High Performance Molecular Visualization and Analysis on GPUs

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/

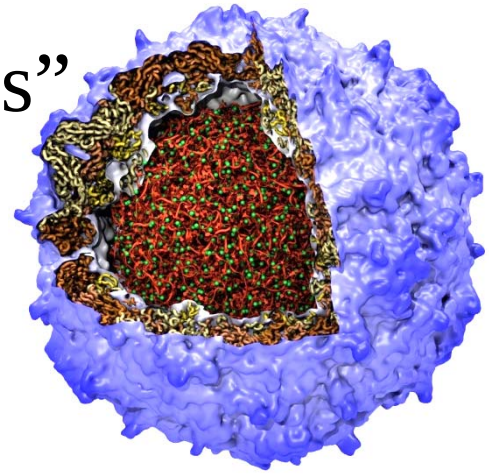
GPU Technology Conference

San Jose Convention Center,

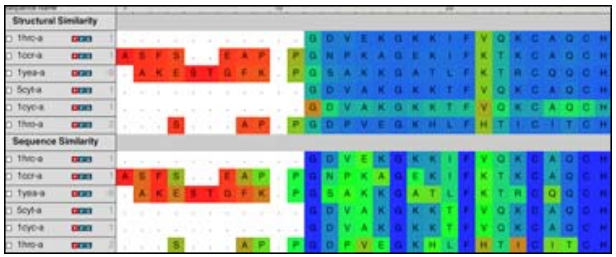
San Jose, CA, May 16, 2012

VMD – “Visual Molecular Dynamics”

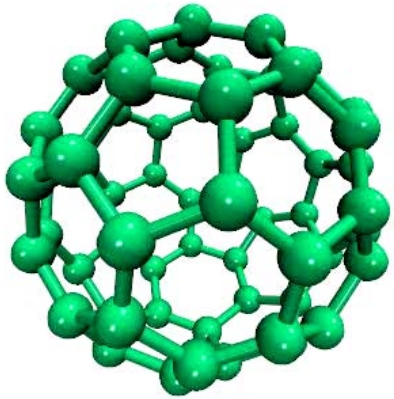
- Visualization and analysis of:
  - molecular dynamics simulations
  - quantum chemistry calculations
  - particle systems and whole cells
  - sequence data
- User extensible w/ scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/



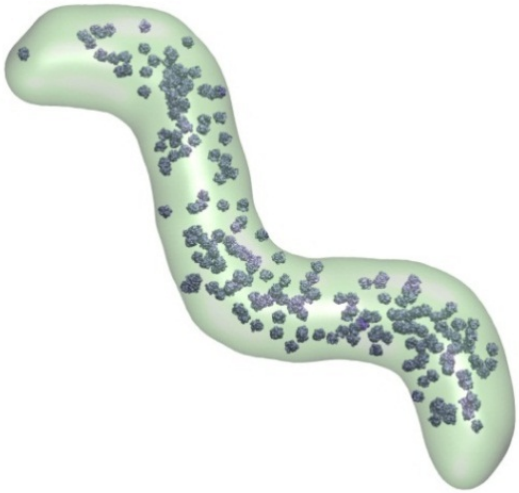
Poliovirus



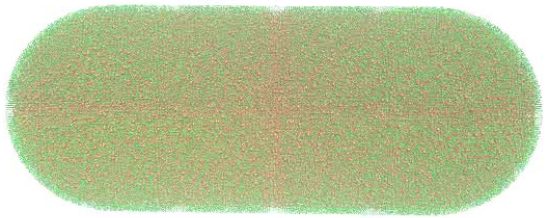
Ribosome Sequences



Electrons in
Vibrating Buckyball



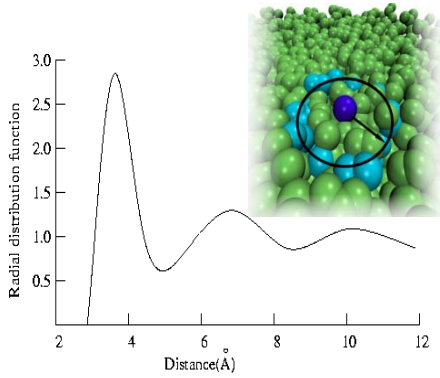
Cellular Tomography,
Cryo-electron Microscopy

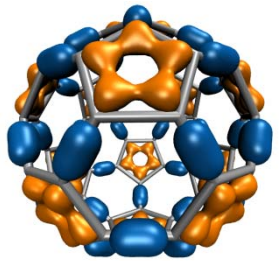


Whole Cell Simulations

GPU Accelerated Trajectory Analysis and Visualization in VMD

GPU-Accelerated Feature	GPU Speedup
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 latest NVIDIA “Kepler” GPU

Early Experiences with Kepler

- **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 greatly improved performance for special functions vs. Fermi:

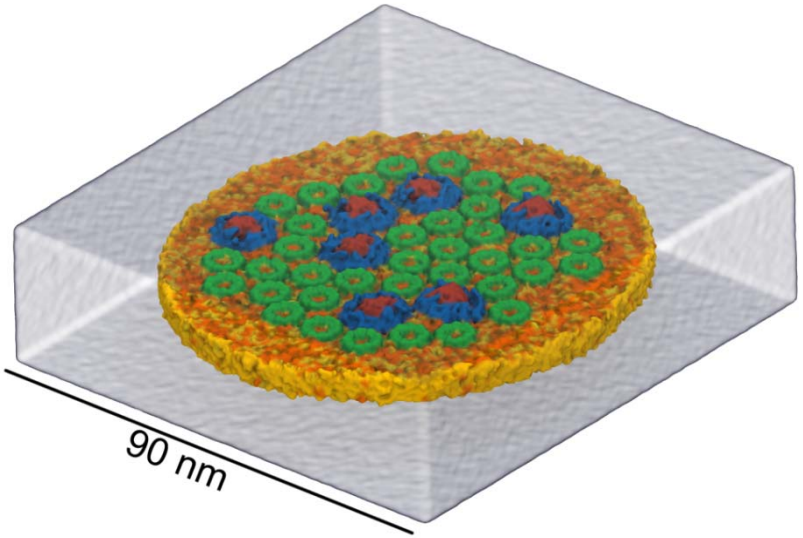
CUDA Kernel	Dominant Arithmetic Operations	Kepler (GeForce 680) Speedup vs. Fermi (Quadro 7000)
Direct Coulomb summation	rsqrtf()	2.4x
Molecular orbital grid evaluation	expf(), exp2f(), Multiply-Add	1.7x

VMD: Improved Support for Large Datasets

- New structure building tools, file formats, and data structures enable VMD to operate efficiently up to 100M atoms
  - Up to 30% more memory efficient
  - Analysis routines optimized for large structures, up to 20x faster for calculations on 100M atom complexes
  - New and revised graphical representations support smooth trajectory animation for multi-million atom complexes; VMD remains interactive even when displaying surface reps for 20M atom membrane patch
- Uses multi-core CPUs and GPUs for the most demanding computations

20M atoms:

membrane patch and solvent



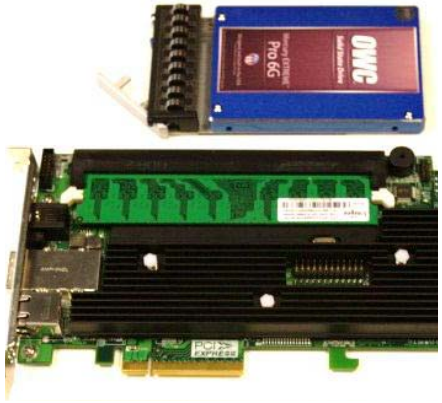
New Interactive Display & Analysis of Terabytes of Data:

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

450MB/sec

to 4GB/sec

A DVD movie per second!

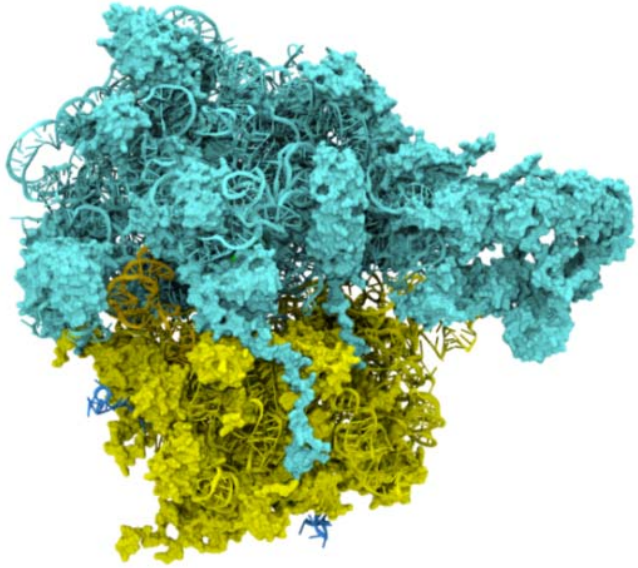


Commodity SSD, SSD RAID

- 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.

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

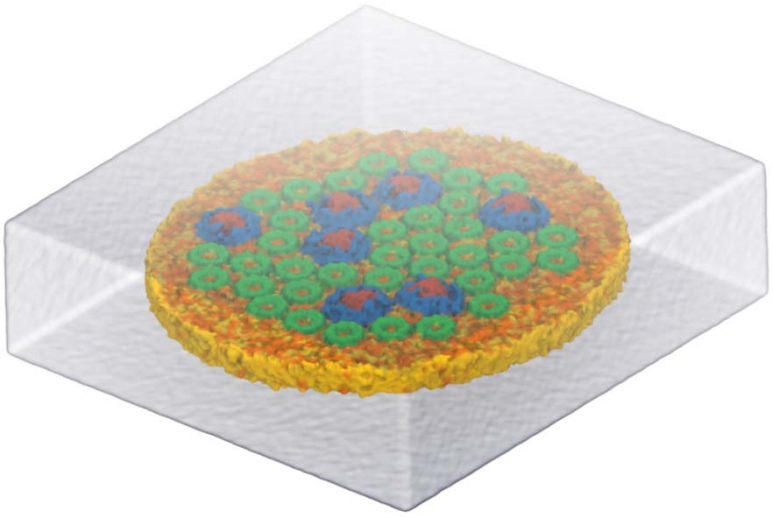


Ribosome w/ solvent

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

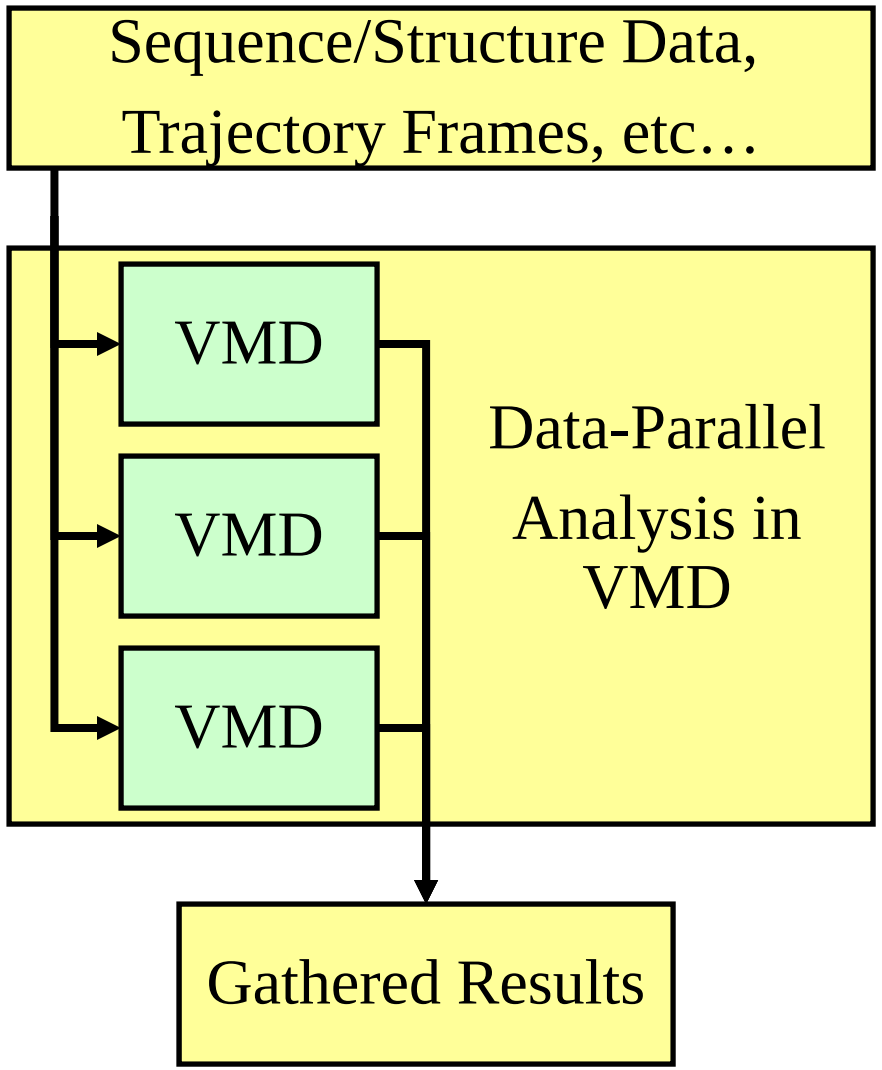
VMD I/O rate ~2.1 GB/sec w/ 8 SSDs

Challenges for High Throughput Trajectory Analysis

- It is not currently possible to fully exploit full I/O bandwidths when streaming data from SSD arrays (>4GB/sec) to GPU global memory
- Need to eliminated copies from disk controllers to host memory – bypass host entirely and perform zero-copy DMA operations straight from disk controllers to GPU global memory

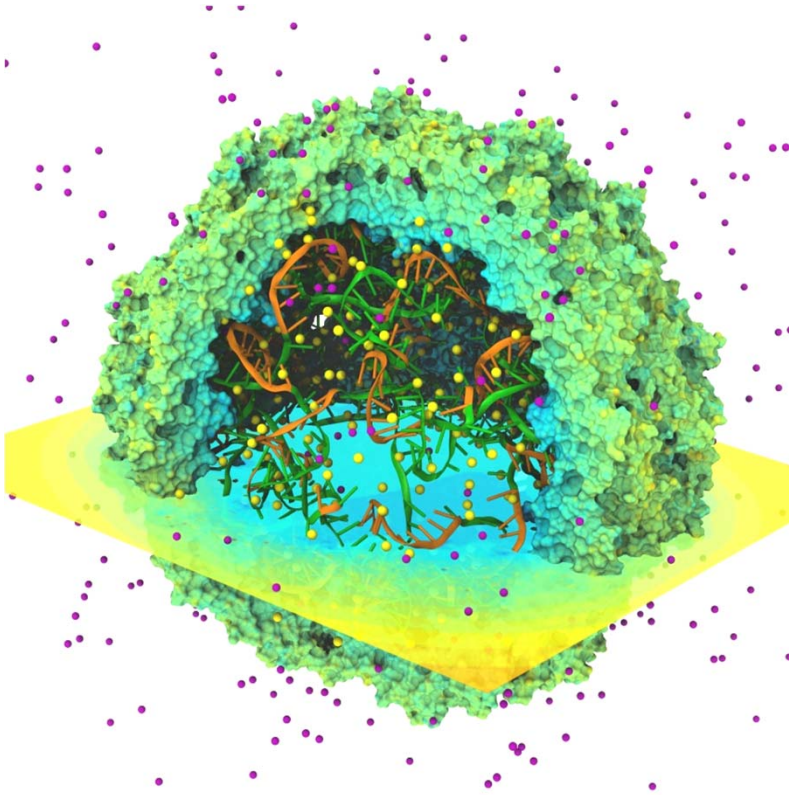
VMD for Demanding Analysis Tasks
Parallel VMD Analysis w/ MPI

- Analyze trajectory frames, structures, or sequences in parallel on clusters and supercomputers:
  - Compute time-averaged electrostatic fields, MDFF quality-of-fit, etc.
  - Parallel rendering, movie making
- Addresses computing requirements beyond desktop
- User-defined parallel reduction operations, data types
- Dynamic load balancing:
  - Tested with up to 15,360 CPU cores
- **Supports GPU-accelerated clusters and supercomputers**



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.

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

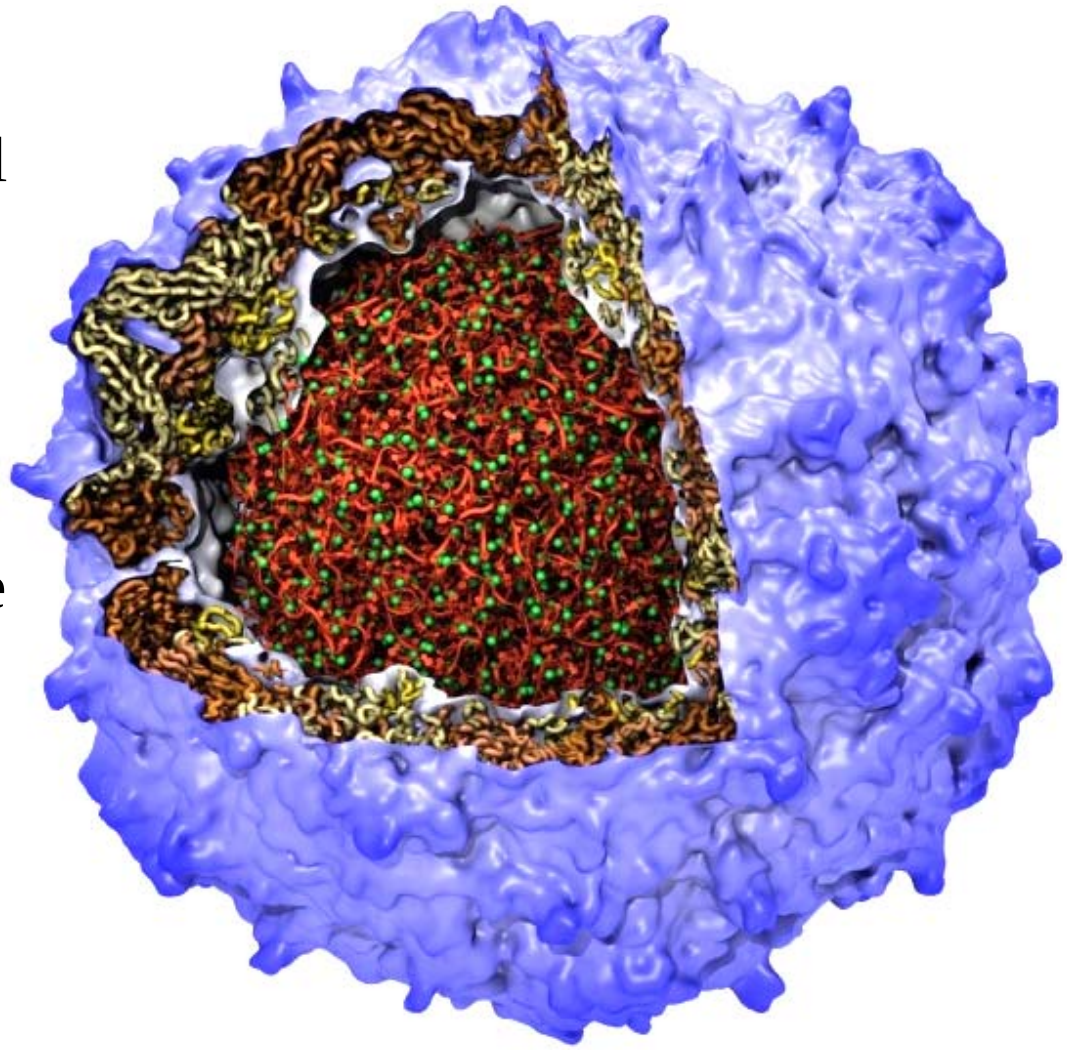
Time-Averaged Electrostatics Analysis on NCSA Blue Waters Early Science System

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

Molecular Surface Visualization

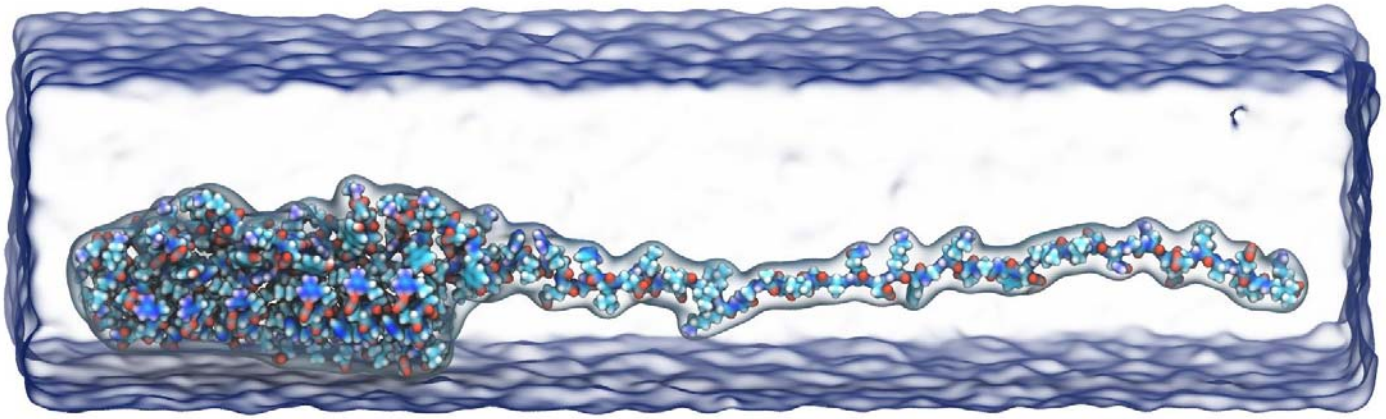
- Large biomolecular complexes are difficult to interpret with atomic detail graphical representations
- Even secondary structure representations become cluttered
- Surface representations are easier to use when greater abstraction is desired, but are computationally costly
- Existing surface display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

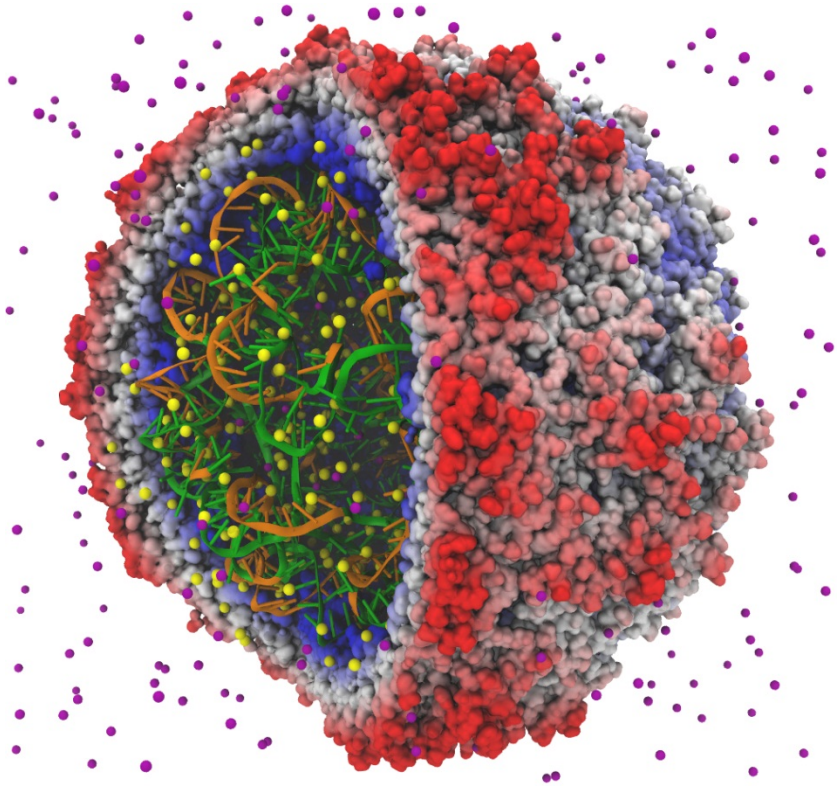


Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.

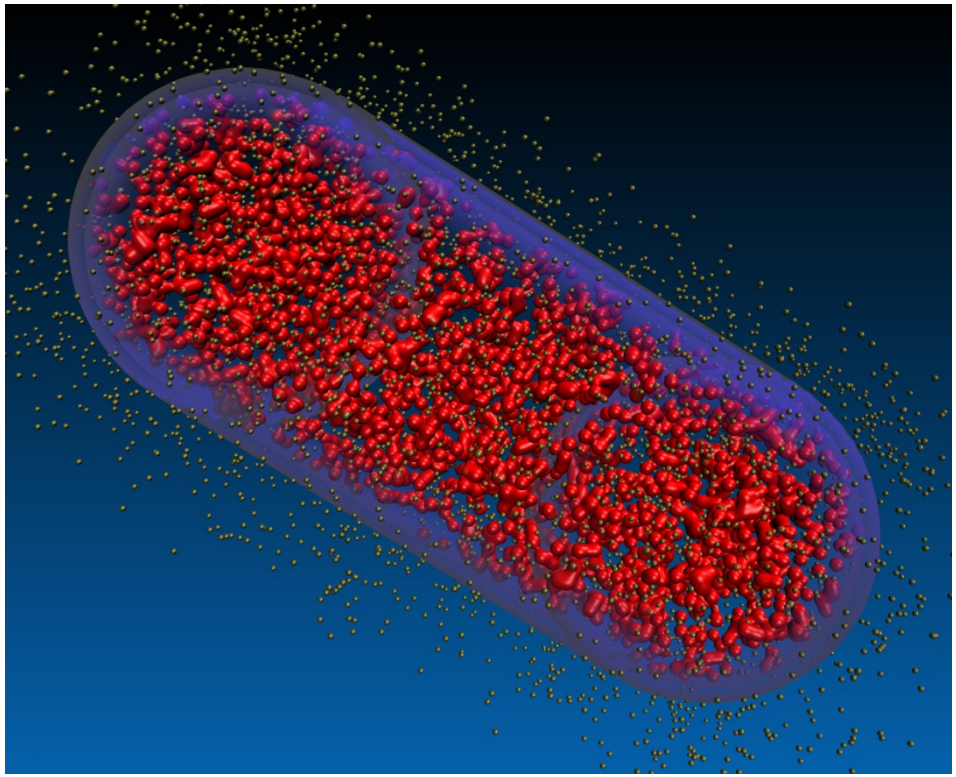
M. Krone, J. Stone, T. Ertl, K. Schulten. *EuroVis* 2012. (*In-press*)

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



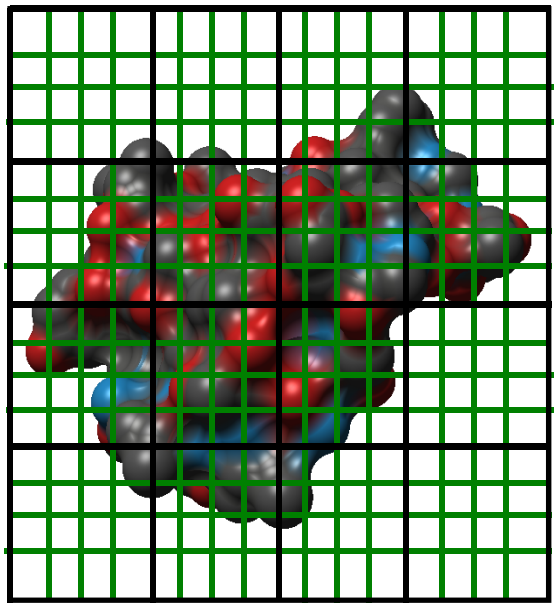
Lattice Cell Simulations

QuickSurf Algorithm Overview

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

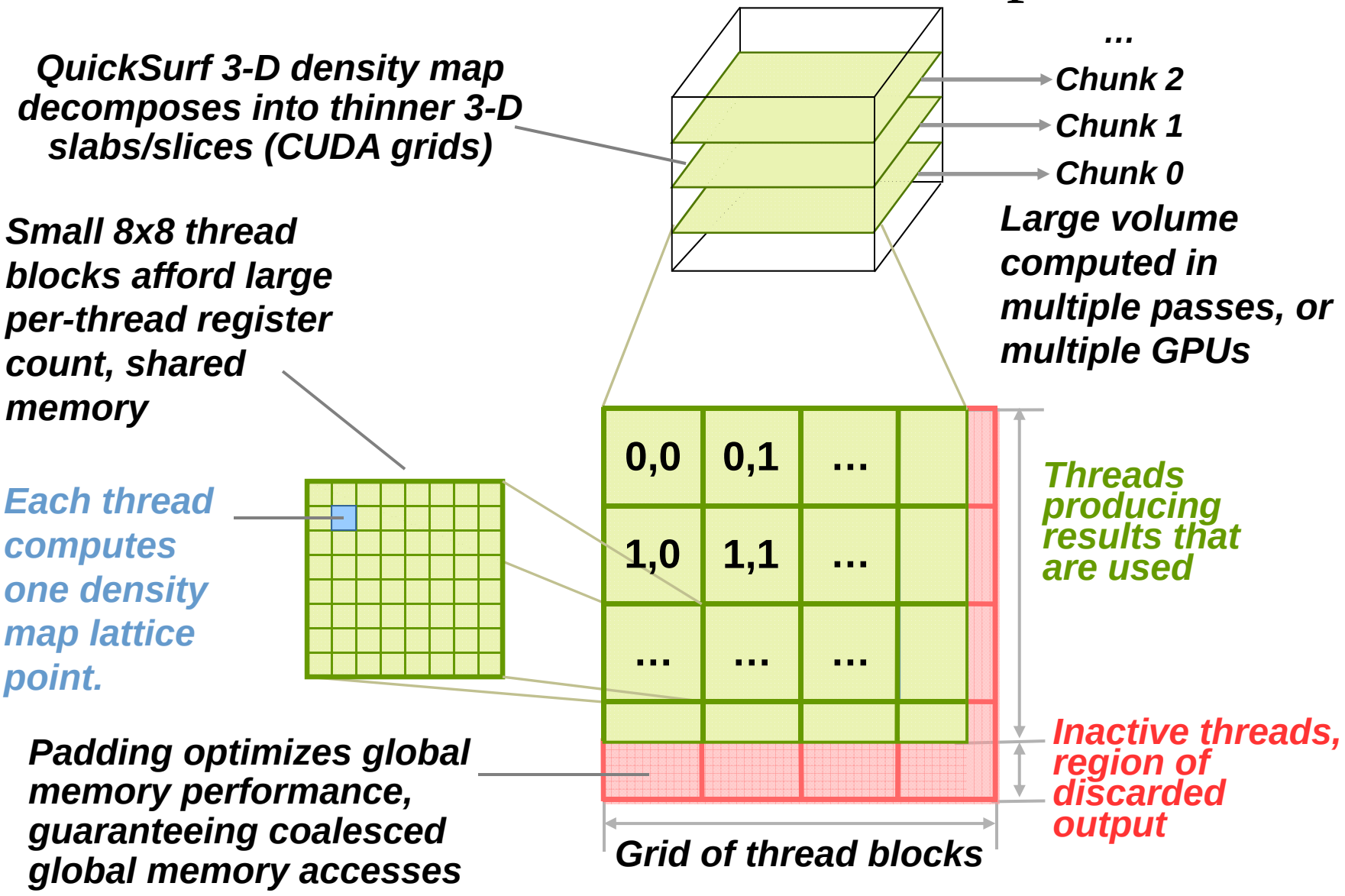
$$\rho(\vec{r}; \vec{r}_1, \vec{r}_2, \ldots, \vec{r}_N) = \sum_{i=1}^{N} e^{\frac{-|\vec{r}-\vec{r}_i|^2}{2\alpha^2}}$$

- Extract isosurface for a user-defined density value



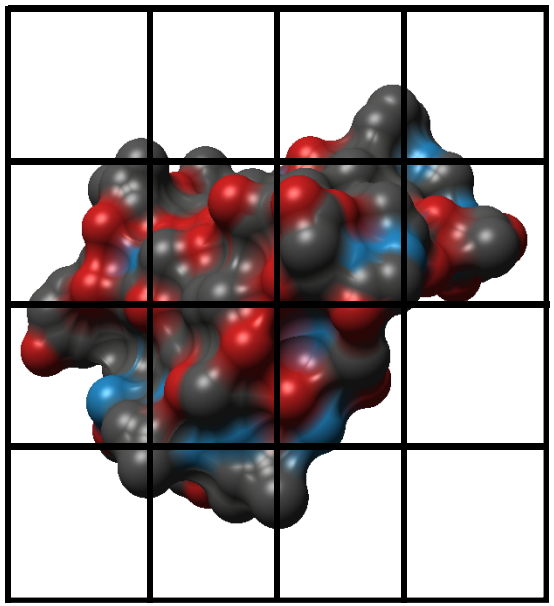
3-D density map lattice and extracted surface

QuickSurf GPU Parallel Decomposition



QuickSurf Particle Sorting, Bead Generation, Spatial Hashing

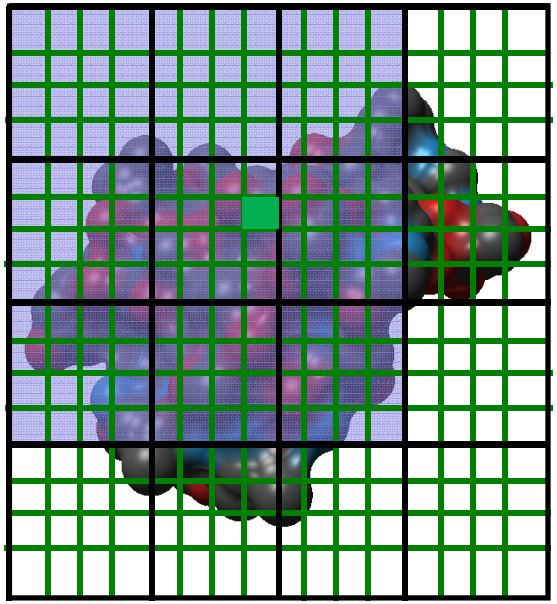
- Particles sorted into spatial acceleration grid:
  - Selected atoms or residue “beads” converted lattice coordinate system
  - Each particle/bead assigned cell index, sorted w/NVIDIA Thrust template library
- Complication:
  - Thrust allocates GPU mem. on-demand, no recourse if insufficient memory, 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



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

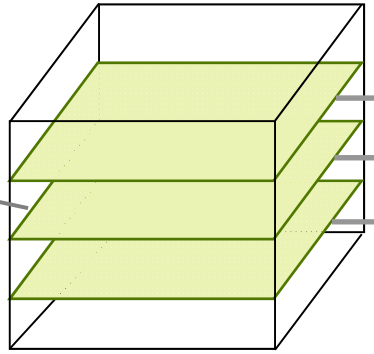


3-D density map lattice point and the neighboring spatial acceleration cells it references

QuickSurf 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 OpenGL interop
- All 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

QuickSurf Marching Cubes Isosurface Extraction

- Our optimized MC implementation computes per-vertex surface normals, colors, and outperforms the NVIDIA SDK sample by a fair margin on Fermi GPUs
- 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
  - The 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

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

QuickSurf Performance
GeForce GTX 580

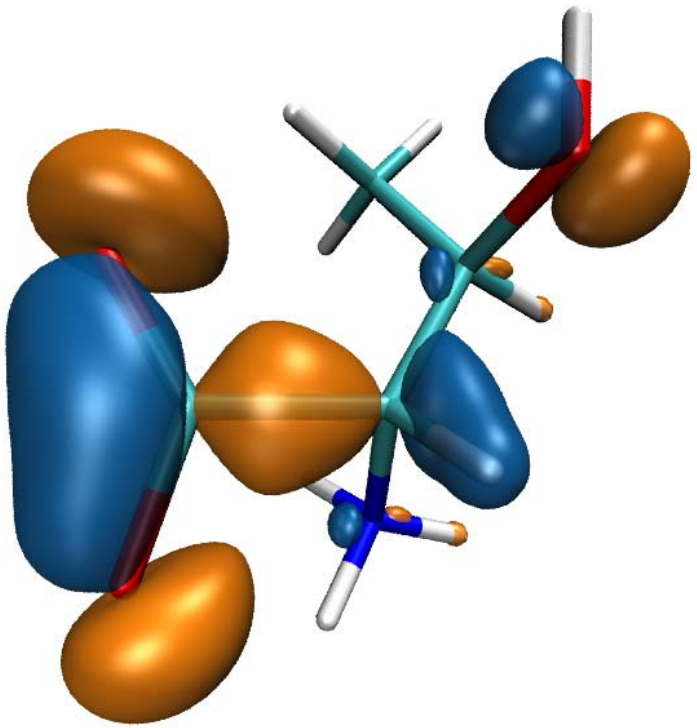
Molecular system	Atoms	Resolution	T_{sort}	$T_{density}$	T_{MC}	# vertices	FPS
MscL	111,016	1.0Å	0.005	0.023	0.003	0.7 M	28
STMV capsid	147,976	1.0Å	0.007	0.048	0.009	2.4 M	13.2
Poliovirus capsid	754,200	1.0Å	0.01	0.18	0.05	9.2 M	3.5
STMV w/ water	955,225	1.0Å	0.008	0.189	0.012	2.3 M	4.2
Membrane	2.37 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.

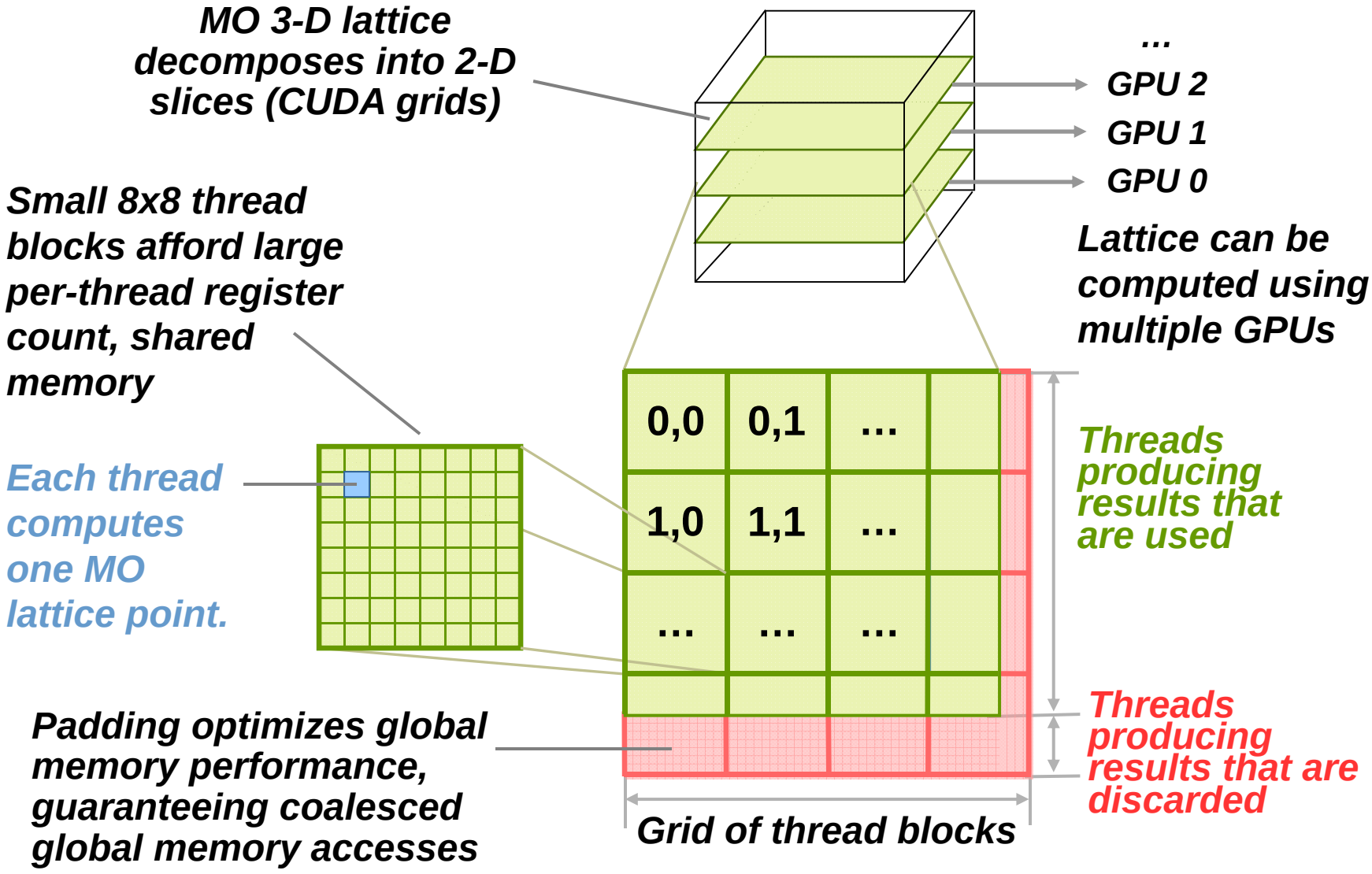
M. Krone, J. Stone, T. Ertl, K. Schulten. *EuroVis* 2012. (*In-press*)

Computing Molecular Orbitals

- Visualization of MOs aids in understanding the chemistry of molecular system
- Calculation of high resolution MO grids for display can require tens to hundreds of seconds on multi-core CPUs, even with the use of hand-coded SSE



MO GPU Parallel Decomposition



VMD MO GPU Kernel Snippet:
Loading Tiles Into Shared Memory On-Demand

```
[… outer loop over atoms …]
  if ((prim_counter + (maxprim<<1)) >= SHAREDSIZE) {
    prim_counter += sblock_prim_counter;
    sblock_prim_counter = prim_counter & MEMCOAMASK;
    s_basis_array[sidx      ] = basis_array[sblock_prim_counter + sidx
    ];
    s_basis_array[sidx +  64] = basis_array[sblock_prim_counter + sidx +
    64];
    s_basis_array[sidx + 128] = basis_array[sblock_prim_counter + sidx +
    128];
    s_basis_array[sidx + 192] = basis_array[sblock_prim_counter + sidx +
    192];
    prim_counter -= sblock_prim_counter;
    __syncthreads();
  }
  for (prim=0; prim < maxprim;  prim++) {
    float exponent       = s_basis_array[prim_counter    ];
    float contract_coeff = s_basis_array[prim_counter + 1];
    contracted_gto += contract_coeff * __expf(-exponent*dist2);
    prim_counter += 2;
  }
[… continue on to angular momenta loop …]
```

Shared memory tiles:

- Tiles are checked and loaded, if necessary, immediately prior to entering key arithmetic loops
- Adds additional control overhead to loops, even with optimized implementation

VMD MO GPU Kernel Snippet:

Fermi/Kepler kernel based on L1 cache

```
[… outer loop over atoms …]
  // loop over the shells/basis funcs belonging to this atom
  for (shell=0; shell < maxshell; shell++) {
    float contracted_gto = 0.0f;
    int maxprim    = shellinfo[(shell_counter<<4)     ];
    int shell_type = shellinfo[(shell_counter<<4) + 1];
    for (prim=0; prim < maxprim; 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;
   }
   […continue on to angular momenta loop …]
```

L1 cache:

- Simplifies code!
- Reduces control overhead
- Gracefully handles arbitrary-sized problems
- Matches performance of constant memory on Fermi, slower on Kepler (GK104)

VMD Single-GPU Molecular Orbital Performance Results for C_{60} on Fermi

Intel X5550 CPU, GeForce GTX 480 GPU

Kernel	Cores/GPUs	Runtime (s)	Speedup
Xeon 5550 ICC-SSE	1	30.64	1.0
Xeon 5550 ICC-SSE	8	4.13	7.4
CUDA shared mem	1	0.37	83
CUDA L1-cache (16KB)	**1**	**0.27**	**113**
CUDA const-cache	1	0.26	117
CUDA const-cache, zero-copy	1	0.25	122

Fermi GPUs have caches: match perf. of hand-coded shared memory kernels. Zero-copy memory transfers improve overlap of computation and host-GPU I/Os.

Preliminary Single-GPU Molecular Orbital Performance Results for C_{60} on Kepler

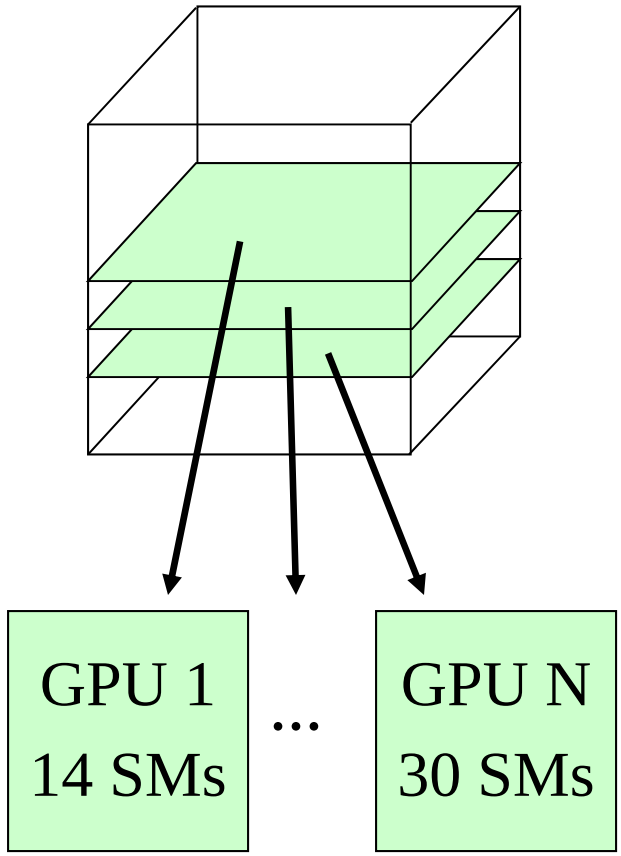
Intel X5550 CPU, GeForce GTX 680 GPU

Kernel	Cores/GPUs	Runtime (s)	Speedup
Xeon 5550 ICC-SSE	1	30.64	1.0
Xeon 5550 ICC-SSE	8	4.13	7.4
CUDA shared mem	1	0.264	116
CUDA L1-cache (16KB)	**1**	**0.228**	**134**
CUDA const-cache	1	0.104	292
CUDA const-cache, zero-copy	1	0.0938	326

Kepler GK104 (GeForce 680) seems to strongly prefer the constant cache kernels vs. the others. Not yet sure of the cause for this significant disparity vs. Fermi.

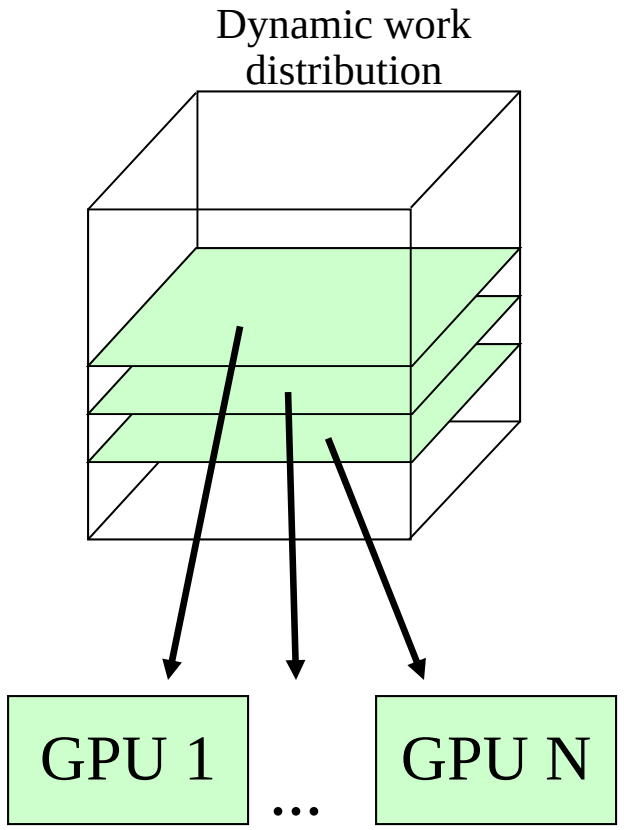
Multi-GPU Load Balance

- 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
}
```



Example Multi-GPU Latencies
4 C2050 GPUs, Intel Xeon 5550

6.3us	CUDA empty kernel (immediate return)
9.0us	Sleeping barrier primitive (non-spinning barrier that uses POSIX condition variables to prevent idle CPU consumption while workers wait at the barrier)
14.8us	pool wake, host fctn exec, sleep cycle (no CUDA)
30.6us	pool wake, 1x(tile fetch, simple CUDA kernel launch), sleep
1817.0us	pool wake, 100x(tile fetch, simple CUDA kernel launch), sleep

VMD Multi-GPU Molecular Orbital Performance Results for C_{60}

Intel X5550 CPU, 4x GeForce GTX 480 GPUs,

Kernel	Cores/GPUs	Runtime (s)	Speedup
Intel X5550-SSE	1	30.64	1.0
Intel X5550-SSE	8	4.13	7.4
GeForce GTX 480	1	0.255	120
GeForce GTX 480	2	0.136	225
GeForce GTX 480	3	0.098	312
GeForce GTX 480	4	0.081	378

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

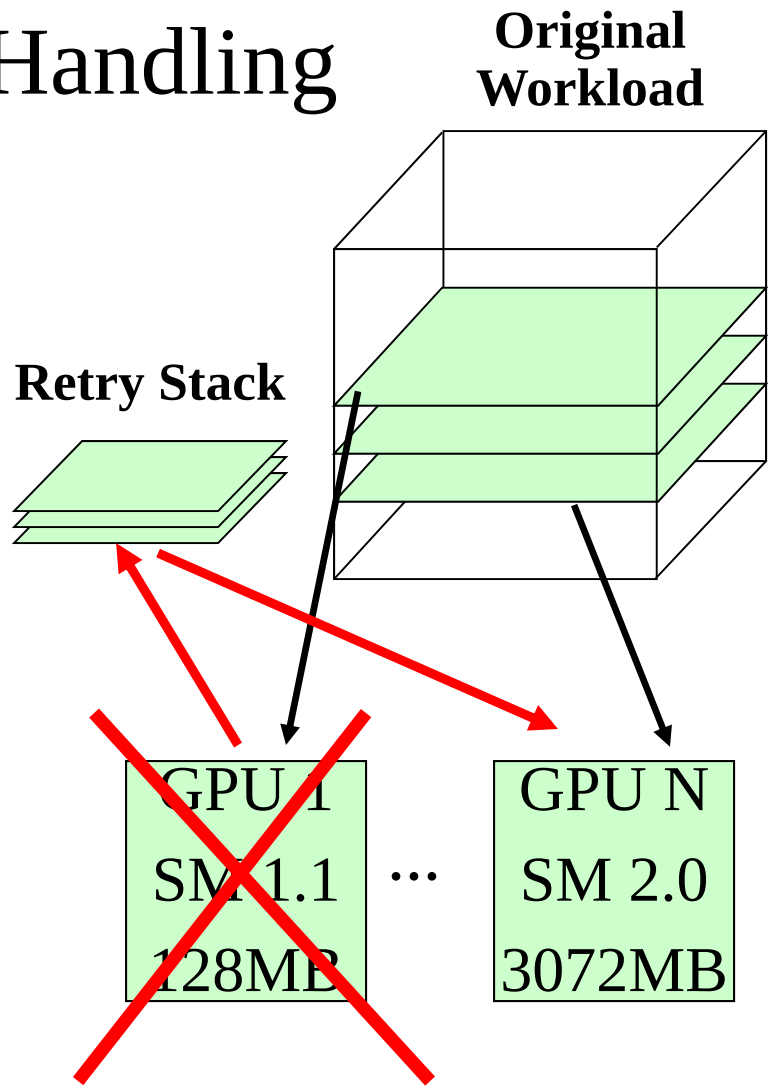
Molecular Orbital Dynamic Scheduling Performance with Heterogeneous GPUs

Kernel	Cores/GPUs	Runtime (s)	Speedup
Intel X5550-SSE	1	30.64	1.0
Quadro 5800	1	0.384	79
Tesla C2050	1	0.325	94
GeForce GTX 480	1	0.255	120
GeForce GTX 480 + Tesla C2050 + Quadro 5800	3	0.114	268 (91% of ideal perf)

Dynamic load balancing enables mixture of GPU generations, SM counts, and clock rates to perform well.

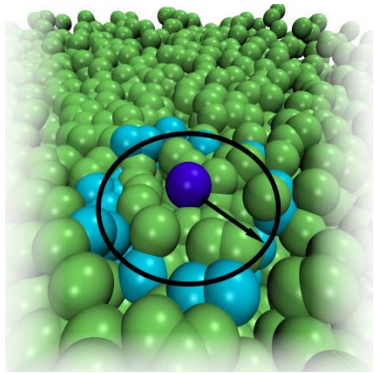
Multi-GPU Runtime Error/Exception Handling

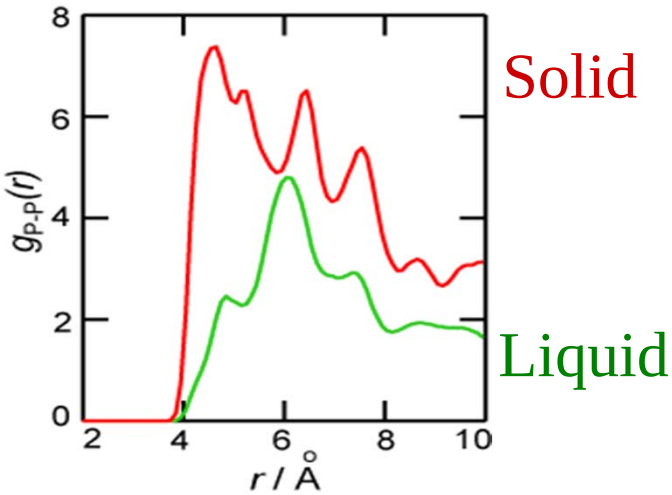
- Competition for resources from other applications can cause runtime failures, e.g. GPU out of memory half way through an algorithm
- Handle exceptions, e.g. convergence failure, NaN result, insufficient compute capability/features
- Handle and/or reschedule failed tiles of work



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 compute the entire histogram in a **single pass**, regardless of the problem size or number of 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…
- 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
- SSE can't be used effectively: lacks ability to "scatter" to memory (e.g. no *scatter-add)*

Computing RDFs on the GPU

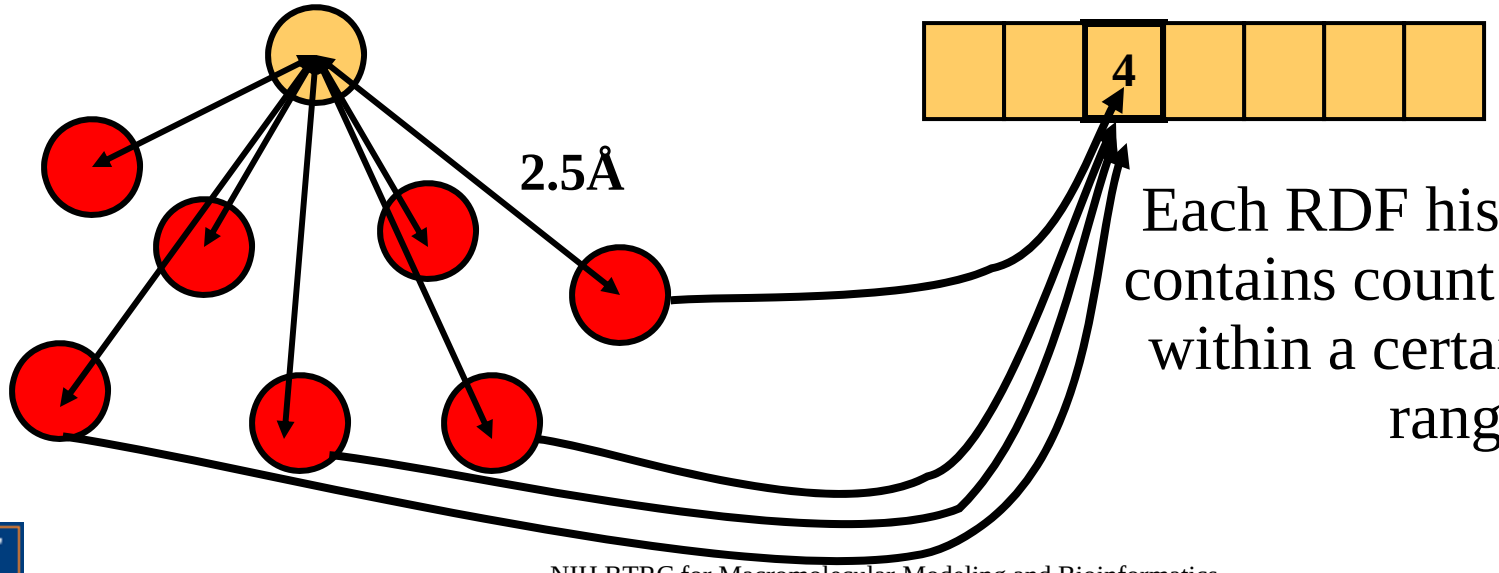
- Need tens of thousands of independent threads
- Each GPU thread computes one or more atom pair distances
- Performance is limited by the speed of histogramming
- Histograms are best stored in fast on-chip shared memory
- Small size of shared memory severely constrains the range of viable histogram 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

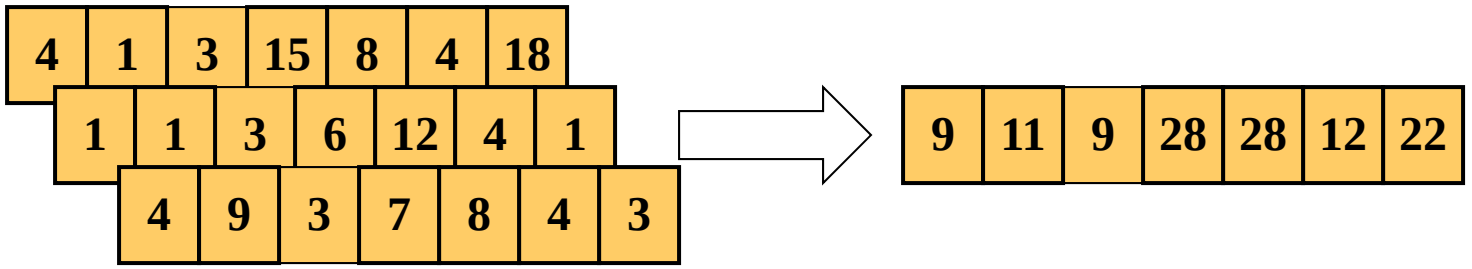
- Each warp maintains its own **private** histogram in on-chip shared memory
- Each thread in the warp computes an atom pair distance and updates a histogram bin in parallel
- Conflicting histogram bin updates are resolved using one of two schemes:
  - Shared memory write combining with thread-tagging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

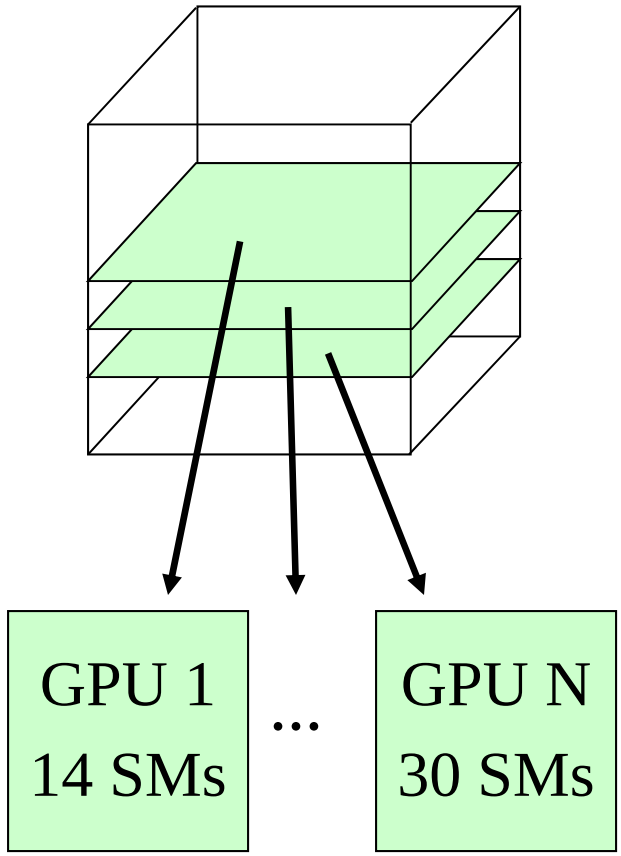


Preventing Integer Overflows

- 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 increments large (e.g. long, or double) histogram counters in host memory after each kernel completes

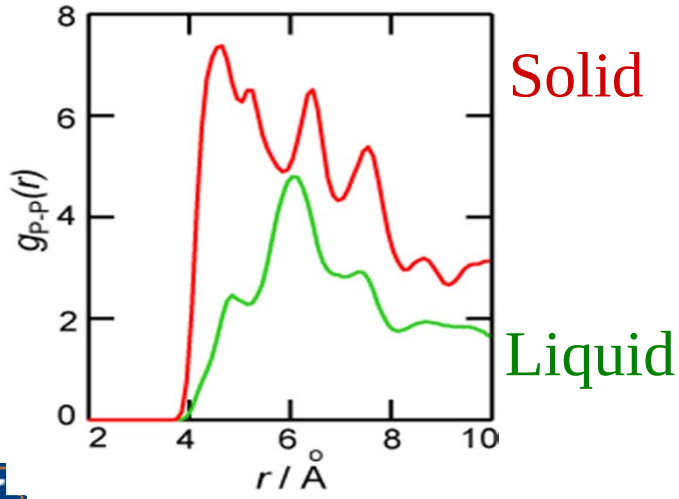
Multi-GPU RDF Calculation

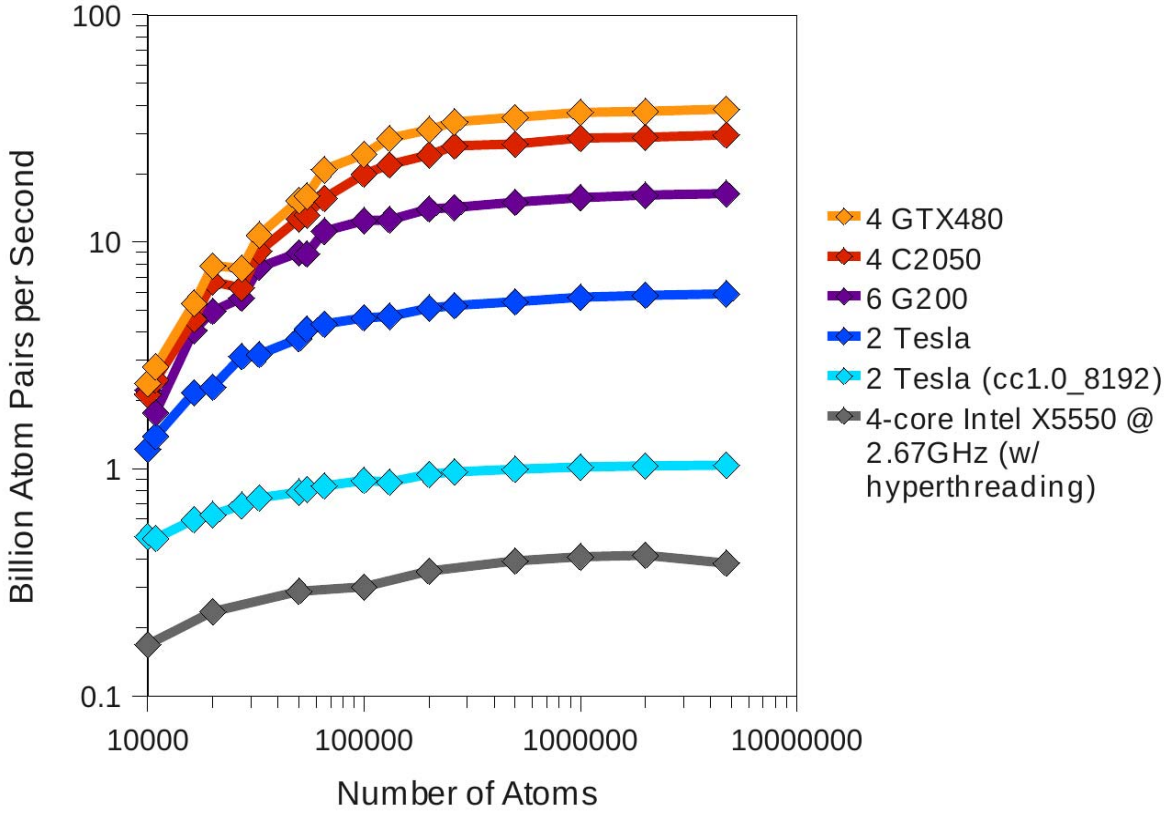
- Distribute combinations of tiles of atoms and histogram regions to different GPUs
- Decomposed over two dimensions to obtain enough work units to balance GPU loads
- Each GPU computes its own histogram, and all results are combined for final histogram



Multi-GPU 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. (submitted)

Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- NCSA Blue Waters Team
- NCSA Innovative Systems Lab
- 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/

- **Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.** M. Krone, J. Stone, T. Ertl, and K. Schulten. *In proceedings EuroVis 2012*, 2012. (In-press)
- **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.