High Performance Molecular Simulation, Visualization, and Analysis on GPUs

John Stone

Theoretical and Computational Biophysics Group

Beckman Institute for Advanced Science and Technology

University of Illinois at Urbana-Champaign

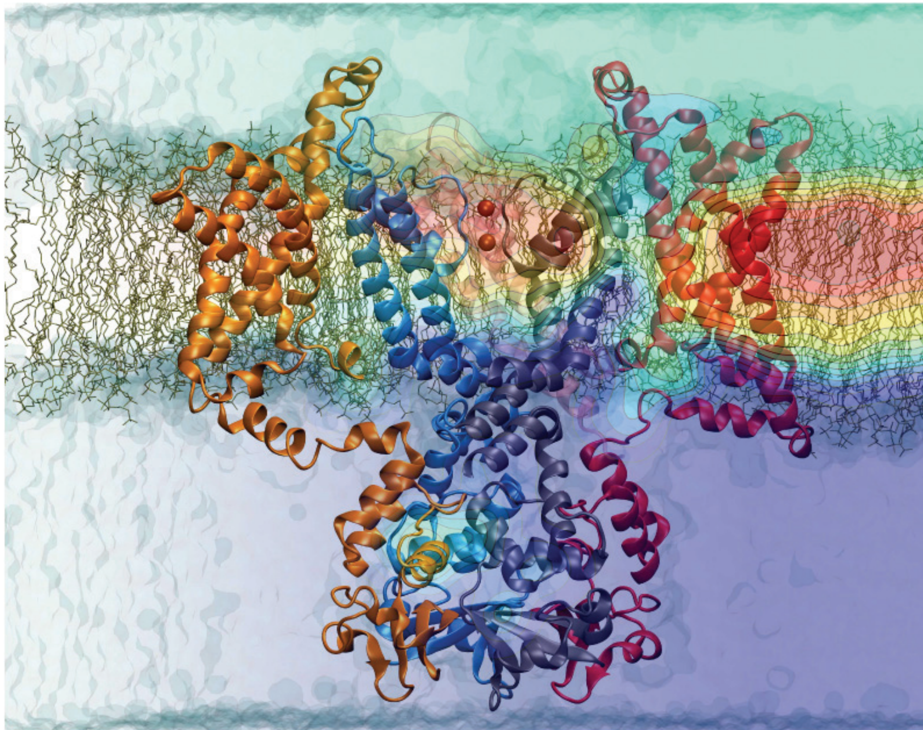
http://www.ks.uiuc.edu/Research/gpu/

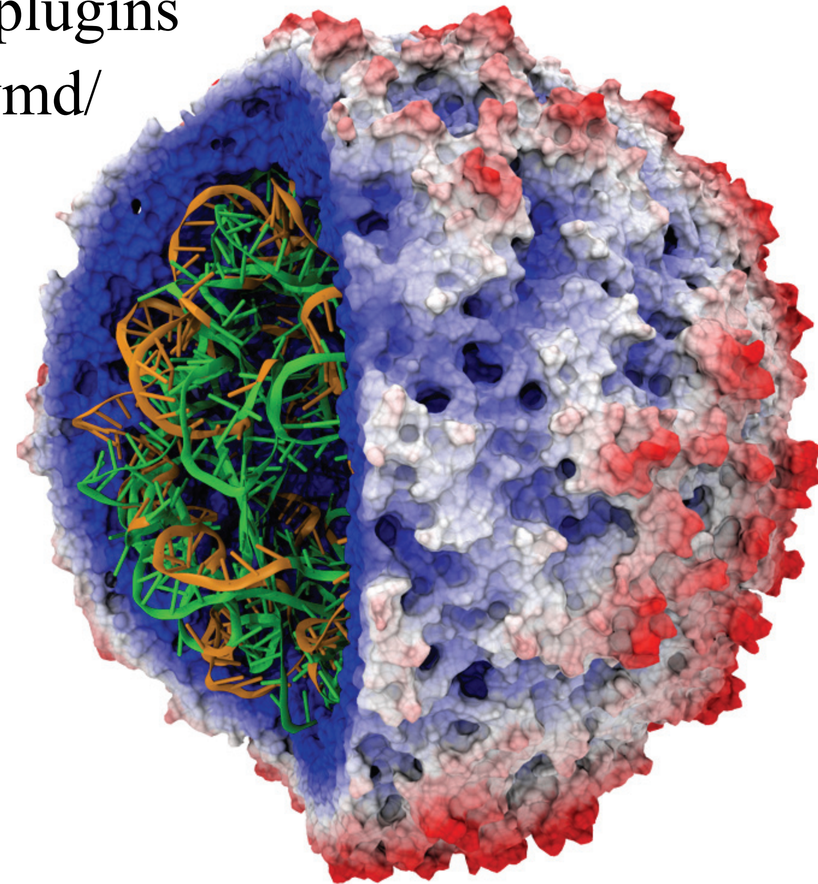
Bio-molecular Simulations on Future Computing Architectures

Oak Ridge National Laboratory, September 16, 2010

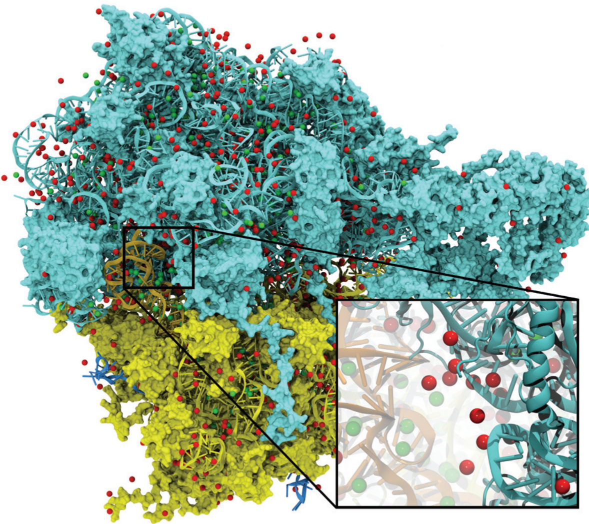
VMD – “Visual Molecular Dynamics”

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



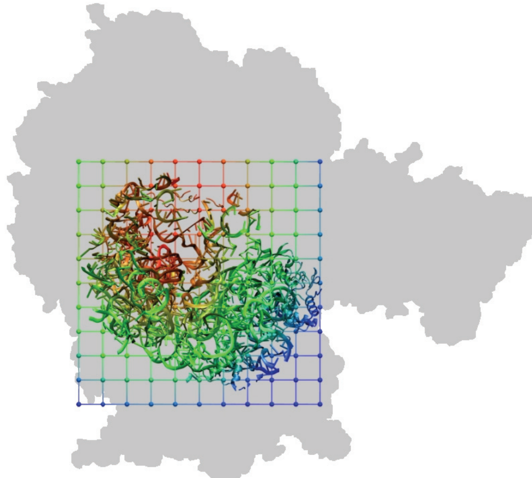


CUDA Algorithms in VMD



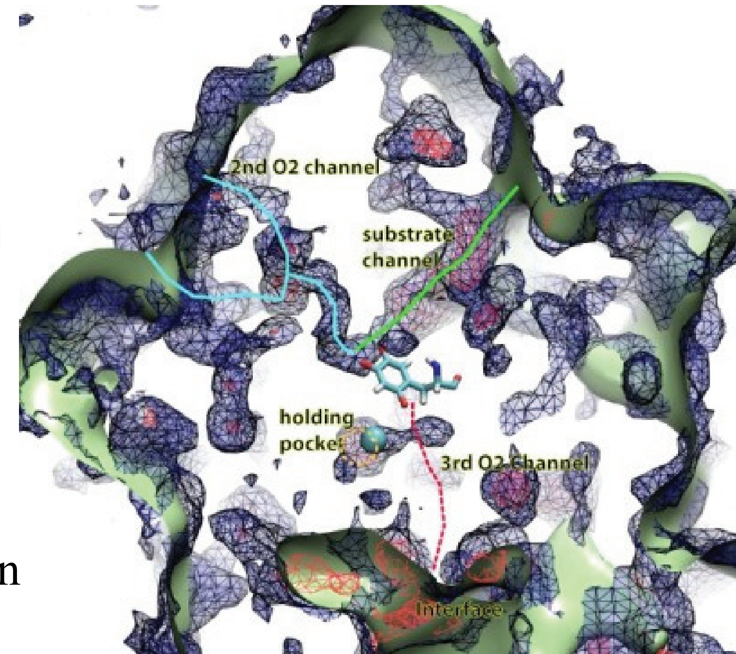
Ion placement

20x to 44x faster



Electrostatic field calculation

31x to 44x faster

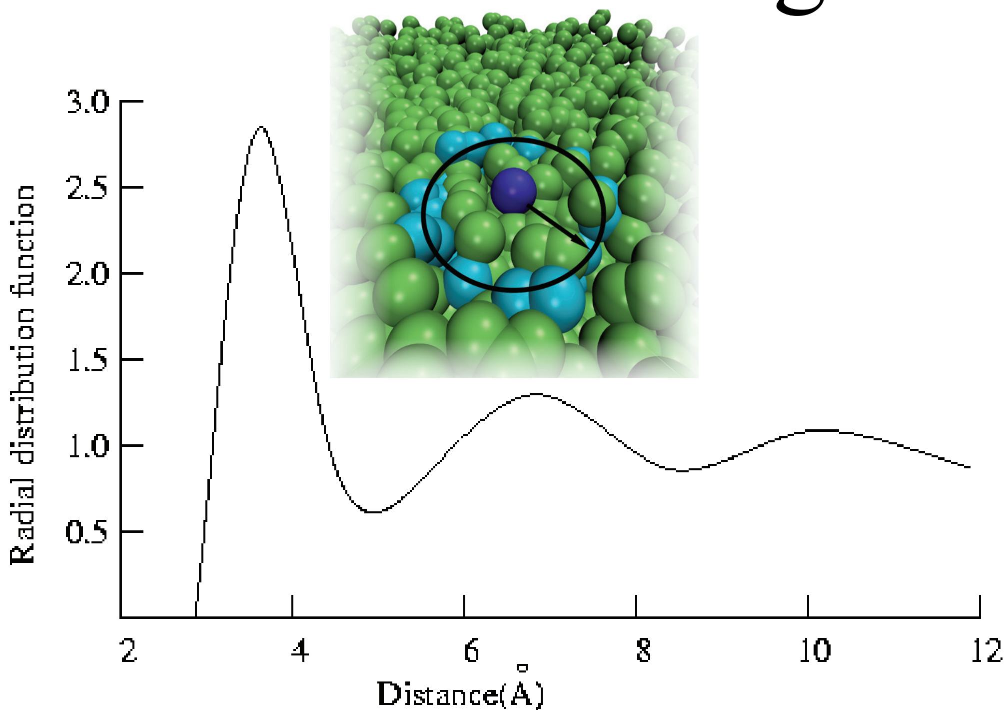


Imaging of gas migration pathways in proteins with implicit ligand sampling

20x to 30x faster

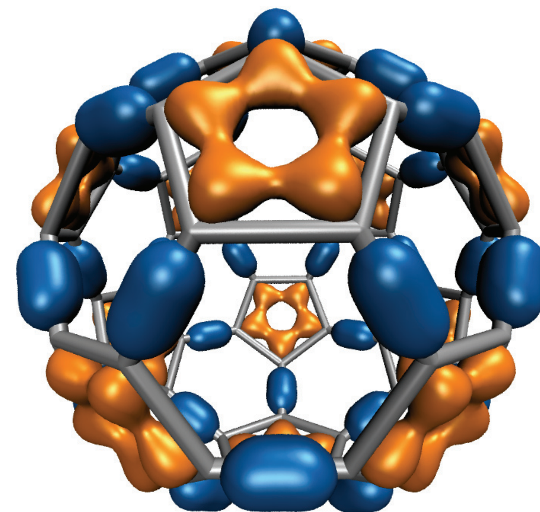
GPU: massively parallel co-processor

CUDA Algorithms in VMD



Radial distribution functions

30x to 92x faster



Molecular orbital

calculation and display

100x to 120x faster

GPU: massively parallel co-processor

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 on GPU clusters
- Updating existing CUDA kernels to take advantage of new hardware features on the latest NVIDIA "Fermi" GPUs
- Adaptation of CUDA kernels to OpenCL, evaluation of JIT techniques with OpenCL

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

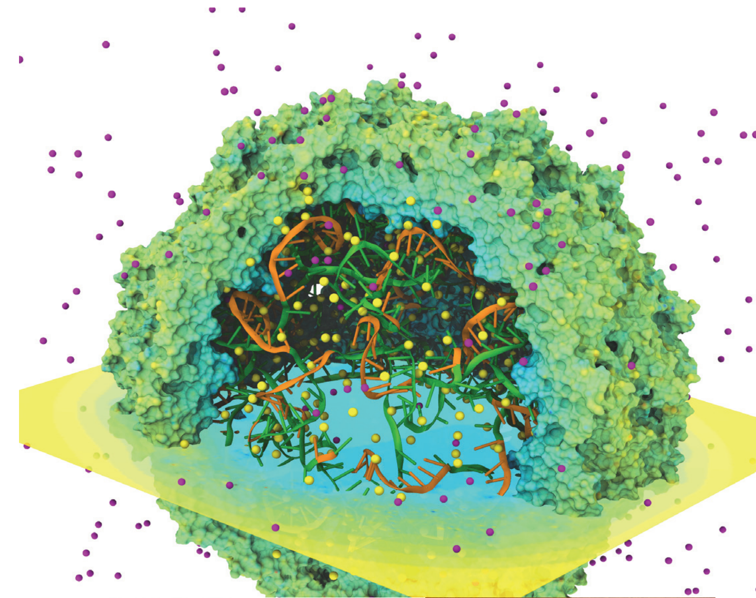
AC GPU Cluster Power Measurements

State	Host Peak (Watt)	Tesla Peak (Watt)	Host power factor (pf)	Tesla power factor (pf)
power off	4	10	.19	.31
pre-GPU use idle	173	178	.98	.96
after NVIDIA driver module unload/reload[1]	173	178	.98	.96
after deviceQuery[2] (idle)	173	365	.99	.99
GPU memtest #10 (stress)	269	745	.99	.99
after memtest kill (idle)	172	367	.99	.99
after NVIDIA module unload/reload[3] (idle)	172	367	.99	.99
VMD Multiply-add	268	598	.99	.99
NAMD GPU STMV	321	521	.97-1.0	.85-1.0[4]

1. Kernel module unload/reload does not increase Tesla power
2. Any access to Tesla (e.g., deviceQuery) results in doubling power consumption after the application exits
3. Note that second kernel module unload/reload cycle does not return Tesla power to normal, only a complete reboot can
4. Power factor stays near one except while load transitions. Range varies with consumption swings

Energy Efficient GPU Computing of Time-Averaged Electrostatics

- **1.5 hour** job reduced to **3 min**
- Electrostatics of thousands of trajectory frames averaged
- Per-node power consumption on NCSA GPU cluster:
  - CPUs-only: 299 watts
  - CPUs+GPUs: 742 watts
- GPU Speedup: **25.5x**
- Power efficiency gain: **10.5x**



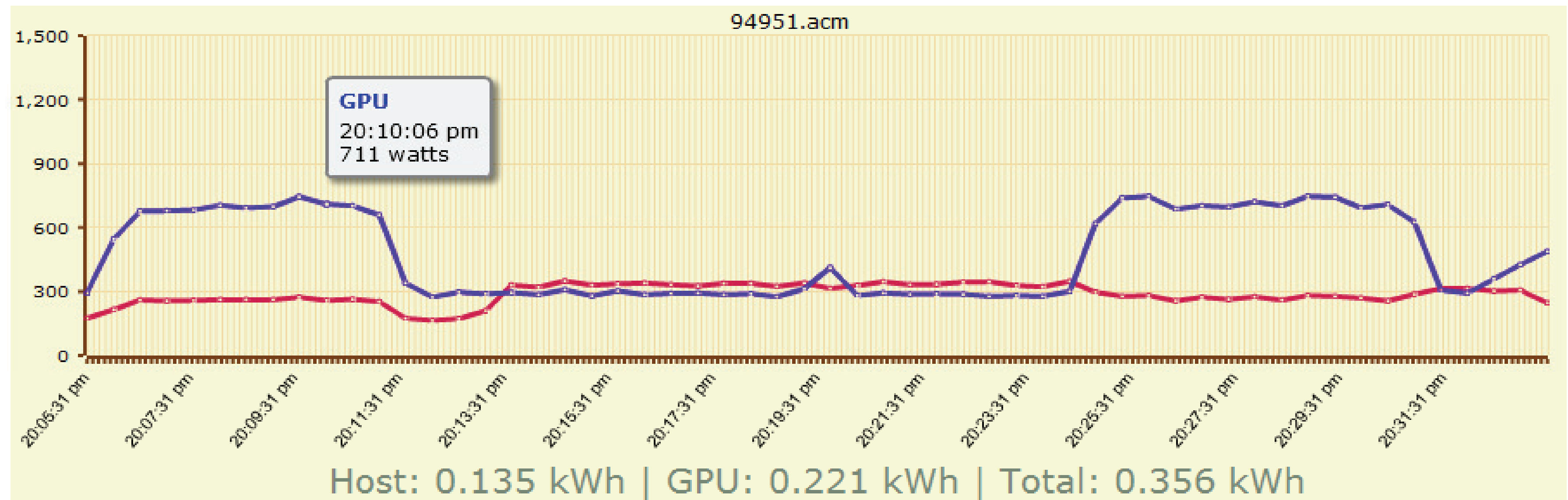
NCSA "AC" GPU cluster and Tweet-a-watt wireless power monitoring device

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,* 2010. In press.

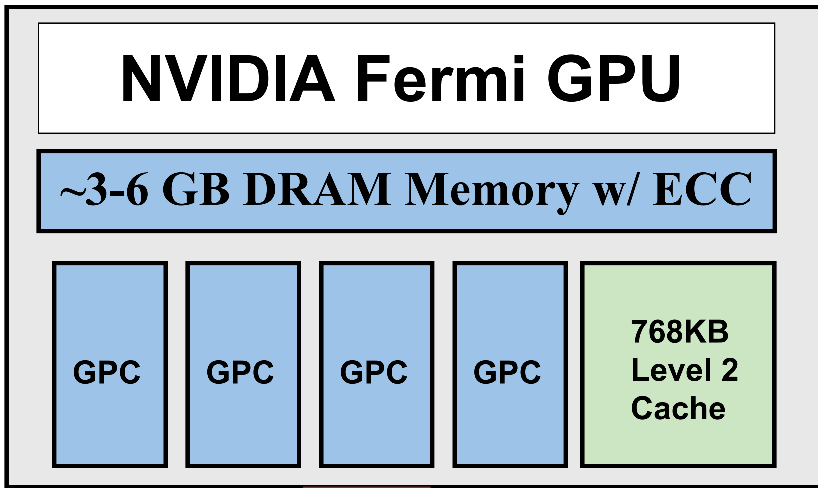
Power Profiling: Example Log

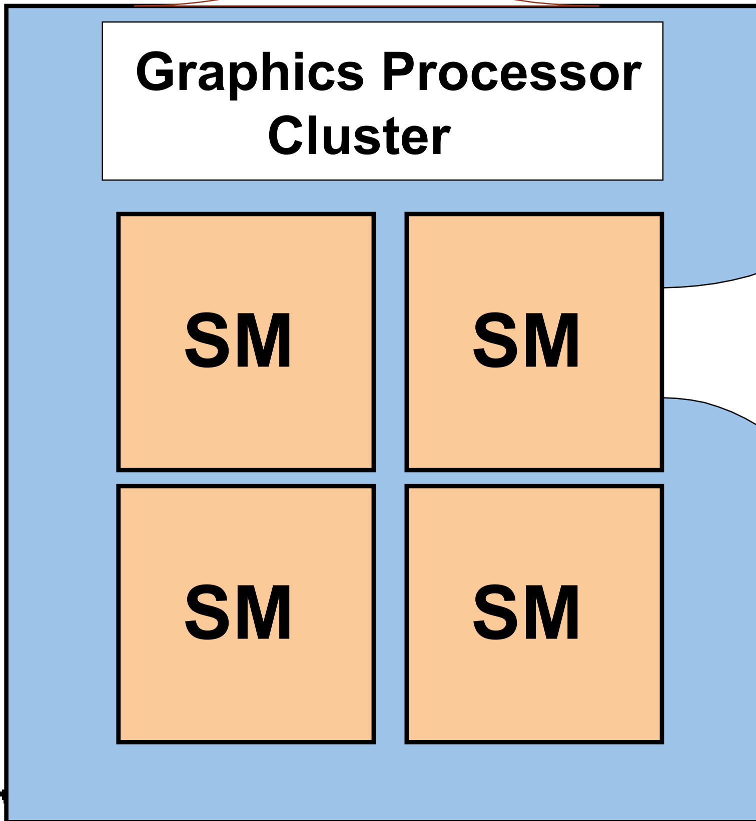


- Mouse-over value displays
- Under curve totals displayed
- If there is user interest, we may support calls to add custom tags from application

Fermi GPUs Bring Higher Performance and Easier Programming

- NVIDIA's latest "Fermi" GPUs bring:
  - Greatly increased peak single- and double-precision arithmetic rates
  - Moderately increased global memory bandwidth
  - Increased capacity on-chip memory partitioned into shared memory and an L1 cache for global memory
  - Concurrent kernel execution
  - Bidirectional asynchronous host-device I/O
  - ECC memory, faster atomic ops, many others…





Streaming Multiprocessor

64KB Constant Cache

64 KB L1 Cache / Shared Memory

SP SP SP SP — LDST LDST — SFU (8 rows of SP ×4 and LDST ×2; 4 SFU)

Tex Tex Tex Tex

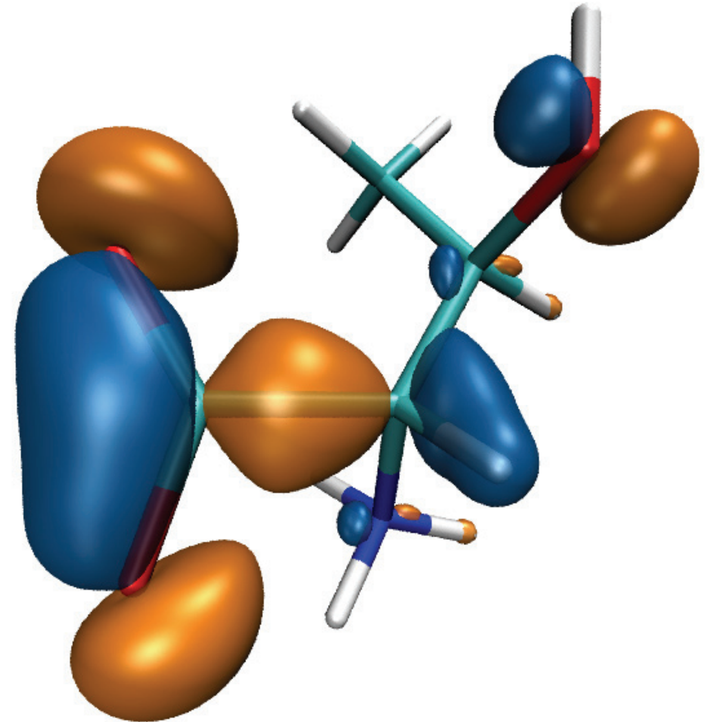
Texture Cache

Early Experiences with Fermi

- The 2x single-precision and up to 8x double-precision arithmetic performance increases vs. GT200 cause more kernels to be memory bandwidth bound…
- …unless they make effective use of the larger on-chip shared mem and L1 global memory cache to improve performance
- **Arithmetic is cheap, memory references are costly** (trend is certain to continue & intensify…)
- Register consumption and GPU "occupancy" are a bigger concern with Fermi than with GT200

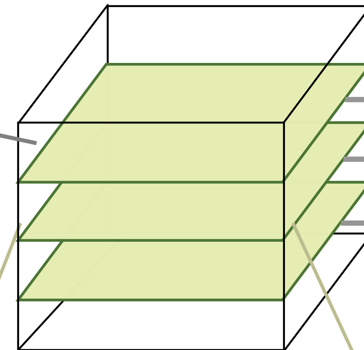
Computing Molecular Orbitals

- 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

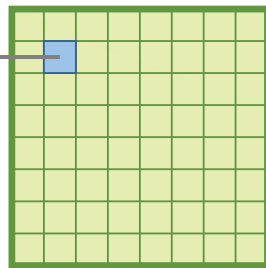
MO 3-D lattice decomposes into 2-D slices (CUDA grids)

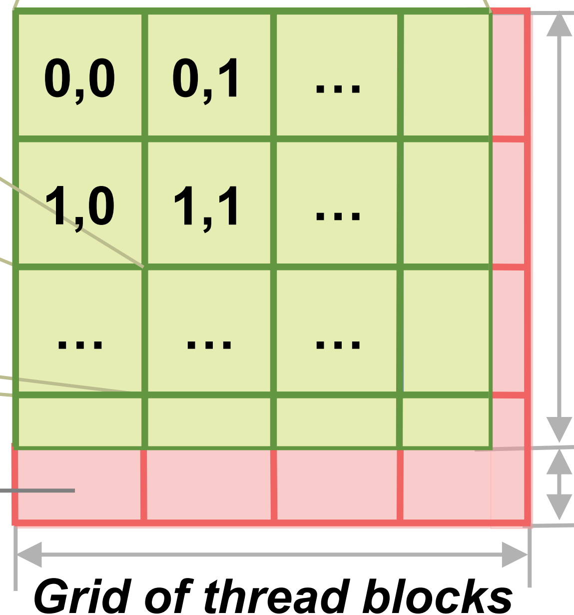


Lattice can be computed using multiple GPUs

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

Each thread computes one MO lattice point.





Threads producing results that are used

Threads producing results that are discarded

Padding optimizes global memory performance, guaranteeing coalesced global memory accesses

Grid of thread blocks

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 kernel based on L1 cache

```
[… outer loop over atoms …]
 // loop over the shells belonging to this atom (or basis function)
 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

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.

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

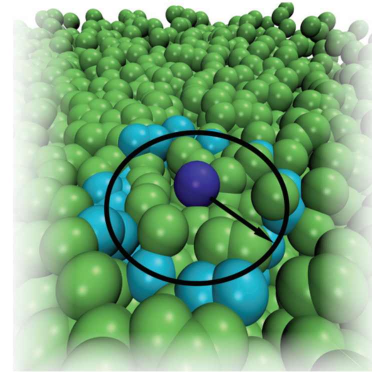
Intel X5550 CPU, 4x GeForce GTX 480 GPUs,

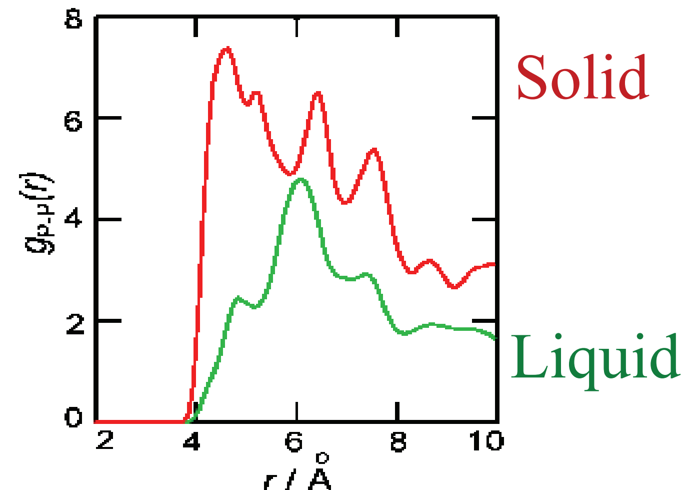
Kernel	Cores/GPUs	Runtime (s)	Speedup
Intel X5550-SSE	1	30.64	1.0
Intel X5550-SSE	8	4.13	7.4
GeForce GTX 480	1	0.255	120
GeForce GTX 480	2	0.136	225
GeForce GTX 480	3	0.098	312
GeForce GTX 480	4	0.081	378

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

Radial Distribution Function

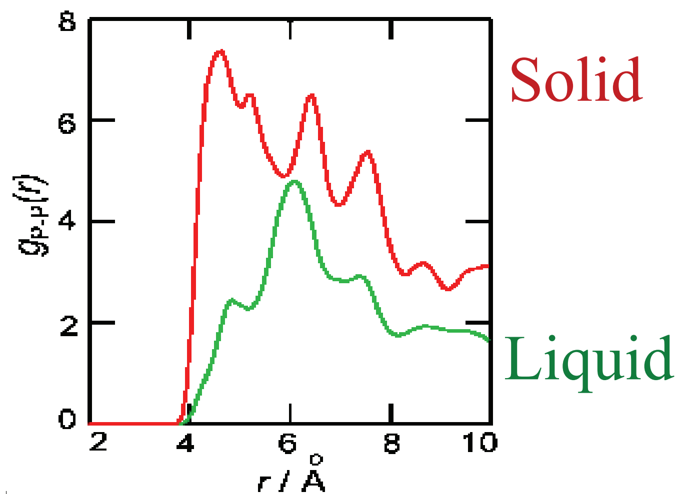
- RDFs describes how atom density varies with distance
- Decays toward unity with increasing distance, for liquids
- Sharp peaks appear for solids, according to crystal structure, etc.
- Quadratic time complexity $O(N^2)$

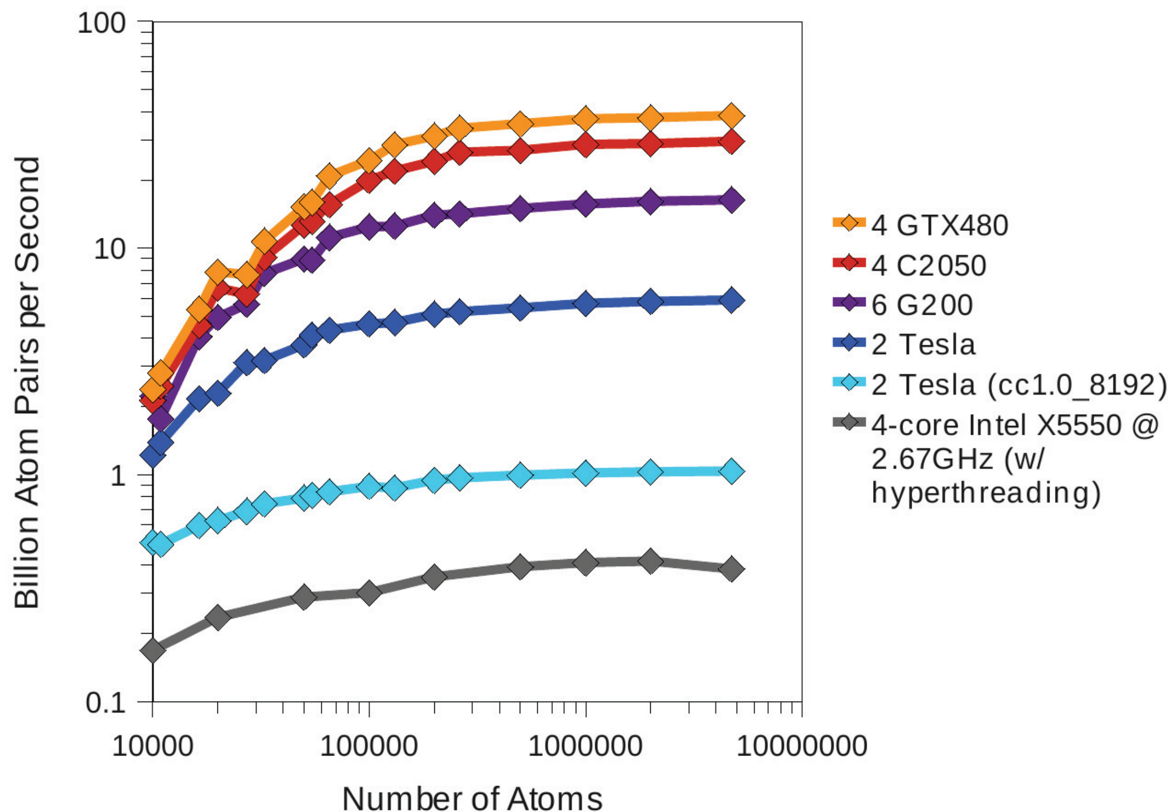




Radial Distribution Functions on Fermi

- 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





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
- Performance tends to be limited primarily by the histogram update step

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…

What About x86 SSE for RDF Histogramming?

- Atom pair distances can be computed four-at-a-time without too much difficulty
- Current generation x86 CPUs don't provide SSE instructions to allow individual SIMD units to scatter results to arbitrary memory locations
- Since the fetch-and-increment operation must be done with scalar code, the histogram updates are a performance bottleneck for the CPU

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; not suited for this purpose
- 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

VMD Multi-core CPU RDF Implementation

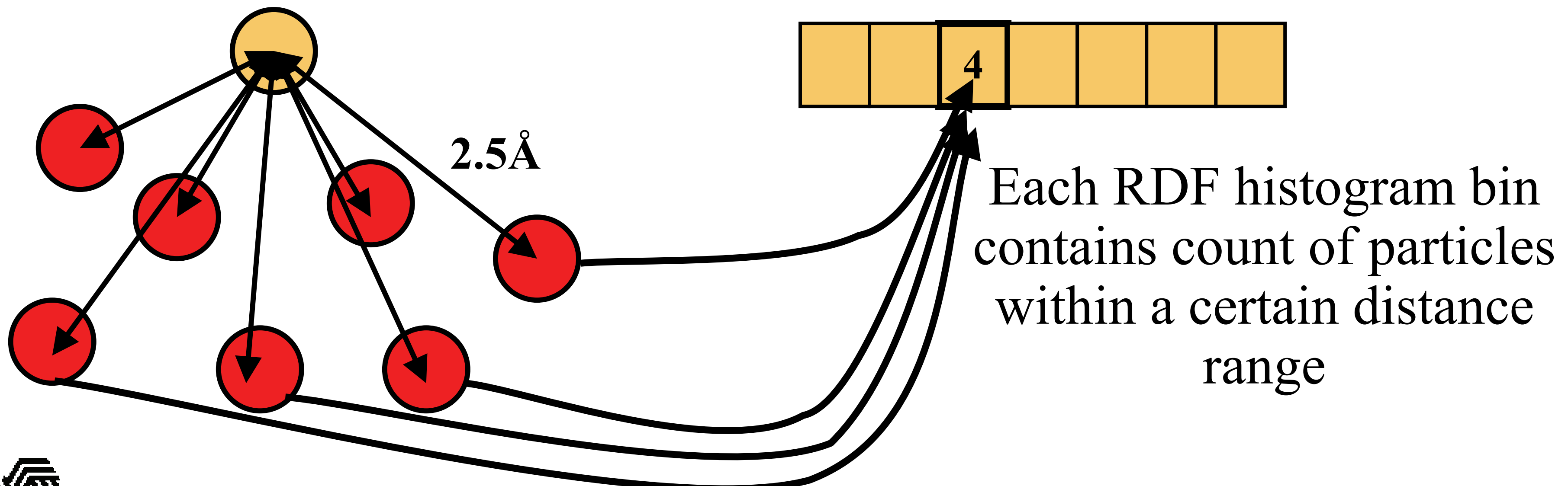
- Each CPU worker thread processes a subset of atom pair distances, maintaining its own histogram
- Threads acquire "tiles" of work from a dynamic work scheduler built into VMD
- When all threads have completed their histograms, the main thread combines the independently computed histograms into a final result histogram
- CPUs compute the entire histogram in a **single pass**, regardless of the problem size or number of histogram bins

Computing RDFs on the GPU

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

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..



Computing Atom Pair Distances on the GPU

- Distances are computed using nearest image convention in the case of periodic boundary conditions
- Since all atom pair combinations will ultimately be computed, the 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

GPU Atom Pair Distance Calculation

- Divide A and B atom selections into fixed size blocks
- Load a large block of A into constant memory
- Load small block of B into thread block's registers
- Each thread in a thread block computes atom pair distances between its atom and all atoms in constant memory, incrementing appropriate histogram bins until all A/B block atom pairs are processed
- Next block(s) are loaded, repeating until done…

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**

Per-warp Histogram Approach

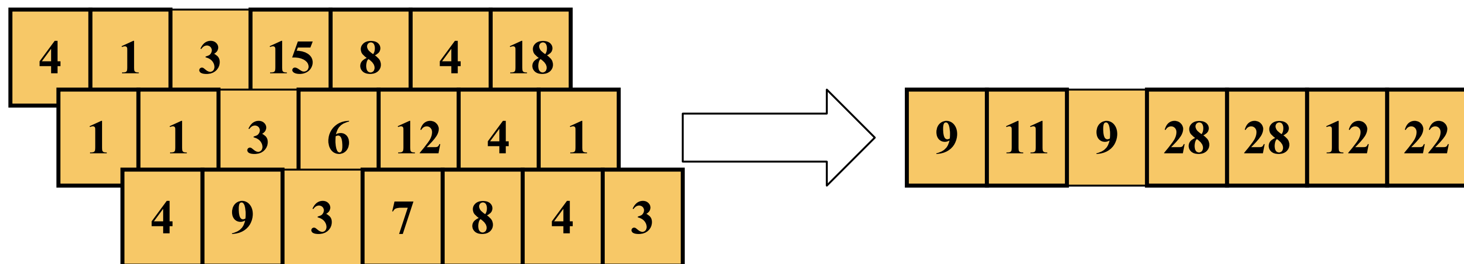
- Each warp maintains its own **private** histogram in on-chip shared memory
- Each thread in the warp computes an atom pair distance and updates a histogram bin in parallel
- Conflicting histogram bin updates are resolved using one of two schemes:
  - Shared memory write combining with thread-tagging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

RDF Inner Loops (Abbrev.)

```
// 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

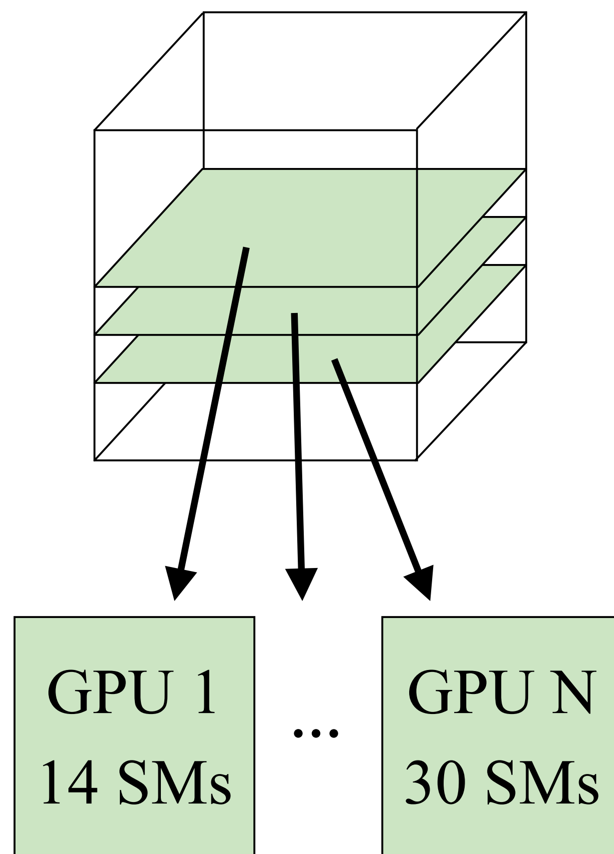


Preventing Integer Overflows

- 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



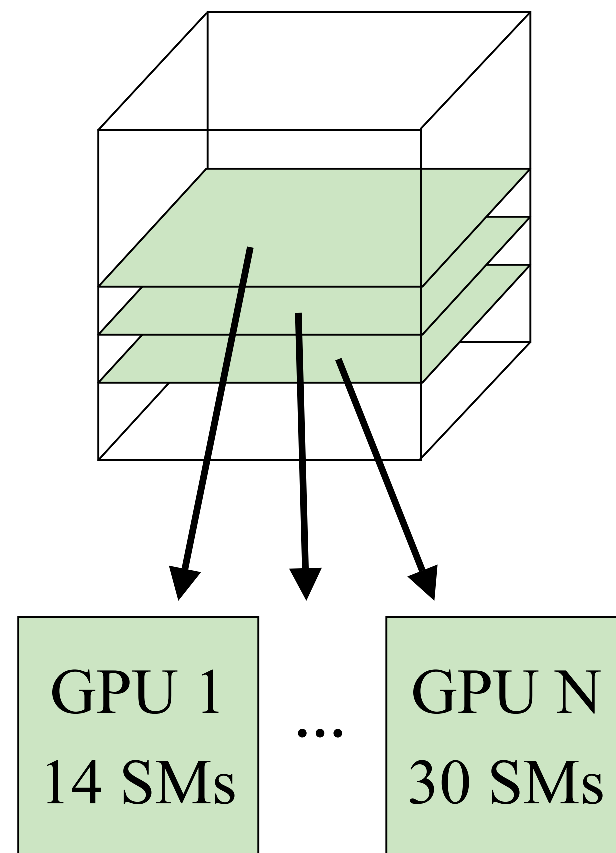
Example Multi-GPU Latencies

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

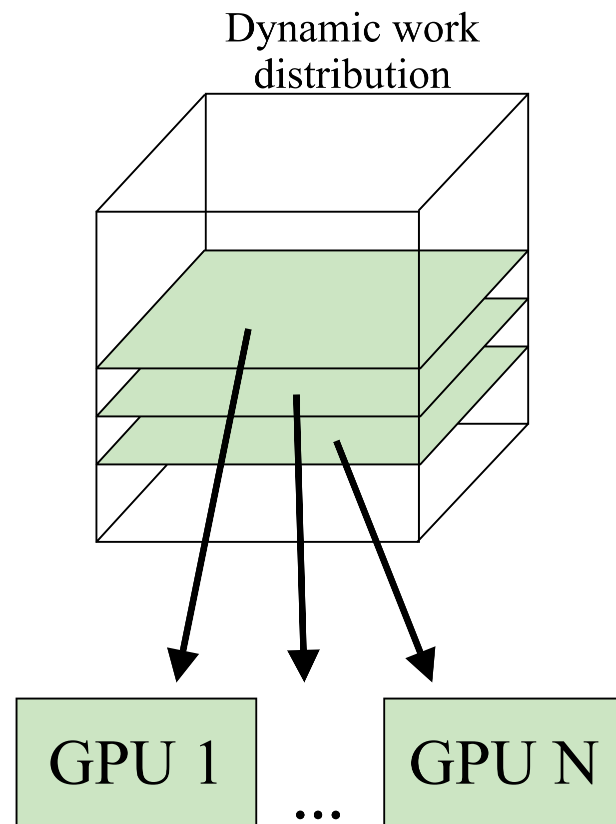
Multi-GPU Load Balance

- 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



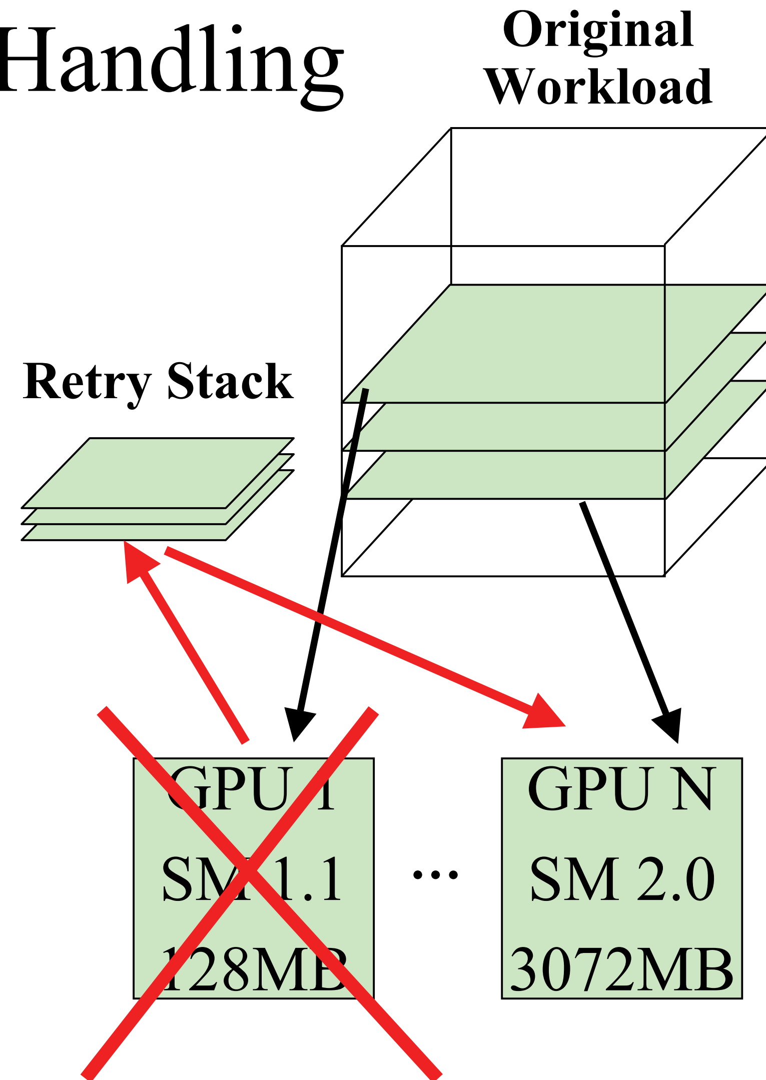
Multi-GPU Dynamic Work Distribution

```
// 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
}
```

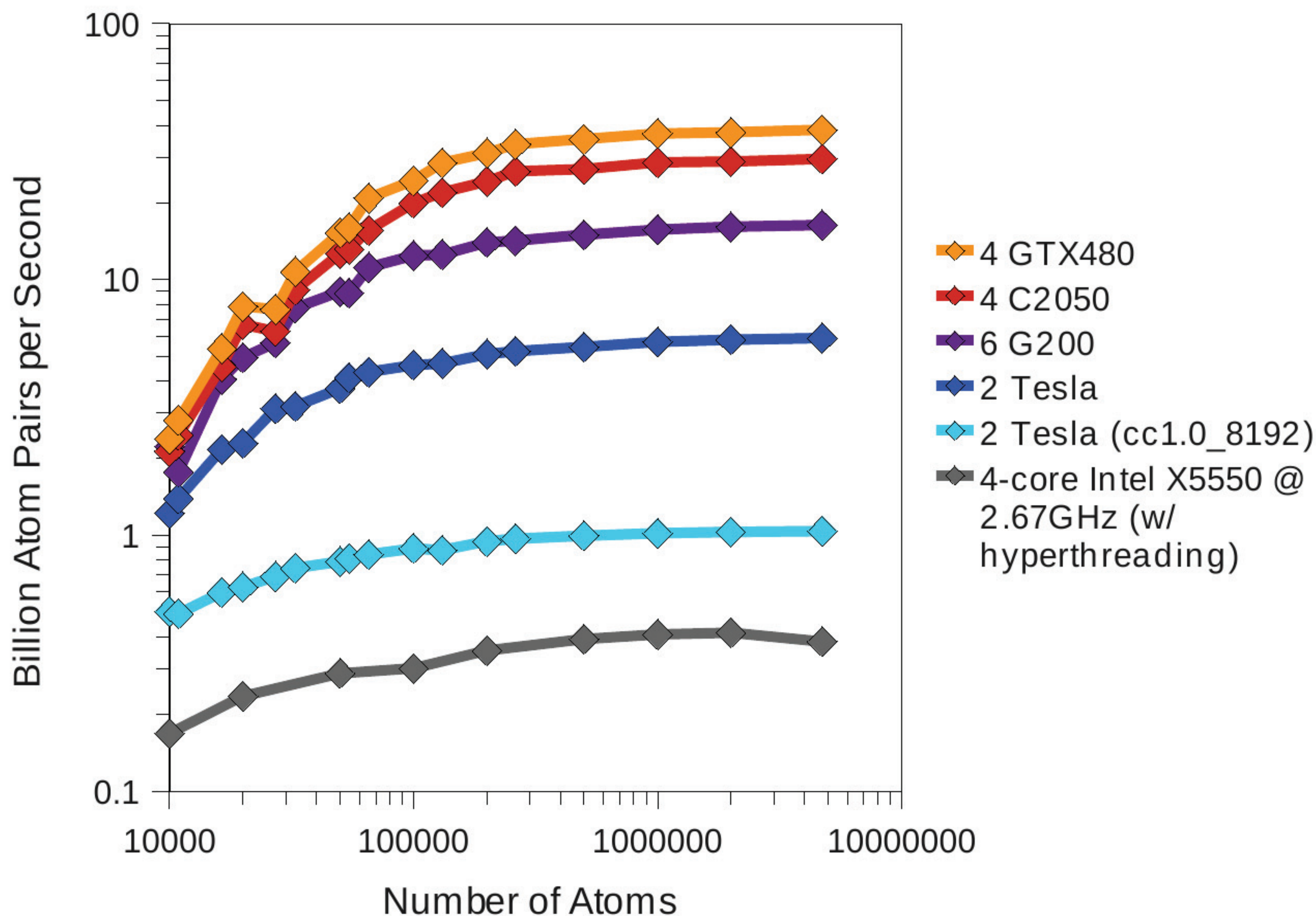


Multi-GPU Runtime Error/Exception Handling

- 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 vs. Problem Size



Acknowledgements

- Ben Levine and Axel Kohlmeyer at Temple University
- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- NVIDIA CUDA Center of Excellence, University of Illinois at Urbana-Champaign
- NCSA Innovative Systems Lab
- The CUDA team at NVIDIA
- NIH support: P41-RR05969

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. *The Work in Progress in Green Computing,* 2010. In press.
- **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/

- **Probing Biomolecular Machines with Graphics Processors**. J. Phillips, J. Stone. *Communications of the ACM,* 52(10):34-41, 2009.
- **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.
- **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.