GPU Acceleration of Cutoff Pair Potentials for Molecular Modeling Applications

Christopher Rodrigues,
David J. Hardy, John E. Stone,
Klaus Schulten, Wen-Mei W. Hwu

University of Illinois at Urbana-Champaign

Computing Frontiers, Ischia, Italy
7 May 2008

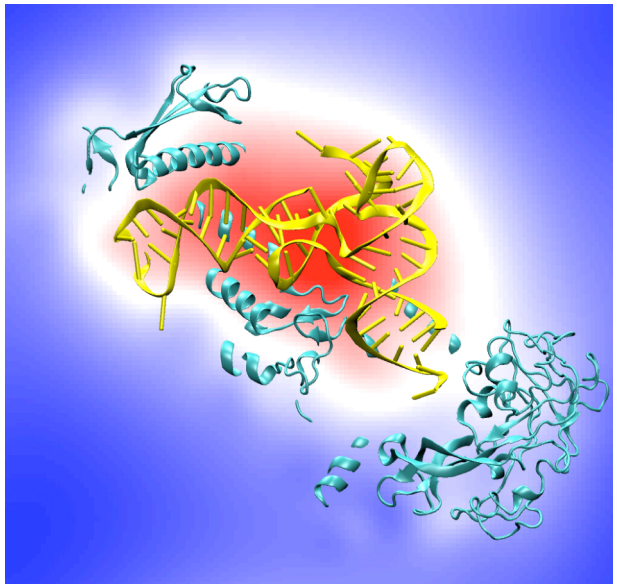
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

IMPACT
ADVANCED COMPILER TECHNOLOGY

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

Cutoff Pair Potentials

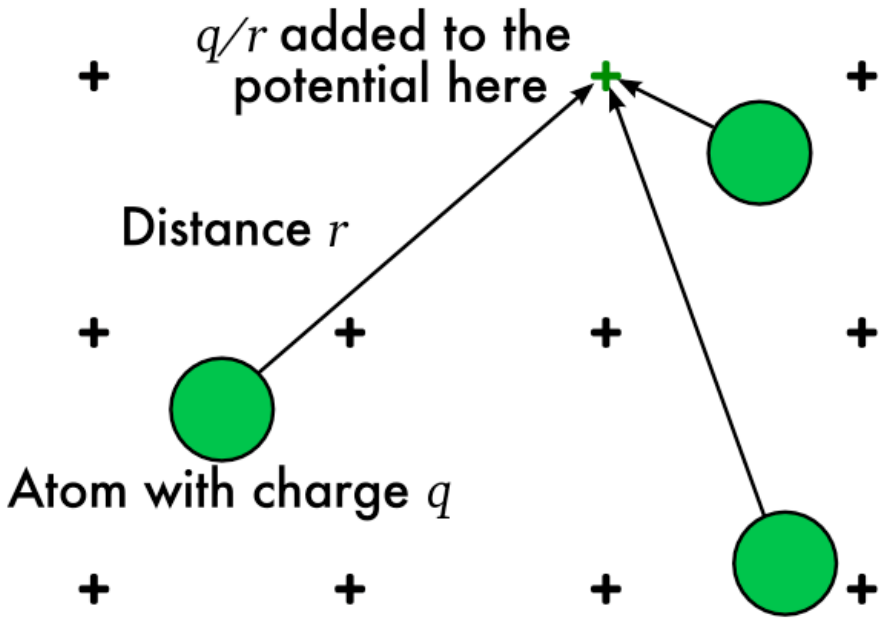
- Essential to molecular modeling applications
  - E.g., van der Waals, electrostatic potential
  - Often the most costly part of computation
- Evaluate a cutoff electrostatic potential on a 3D lattice
- Applications include
  - Structure building
    - Ion placement
    - Time-averaged potential
  - Analysis
    - Visualizing electrostatic potential



2D slice through an electrostatic potential map

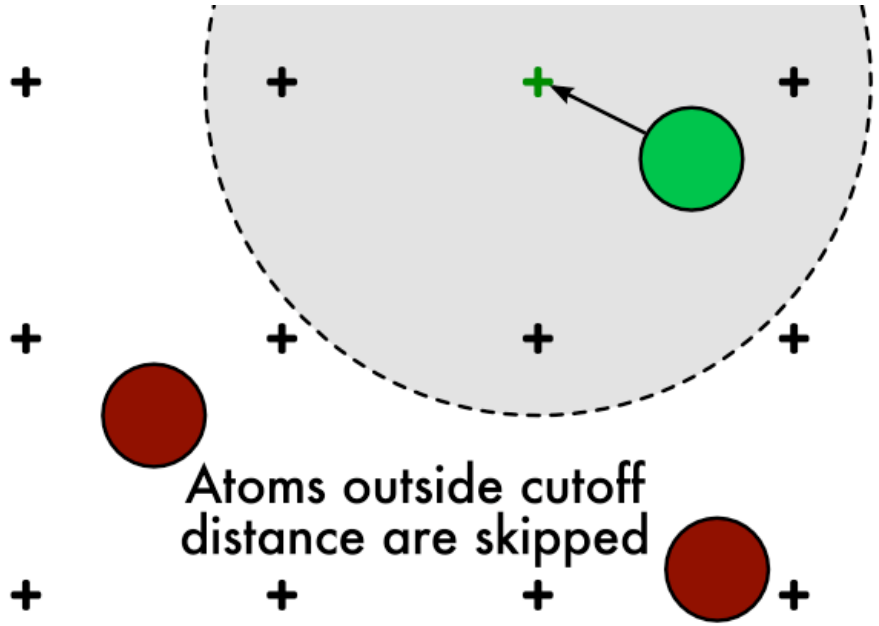
Red: positive Blue: negative

Algorithm for Pair Potentials



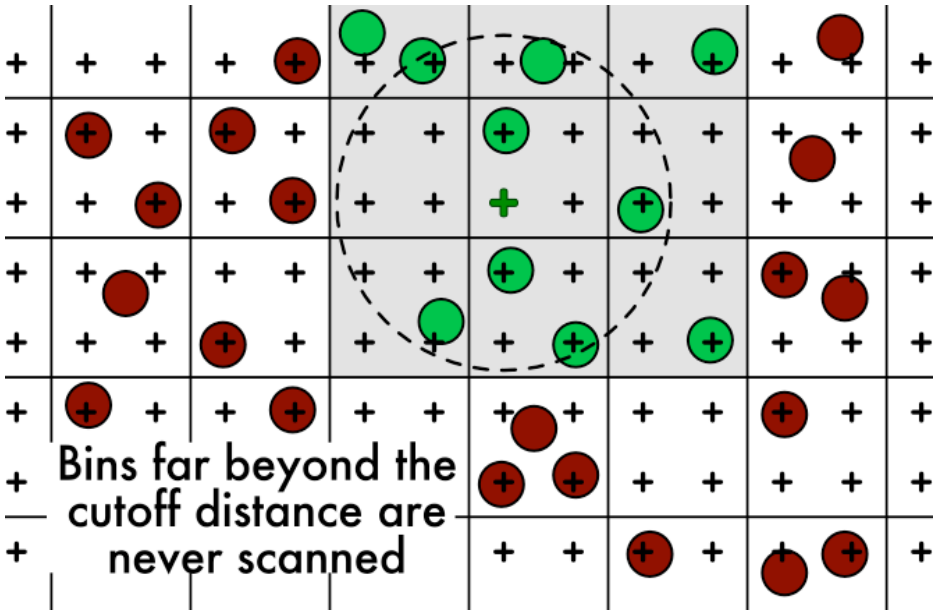
- At each grid point, sum the electrostatic potential from all atoms
- Highly data-parallel
- But has quadratic complexity
  - Number of grid points × number of atoms
  - Both proportional to volume

Algorithm for Pair Potentials With a Cutoff



- Ignore atoms beyond a *cutoff distance*, r_c
  - Typically 8Å–12Å
  - Long-range potential may be computed separately
- Number of atoms within cutoff distance is roughly constant
  - On the order of 1000

Spatial Sorting



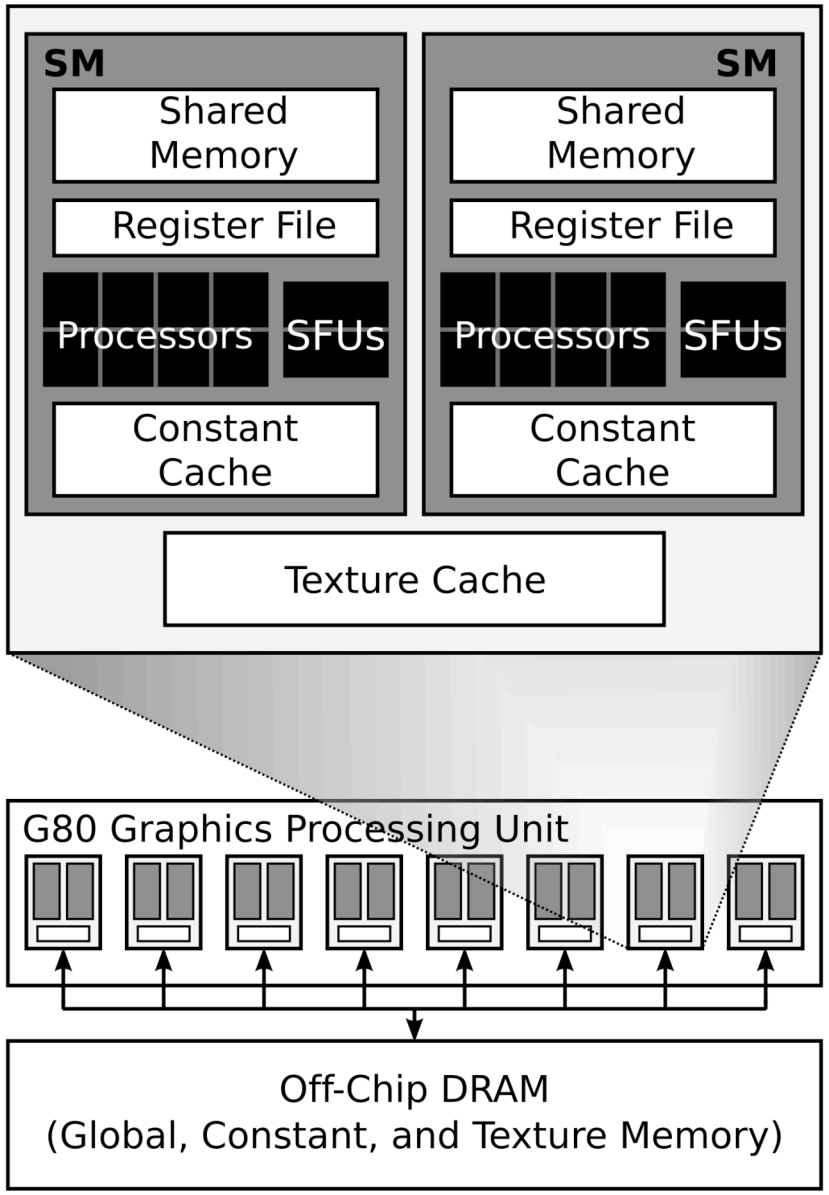
- Presort atoms into *bins* by location in space
- Each bin holds several atoms
- Cutoff potential only uses bins within r_c
  - Yields a linear complexity cutoff potential algorithm

The Draw of GPU Computing

- Raw power
  - Highly parallel, throughput-oriented design
  - 345.6 GFLOPS peak performance
- Programmability
  - C-language programming interface via CUDA
- Adoptability
  - Commodity hardware, easy for users to add to a desktop computer
  - Large install base
    (1 million CUDA-capable GPUs sold per week)

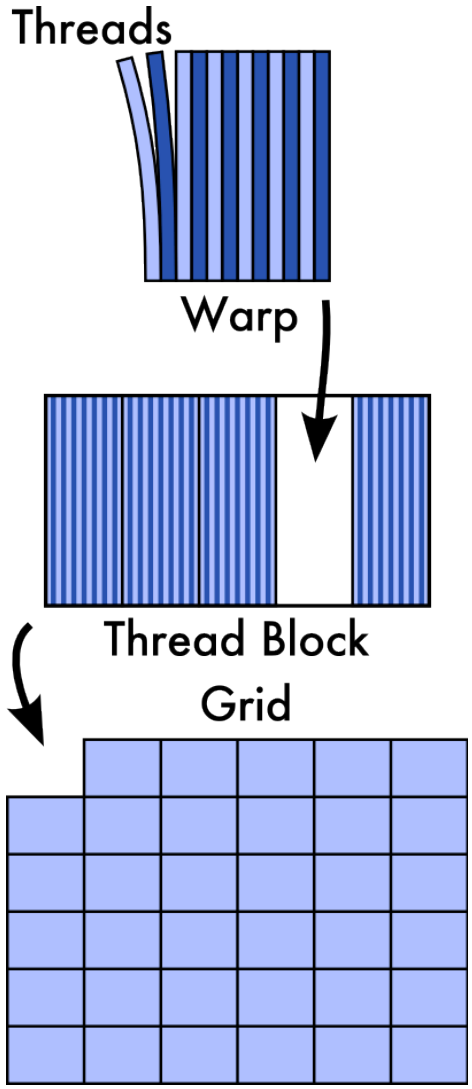
Architecture of the G80 GPU

- 16 Streaming Multiprocessors
  - 8 processors
  - 768 thread contexts
  - Groups of 32 threads (**warps**) run in lockstep
- Large, long-latency, off-chip **global memory**
  - > 200 cycles
  - 64-byte, aligned accesses are most efficient
    - Effected with 16 consecutive accesses from a half-warp
- Scratchpad **shared memory**
  - 16 single-ported banks
- No general-purpose cache



CUDA Programming Model

- Lightweight **threads** multiplexed onto processors
- 32 threads bundled into a **warp**
  - SIMD-like simultaneous instruction issue
- Warps grouped into **thread blocks**
  - Share resources on one Streaming Mutiprocessor
- CPU launches a single **grid** of many thread blocks
  - Thread blocks start executing asynchronously as resources become available



GPU Programming Principles

- Create hundreds of thousands of small, independent threads
  - Keep each thread's resource use small
    - Registers, shared memory
  - Allows many threads to be active simultaneously
- Exploit data locality and conserve memory bandwidth
  - Avoid waiting for off-chip memory accesses
  - Threads in a thread block can take advantage of shared memory on an SM

Previous Cutoff Kernel

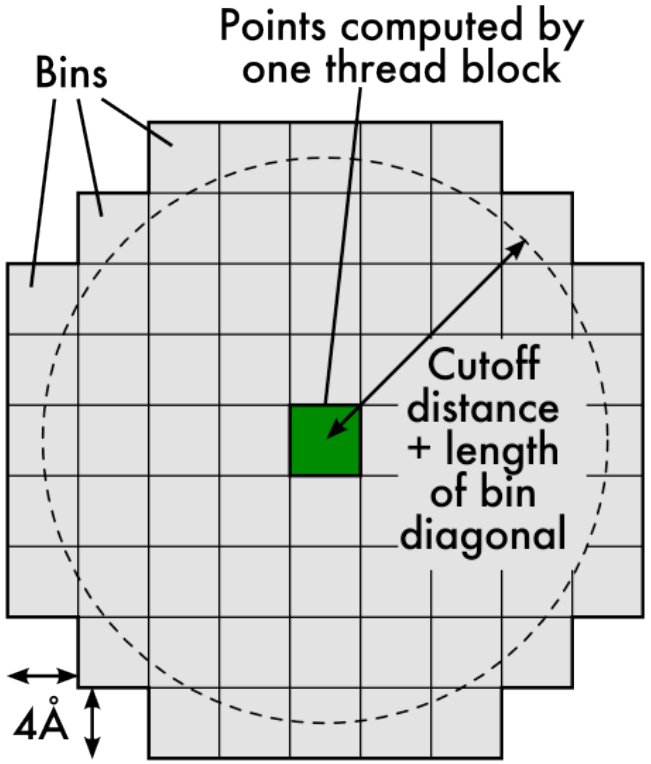
- 6× speedup relative to CPU version
- Work-inefficient
  - Coarse spatial hashing into $(24Å)^3$ bins
  - Only 6.5% of the atoms a thread tests are within the cutoff distance
- Better adaptation of the algorithm to the GPU will gain another 2.5×

Design Considerations for the New Cutoff Kernel

- High memory throughput to atom data essential
  - Group threads together for locality
  - Fetch blocks of data into shared memory
  - Structure atom data to allow fetching
- After taking care of memory demand, optimize to reduce instruction count
  - Loop and instruction-level optimization

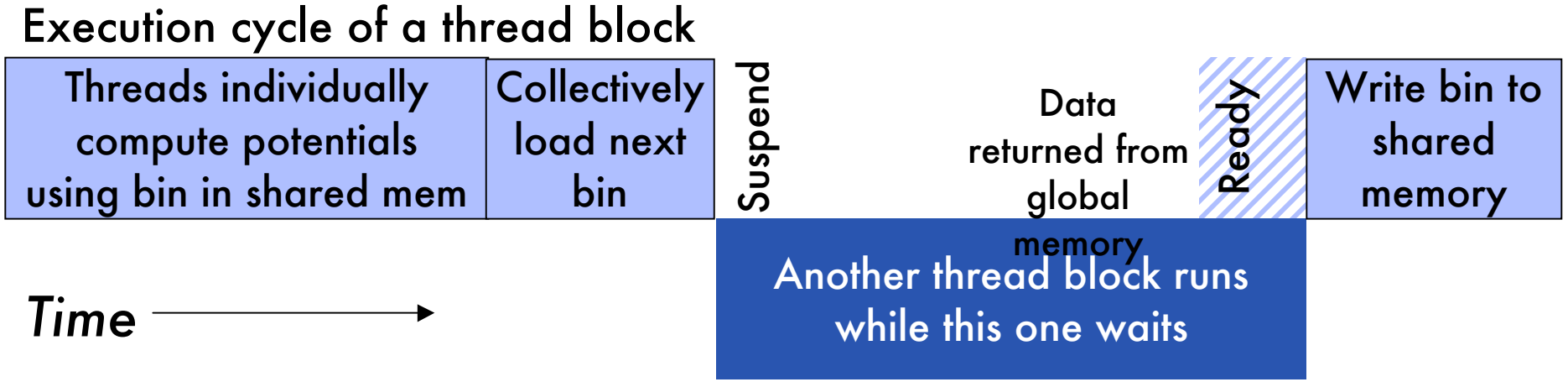
Improving Work Efficiency

- $(4Å)^3$ cube of the potential map computed by each thread block
  - 8×8×8 potential map points
  - 128 threads per block
  - 34% of atoms are within cutoff distance
- Thread block needs atom data up to the cutoff distance
  - Use a sphere of bins
  - All threads in a block scan the same atoms
    - No hardware penalty for multiple simultaneous reads of the same address
    - Simplifies fetching of data



Caching Atom Data

- >200 cycle global memory latency
- Effectively 1 cycle shared memory latency
- Shared memory used in software as a cache
  - Threads in a thread block collectively load one bin at a time into shared memory
  - Once loaded, threads scan atoms in shared memory
  - Reuse: Loaded bins used 128 times



High-Throughput Access to Atom Data

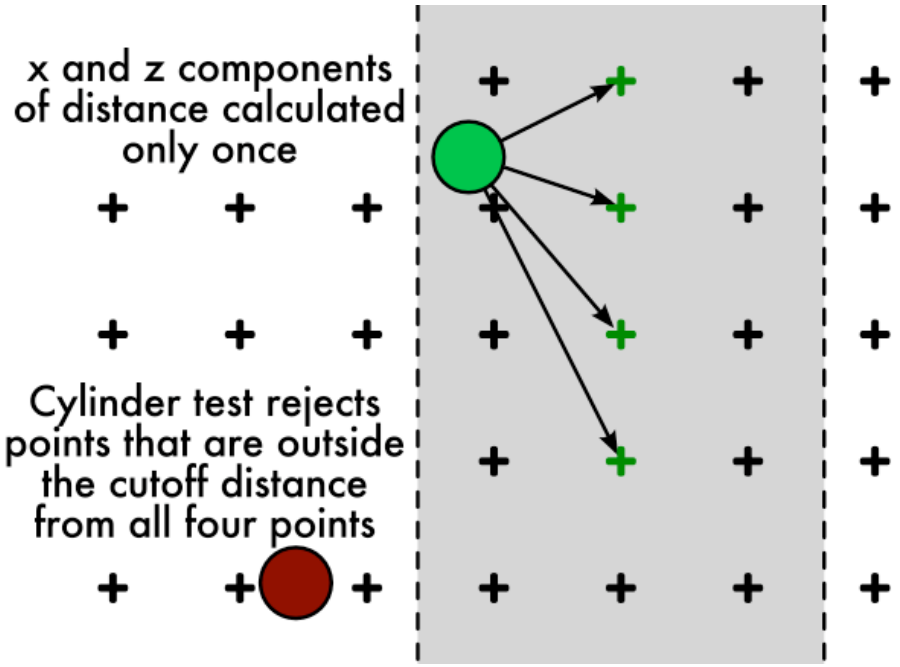
- Full global memory bandwidth only with 64-byte, 64-byte-aligned memory accesses
  - Each bin is exactly 128 bytes
  - Bins stored in a 3D array
- 128 bytes = 8 atoms (x,y,z,q)
  - Nearly uniform density of atoms in typical systems
    - 1 atom per 10 $Å^3$
  - Bins hold atoms from exactly $(4Å)^3$ of space
  - Number of atoms in a bin varies
    - For water test systems, 5.35 atoms in a bin on average
    - Some bins overfull

Handling Overfull Bins

- 2.6% of atoms exceed bin capacity
- Spatial sorting puts these into a list of extra atoms
- Extra atoms processed by the CPU
  - Computed with CPU-optimized algorithm
  - Takes about 66% as long as GPU computation
  - Overlapping GPU and CPU computation yields in additional speedup

GPU Thread Optimization

- Each thread computes potentials at four potential map points
  - Reuse x and z components of distance calculation
  - Check x and z components against cutoff distance (cylinder test)
- Exit inner loop early upon encountering the first empty slot in a bin



GPU Thread Inner Loop

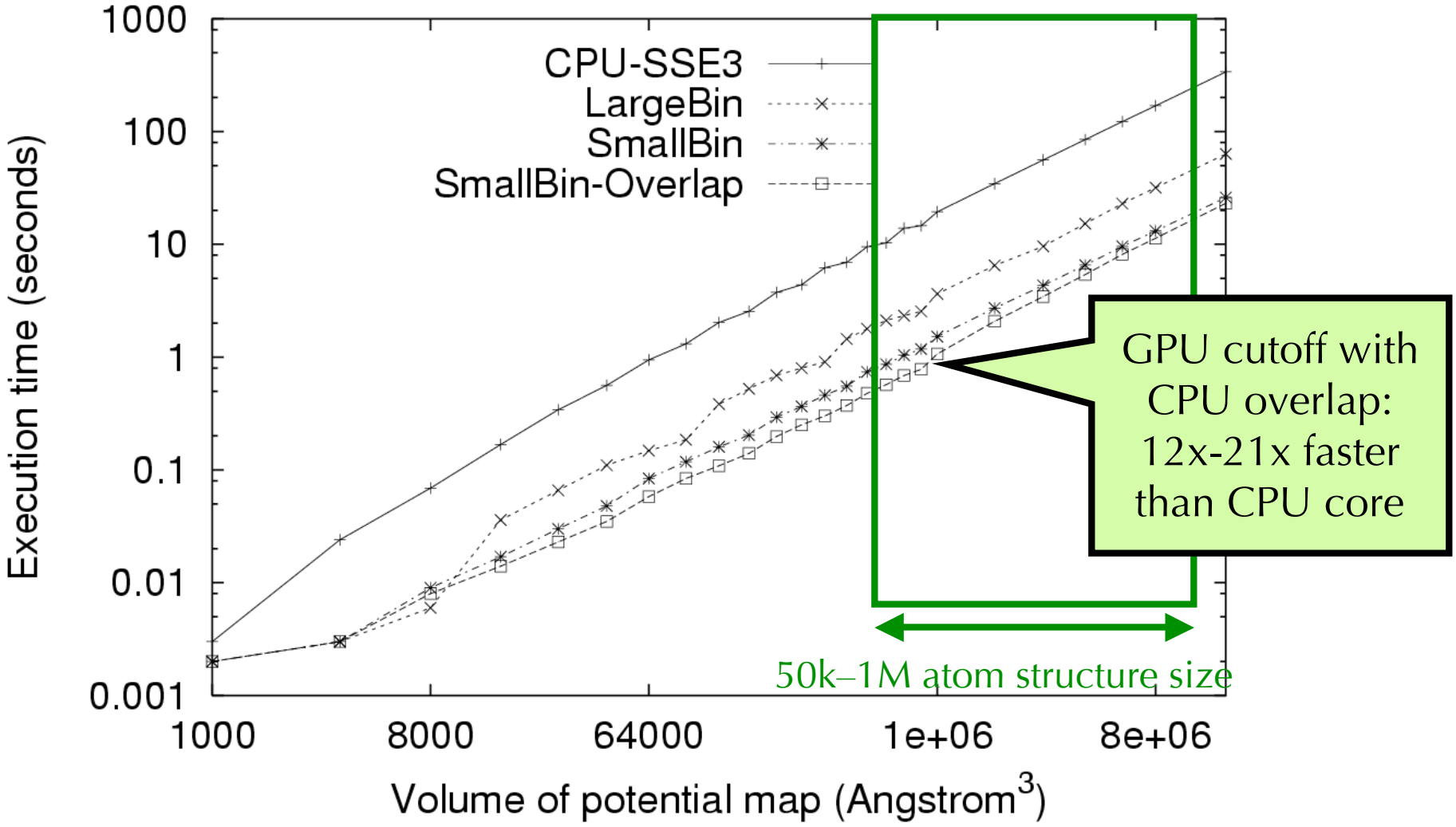
Exit when an empty atom bin entry is encountered

Cylinder test

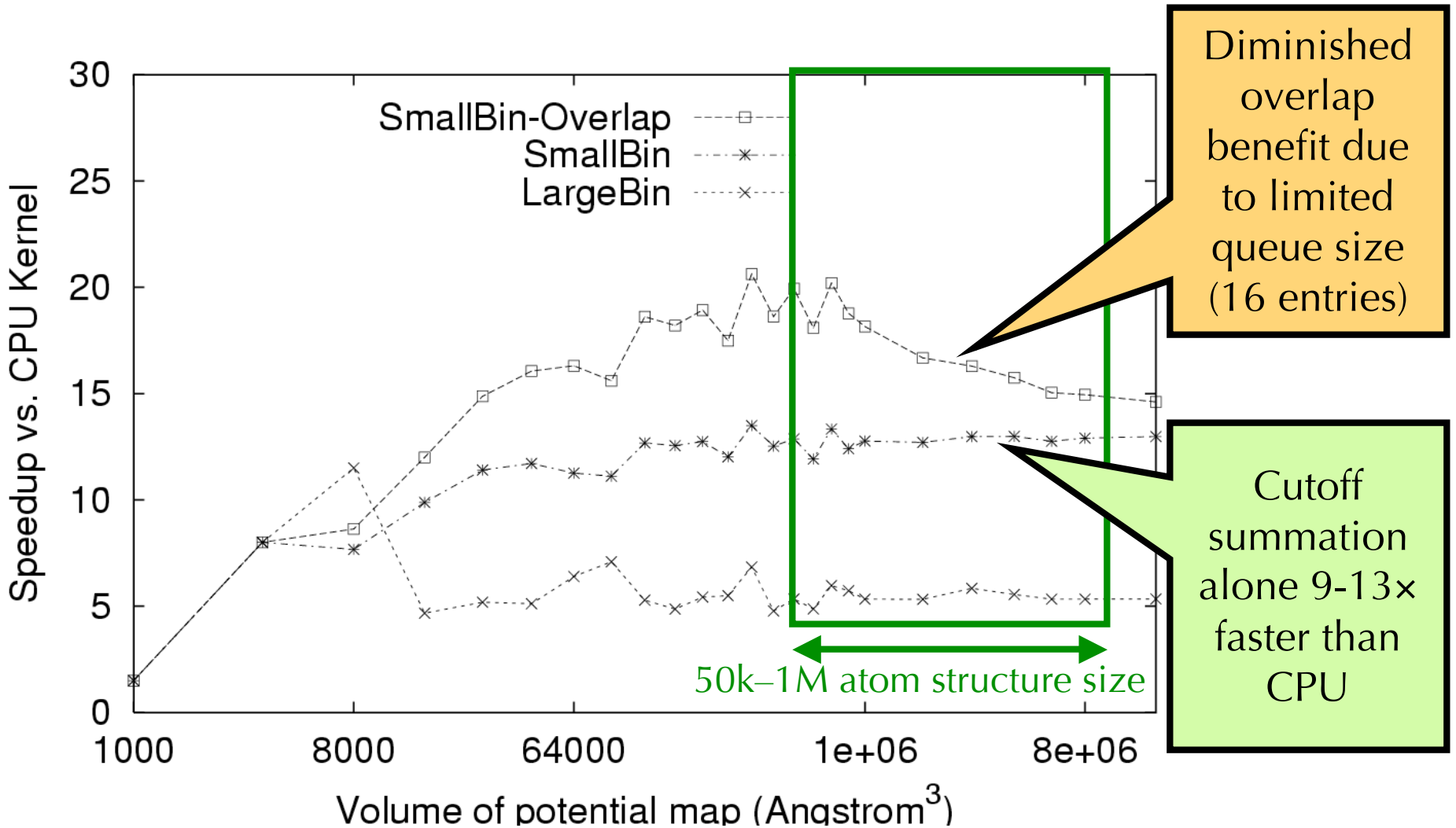
Cutoff test and potential value calculation

```
for (i = 0;  i < BIN_DEPTH;  i++) {
  aq = AtomBinCache[i].w;
  if (aq == 0) break;

  dx = AtomBinCache[i].x - x;
  dz = AtomBinCache[i].z - z;
  dxdz2 = dx*dx + dz*dz;
  if (dxdz2 < cutoff2) continue;

  dy = AtomBinCache[i].y - y;
  r2 = dy*dy + dxdz2;
  if (r2 < cutoff2)
    poten0 += aq * rsqrtf(r2);

  dy = dy - 2 * grid_spacing;
  /* Repeat three more times */
}
```

Cutoff Summation Runtime



Cutoff Summation Speedup



Improving Floating-Point Accuracy

- GPU provides single-precision FP with slightly reduced accuracy on some operations
- Accuracy depends on summation order
  - FP addition is not associative
- Compensated summation improves accuracy
  - Less than 10% performance cost on GPU

	Maximum % relative error	Maximum % absolute error
CPU	0.4793	0.0000939
GPU	0.8715	0.0001579
GPU with Compensated Summation	0.5710	0.0001579

Error relative to double-precision floating point on CPU

Summary

- Cutoff pair potentials heavily used in molecular modeling applications
- Use CPU to regularize the work given to the GPU to optimize its performance
  - GPU performs very well on 64-byte-aligned array data
- Run CPU and GPU concurrently to improve performance
- Use shared memory as a program-managed cache

Thank You

Publications in Molecular Modeling:

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.

in GPU Optimization:

Program Optimization Space Pruning for a Multithreaded GPU. S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S.-Z. Ueng, J. A. Stratton, W.-M. W. Hwu. in *CGO* 2008.

Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA. S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, W.-M. W. Hwu. in *PPoPP* 2008.

Backup Slides

Ion Placement

- Selection of initial conditions for a negatively charged virus in water
- Neutralize charge by adding positively charged ions
  - Also stabilizes the virus structure
- Ions placed at sites with most negative electrostatic potential

Simplified Pseudocode of the Cutoff Pair Potentials Algorithm

```
for each grid point, r_j:
 for each nearby bin, B:
  for each atom (q,r) in B:
    dr = |r - r_j|
    if dr < r_c:
      s = (1 - (dr/r_c)^2)^2
      V(r_j) = V(r_j) + q/dr * s
```