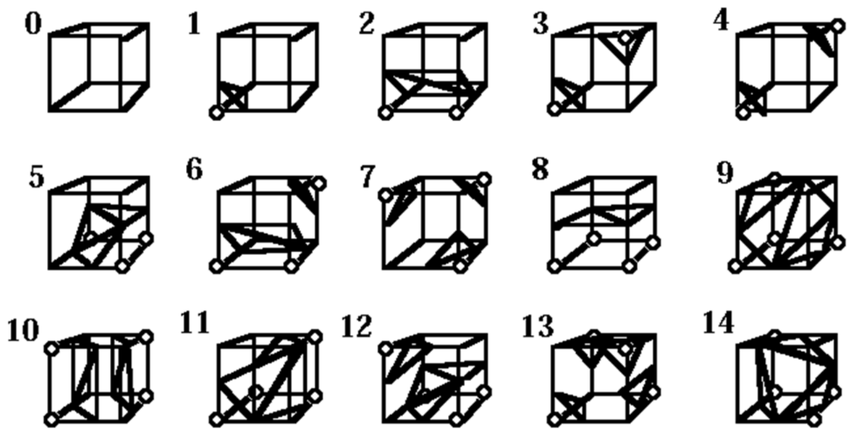
QuickSurf Marching Cubes Isosurface Extraction

- Isosurface is extracted from each density map “chunk”, and either copied back to the host, or rendered directly out of GPU global memory via CUDA/OpenGL interop
- All MC memory buffers are pre-allocated to prevent significant overhead when animating a simulation trajectory


QuickSurf 3-D density map decomposes into thinner 3-D slabs/slices (CUDA grids)
…
Chunk 2
Chunk 1
Chunk 0
Large volume computed in multiple passes

(Very) Brief Marching Cubes Isosurface Extraction Overview

- Given a 3-D volume of scalar density values and a requested surface density value, marching cubes computes vertices and triangles that compose the requested surface triangle mesh
- Each MC "cell" (a cube with 8 density values at its vertices) produces a variable number of output vertices depending on how many edges of the cell contain the requested isovalue…
- Use **scan()** to compute the output indices so that each worker thread has **conflict-free output** of vertices/triangles



(Very) Brief Marching Cubes Isosurface Extraction Overview

- Once the output vertices have been computed and stored, we compute surface normals and colors for each of the vertices
- Although the separate normals+colors pass reads the density map again, molecular surfaces tend to generate a small percentage of MC cells containing triangles, we avoid wasting interpolation work
- We use CUDA **tex3D()** hardware 3-D texture mapping:
  - Costs double the texture memory and a one copy from GPU global memory to the target texture map with **cudaMemcpy3D()**
  - Still roughly 2x faster than doing color interpolation without the texturing hardware, at least on GT200 and Fermi hardware
  - Kepler has new texture cache memory path that may make it feasible to do our own color interpolation and avoid the use of extra 3-D texture memory and associated copy, with acceptable performance

QuickSurf Marching Cubes Isosurface Extraction

- Our optimized MC implementation computes per-vertex surface normals, colors, and outperforms the NVIDIA SDK sample on Fermi GPUs by using vector **intN** and **floatN** types to achieve better memory bandwidth on **scan()** etc…
- Complications:
  - Even on a 6GB Quadro 7000, GPU global memory is under great strain when working with large molecular complexes, e.g. viruses
  - Marching cubes involves a parallel prefix sum (scan) to compute target indices for writing resulting vertices
  - We use Thrust for scan, has the same memory allocation issue mentioned earlier for the sort, so we use the same workaround
  - Worst-case number of output vertices can be huge, but we rarely have sufficient GPU memory for this – we use a fixed size vertex output buffer and hope our heuristics don't fail us

QuickSurf Performance GeForce GTX 580

Molecular system	Atoms	Resolution	T_{sort}	$T_{density}$	T_{MC}	# vertices	FPS
MscL	111,016	1.0Å	0.005	0.023	0.003	0.7 M	28
STMV capsid	147,976	1.0Å	0.007	0.048	0.009	2.4 M	13.2
Poliovirus capsid	754,200	1.0Å	0.01	0.18	0.05	9.2 M	3.5
STMV w/ water	955,225	1.0Å	0.008	0.189	0.012	2.3 M	4.2
Membrane	2.37 M	2.0Å	0.03	0.17	0.016	5.9 M	3.9
Chromatophore	9.62 M	2.0Å	0.16	0.023	0.06	11.5 M	3.4
Membrane w/ water	22.77 M	4.0Å	4.4	0.68	0.01	1.9 M	0.18

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

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

Extensions and Analysis Uses for QuickSurf Triangle Mesh

- Curved PN triangles:
  - We have performed tests with post-processing the resulting triangle mesh and using curved PN triangles to generate smooth surfaces with a larger grid spacing, for increased performance
  - Initial results demonstrate some potential, but there can be pathological cases where MC generates long skinny triangles, causing unsightly surface creases
- Analysis uses (beyond visualization):
  - Minor modifications to the density map algorithm allow rapid computation of solvent accessible surface area by summing the areas in the resulting triangle mesh
  - Modifications to the density map algorithm will allow it to be used for MDFF (molecular dynamics flexible fitting)
  - Surface triangle mesh can be used as the input for computing the electrostatic potential field for mesh-based algorithms

New Interactive Display & Analysis of Terabytes of Data:

Out-of-Core Trajectory I/O w/ Solid State Disks

450MB/sec

to 4GB/sec

A DVD movie per second!

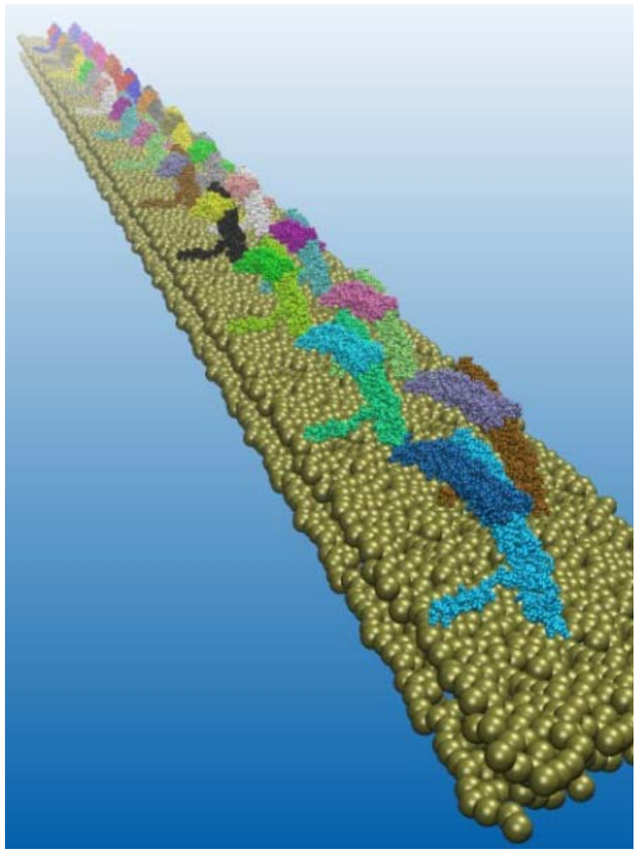
Commodity SSD, SSD RAID

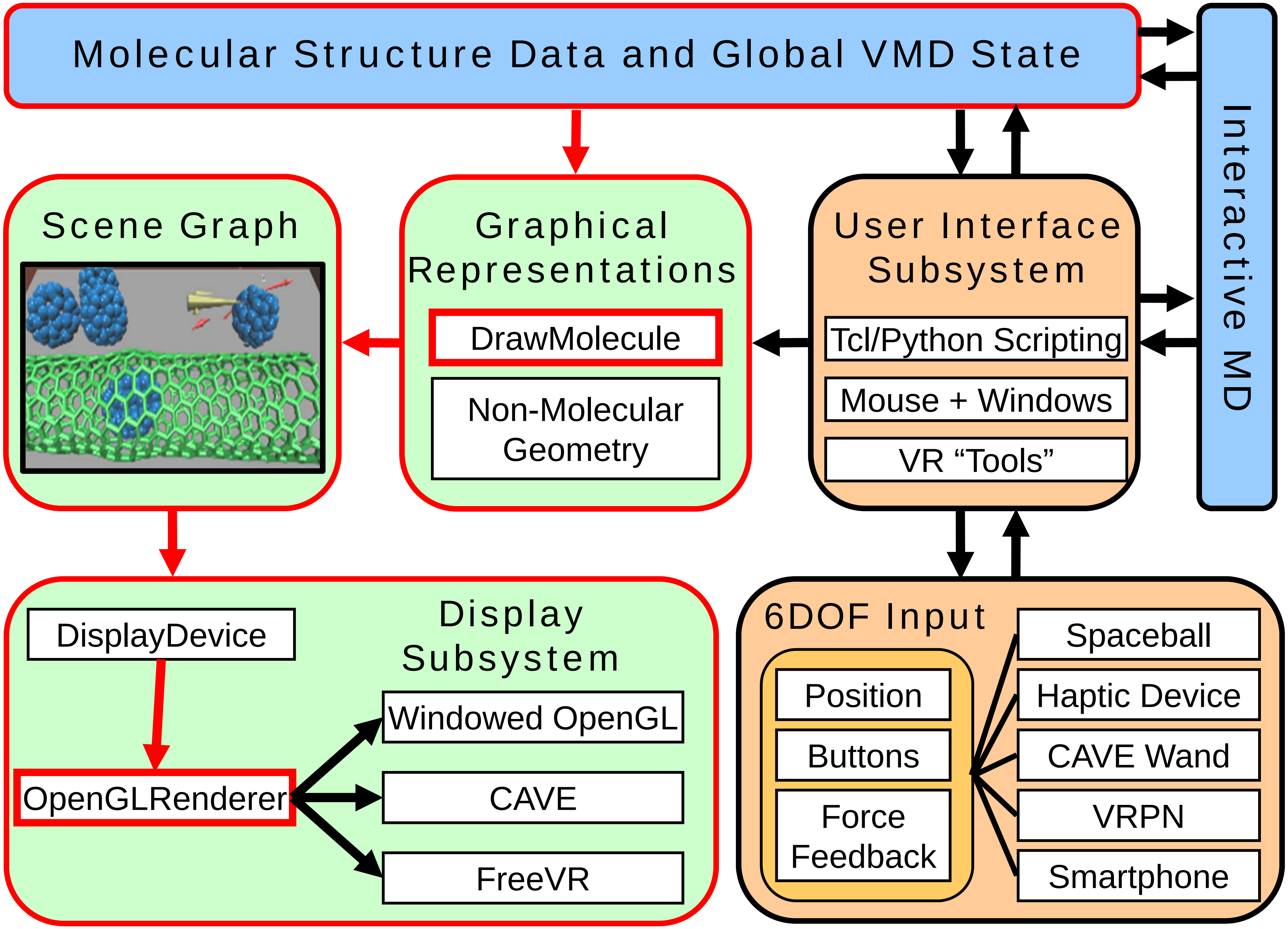
- Timesteps loaded on-the-fly (out-of-core)
  - Eliminates memory capacity limitations, even for multi-terabyte trajectory files
  - High performance achieved by new trajectory file formats, optimized data structures, and efficient I/O
- Analyze long trajectories significantly faster using just a personal computer

Immersive out-of-core visualization of large-size and long-timescale molecular dynamics trajectories. J. Stone, K. Vandivort, and K. Schulten. *Lecture Notes in Computer Science*, 6939:1-12, 2011.

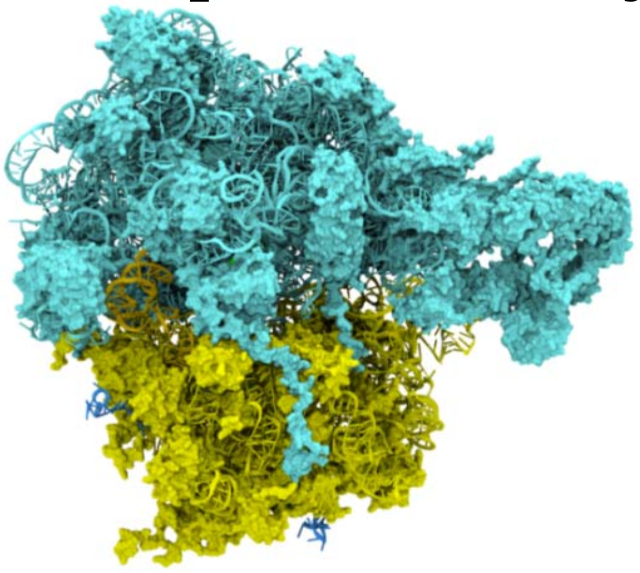
Challenges for Immersive Visualization of Dynamics of Large Structures

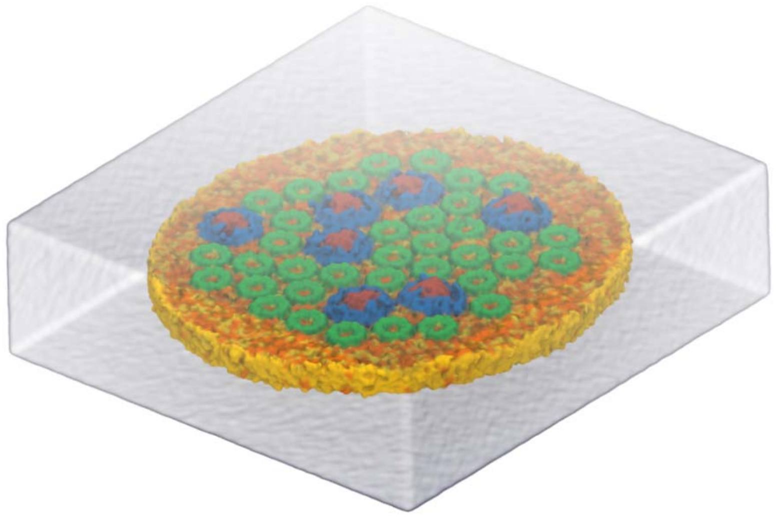
- Graphical representations re-generated for each simulation timestep:
  - Dependent on user-defined atom selections
- Although visualizations often focus on interesting regions of substructure, fast display updates require rapid traversal of molecular data structures
- Optimized per-frame atom selection traversal:
  - Increased performance of per-frame updates by ~10x for 116M atom BAR case with 200,000 selected atoms
- New GLSL point sprite sphere shader:
  - Reduce host-GPU bandwidth for displayed geometry
  - Over 20x faster than old GLSL spheres drawn using display lists — drawing time is now inconsequential
- Optimized all graphical representation generation routines for large atom counts, sparse selections


116M atom BAR domain test case:
200,000 selected atoms,
stereo trajectory animation 70 FPS,
static scene in stereo 116 FPS


Molecular Structure Data and Global VMD State
Interactive MD
Scene Graph
Graphical Representations
DrawMolecule
Non-Molecular Geometry
User Interface Subsystem
Tcl/Python Scripting
Mouse + Windows
VR “Tools”
Display Subsystem
DisplayDevice
OpenGLRenderer
Windowed OpenGL
CAVE
FreeVR
6DOF Input
Position
Buttons
Force Feedback
Spaceball
Haptic Device
CAVE Wand
VRPN
Smartphone

VMD Out-of-Core Trajectory I/O Performance: SSD-Optimized Trajectory Format, 8-SSD RAID





Ribosome w/ solvent

3M atoms

3 frames/sec w/ HD

60 frames/sec w/ SSDs

Membrane patch w/ solvent

20M atoms

0.4 frames/sec w/ HD

8 frames/sec w/ SSDs

New SSD Trajectory File Format 2x Faster vs. Existing Formats

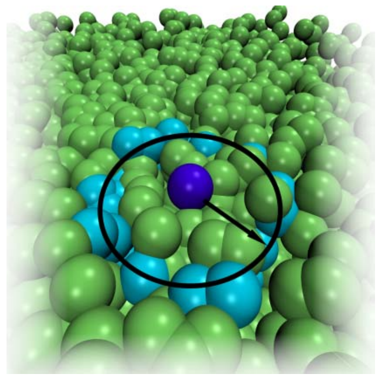
VMD I/O rate ~2.1 GB/sec w/ 8 SSDs

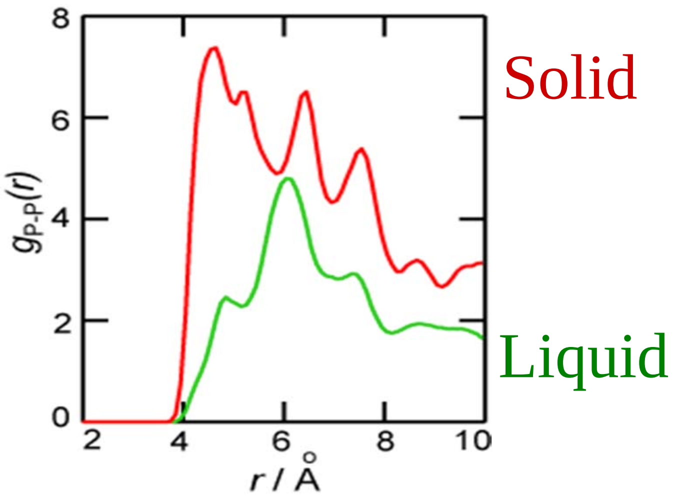
Challenges for High Throughput Trajectory Visualization and Analysis

- It is not currently possible to exploit full disk I/O bandwidths when streaming data from SSD arrays (>4GB/sec) to GPU global memory
- Need to eliminate copies from disk controllers to host memory – bypass host entirely and perform zero-copy DMA operations straight from disk controllers to GPU global memory
- **Goal: GPUs directly pull in pages from storage systems bypassing host memory entirely**

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
- Quadratic time complexity $O(N^2)$





Computing RDFs

- Compute distances for all pairs of atoms between two groups of atoms A and B
- A and B may be the same, or different
- Use nearest image convention for periodic systems
- Each pair distance is inserted into a histogram
- Histogram is normalized one of several ways depending on use, but usually according to the volume of the spherical shells associated with each histogram bin

Computing RDFs on CPUs

- Atom coordinates can be traversed in a strictly consecutive access pattern, yielding good cache utilization
- Since RDF histograms are usually small to moderate in size, they normally fit entirely in L2 cache
- CPUs can compute the entire histogram in a **single pass**, regardless of the problem size or number of histogram bins

Histogramming on the CPU (slow-and-simple C)

```
memset(histogram, 0, sizeof(histogram));
for (i=0; i<numdata; i++) {
  float val = data[i];
  if (val >= minval  && val <= maxval) {
    int bin = (val - minval) / bindelta;
    histogram[bin]++;
  }
}
```

Fetch-and-increment:

random access updates to histogram bins…

Parallel Histogramming on Multi-core CPUs

- Parallel updates to a single histogram bin creates a **potential output conflict**
- CPUs have atomic increment instructions, but they often take hundreds of clock cycles; unsuitable…
- SSE can't be used effectively: lacks ability to "scatter" to memory (e.g. no *scatter-add*, no indexed store instructions)
- For small numbers of CPU cores, it is best to **replicate** and **privatize** the histogram for each CPU thread, compute them independently, and combine the separate histograms in a final reduction step

Computing RDFs on the GPU

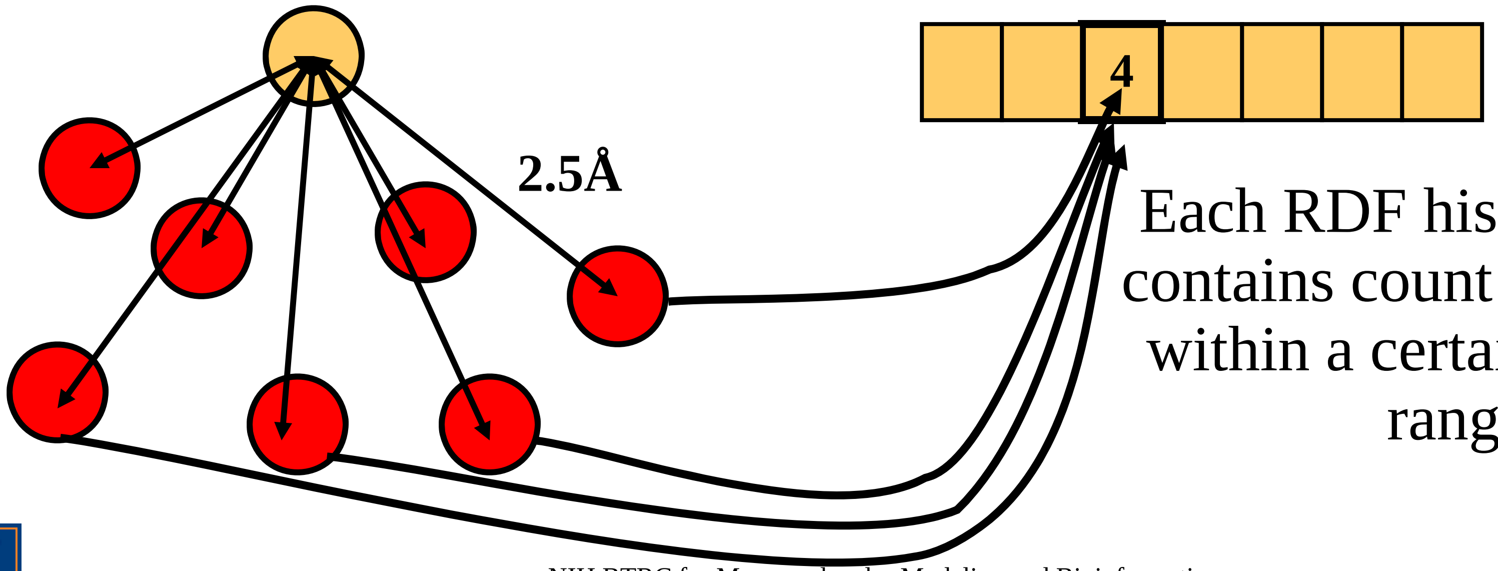
- Need tens of thousands of independent threads
- Each GPU thread computes one or more atom pair distances
- Performance is limited by the speed of histogramming
- Histograms are best stored in fast on-chip shared memory
- Small size of shared memory severely constrains the range of viable histogram update techniques
- **Fast CUDA implementation on Fermi: 30-92x faster than CPU**

Computing Atom Pair Distances on the GPU

- Memory access pattern is simple
- Primary consideration is **amplification of effective memory bandwidth**, through use of GPU on-chip shared memory, caches, and broadcast of data to multiple or all threads in a thread block

Radial Distribution Functions on GPUs

- Load blocks of atoms into shared memory and constant memory, compute periodic boundary conditions and atom-pair distances, all in parallel…
- Each thread computes all pair distances between its atom and all atoms in constant memory, incrementing the appropriate bin counter in the RDF histogram..



Each RDF histogram bin contains count of particles within a certain distance range

GPU Histogramming

- Tens of thousands of threads concurrently computing atom distance pairs…
- Far too many threads for a simple per-thread histogram privatization approach like CPU…
- Viable approach: **per-warp histograms**
- Fixed size shared memory limits histogram size that can be computed in a single pass
- Large histograms require **multiple passes**, but we can skip block pairs that are known not to contribute to a histogram window

Per-warp Histogram Approach

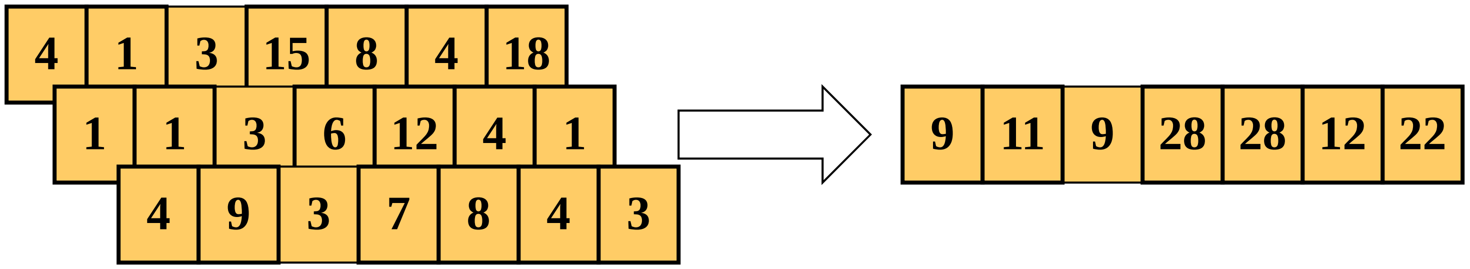
- Each warp maintains its own **private** histogram in on-chip shared memory
- Each thread in the warp computes an atom pair distance and updates a histogram bin in parallel
- Conflicting histogram bin updates are resolved using one of two schemes:
  - Shared memory write combining with thread-tagging technique (older hardware, e.g. G80, G9x)
  - **atomicAdd()** to shared memory (new hardware)

RDF Inner Loops (abbreviated, xdist-only)

```
// loop over all atoms in constant memory
for (iblock=0; iblock<loopmax2; iblock+=3*NCUDABLOCKS*NBLOCK) {
   __syncthreads();
   for (i=0; i<3; i++) xyzi[threadIdx.x + i*NBLOCK]=pxi[iblock + i*NBLOCK]; // load coords…
   __syncthreads();
   for (joffset=0; joffset<loopmax; joffset+=3) {
      rxij=fabsf(xyzi[idxt3  ] - xyzj[joffset  ]);  // compute distance, PBC min image convention
      rxij2=celld.x - rxij;
      rxij=fminf(rxij, rxij2);
      rij=rxij*rxij;
      […other distance components…]
      rij=sqrtf(rij + rxij*rxij);
      ibin=__float2int_rd((rij-rmin)*delr_inv);
      if (ibin<nbins && ibin>=0 && rij>rmin2) {
        atomicAdd(llhists1+ibin, 1U);
      }
   } //joffset
} //iblock
```

Writing/Updating Histogram in Global Memory

- When thread block completes, add independent per-warp histograms together, and write to per-thread-block histogram in global memory
- Final reduction of all per-thread-block histograms stored in global memory

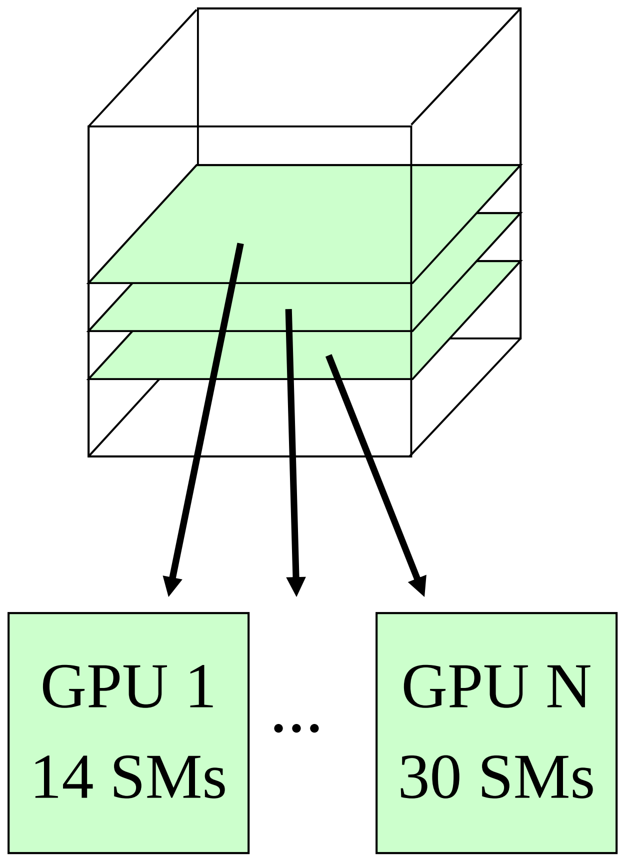


Preventing Integer Overflows

- Since all-pairs RDF calculation computes many billions of pair distances, we have to **prevent integer overflow** for the 32-bit histogram bin counters (supported by the **atomicAdd()** routine)
- We compute full RDF calculation in **multiple kernel launches**, so each kernel launch computes partial histogram
- Host routines read GPUs and increment large (e.g. long, or double) histogram counters in host memory after each kernel completes

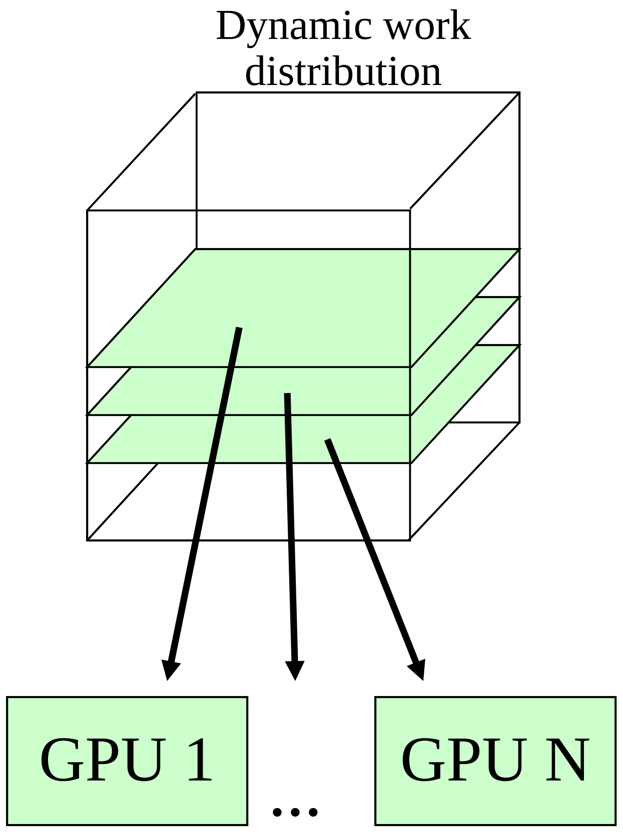
Multi-GPU Load Balance

- Many early CUDA codes assumed all GPUs were identical
- Host machines may contain a diversity of GPUs of varying capability (discrete, IGP, etc)
- Different GPU on-chip and global memory capacities may need different problem "tile" sizes
- Static decomposition works poorly for non-uniform workload, or diverse GPUs



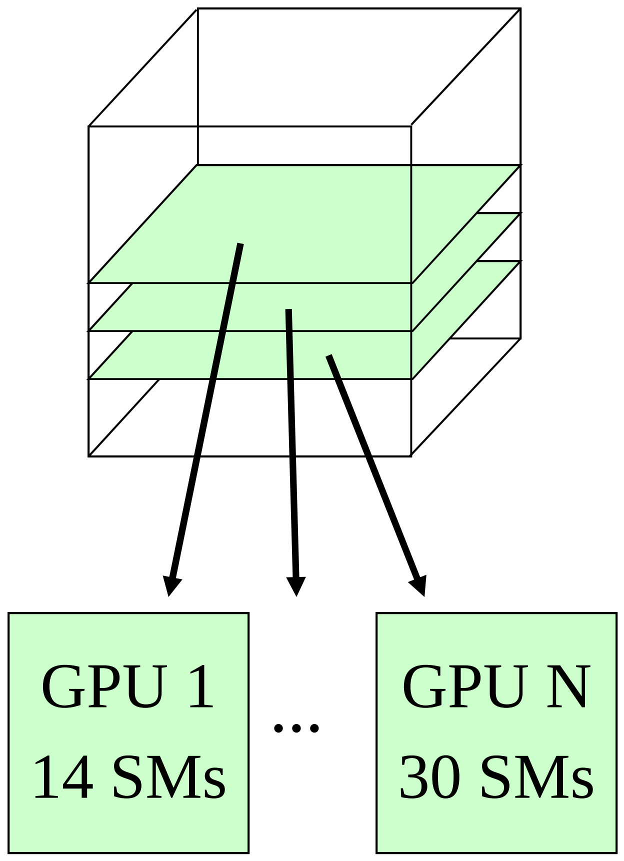
Multi-GPU Dynamic Work Distribution

```
// Each GPU worker thread loops over
// subset of work items…
while (!threadpool_next_tile(&parms,
    tilesize, &tile)){
  // Process one work item…
  // Launch one CUDA kernel for each
  //   loop iteration taken…
  // Shared iterator automatically
  //   balances load on GPUs
}
```



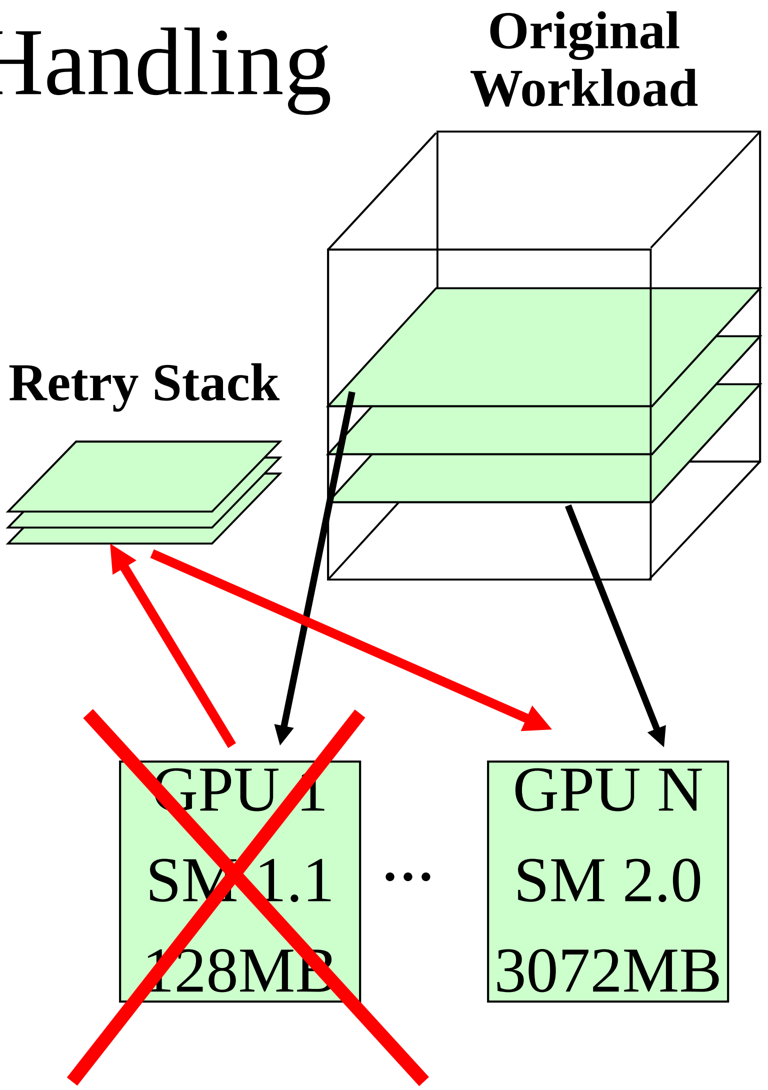
Multi-GPU RDF Calculation

- Distribute combinations of tiles of atoms and histogram regions to different GPUs
- Decomposed over two dimensions to obtain enough work units to balance GPU loads
- Each GPU computes its own histogram, and all results are combined for final histogram



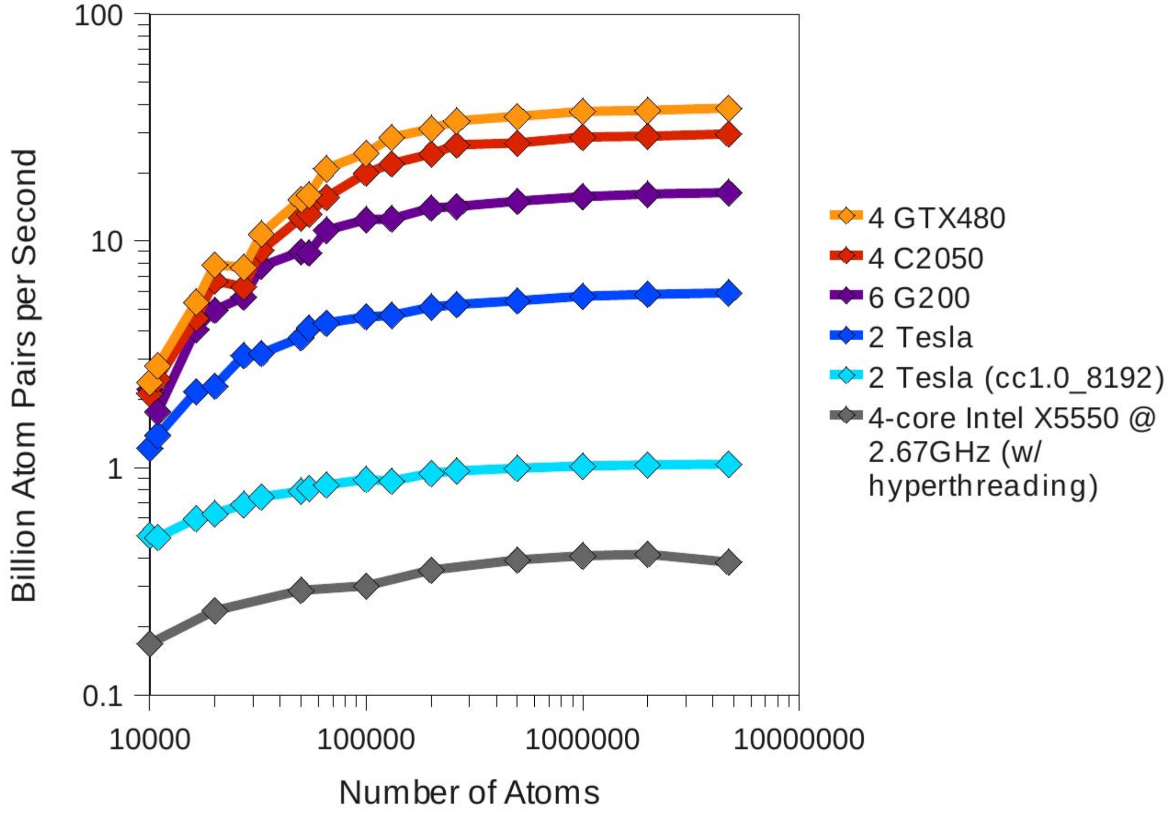
Multi-GPU Runtime Error/Exception Handling

- Competition for resources from other applications can cause runtime failures, e.g. GPU out of memory half way through an algorithm
- Handle exceptions, e.g. convergence failure, NaN result, insufficient compute capability/features
- Handle and/or reschedule failed tiles of work



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. *J. Comp. Physics*, 230(9):3556-3569, 2011.

Acknowledgements

- Theoretical and Computational Biophysics Group, Luthey-Schulten Group, IMPACT Group, University of Illinois at Urbana-Champaign
- NCSA Blue Waters Team
- NCSA Innovative Systems Lab
- Ben Levine (MSU), Axel Kohlmeyer (Temple)
- NVIDIA CUDA Center of Excellence, University of Illinois at Urbana-Champaign
- The CUDA team at NVIDIA
- NIH support: P41-RR005969

GPU Computing Publications

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

- **Lattice Microbes: High-Performance Stochastic Simulation Method for the Reaction-Diffusion Master Equation.** E. Roberts, J. Stone, Z. Luthey-Schulten. 2012. (Submitted)
- **Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.** M. Krone, J. 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.