Programming in CUDA: the Essentials, Part 1

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/

Cape Town GPU Workshop

Cape Town, South Africa, April 29, 2013

Evolution of Graphics Hardware Towards Programmability

- As graphics accelerators became more powerful, an increasing fraction of the graphics processing pipeline was implemented in hardware
- For performance reasons, this hardware was highly optimized and task-specific
- Over time, with ongoing increases in circuit density and the need for flexibility in lighting and texturing, graphics pipelines gradually incorporated programmability in specific pipeline stages
- Modern graphics accelerators are now complete processors in their own right (thus the new term **"GPU"**), and are composed of large arrays of programmable processing units

Origins of Computing on GPUs

- Widespread support for programmable shading led researchers to begin experimenting with the use of GPUs for general purpose computation, **“GPGPU”**
- Early GPGPU efforts used existing graphics APIs to express computation in terms of drawing
- As expected, expressing general computation problems in terms of triangles and pixels and “drawing the answer” is obfuscating and painful to debug…
- Soon researchers began creating dedicated GPU programming tools, starting with Brook and Sh, and ultimately leading to a variety of commercial tools such as RapidMind, CUDA, OpenCL, and others...

GPU Computing

- Commodity devices, omnipresent in modern computers (over a **million** sold per **week**)
- Massively parallel hardware, hundreds of processing units, **throughput oriented architecture**
- Standard integer and floating point types supported
- Programming tools allow software to be written in dialects of familiar C/C++ and integrated into legacy software
- GPU algorithms are often multicore friendly due to attention paid to **data locality** and **data-parallel work decomposition**

Benefits of GPUs vs. Other Parallel Computing Approaches

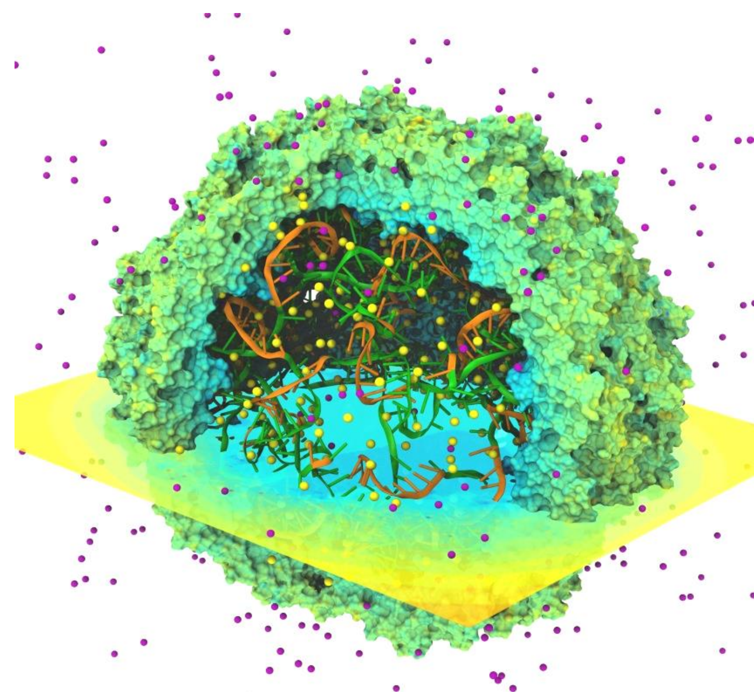
- Increased compute power per unit volume
- Increased FLOPS/watt power efficiency
- Desktop/laptop computers easily incorporate GPUs, no need to teach non-technical users how to use a remote cluster or supercomputer
- GPU can be upgraded without new OS license fees, low cost hardware

What Speedups Can GPUs Achieve?

- Single-GPU speedups of **10x** to **30x** vs. one CPU core are very 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

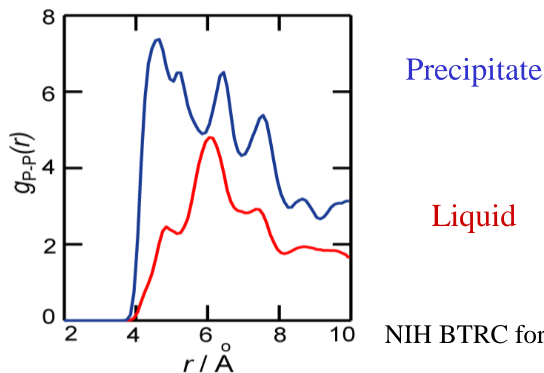
GPU Solution: Time-Averaged Electrostatics

- Thousands of trajectory frames
- **1.5 hour** job reduced to **3 min**
- GPU Speedup: **25.5x**
- Per-node power consumption on NCSA GPU cluster:
  - CPUs-only: 448 Watt-hours
  - CPUs+GPUs: 43 Watt-hours
- Power efficiency gain: **10x**



GPU Solution: Radial Distribution Function Histogramming

- 4.7 million atoms
- 4-core Intel X5550 CPU: **15 hours**
- 4 NVIDIA C2050 GPUs: **10 minutes**
- Fermi GPUs ~3x faster than GT200 GPUs: larger on-chip shared memory



Intel X5550, 4-cores @ 2.66GHz
6x NVIDIA Tesla C1060 (GT200)
4x NVIDIA Tesla C2050 (Fermi)

Billion atom pairs/sec

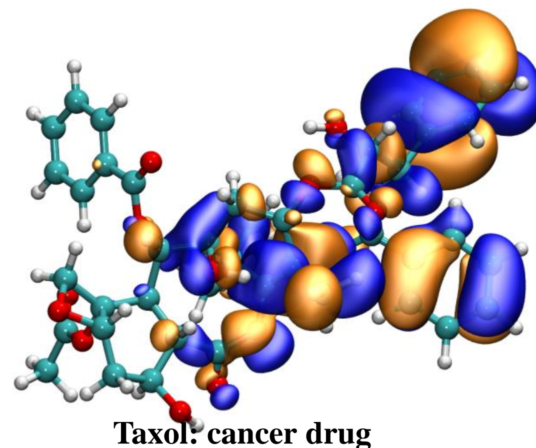
100
10
1
0.1

10,000
100,000
1,000,000
5,000,000

Atoms

Science 5: Quantum Chemistry Visualization

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

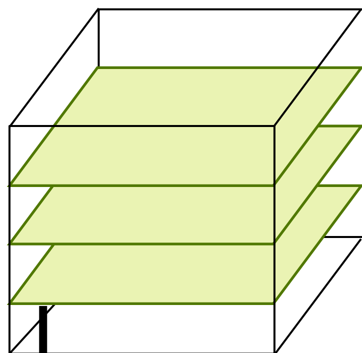


Taxol: cancer drug

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

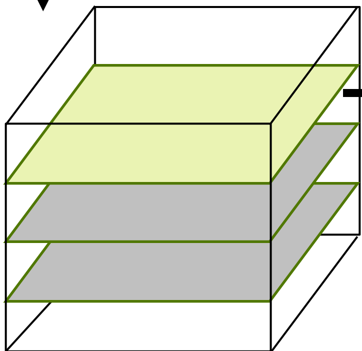
GPU Solution: Computing C_{60} Molecular Orbitals

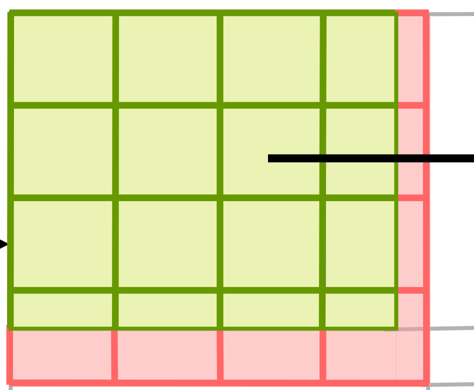
3-D orbital lattice: millions of points



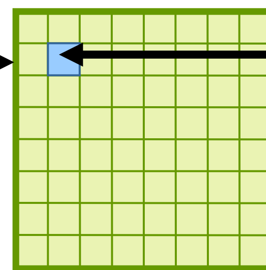
Device	CPUs, GPUs	Runtime (s)	Speedup
Intel X5550-SSE	1	30.64	1.0
Intel X5550-SSE	8	4.13	7.4
GeForce GTX 480	1	0.255	120
GeForce GTX 480	4	0.081	378

Lattice slices computed on multiple GPUs





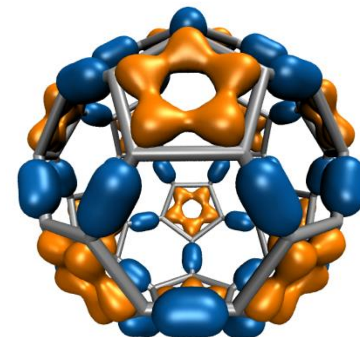
2-D CUDA grid on one GPU



CUDA thread blocks

GPU threads each compute one point.

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

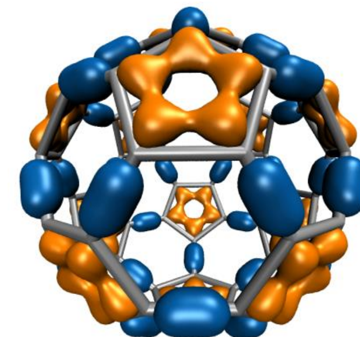


```
for (shell=0; shell < maxshell; shell++) {
  __m128 Cgto = _mm_setzero_ps();
  for (prim=0; prim<num_prim_per_shell[shell_counter]; prim++) {
    float exponent       = -basis_array[prim_counter     ];
    float contract_coeff =  basis_array[prim_counter + 1];
    __m128 expval = _mm_mul_ps(_mm_load_ps1(&exponent), dist2);
    __m128 ctmp = _mm_mul_ps(_mm_load_ps1(&contract_coeff), exp_ps(expval));
    Cgto = _mm_add_ps(contracted_gto, ctmp);
    prim_counter += 2;
  }
  __m128 tshell = _mm_setzero_ps();
  switch (shell_types[shell_counter]) {
    case S_SHELL:
      value = _mm_add_ps(value, _mm_mul_ps(_mm_load_ps1(&wave_f[ifunc++]), Cgto));   break;
    case P_SHELL:
      tshell = _mm_add_ps(tshell, _mm_mul_ps(_mm_load_ps1(&wave_f[ifunc++]), xdist));
      tshell = _mm_add_ps(tshell, _mm_mul_ps(_mm_load_ps1(&wave_f[ifunc++]), ydist));
      tshell = _mm_add_ps(tshell, _mm_mul_ps(_mm_load_ps1(&wave_f[ifunc++]), zdist));
      value = _mm_add_ps(value, _mm_mul_ps(tshell, Cgto));     break;
```

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

Molecular Orbital Inner Loop in CUDA

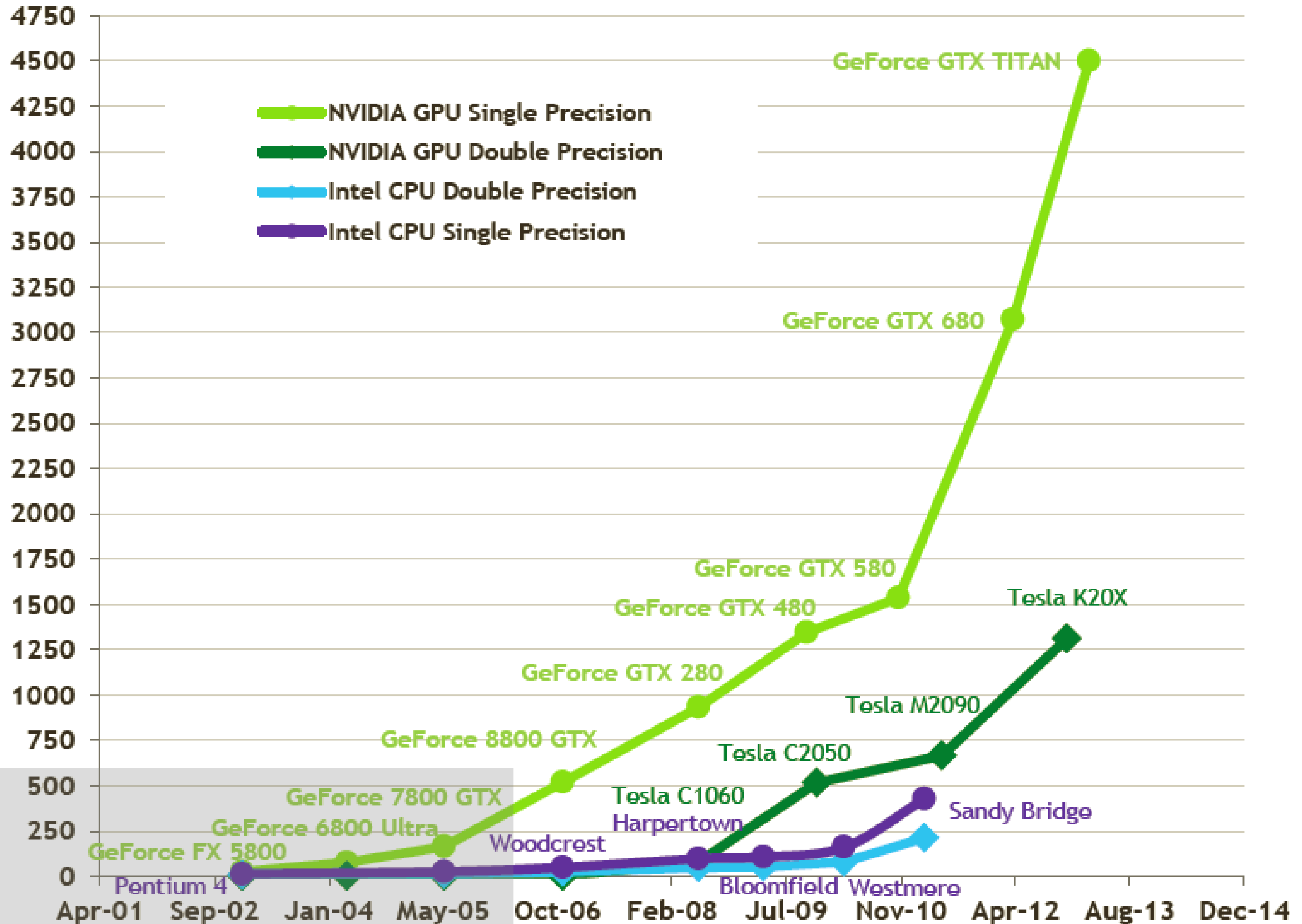
```
for (shell=0; shell < maxshell; shell++) {
   float contracted_gto = 0.0f;
   for (prim=0; prim<num_prim_per_shell[shell_counter]; prim++) {
    float exponent       = const_basis_array[prim_counter    ];
    float contract_coeff = const_basis_array[prim_counter + 1];
    contracted_gto += contract_coeff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



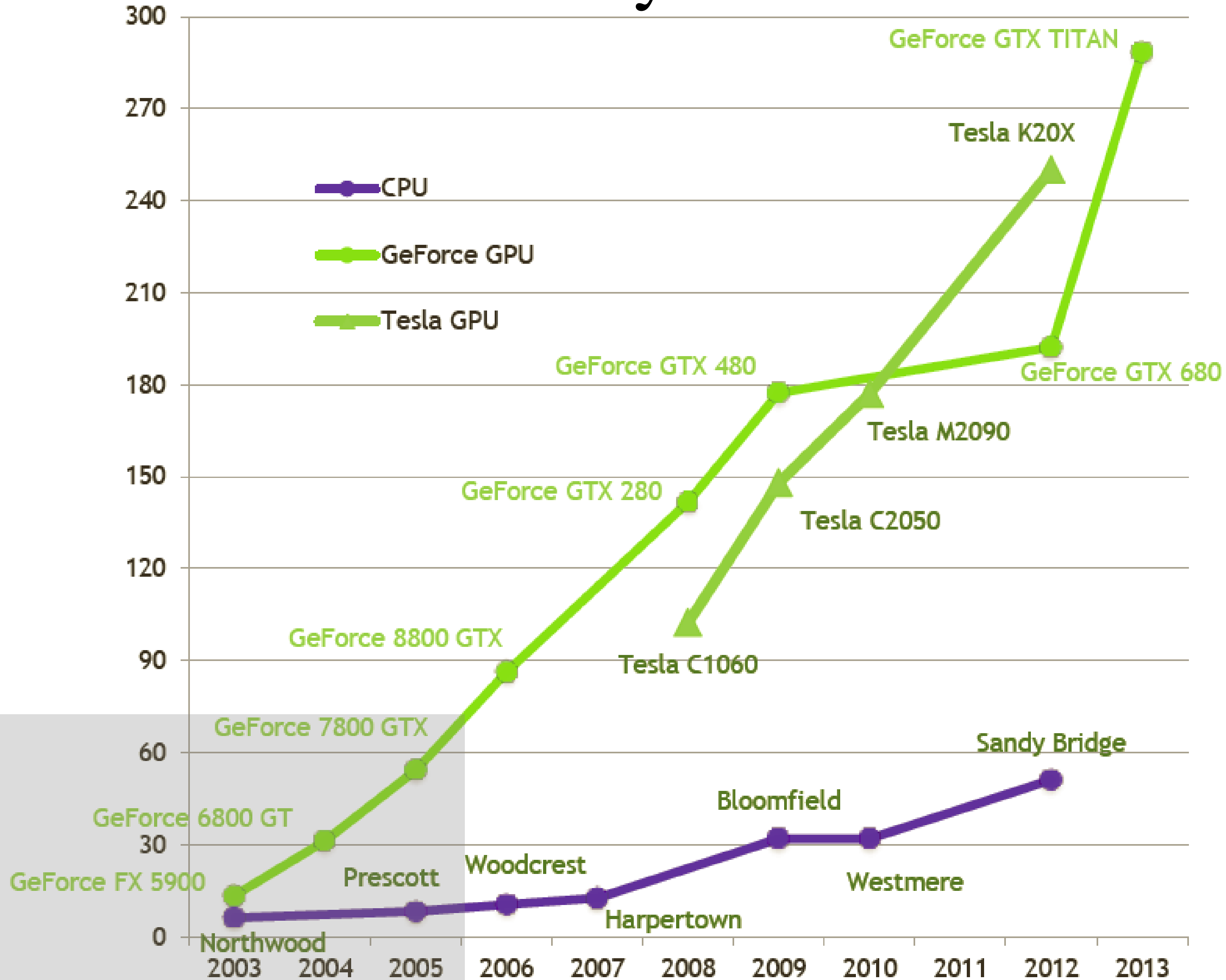
Aaaaahhhh….

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

Peak Arithmetic Performance Trend



Peak Memory Bandwidth Trend

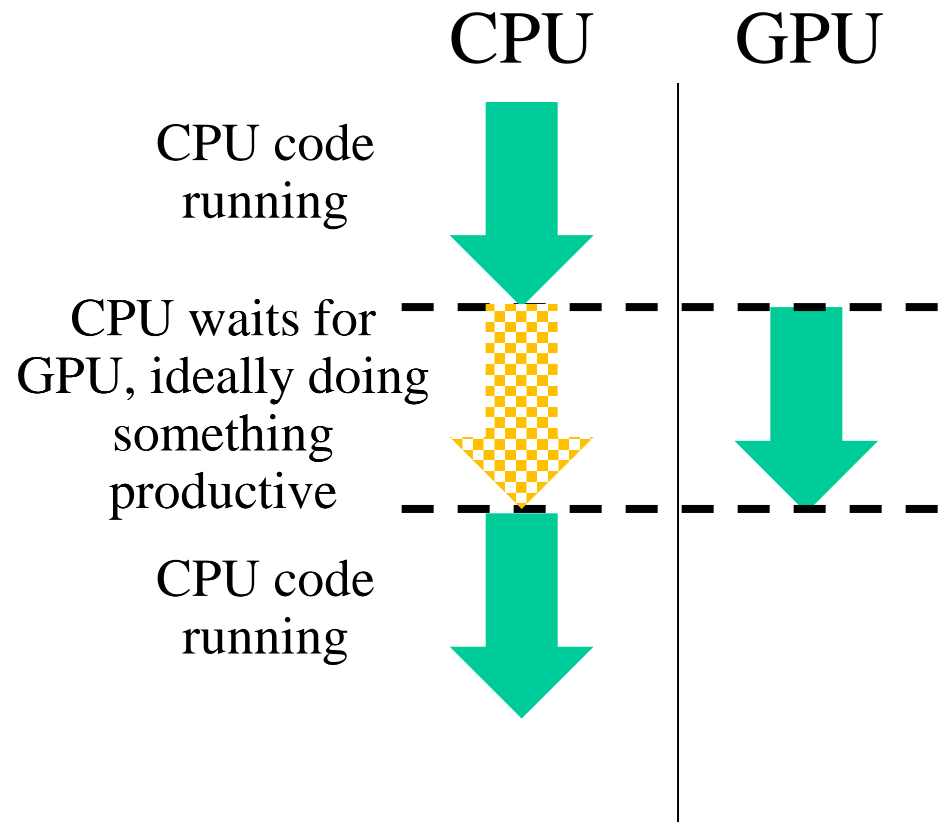


What Runs on a GPU?

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

CUDA Stream of Execution

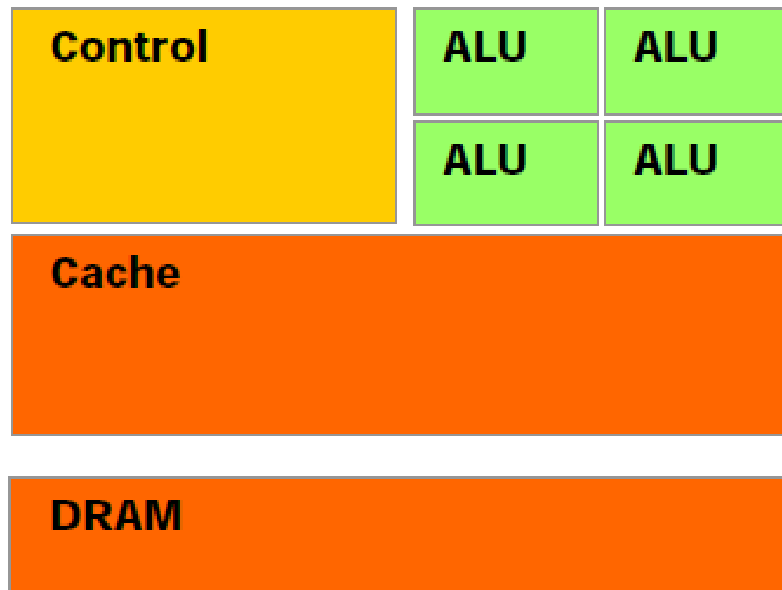
- Host CPU thread launches a CUDA "kernel", a memory copy, etc. on the GPU
- GPU action runs to completion
- Host synchronizes with completed GPU action

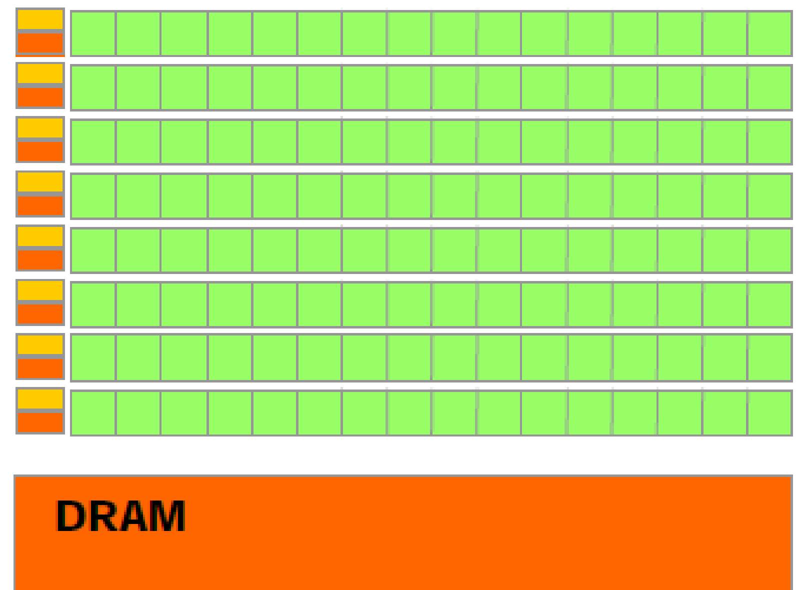


Comparison of CPU and GPU Hardware Architecture

CPU: Cache heavy, focused on individual thread performance

GPU: ALU heavy, massively parallel, throughput oriented





GPU: Throughput-Oriented Hardware Architecture

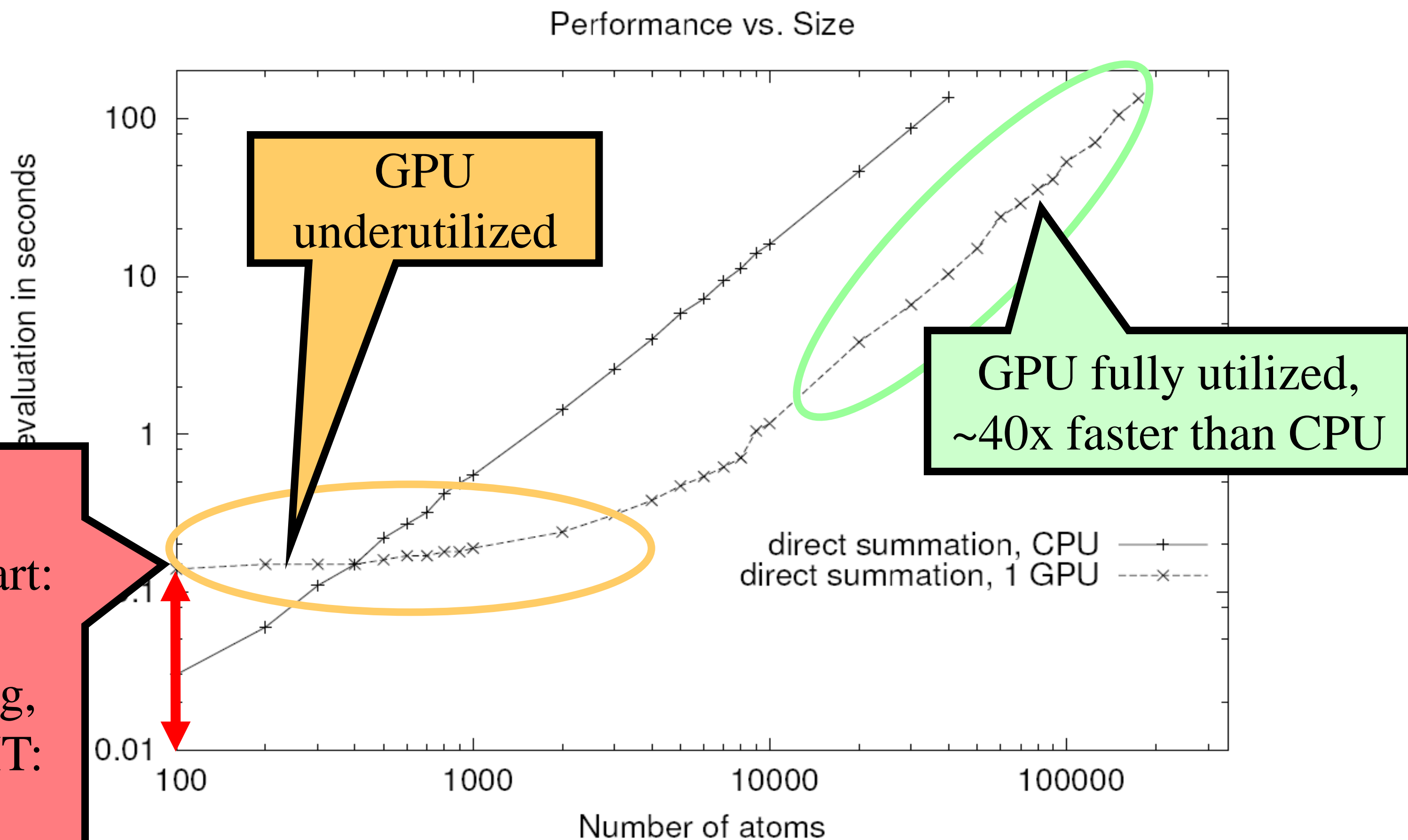
- GPUs have very small on-chip caches
- Main memory latency (several hundred clock cycles!) is tolerated through hardware multithreading – **overlap memory transfer latency with execution of other work**
- When a GPU thread stalls on a memory operation, the hardware immediately switches context to a ready thread
- Effective latency hiding requires saturating the GPU with lots of work – tens of thousands of independent work items

GPU Memory Systems

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

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

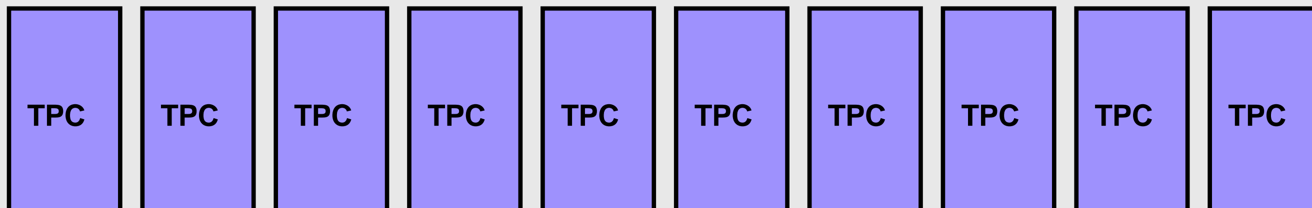
Lower
is better



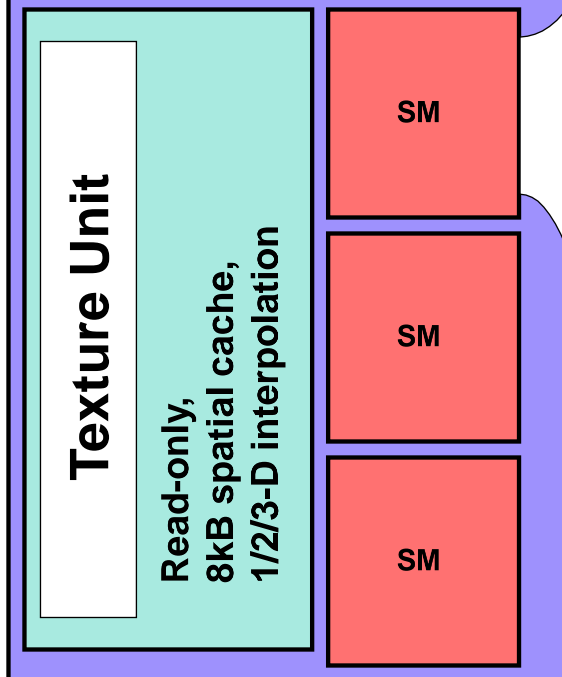
Host thread GPU Cold Start: context init, device binding, kernel PTX JIT: ~110ms

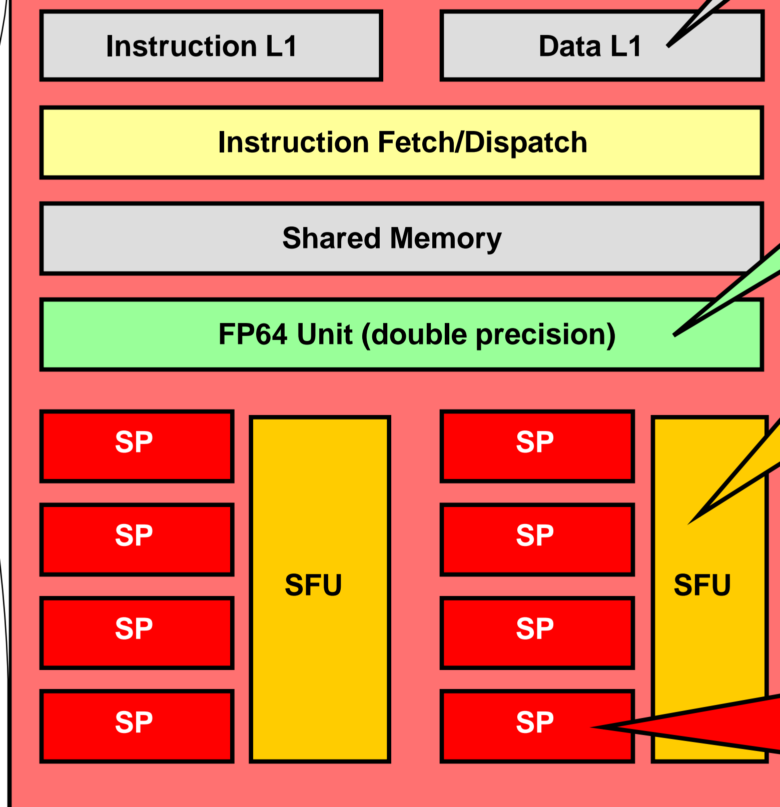
Accelerating molecular modeling applications with graphics processors.
J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, K. Schulten.
J. Comp. Chem., 28:2618-2640, 2007.

Streaming Processor Array



NVIDIA GT200





Constant Cache

64kB, read-only

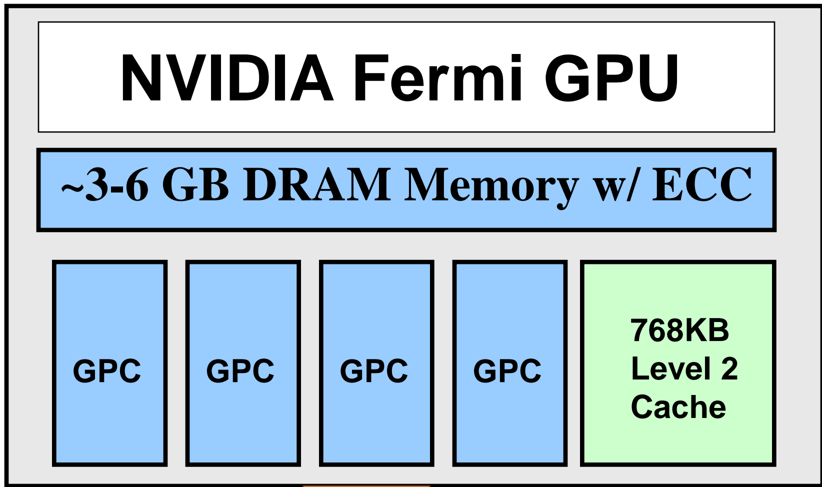
FP64 Unit

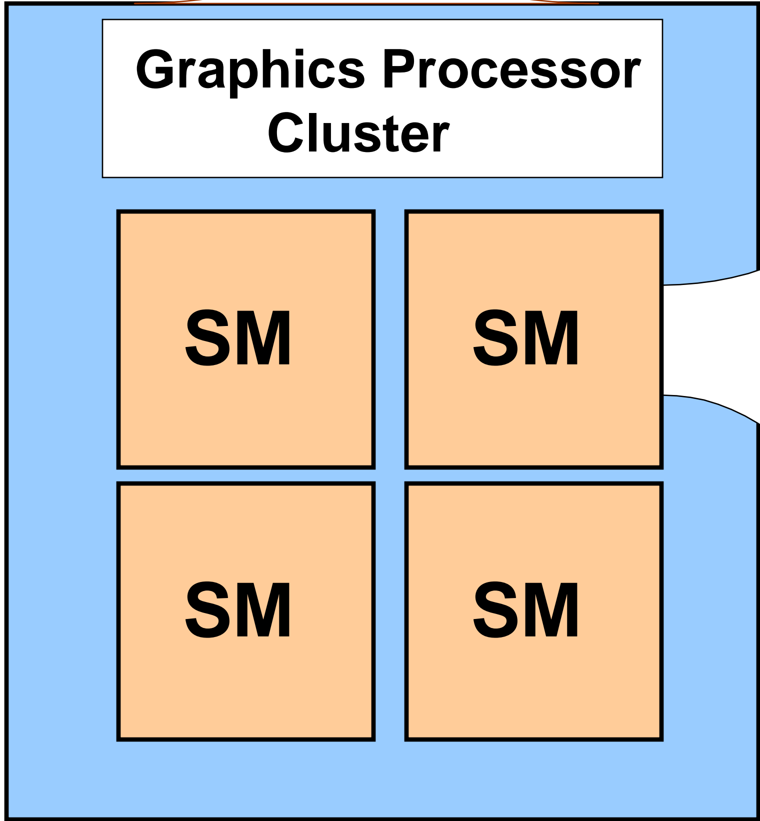
Special Function Unit

SIN, EXP, RSQRT, Etc…

Streaming Processor

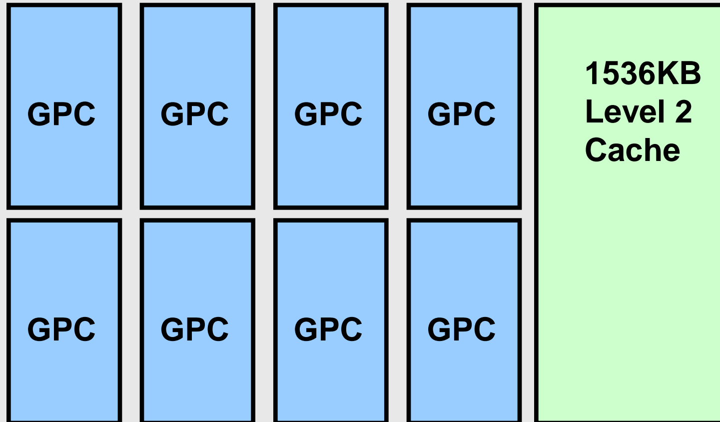
ADD, SUB MAD, Etc…


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
LDST
SFU
Tex
Tex
Tex
Tex
Texture Cache

NVIDIA Kepler GPU

~3-6 GB DRAM Memory w/ ECC



Graphics Processor Cluster

SMX

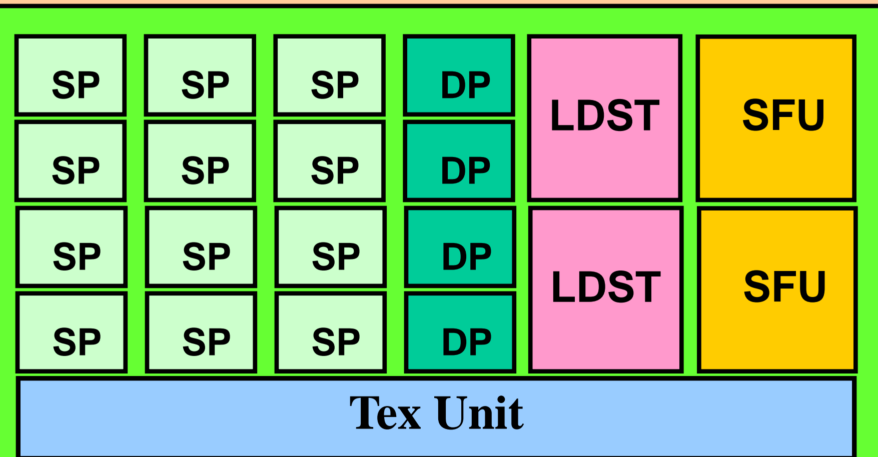
SMX

Streaming Multiprocessor - SMX

64 KB Constant Cache

64 KB L1 Cache / Shared Memory

48 KB Tex + Read-only Data Cache



16 × Execution block =

192 SP, 64 DP,

32 SFU, 32 LDST

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/

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

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

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

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

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

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

- **Adapting a message-driven parallel application to GPU-accelerated clusters**. J. Phillips, J. Stone, K. Schulten. *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, IEEE Press, 2008.
- **GPU acceleration of cutoff pair potentials for molecular modeling applications**. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, and W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.
- **GPU computing**. J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips. *Proceedings of the IEEE*, 96:879-899, 2008.
- **Accelerating molecular modeling applications with graphics processors**. J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, K. Schulten. *J. Comp. Chem.*, 28:2618-2640, 2007.
- **Continuous fluorescence microphotolysis and correlation spectroscopy**. A. Arkhipov, J. Hüve, M. Kahms, R. Peters, K. Schulten. *Biophysical Journal*, 93:4006-4017, 2007.