Faster, Cheaper, and Better Science: Molecular Modeling 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/

Fall National Meeting of the American Chemical Society,
Boston, MA, August 22, 2010

Potential Impact of GPU Computing

- State-of-the-art GPUs achieve over 1.0 TFLOPS single-precision, and 0.5 TFLOPS double-precision
- GPUs are an inexpensive commodity technology that is already ubiquitous, and are well-supported by existing operating systems, unlike previous accelerators
- End-users do not need to become HPC gurus to run GPU-accelerated software

GPU Hardware Platforms

- GPUs are equally applicable to accelerating applications on laptops, desktops, and supercomputers
- GPUs are available in HPC and data-center-friendly rack mounts, with ECC protected memory, supporting hardware monitoring features of interest to cluster builders
- 2 of the top 10 supercomputers in the latest Top500 list are currently GPU-accelerated systems
- 2 of the top 10 supercomputers in the Green500 list are GPU-accelerated, and are over 3x more efficient than the average of all Green500 systems!

Programmable Graphics Hardware

Groundbreaking research systems:

AT&T Pixel Machine (1989):

82 x DSP32 processors

UNC PixelFlow (1992-98):

64 x (PA-8000 +

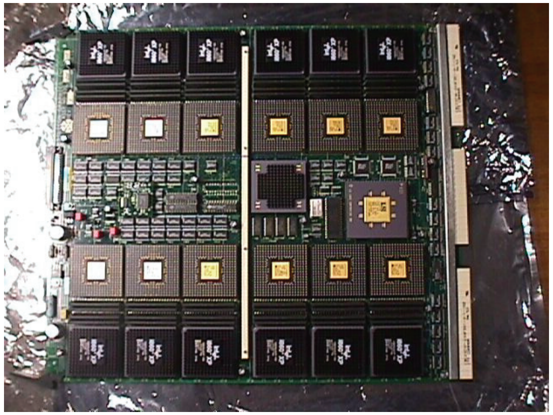
8,192 bit-serial SIMD)

SGI RealityEngine (1990s):

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

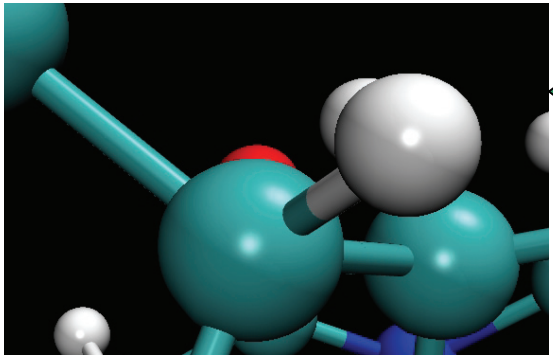
All mainstream GPUs now incorporate fully programmable processors

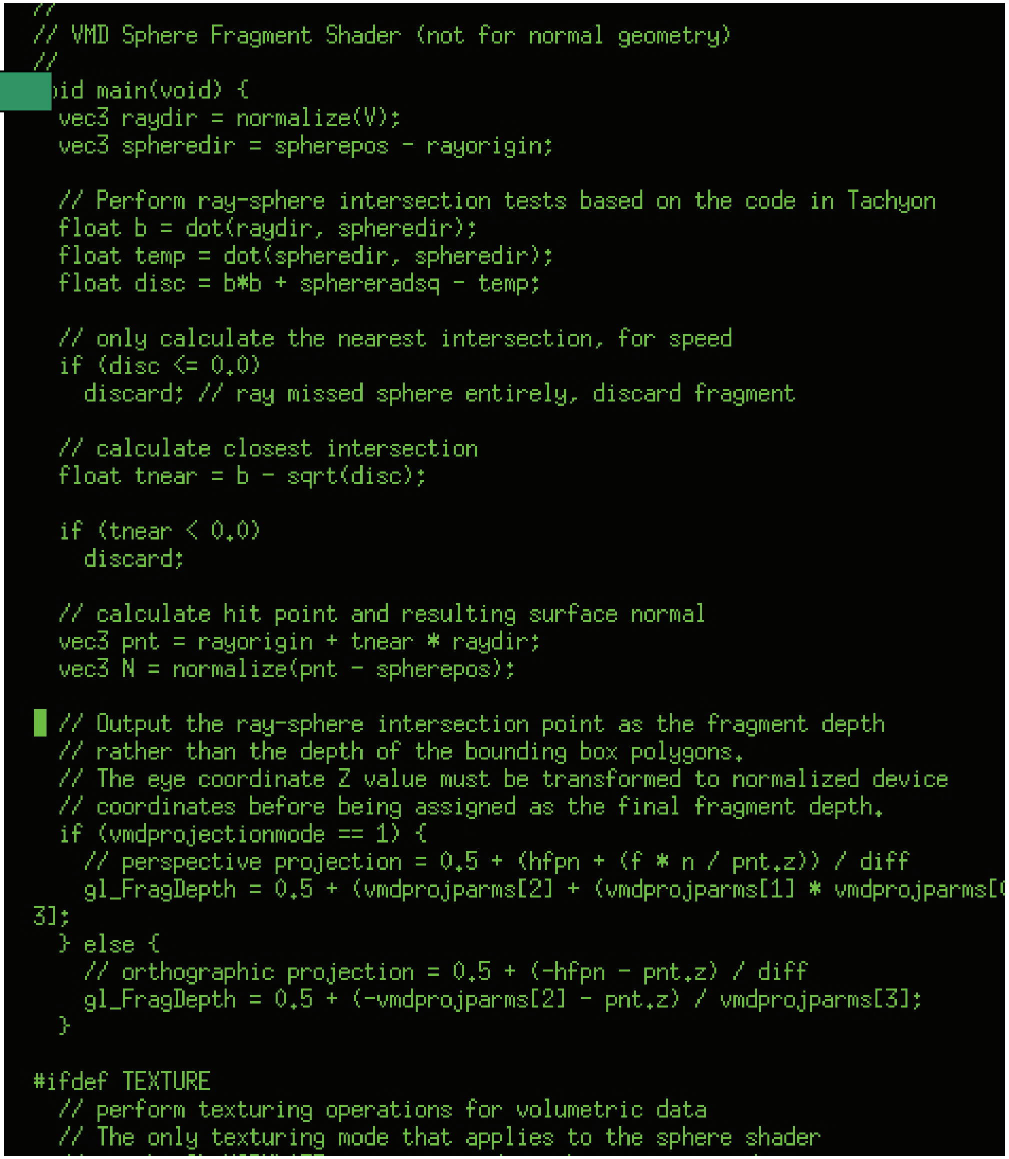
UNC PixelFlow Rack



SGI Reality Engine i860
Vertex Processors

GLSL Sphere Fragment Shader





```
//
// VMD Sphere Fragment Shader (not for normal geometry)
//
oid main(void) {
  vec3 raydir = normalize(V);
  vec3 spheredir = spherepos - rayorigin;

  // Perform ray-sphere intersection tests based on the code in Tachyon
  float b = dot(raydir, spheredir);
  float temp = dot(spheredir, spheredir);
  float disc = b*b + sphereradsq - temp;

  // only calculate the nearest intersection, for speed
  if (disc <= 0.0)
    discard; // ray missed sphere entirely, discard fragment

  // calculate closest intersection
  float tnear = b - sqrt(disc);

  if (tnear < 0.0)
    discard;

  // calculate hit point and resulting surface normal
  vec3 pnt = rayorigin + tnear * raydir;
  vec3 N = normalize(pnt - spherepos);

  // Output the ray-sphere intersection point as the fragment depth
  // rather than the depth of the bounding box polygons.
  // The eye coordinate Z value must be transformed to normalized device
  // coordinates before being assigned as the final fragment depth.
  if (vmdprojectionmode == 1) {
    // perspective projection = 0.5 + (hfpn + (f * n / pnt.z)) / diff
    gl_FragDepth = 0.5 + (vmdprojparms[2] + (vmdprojparms[1] * vmdprojparms[0
3];
  } else {
    // orthographic projection = 0.5 + (-hfpn - pnt.z) / diff
    gl_FragDepth = 0.5 + (-vmdprojparms[2] - pnt.z) / vmdprojparms[3];
  }

#ifdef TEXTURE
  // perform texturing operations for volumetric data
  // The only texturing mode that applies to the sphere shader
```

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

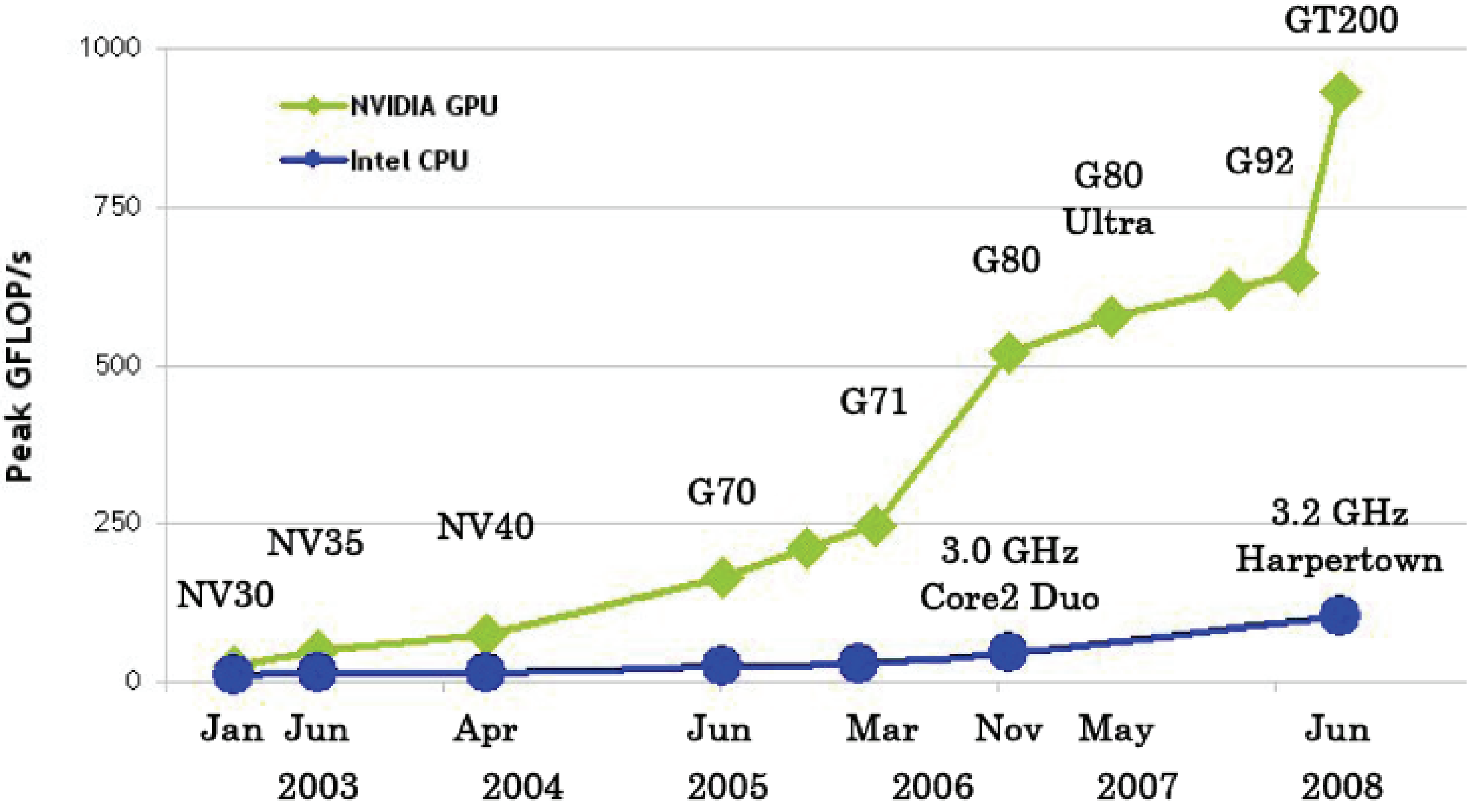
GPU Computing

- Commodity devices, omnipresent in modern computers
- Massively parallel hardware, hundreds of processing units, throughput oriented design
- Support all standard integer and floating point types
- CUDA and OpenCL allow GPU software to be written in dialects of familiar C/C++ and integrated into legacy software
- GPU algorithms are often multicore-friendly due to attention paid to data locality and work decomposition, and can be successfully executed on multi-core CPUs as well, using special runtime systems (e.g. MCUDA)

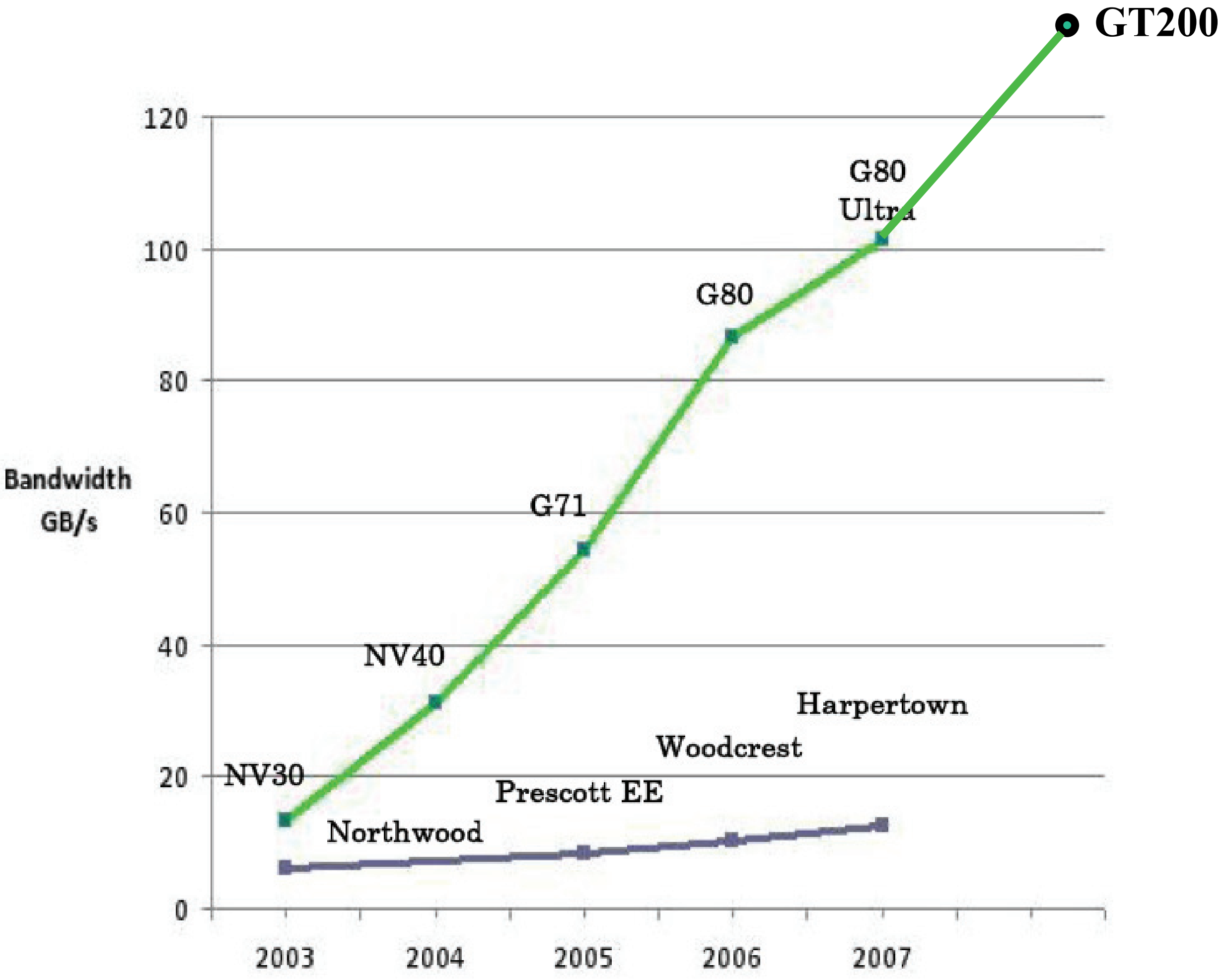
What Speedups Can GPUs Achieve?

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

GPU Peak Single-Precision Performance: **Exponential Trend**

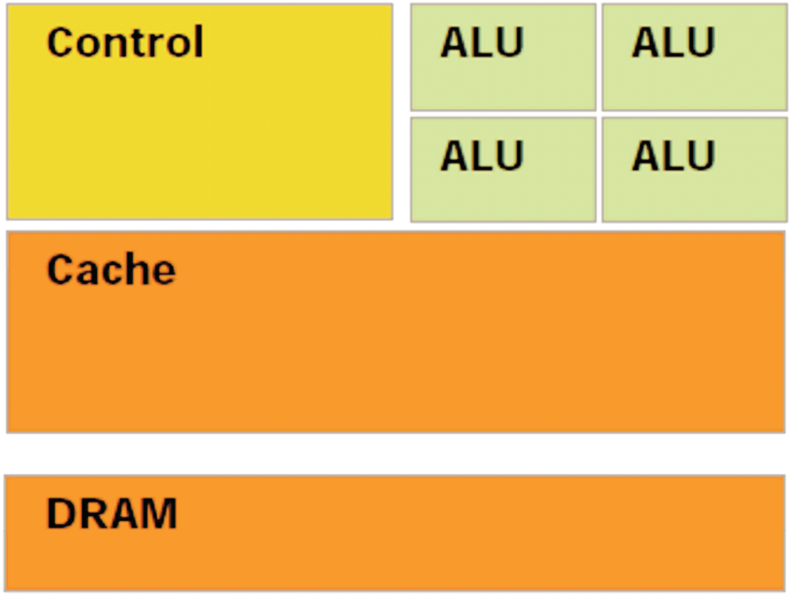


GPU Peak Memory Bandwidth: **Linear Trend**

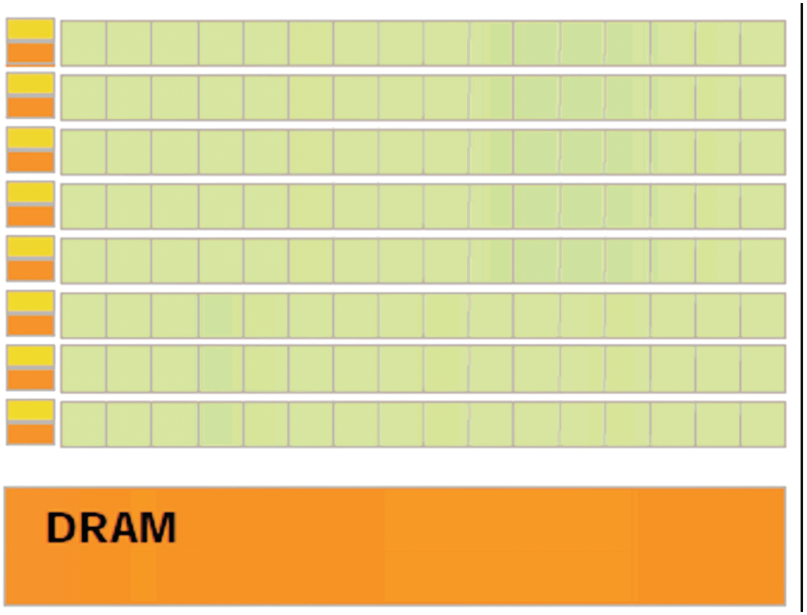


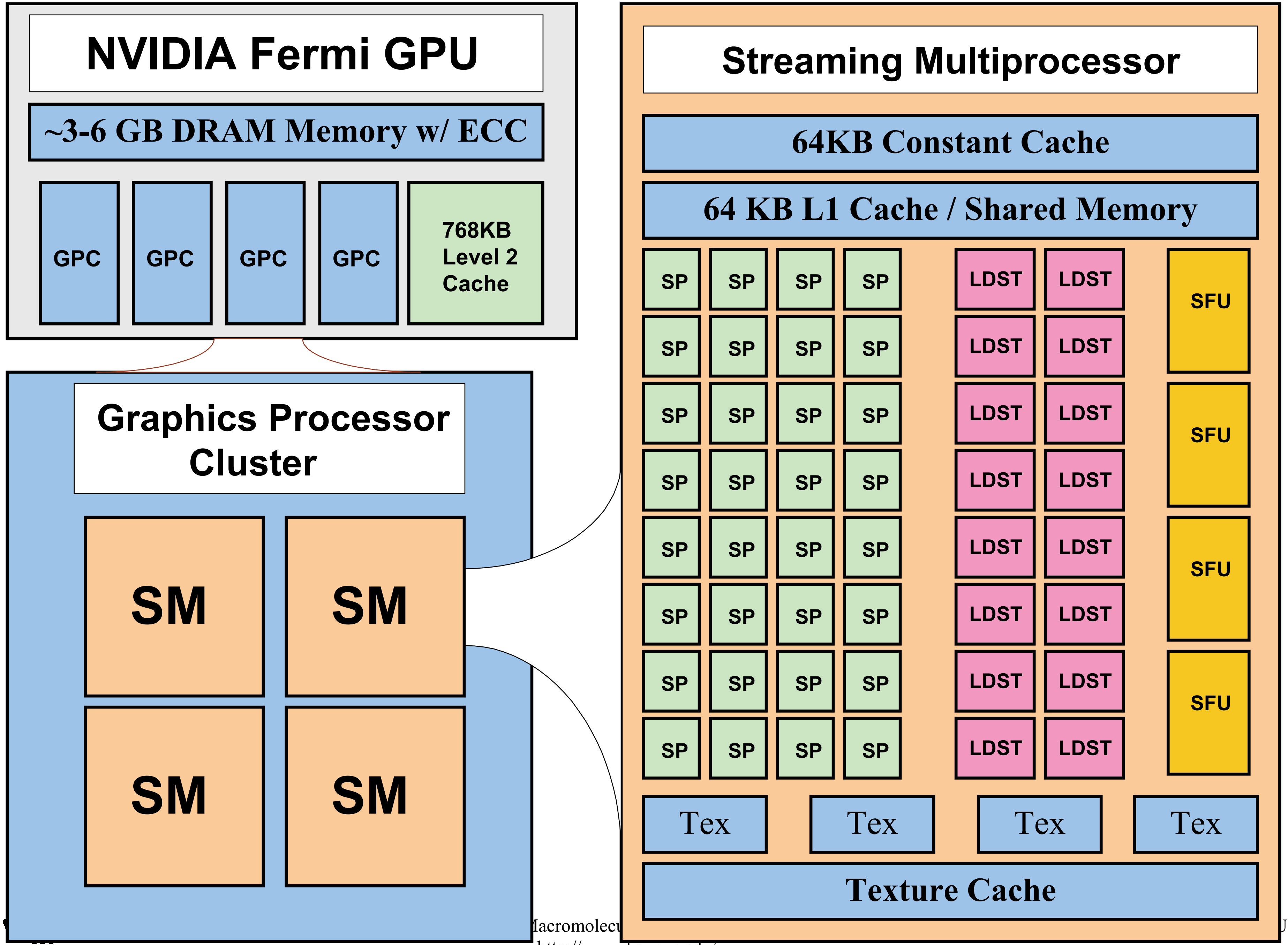
Comparison of CPU and GPU Hardware Architecture

CPU: Cache heavy, focused on individual thread performance



GPU: ALU heavy, massively parallel, throughput oriented





NVIDIA Fermi GPU

~3-6 GB DRAM Memory w/ ECC

GPC GPC GPC GPC

768KB Level 2 Cache

Graphics Processor Cluster

SM SM

SM SM

Streaming Multiprocessor

64KB Constant Cache

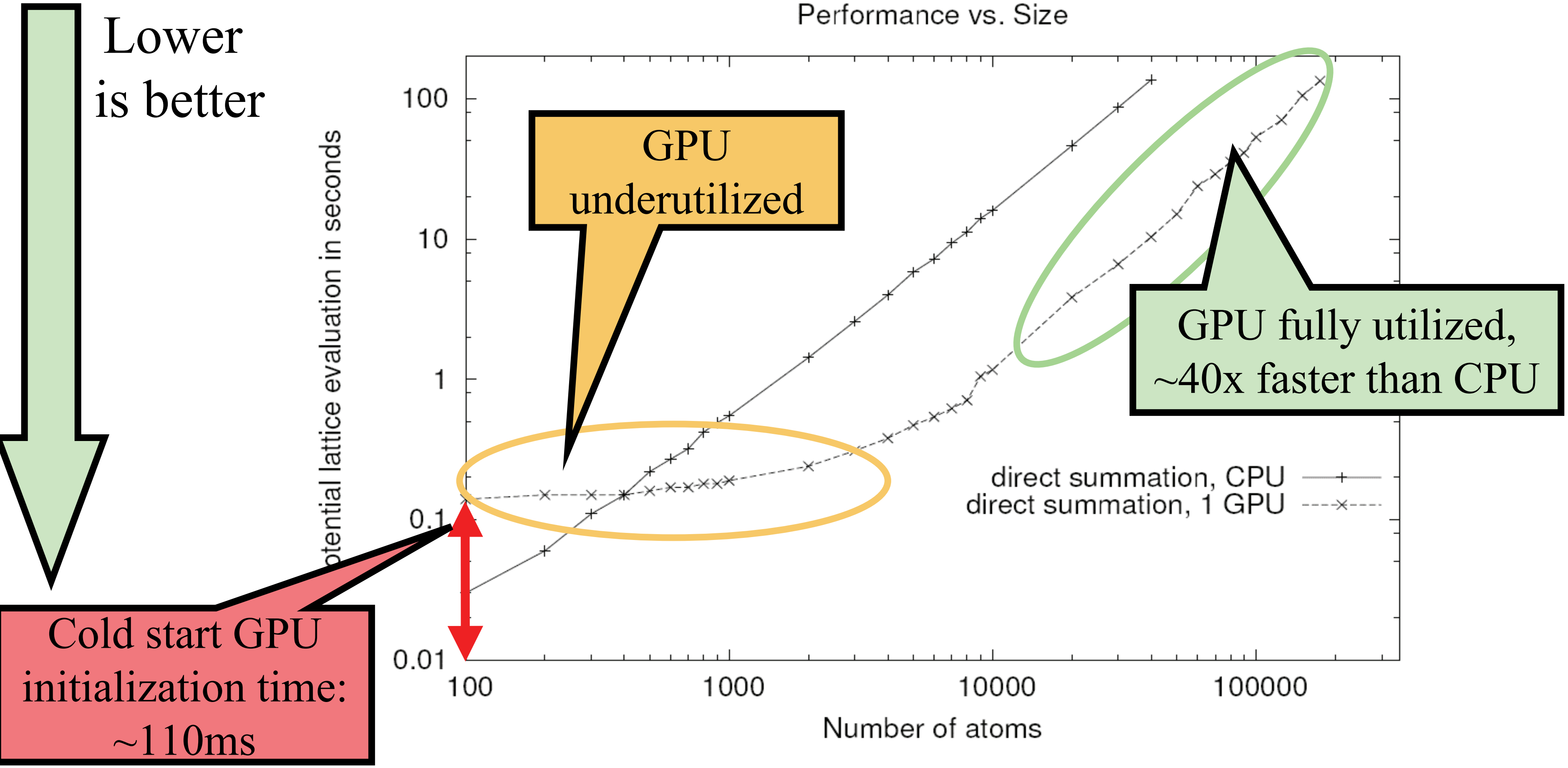
64 KB L1 Cache / Shared Memory

SP SP SP SP LDST LDST SFU
SP SP SP SP LDST LDST
SP SP SP SP LDST LDST SFU
SP SP SP SP LDST LDST
SP SP SP SP LDST LDST SFU
SP SP SP SP LDST LDST
SP SP SP SP LDST LDST SFU
SP SP SP SP LDST LDST

Tex Tex Tex Tex

Texture Cache

GPUs Require Lots of Parallelism, Favor Large Problem Sizes



Lower is better

GPU underutilized

GPU fully utilized, ~40x faster than CPU

Cold start GPU initialization time: ~110ms

Accelerating molecular modeling applications with graphics processors. J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, K. Schulten. *J. Comp. Chem.*, 28:2618-2640, 2007.

NAMD Molecular Dynamics on GPUs

NVIDIA Tesla

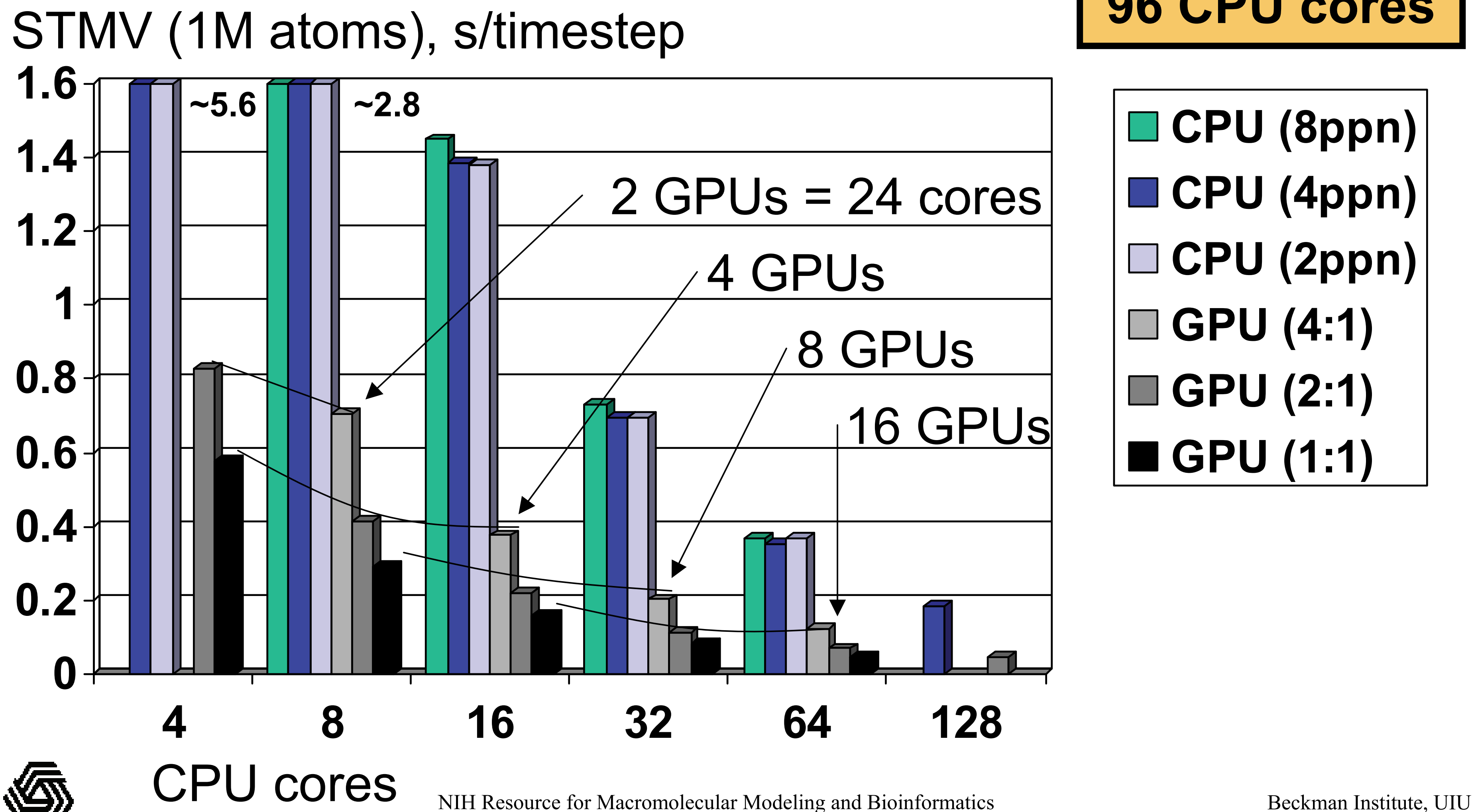
LINCOLN
LINCOLN
Peak 47.5 Teraflops
Architecture Dell PowerEdge Xeon cluster with NVIDIA Tesla accelerators
Total CPUs 1536 CPUs, 384 GPUs
Total Memory 3.072 Terabytes
Launch Date 2009
NCSA
DELL

NCSA Lincoln GPU Cluster

NAMD Performance on NCSA “Lincoln” GPU Cluster

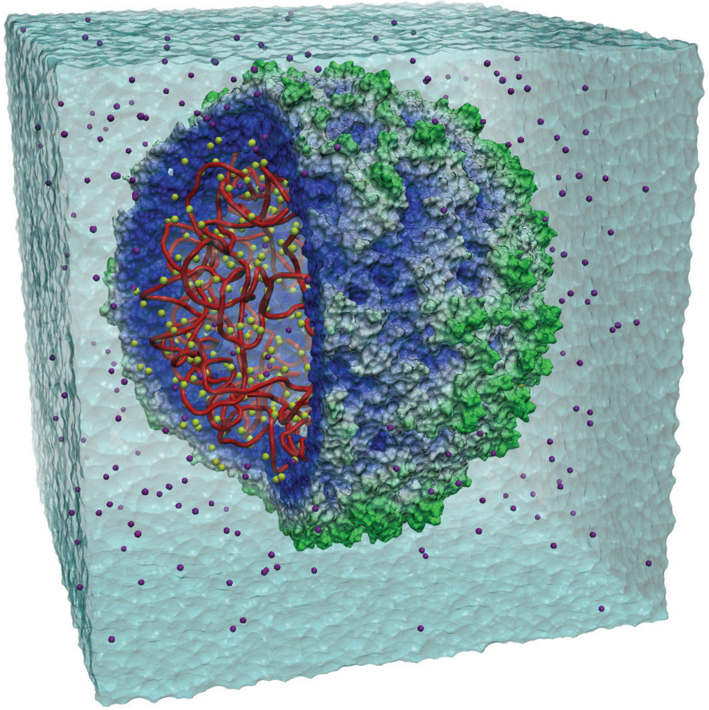
(8 cores and 2 GPUs per node)

8 GPUs = 96 CPU cores



NAMD 2.7b3 w/ CUDA Available

- CUDA-enabled NAMD binaries for 64-bit Linux are available on the NAMD web site now! http://www.ks.uiuc.edu/Research/namd/2.7b3/announce.html
- Direct download link: http://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=NAMD



GPU-Accelerated NAMD Plans

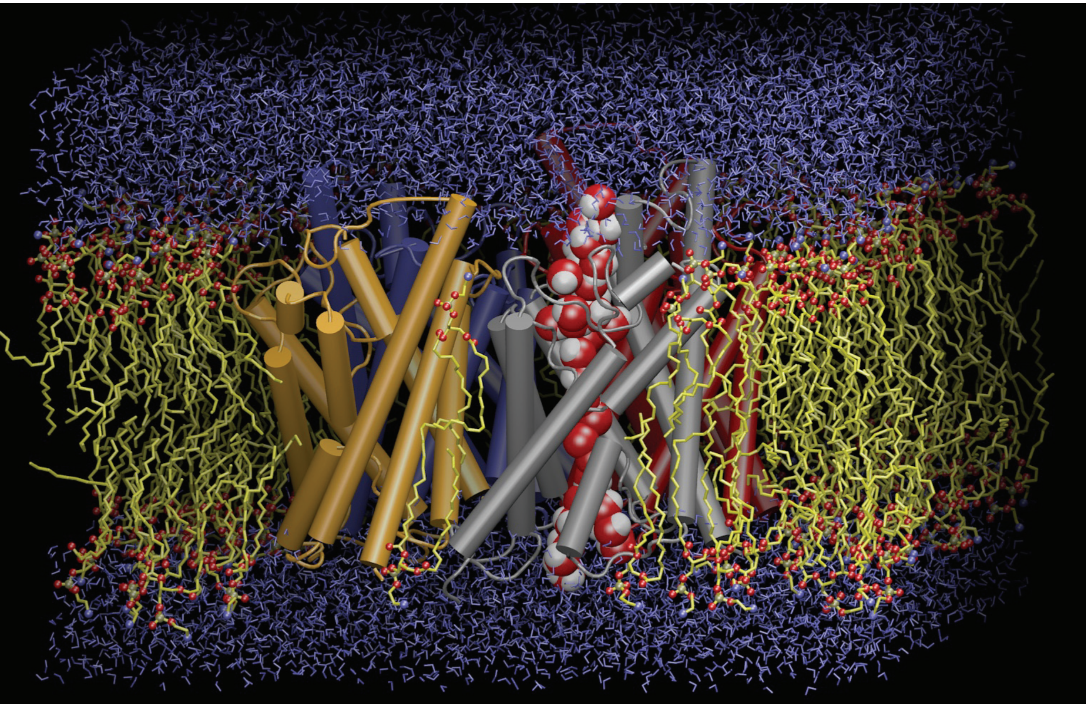
- Serial performance
  - Target NVIDIA Fermi architecture
  - Revisit GPU kernel design decisions made in 2007
  - Improve performance of remaining CPU code
- Parallel scaling
  - Target NSF Track 2D Keeneland cluster at ORNL
  - Finer-grained work units on GPU (feature of Fermi)
  - One process per GPU, one thread per CPU core
  - Dynamic load balancing of GPU work
- Wider range of simulation options and features

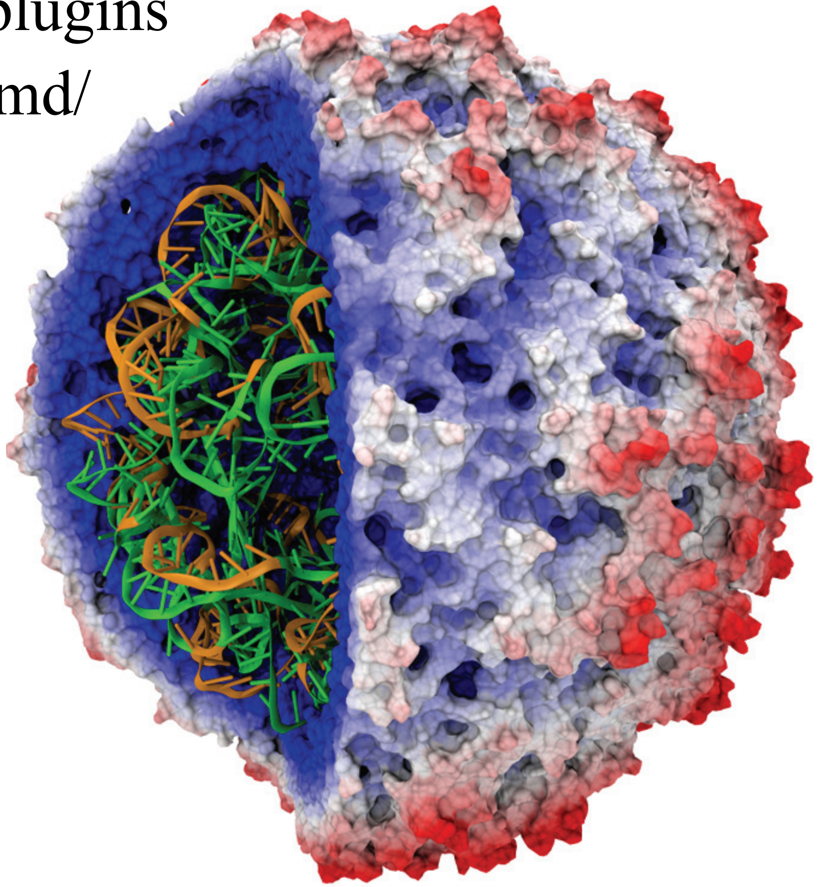
Science Benefits from GPU-Accelerated Molecular Dynamics

- Run simulations of larger molecular complexes
- Run longer timescale simulations
- Run simulations on desktop or departmental sized cluster at speeds previously accessible only on larger machines
- Future: enable higher fidelity modeling
  - Polarizable force fields
  - Better sampling

VMD – "Visual Molecular Dynamics"

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

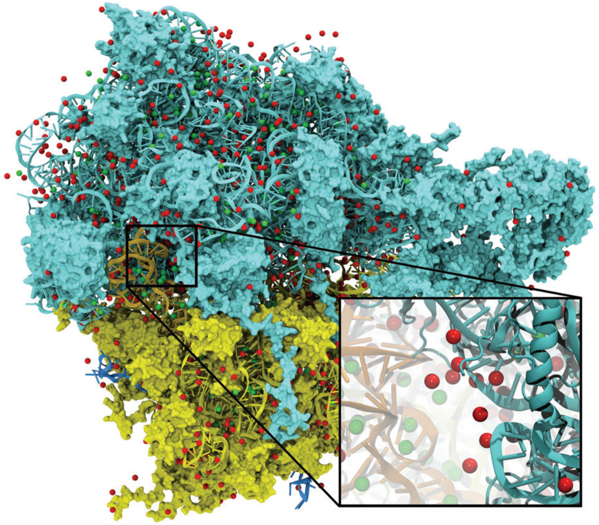




Motivation for GPU Acceleration in VMD

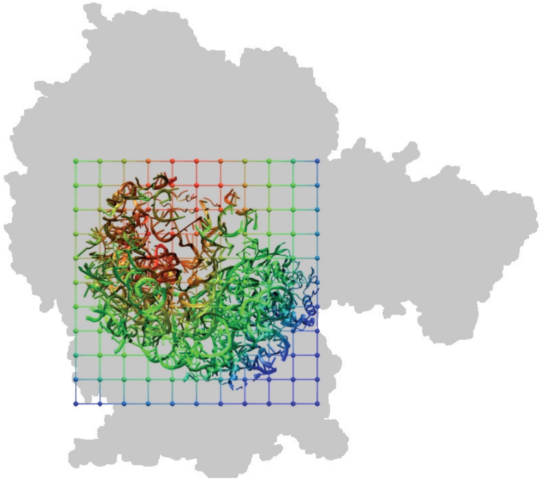
- Increases in supercomputing resources at NSF centers such as NCSA enable increased simulation complexity, fidelity, and longer time scales…
- Drives need for more visualization and analysis capability at the desktop and on clusters
- Desktop use is the most compute-limited scenario, where **GPUs can make a big impact**…
- GPU acceleration provides an opportunity to make some **slow, or batch** calculations capable of being run **interactively, or on-demand…**

GPU Algorithms in VMD



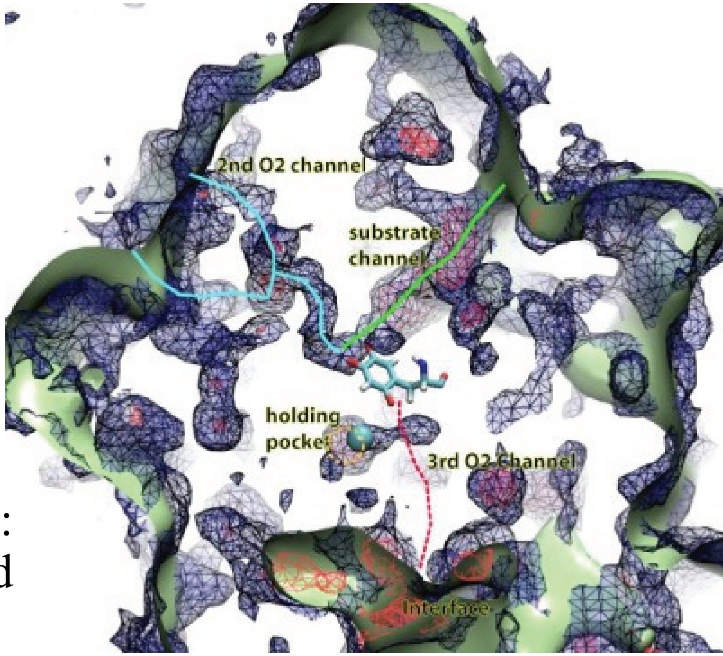
Ion placement

20x to 44x faster



Electrostatic field calculation: multilevel summation method

31x to 44x faster

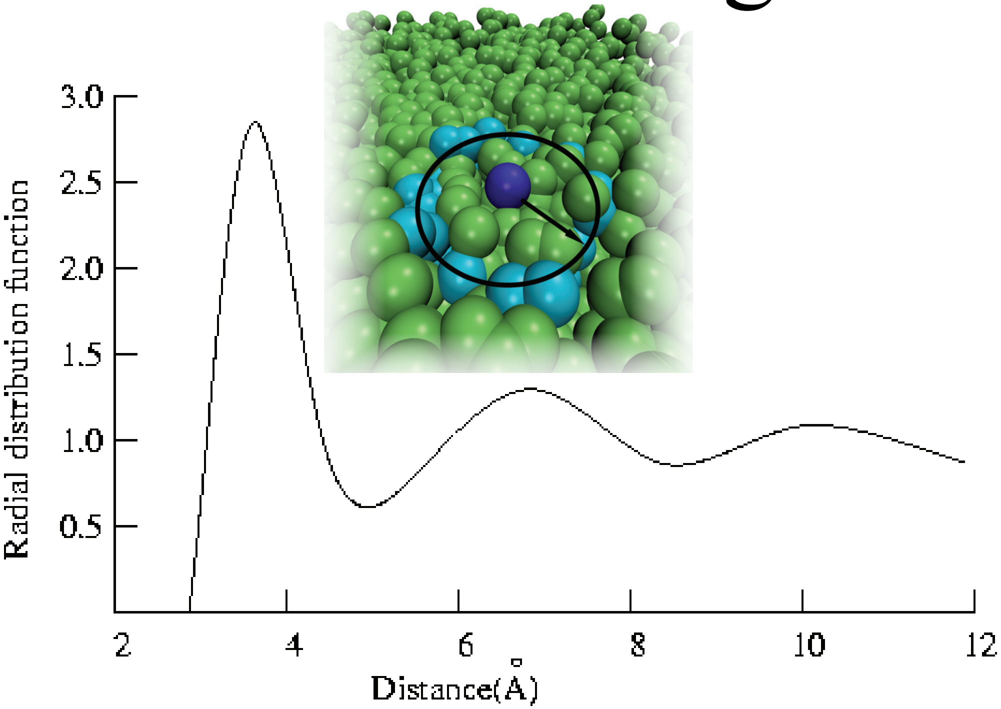


Imaging of gas migration pathways in proteins with implicit ligand sampling

20x to 30x faster

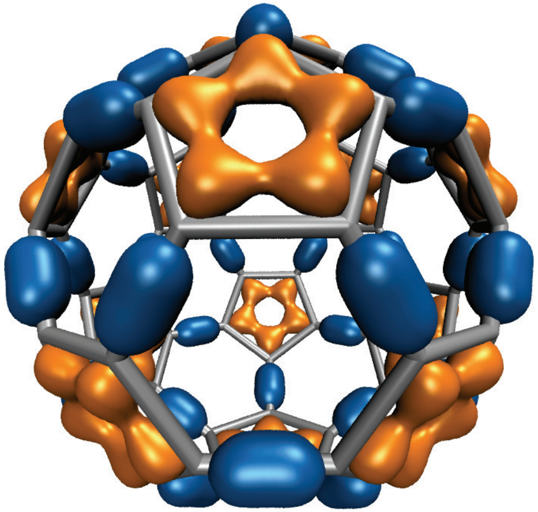
GPU: massively parallel co-processor

GPU Algorithms in VMD



Radial distribution functions

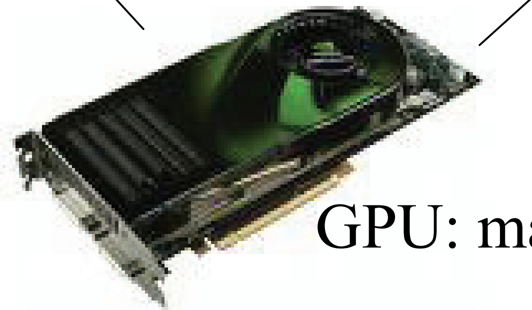
30x to 70x faster



Molecular orbital

calculation and display

100x to 120x faster



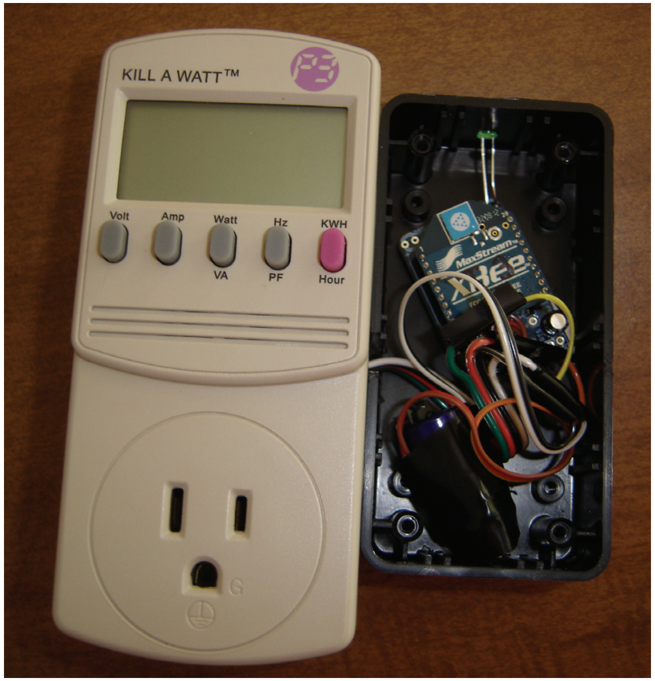
GPU: massively parallel co-processor

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

Trajectory Analysis on NCSA GPU Cluster with MPI-enabled VMD

- Short time-averaged electrostatic field test case (few hundred frames, 700,000 atoms)
- 1:1 CPU/GPU ratio
- Power measured on a single node w/ NCSA monitoring tools
- CPUs-only: 1465 sec, 299 watts
- CPUs+GPUs: 57 sec, 742 watts
- Speedup 25.5 x
- Power efficiency gain: 10.5 x



NCSA Tweet-a-watt power monitoring device

Latest 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…

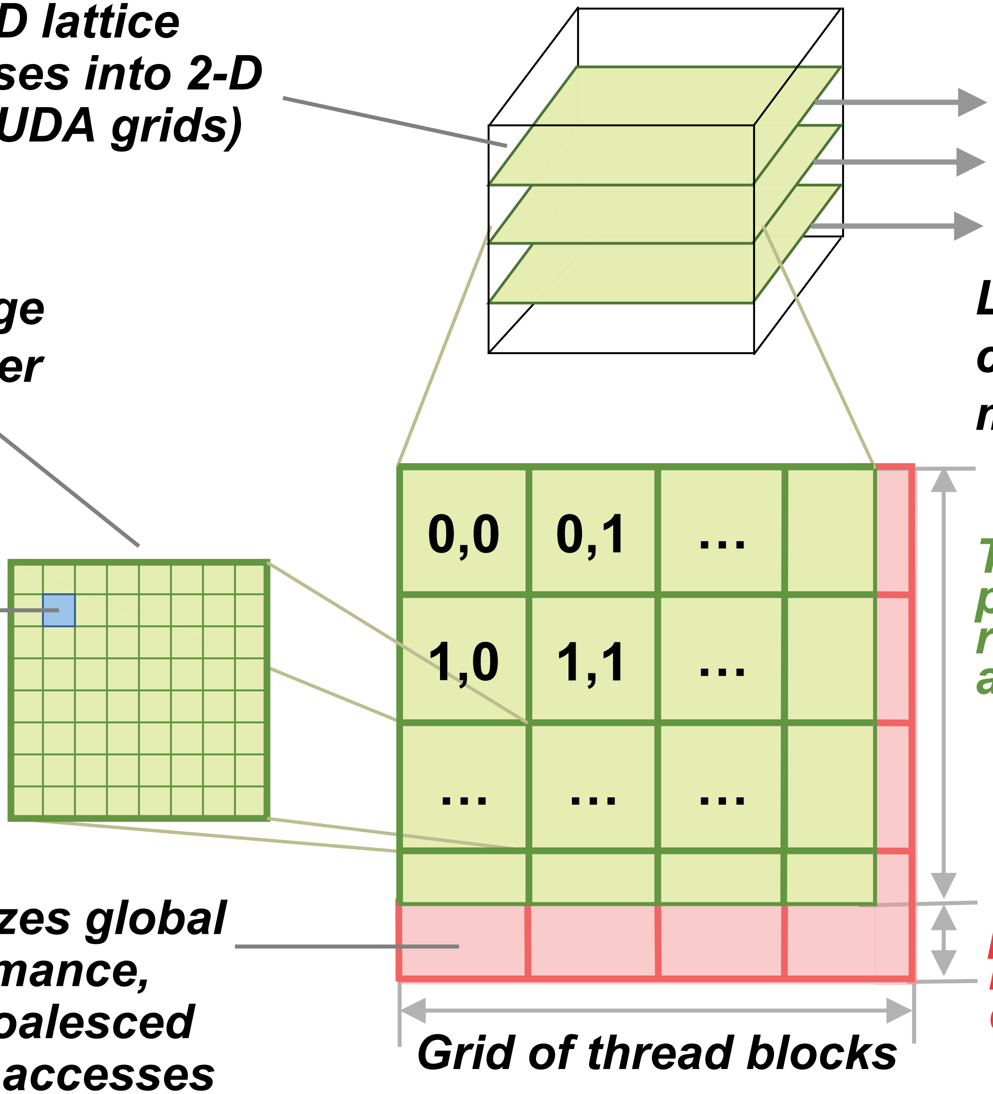
Early Experiences with Fermi

- 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

MO GPU Parallel Decomposition

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

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



Lattice can be computed using multiple GPUs

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

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 MO Performance Results for C_{60}

2.6GHz Intel X5550 vs. NVIDIA GTX 480

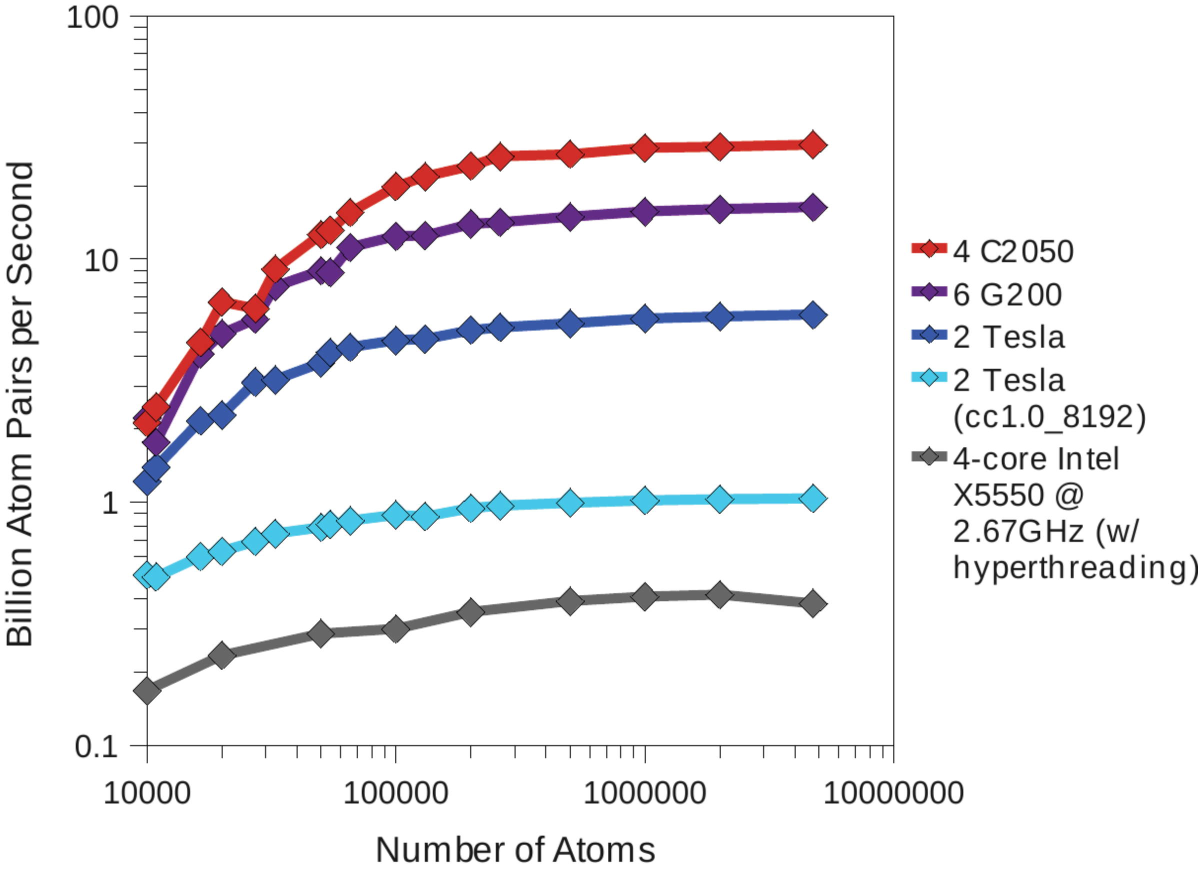
Kernel	Cores/GPUs	Runtime (s)	Speedup
CPU ICC-SSE	1	30.64	1.0
CPU ICC-SSE	8	4.13	7.4
CUDA-tiled-shared	1	0.37	83
CUDA-Fermi-L1-cache (16kB)	1	0.27	113
CUDA-const-cache	1	0.26	117

C_{60} basis set 6-31Gd. We used a high resolution MO grid for accurate timings. A more typical calculation has 1/8th the grid points.

Fermi L1 cache supports arbitrary sized problems, at near peak performance, with much simpler kernel design…

Computing Radial Distribution Functions (Histogramming) on GPUs

- Fermi: larger shared memory and faster atomic ops increase histogramming performance
- ~3x faster than GT200
- Up to 70x faster than 4-core Intel X5550 CPU



Acknowledgements

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