Summer School

e-Science with Many-core CPU/GPU Processors

Lecture 11: Case Study 4

Cut-off Binning for Data and Parallelism Scalability

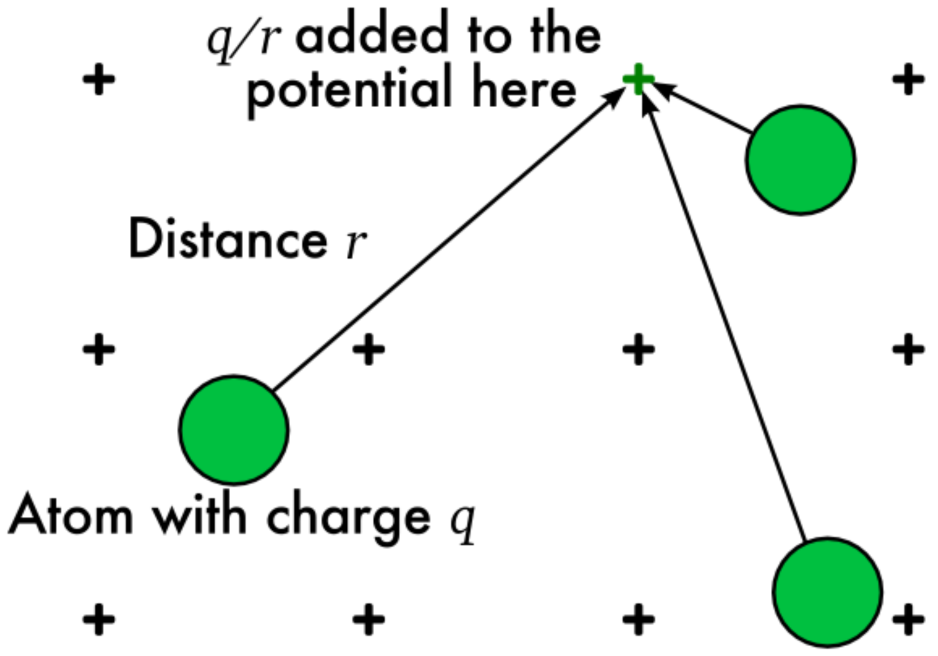
Acknowledgement

- Additional Information and References:
  - http://www.ks.uiuc.edu/Research/gpu/
  - http://www.ks.uiuc.edu/Research/vmd/
- Acknowledgement, questions, source code requests:
  - Chris Rodrigues
  - John Stone johns@ks.uiuc.edu
  - Klaus Schulten
  - Theoretical and Computational Biophysics Group, NIH Resource for Macromolecular Modeling and Bioinformatics Beckman Institute for Advanced Science and Technology
- NIH support: P41-RR05969

Outline

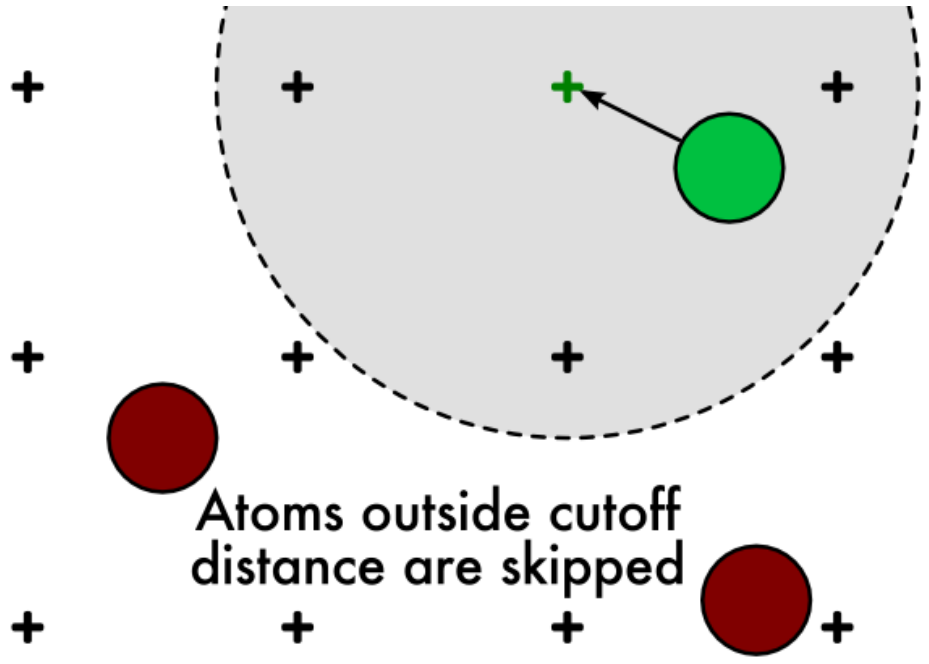
- Explore CUDA versions of the direct Coulomb summation (DCS) algorithm
  - Used for ion placement and time-averaged electrostatic potential calculations
  - Detailed look at a few CUDA implementations of DCS
  - Multi-GPU DCS potential map calculation
- Experiences integrating CUDA kernels into VMD

Algorithm for Pair Potentials



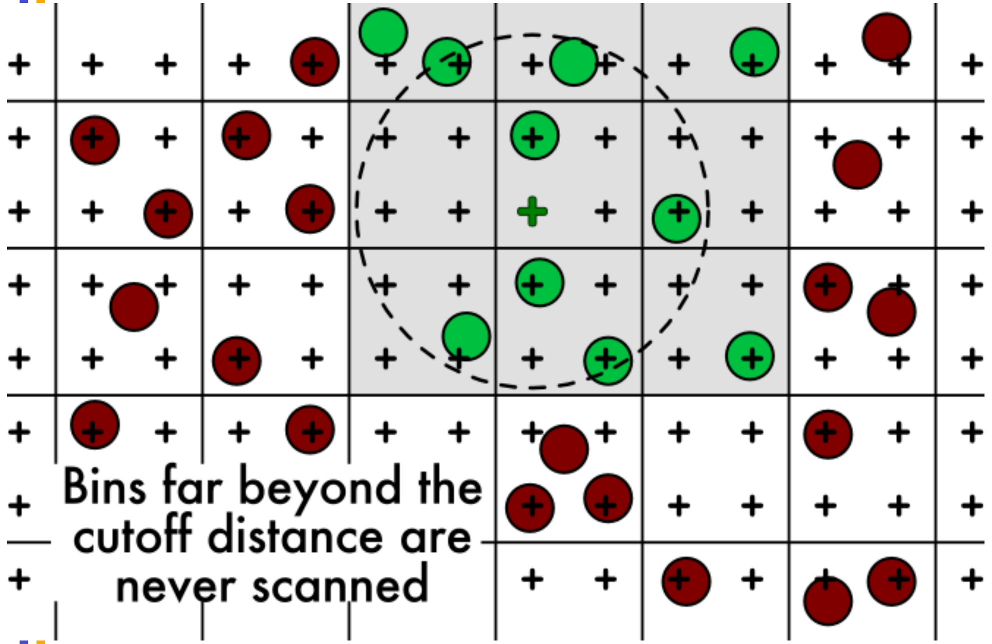
- At each grid point, sum the electrostatic potential from all atoms
- Highly data-parallel
- But has quadratic complexity
  - Number of grid points × number of atoms
  - Both proportional to volume

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

Large-bin Cutoff Kernel

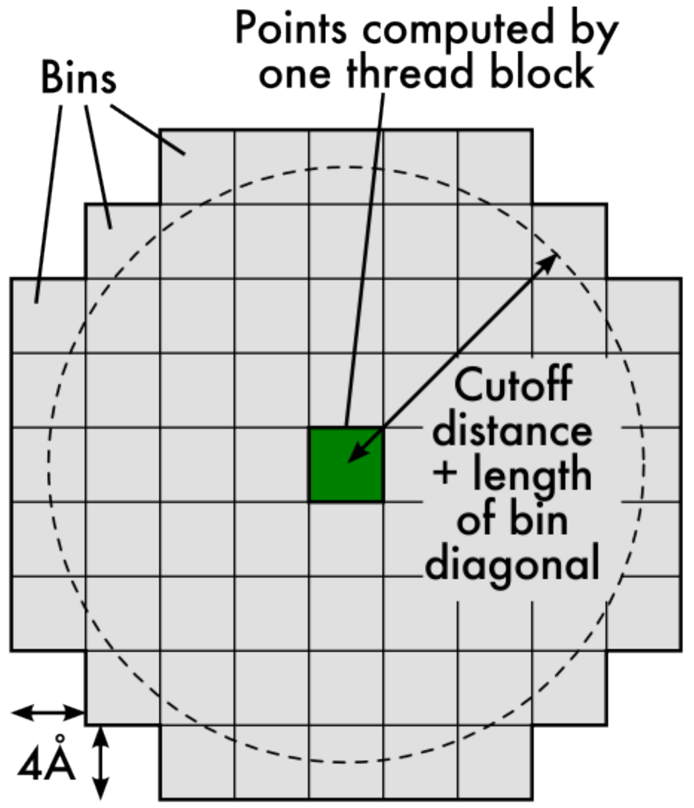
- 6× speedup relative to CPU version
- Work-inefficient
  - Coarse spatial hashing into (24Å)^3 bins
  - Only 6.5% of the atoms a thread tests are within the cutoff distance
- Better adaptation of the algorithm to the GPU will gain another 2.5×

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

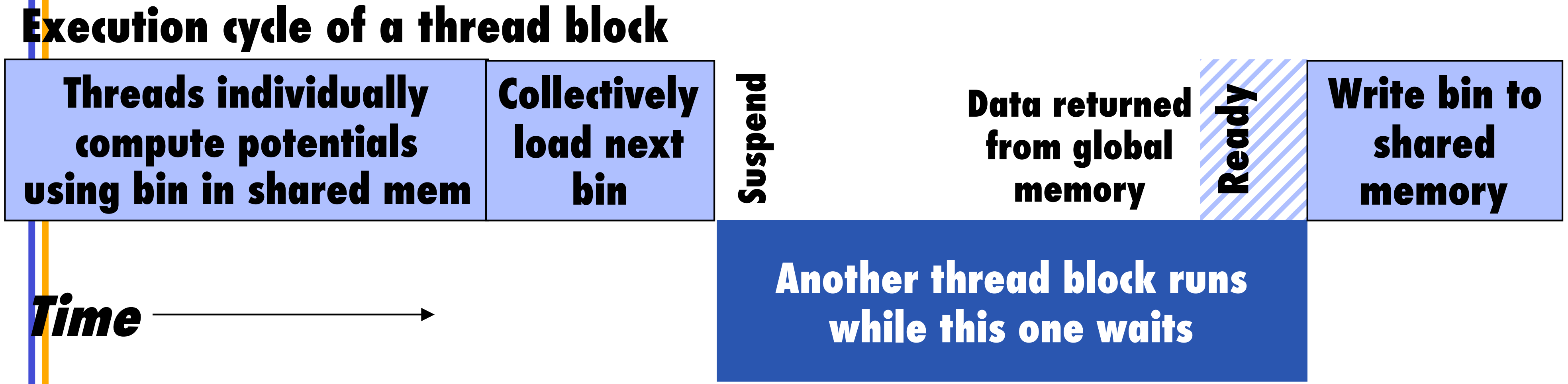
Improving Work Efficiency

- $(4\text{Å})^3$ 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


Bins
Points computed by one thread block
Cutoff distance + length of bin diagonal
4Å

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


Execution cycle of a thread block
Threads individually compute potentials using bin in shared mem
Collectively load next bin
Suspend
Data returned from global memory
Ready
Write bin to shared memory
Another thread block runs while this one waits
Time

High-Throughput Access to Atom Data

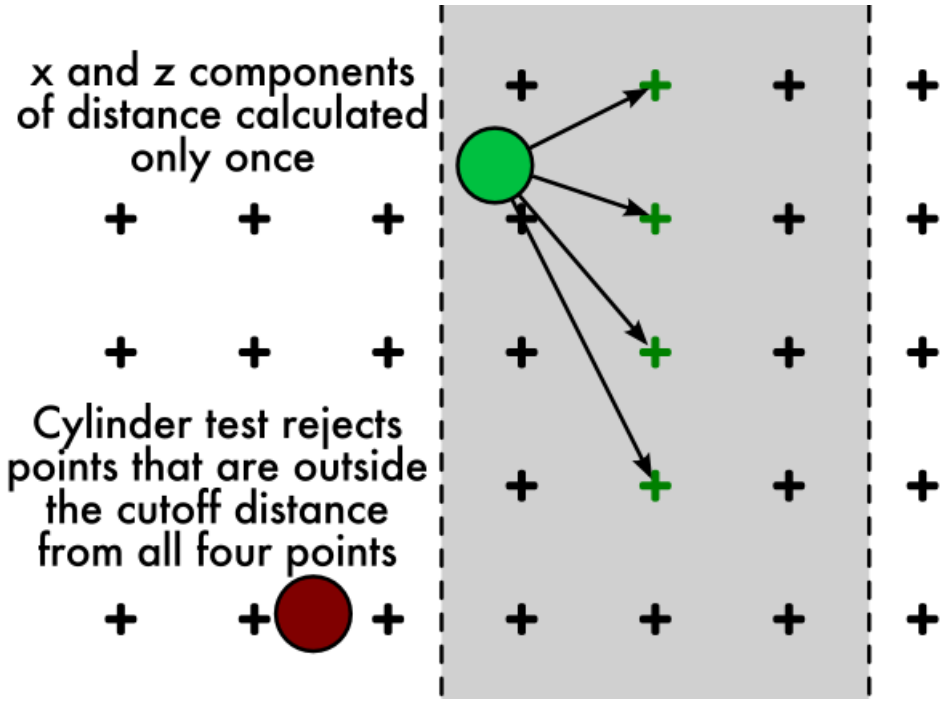
- Full global memory bandwidth only with 64-byte, 64-byte-aligned memory accesses
  - Each bin is exactly 128 bytes
  - Bins stored in a 3D array
  - 32 threads in each block load one bin, which is processed by all threads in the block
- 128 bytes = 8 atoms (x,y,z,q)
  - Nearly uniform density of atoms 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


x and z components of distance calculated only once
Cylinder test rejects points that are outside the cutoff distance from all four points

GPU Thread Inner Loop

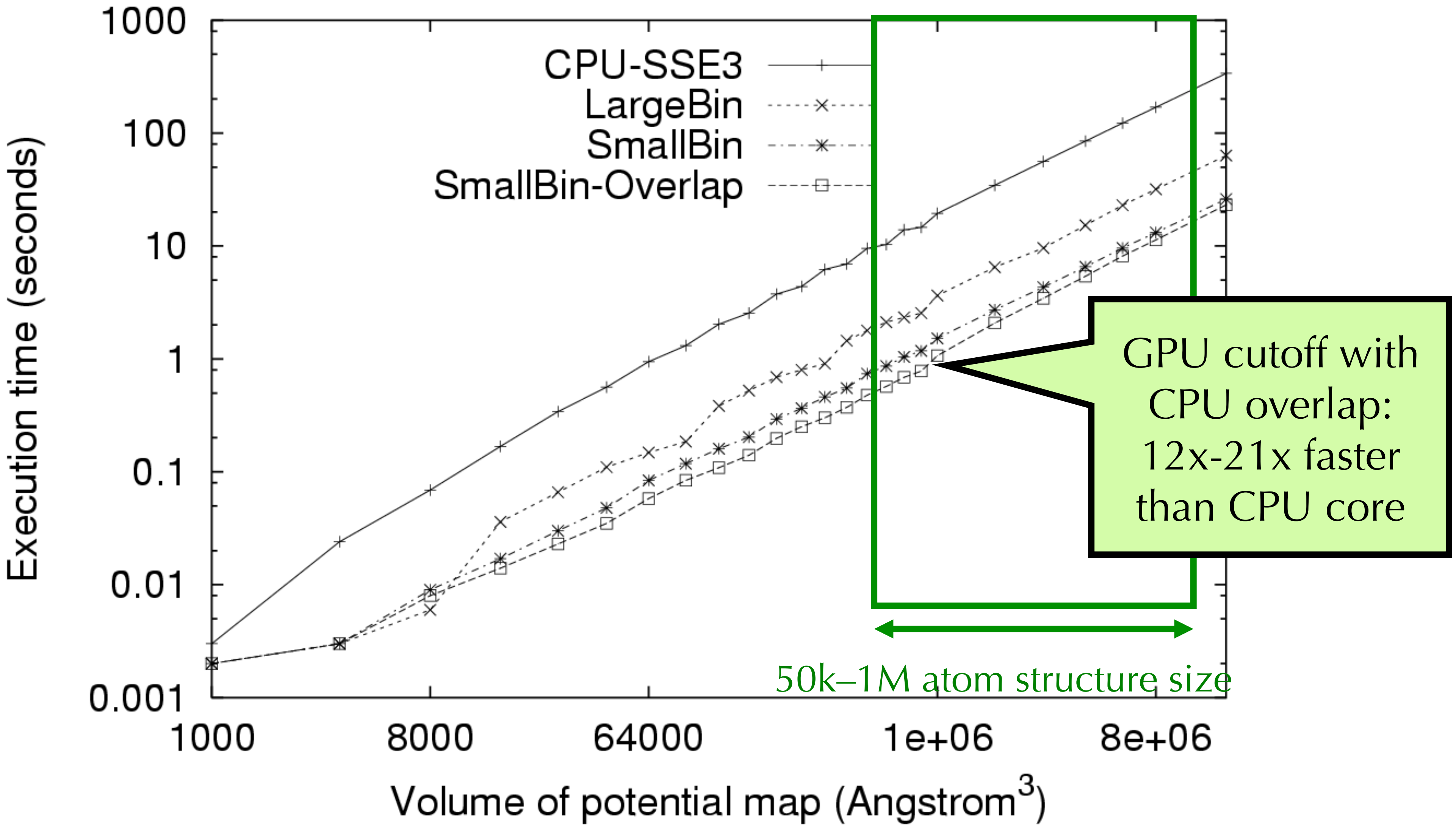
Exit when an empty atom bin entry is encountered

Cylinder test

Cutoff test and potential value calculation

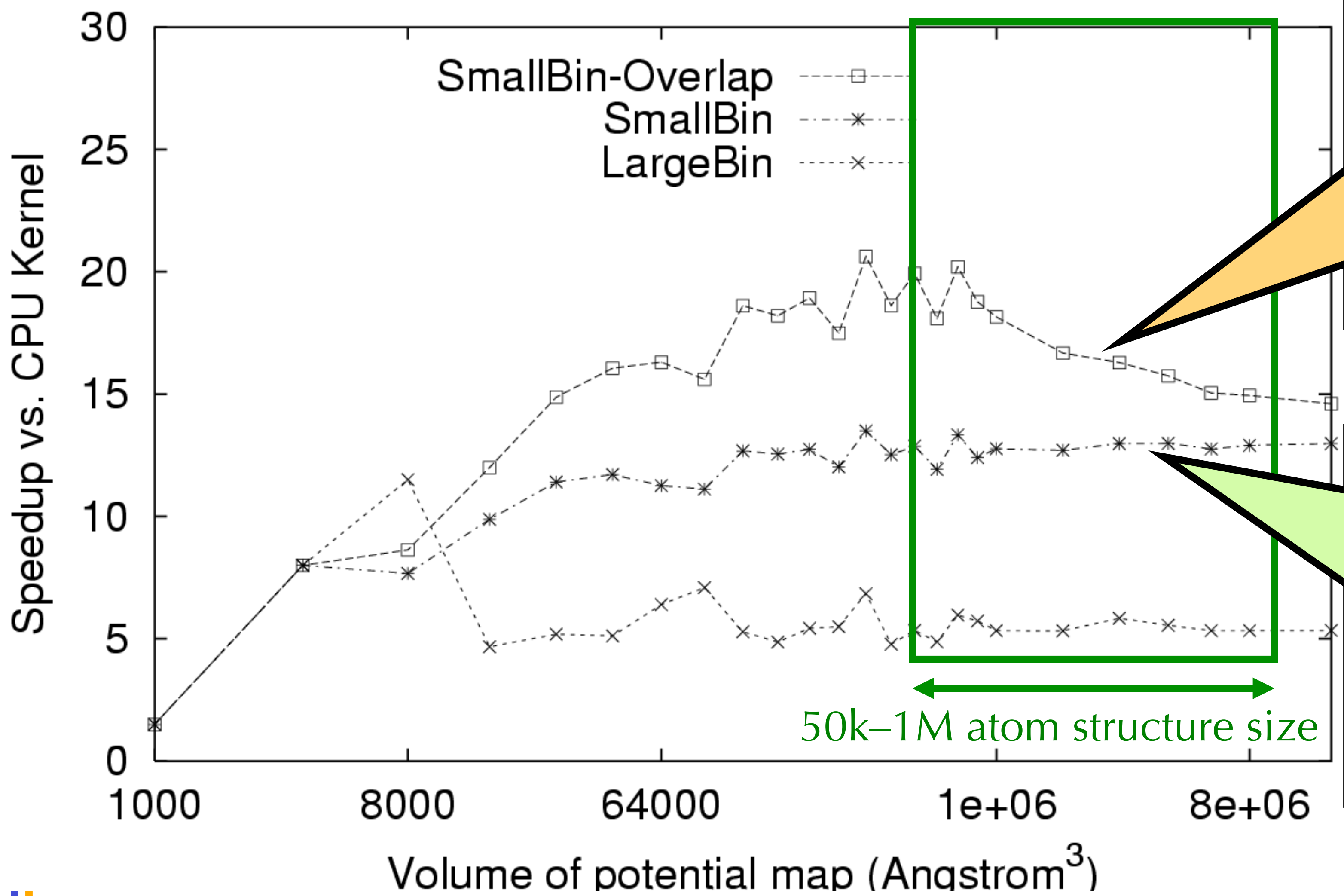
```
for (i = 0;  i < BIN_DEPTH;  i++) {
  aq = AtomBinCache[i].w;
  if (aq == 0) break;

  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



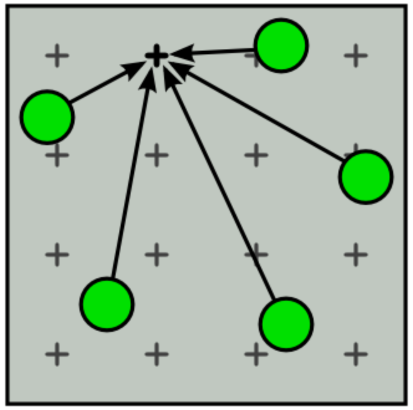
GPU cutoff with CPU overlap: 12x-21x faster than CPU core

Cutoff Summation Speedup

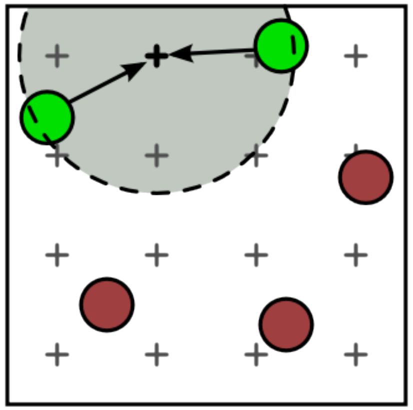


Diminished overlap benefit due to limited queue size (16 entries)

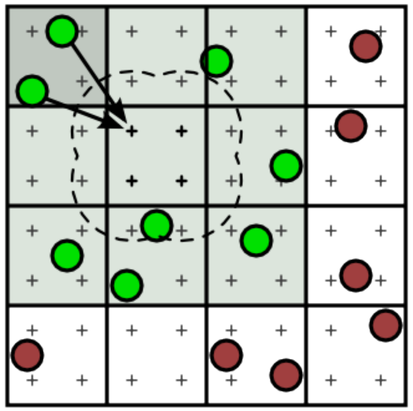
Cutoff summation alone 9-13× faster than CPU



(a) Direct summation
At each grid point, sum the electrostatic potential from all charges



(b) Cutoff summation
Electrostatic potential from nearby charges summed; spatially sort charges first



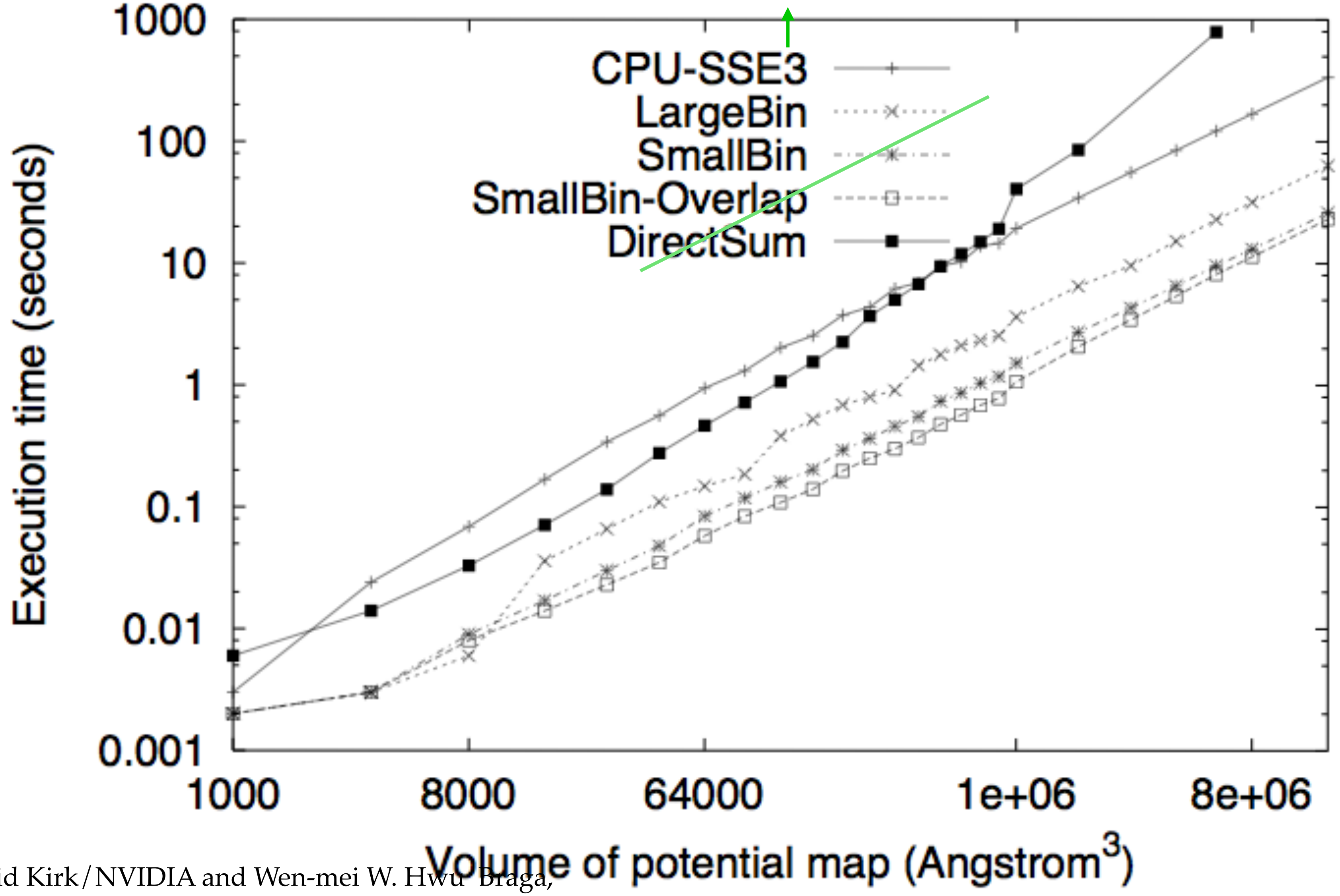
(c) Cutoff summation using direct summation kernel
Spatially sort charges into bins; adapt direct summation to process a bin

Figure 10.2 Cutoff Summation algorithm

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

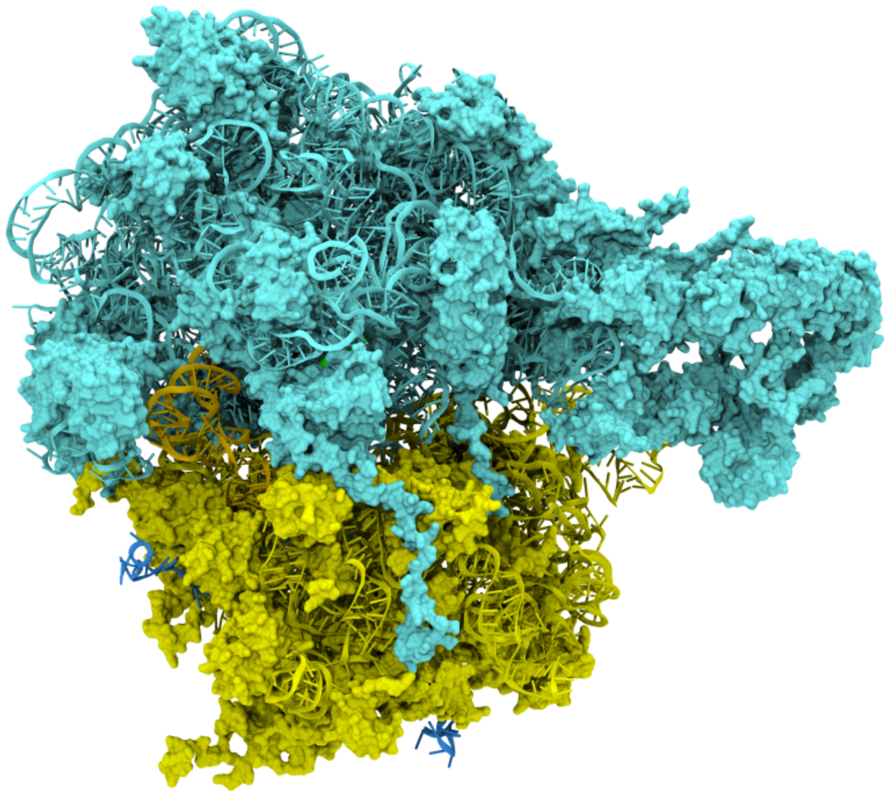
Data Scalability with Cut-off Methods



Rodrigues, et al, ACM Computing Frontier 2008

Experiences Integrating CUDA Kernels Into VMD

- VMD: molecular visualization and analysis
- State-of-the-art simulations require more viz/analysis power than ever before
- For some algorithms, CUDA can bring what was previously supercomputer class performance to an appropriately equipped desktop workstation



Ribosome: 260,790 atoms before adding solvent/ ions

VMD/CUDA Integration Observations

- Single VMD binary must run on all hardware, whether CUDA accelerators are installed or not
- Must maintain both CPU and CUDA versions of kernels
- High performance requirements mean that the CPU kernel may use a different memory layout and algorithm strategy than CUDA, so they could be entirely different bodies of code to maintain
- Further complicated by the need to handle both single-threaded and multithreaded compilations, support for many platforms, etc…

VMD/CUDA Integration Observations (2)

- Evolutionary approach to acceleration:
  As new CUDA kernels augment existing CPU kernels, the original class/function becomes a wrapper that dynamically executes the best CPU/GPU kernels at runtime
- VMD’s current CUDA kernels are always faster than the CPU, so its runtime strategy can be nearly as simple as:

```
int err = 1; // force CPU execution if CUDA is not compiled in
#if defined(VMDCUDA)
if (cudagpucount > 0)
  err=CUDAKernel(); // try CUDA kernel if GPUs are available
#endif
if (err)
  err=CPUKernel(); // if no CUDA GPUs or an error occurred, try on CPU
…
```

VMD/CUDA Integration Observations (3)

- Graceful behavior under errors or resource exhaustion conditions is trickier to deal with:
  - CPU kernel becomes the fallback in most cases
  - What to do when the CPU version is 100x slower than CUDA?!? A CPU fallback isn't very helpful in this case. Aborting or issuing a performance warning to the user may be more appropriate.
- All of these design problems already existed:
  - Not specific to CUDA
  - CUDA just adds another ply to the existing situation for codes that employ multiple computation strategies

VMD/CUDA Code Organization

- Main application holds data needed for execution strategy, CPU/GPU load balancing, etc.
- Single header file containing all the CUDA kernel function prototypes, easy inclusion in other src files
- Separate .cu files for each kernel:
  - each in their compilation unit
  - no need to worry about multiple kernels sharing space for constant buffers etc…