M02: High Performance Computing with CUDA

Molecular Visualization and Analysis

John Stone

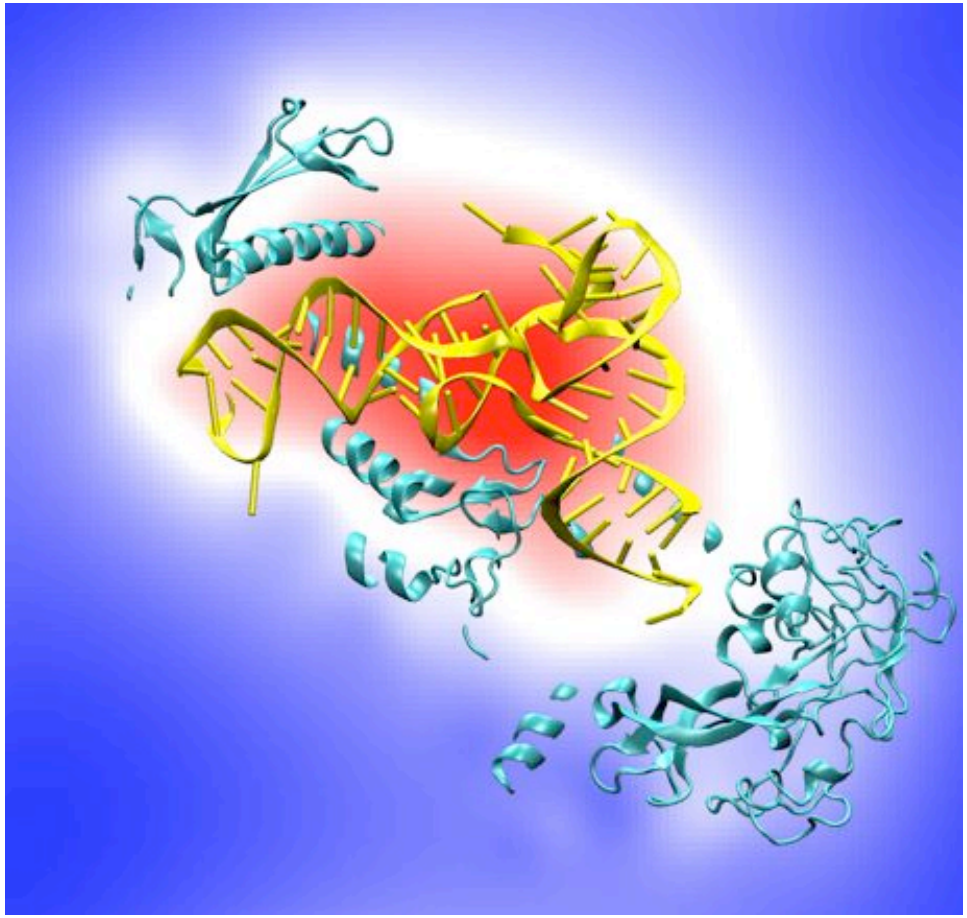
University of Illinois at Urbana-Champaign

Outline

- Explore CUDA algorithms for computing electrostatic fields around molecules
  - Detailed look at a few CUDA implementations of a simple direct Coulomb summation algorithm
    - Basic algorithm, related algorithms
    - Improving per-thread performance with loop unroll-and-jam, and other optimizations
    - Trade-off between concurrency and per-thread performance
  - Multi-GPU direct Coulomb summation
  - Cutoff (range-limited) summation algorithm
    - Using the CPU to optimize GPU performance
    - Using the CPU and GPU concurrently
- Lessons learned

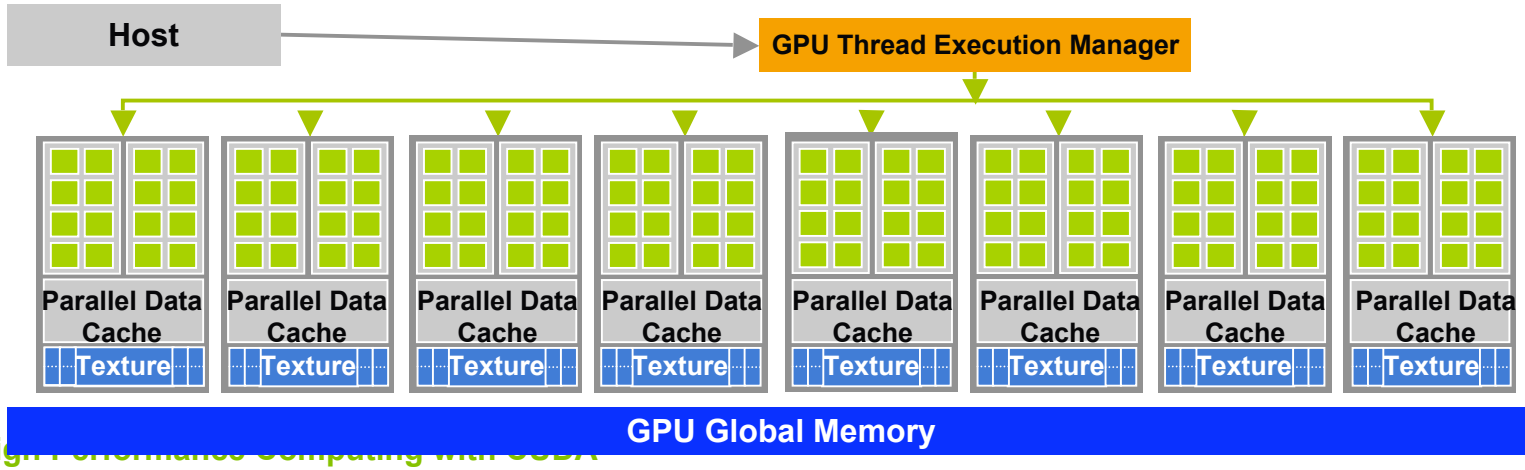
Calculating Electrostatic Potential Maps

- Used in molecular structure building, analysis, visualization, simulation
- Electrostatic potentials evaluated on a uniformly spaced 3-D lattice
- Each lattice point contains sum of electrostatic contributions of all atoms
- Electrostatic potential computation is quite similar to several other particle-grid problems in molecular modeling, these kernels form a template for successfully implementing other GPU accelerated algorithms of this type



Direct Coulomb Summation on the GPU

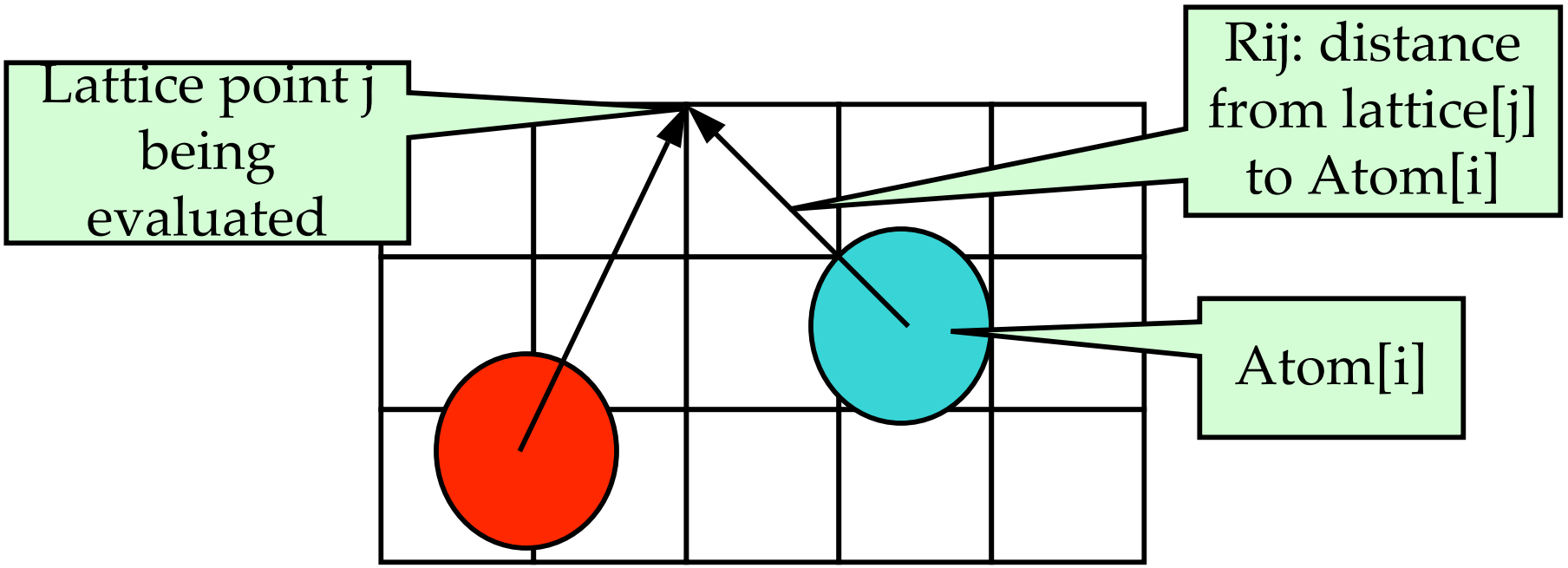
- Starting point for more sophisticated algorithms
- GPU outruns a CPU core by 44x
- Work is decomposed into tens of thousands of independent threads, multiplexed onto hundreds of GPU processor cores
- Single-precision FP arithmetic is adequate for intended application
- Numerical accuracy can be further improved by compensated summation, spatially ordered summation groupings, or accumulation of potential in double-precision



Direct Coulomb Summation

- At each lattice point, sum potential contributions for all atoms in the simulated structure:

  potential[j] += charge[i] / Rij



DCS Algorithm Design Observations

- Atom list has the smallest memory footprint, best choice for the inner loop (both CPU and GPU)
- Lattice point coordinates are computed on-the-fly
- Atom coordinates made relative to the origin of the potential map, eliminating redundant arithmetic
- Arithmetic can be significantly reduced by precalculating and reusing distance components, e.g. create a new array containing X, Q, and dy^2 + dz^2, updated on-the-fly for each row (CPU)
- Vectorized CPU versions benefit greatly from SSE instructions

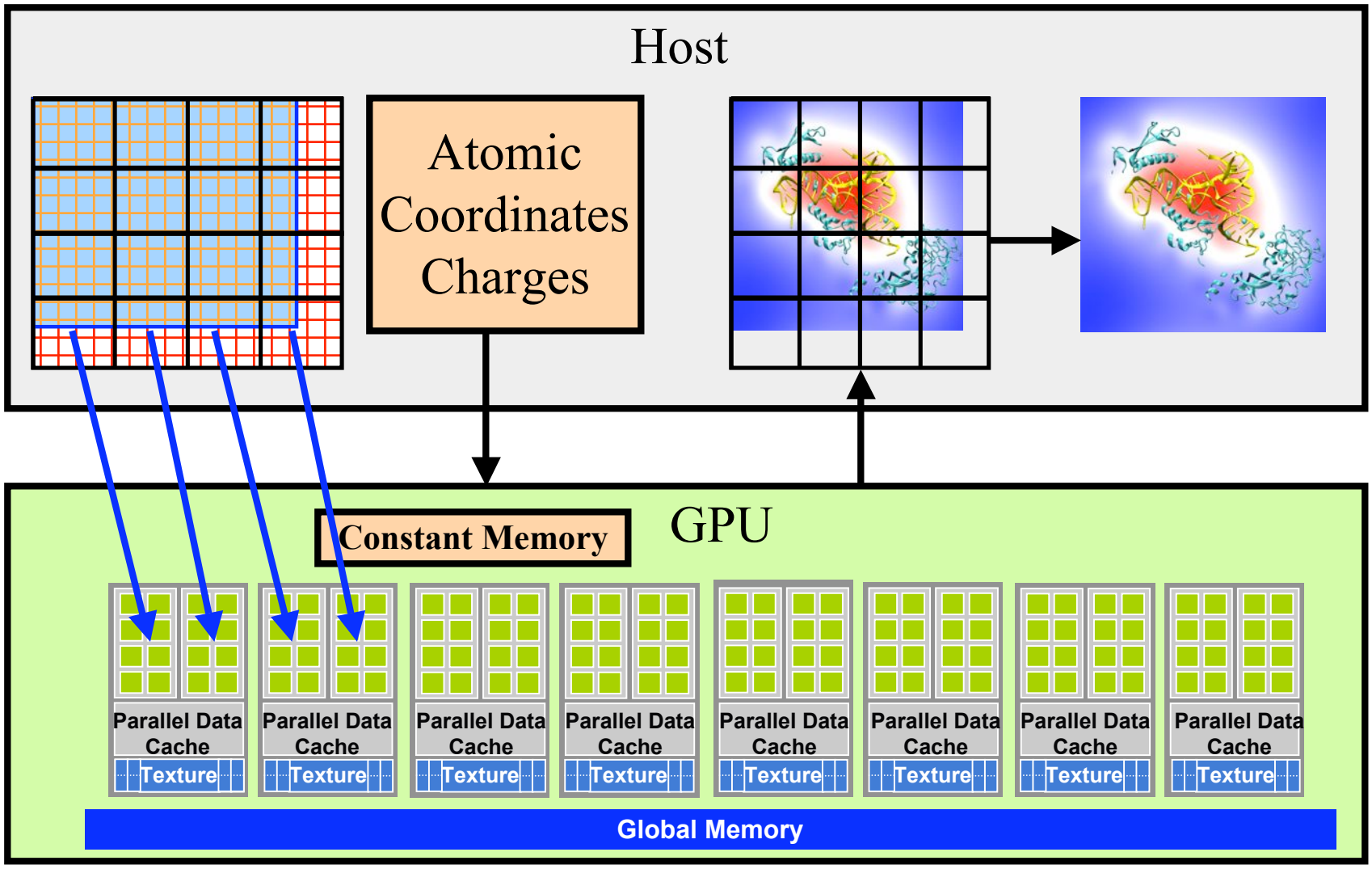
Single Slice DCS: Simple (Slow) C Version


```
void cenergy(float *energygrid, dim3 grid, float gridspacing, float z, const float *atoms,
      int numatoms) {
  int i,j,n;
  int atomarrdim = numatoms * 4;
  for (j=0; j<grid.y; j++) {
    float y = gridspacing * (float) j;
    for (i=0; i<grid.x; i++) {
      float x = gridspacing * (float) i;
      float energy = 0.0f;
      for (n=0; n<atomarrdim; n+=4) {     // calculate potential contribution of each atom
        float dx = x - atoms[n    ];
        float dy = y - atoms[n+1];
        float dz = z - atoms[n+2];
        energy += atoms[n+3] / sqrtf(dx*dx + dy*dy + dz*dz);
      }
      energygrid[grid.x*grid.y*k + grid.x*j + i] = energy;
    }
  }
}
```

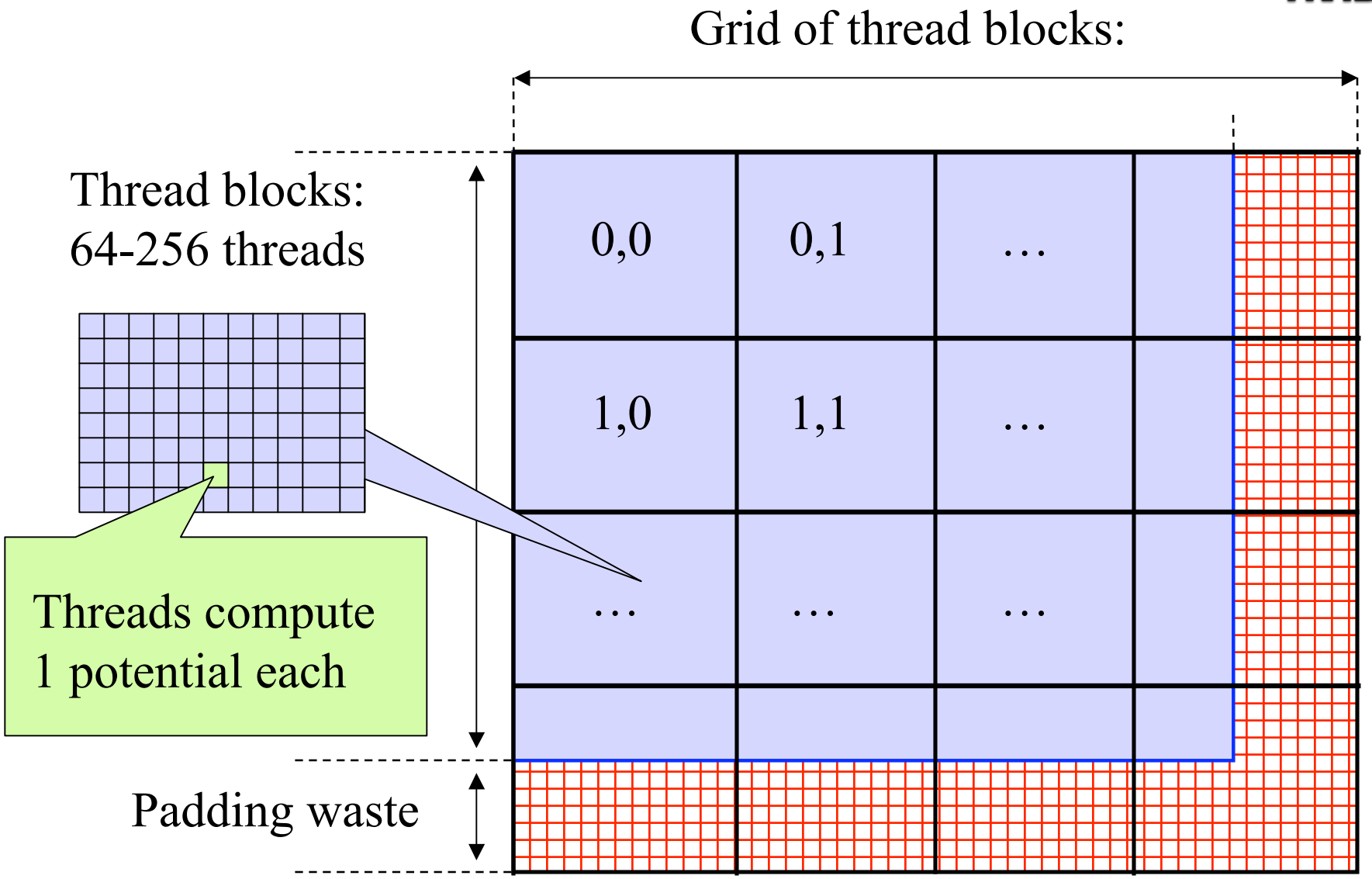

DCS Observations for GPU Implementation

- Straightforward implementation has a low ratio of FLOPS to memory operations (for a GPU…)
- The innermost loop will consume operands VERY quickly
- Since atoms are read-only, they are ideal candidates for texture memory or constant memory
- GPU implementations must access constant memory efficiently, avoid shared memory bank conflicts, and overlap computations with global memory latency
- Map is padded out to a multiple of the thread block size:
  - Eliminates conditional handling at the edges, thus also eliminating the possibility of branch divergence
  - Assists with memory coalescing

Direct Coulomb Summation on the GPU



DCS CUDA Block/Grid Decomposition
(non-unrolled)



DCS Version 1: Const+Precalc
187 GFLOPS, 18.6 Billion Atom Evals/Sec

- Pros:
  - Pre-compute dz^2 for entire slice
  - Inner loop over read-only atoms, const memory ideal
  - If all threads read the same const data at the same time, performance is similar to reading a register
- Cons:
  - Const memory only holds ~4000 atom coordinates and charges
  - Potential summation must be done in multiple kernel invocations per slice, with const atom data updated for each invocation
  - Host must shuffle data in/out for each pass

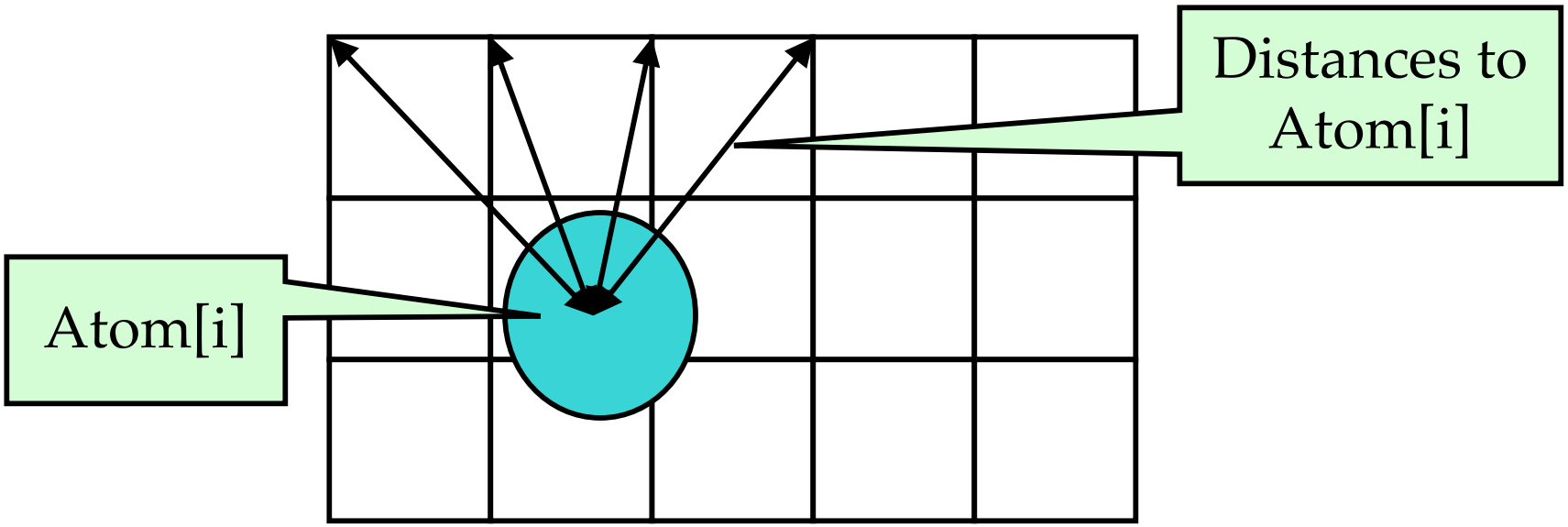
DCS Version 1: Kernel Structure


```
…
 float curenergy = energygrid[outaddr];  // start global mem read very early
 float coorx = gridspacing * xindex;
 float coory = gridspacing * yindex;
 int atomid;
 float energyval=0.0f;

 for (atomid=0; atomid<numatoms; atomid++) {
   float dx = coorx - atominfo[atomid].x;
   float dy = coory - atominfo[atomid].y;
   energyval += atominfo[atomid].w * rsqrtf(dx*dx + dy*dy + atominfo[atomid].z);
 }
 energygrid[outaddr] = curenergy + energyval;
```


DCS CUDA Algorithm: Unrolling Loops

- Reuse atom data and partial distance components multiple times
- Add each atom's contribution to several lattice points at a time, where distances only differ in one component
- Use "unroll and jam" to unroll the outer loop into the inner loop
- Uses more registers, but increases arithmetic intensity significantly



DCS Inner Loop (Unroll and Jam)

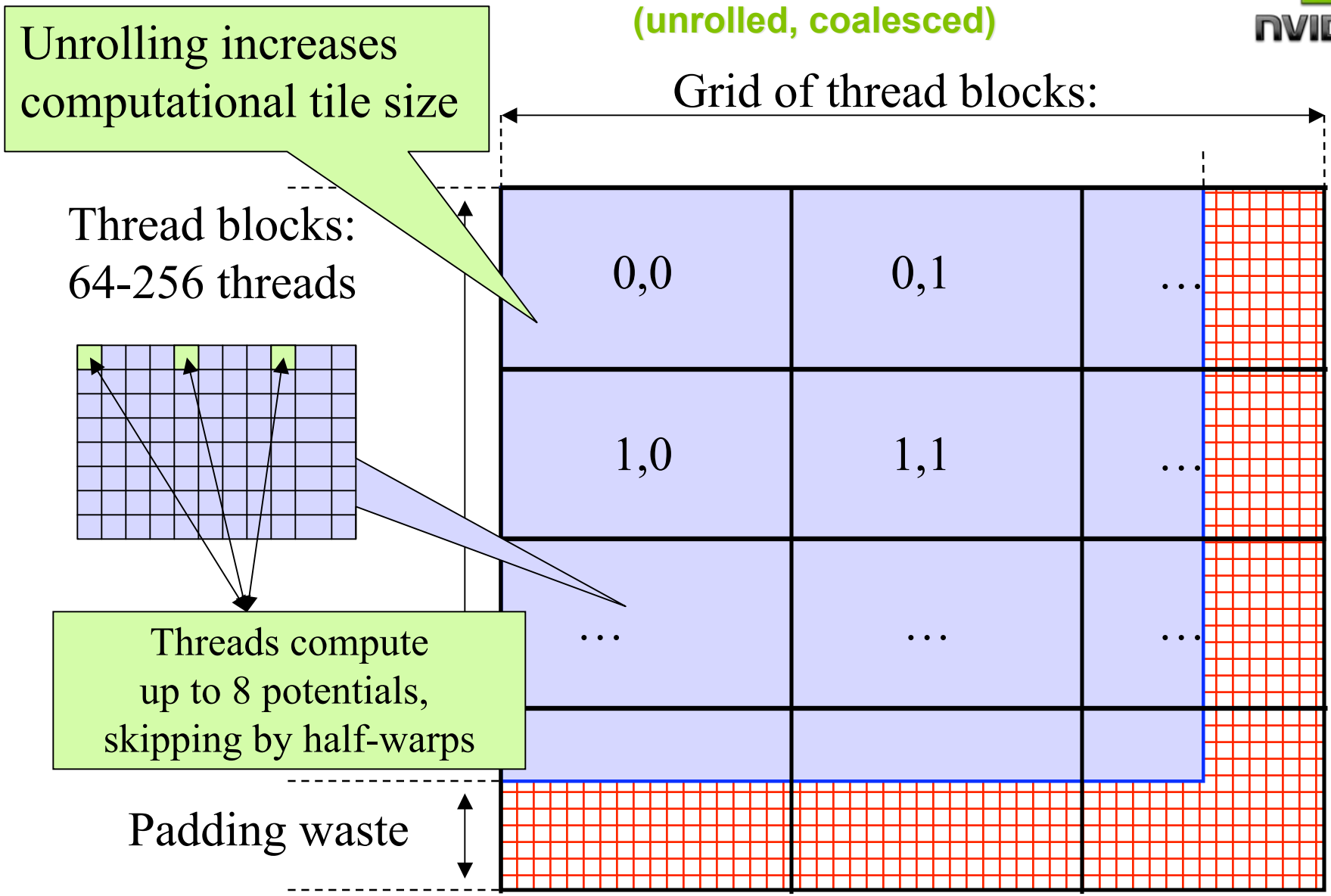

```
…
 for (atomid=0; atomid<numatoms; atomid++) {
   float dy = coory - atominfo[atomid].y;
   float dysqpdzsq = (dy * dy) + atominfo[atomid].z;
   float dx1 = coorx1 - atominfo[atomid].x;
   float dx2 = coorx2 - atominfo[atomid].x;
   float dx3 = coorx3 - atominfo[atomid].x;
   float dx4 = coorx4 - atominfo[atomid].x;
   energyvalx1 += atominfo[atomid].w * rsqrtf(dx1*dx1 + dysqpdzsq);
   energyvalx2 += atominfo[atomid].w * rsqrtf(dx2*dx2 + dysqpdzsq);
   energyvalx3 += atominfo[atomid].w * rsqrtf(dx3*dx3 + dysqpdzsq);
   energyvalx4 += atominfo[atomid].w * rsqrtf(dx4*dx4 + dysqpdzsq);
 }
…
```


DCS CUDA Block/Grid Decomposition (unrolled)

- Kernel variations that calculate more than one lattice point per thread, result in larger computational tiles:
  - Thread count per block must be decreased to reduce computational tile size as unrolling is increased
  - Otherwise, tile size gets bigger as threads do more than one lattice point evaluation, resulting on a significant increase in padding and wasted computations at edges
- It is often beneficial to use templates and/or other techniques to generate kernels tuned for small, medium, and large size problems to achieve peak performance

DCS CUDA Block/Grid Decomposition

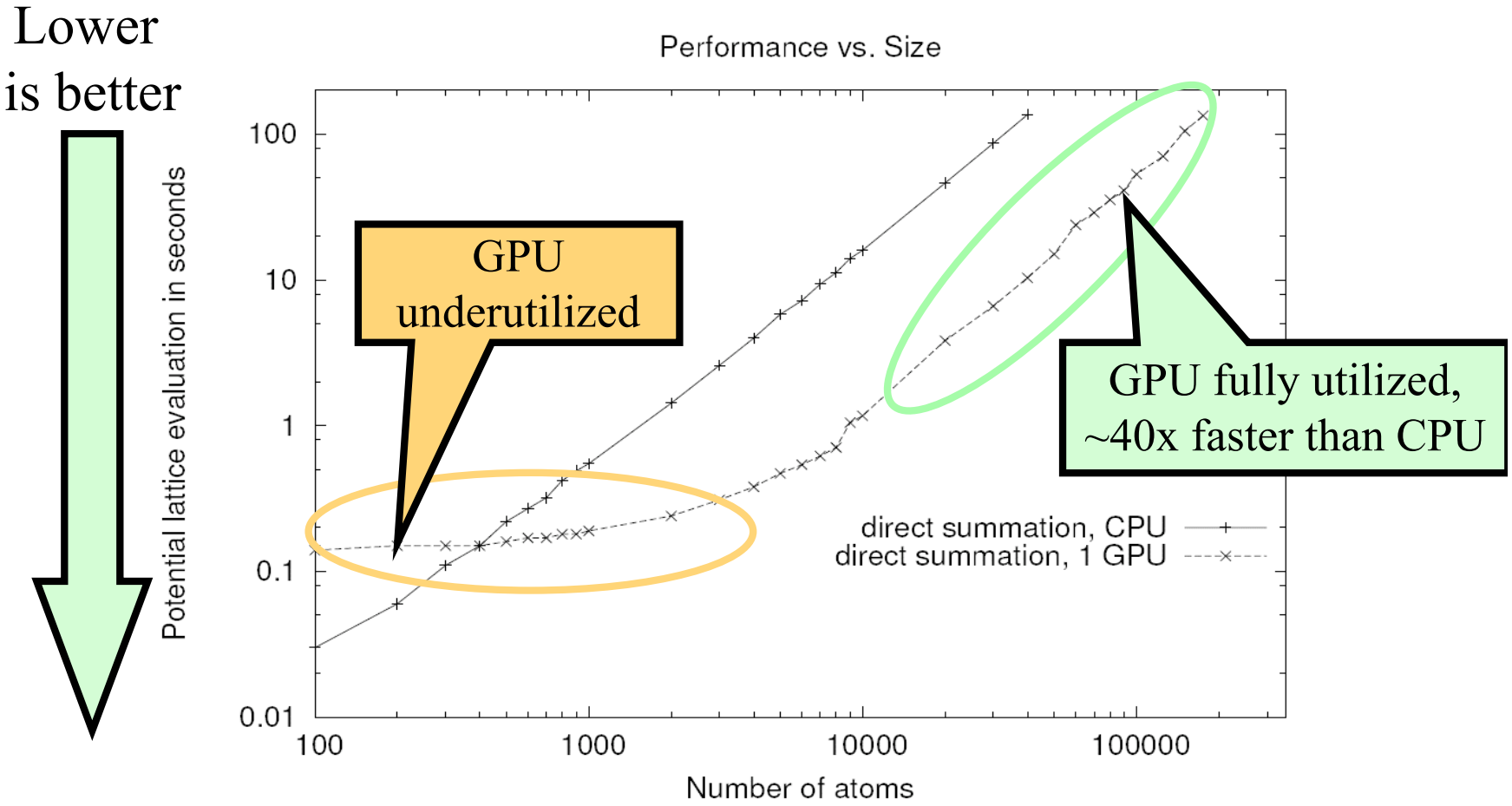
(unrolled, coalesced)



DCS Version 4: Kernel Structure
291.5 GFLOPS, 39.5 Billion Atom Evals/Sec

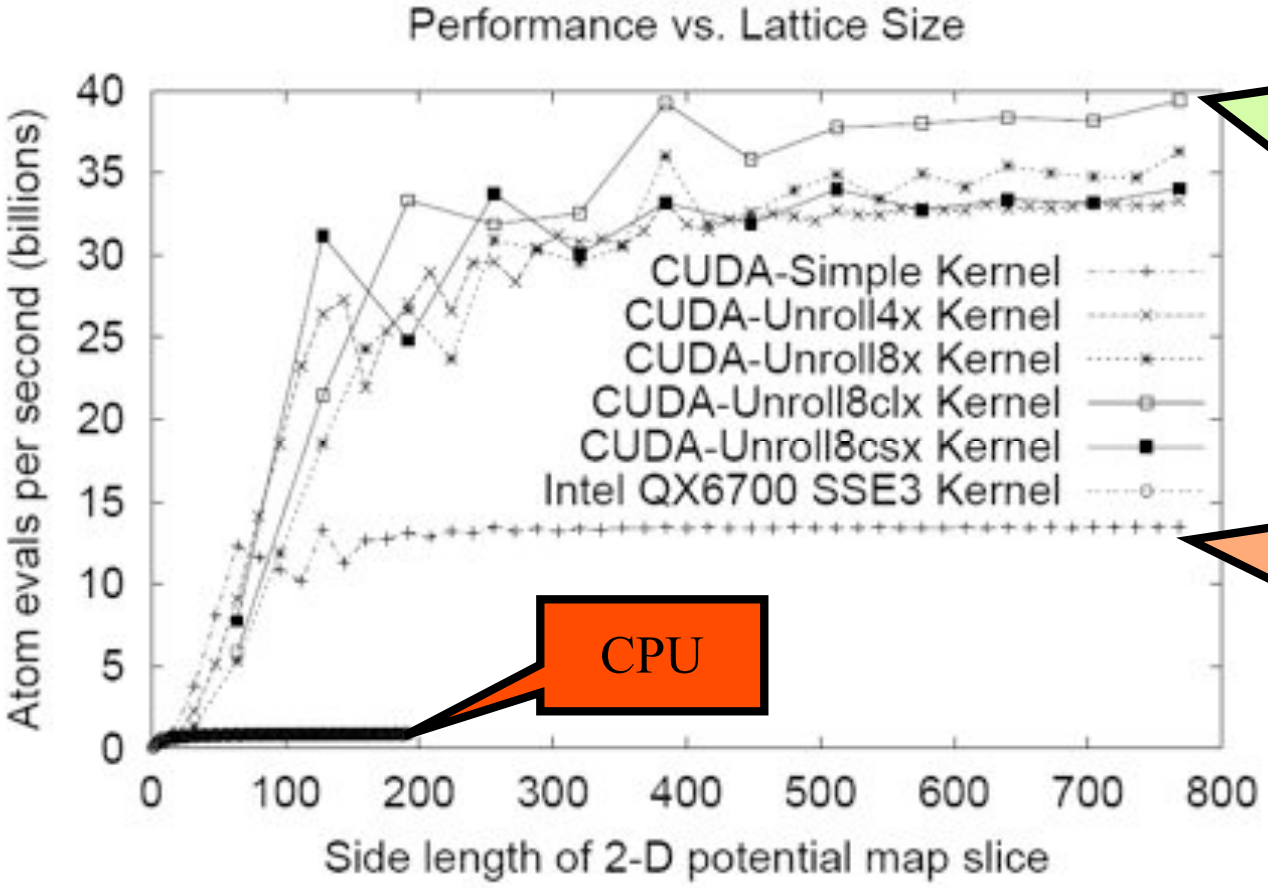
- Processes 8 lattice points at a time in the inner loop
- Subsequent lattice points computed by each thread are offset to guarantee coalesced memory accesses
- Loads and increments 8 potential map lattice points from global memory at completion of of the summation, avoiding register consumption
- Code is too long to show, but is available by request

Direct Coulomb Summation Runtime



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.

Direct Coulomb Summation Performance



CUDA-Unroll8clx: fastest GPU kernel, 44x faster than CPU, 291 GFLOPS on GeForce 8800GTX

CUDA-Simple: 14.8x faster, 33% of fastest GPU kernel

GPU computing. J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips. *Proceedings of the IEEE*, 96:879-899, 2008.

Multi-GPU DCS Algorithm:

- One host thread is created for each CUDA GPU, attached according to host thread ID:
  - First CUDA call binds that thread's CUDA context to that GPU for life
- Map slices are decomposed cyclically onto the available GPUs
- Map slices are usually larger than the host memory page size, so false sharing and related effects are easily avoided for this application

Multi-GPU Direct Coulomb Summation

- Effective memory bandwidth scales with the number of GPUs utilized
- PCIe bus bandwidth not a bottleneck for this algorithm
- 117 billion evals/sec
- 863 GFLOPS
- 131x speedup vs. CPU core
- Power: 700 watts during benchmark

Quad-core Intel QX6700

Three NVIDIA GeForce 8800GTX

Multi-GPU Direct Coulomb Summation

- **4-GPU (2 Quadroplex) Opteron node at NCSA**
- **157 billion evals/sec**
- **1.16 TFLOPS**
- **176x speedup vs. Intel QX6700 CPU core w/ SSE**

- 4-GPU (GT200)
- 241 billion evals/sec
- 1.78 TFLOPS
- 271x speedup vs. Intel QX6700 CPU core w/ SSE

NCSA GPU Cluster

http://www.ncsa.uiuc.edu/Projects/GPUcluster/

Infinite vs. Cutoff Potentials

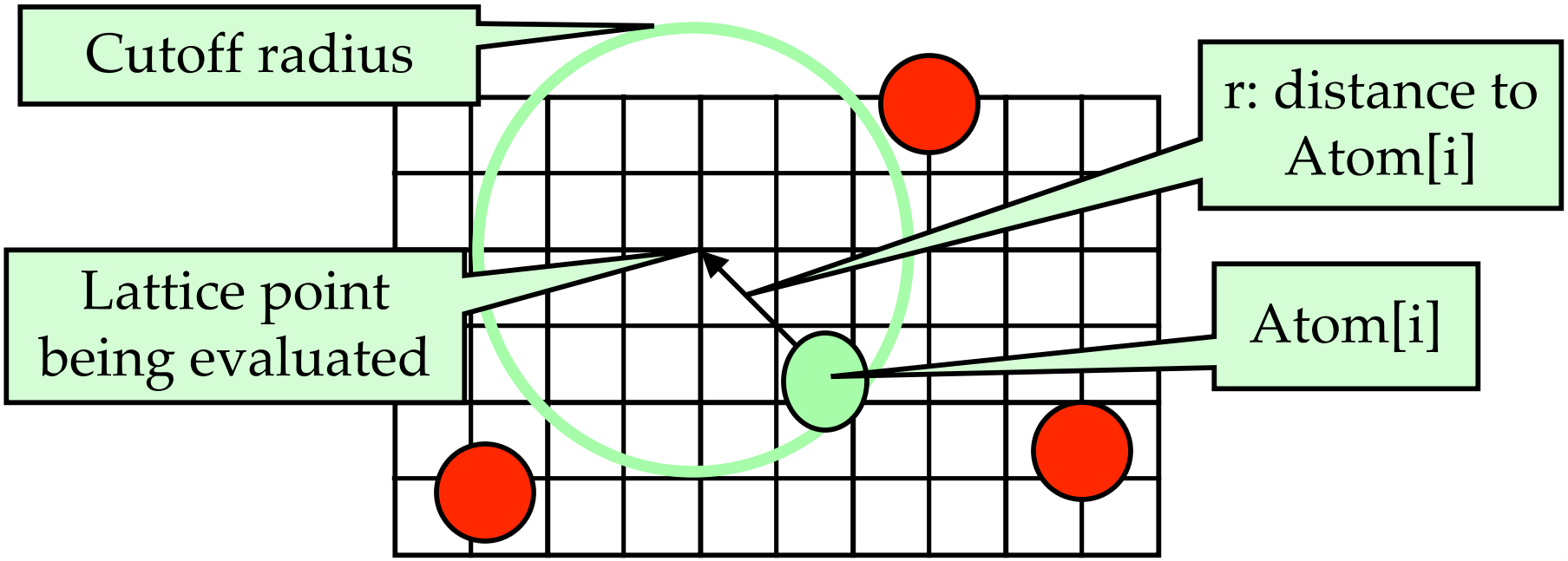
- Infinite range potential:
  - All atoms contribute to all lattice points
  - Summation algorithm has quadratic complexity
- Cutoff (range-limited) potential:
  - Atoms contribute within cutoff distance to lattice points
  - Summation algorithm has linear time complexity
  - Has many applications in molecular modeling:
    - Replace electrostatic potential with shifted form
    - Short-range part for fast methods of approximating full electrostatics
    - Used for fast decaying interactions (e.g. Lennard-Jones, Buckingham)

Cutoff Summation

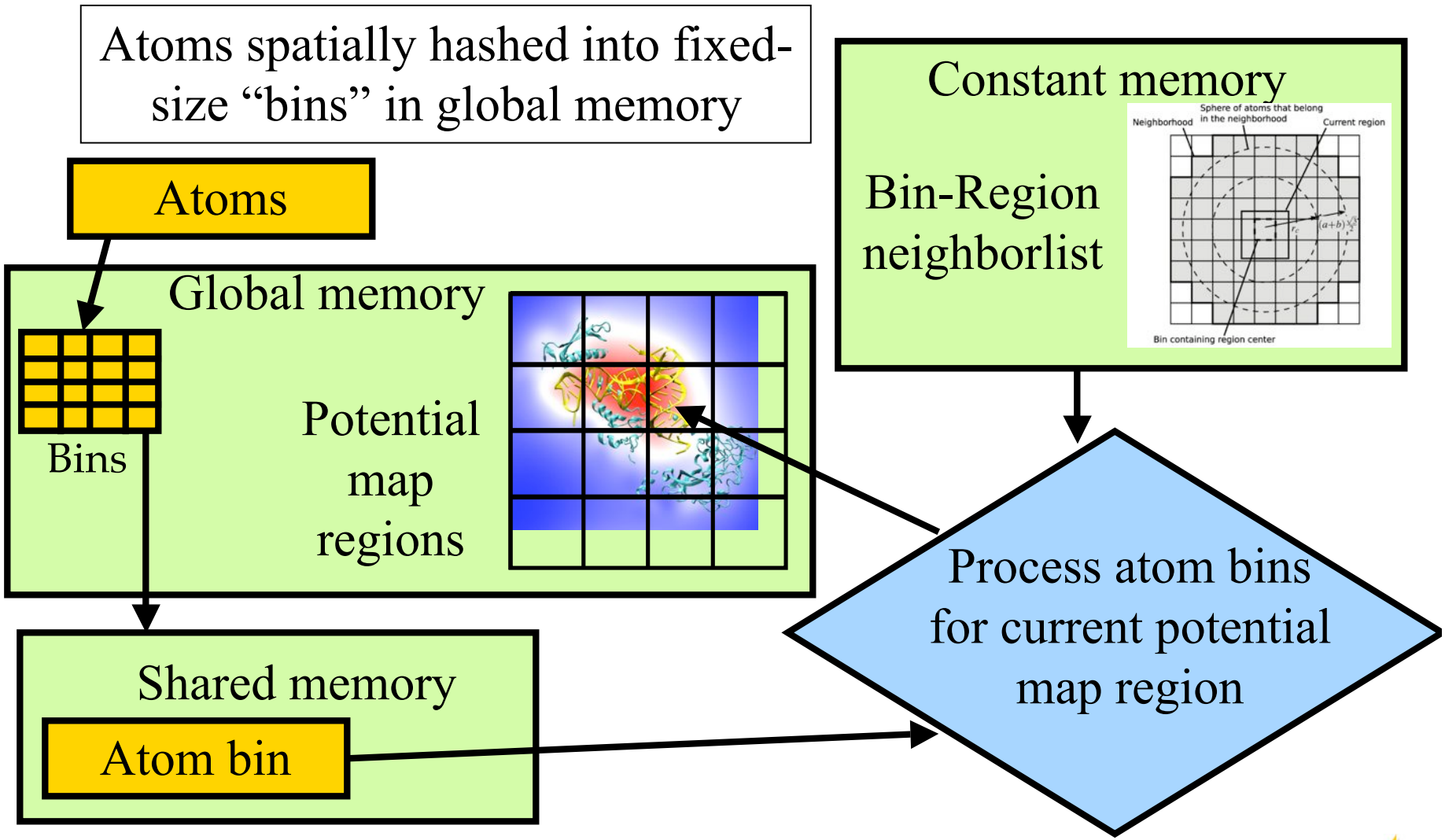
- At each lattice point, sum potential contributions for atoms within cutoff radius:

```
if (distance to atom[i] < cutoff)
  potential += (charge[i] / r) * s(r)
```

- Smoothing function s(r) is algorithm dependent



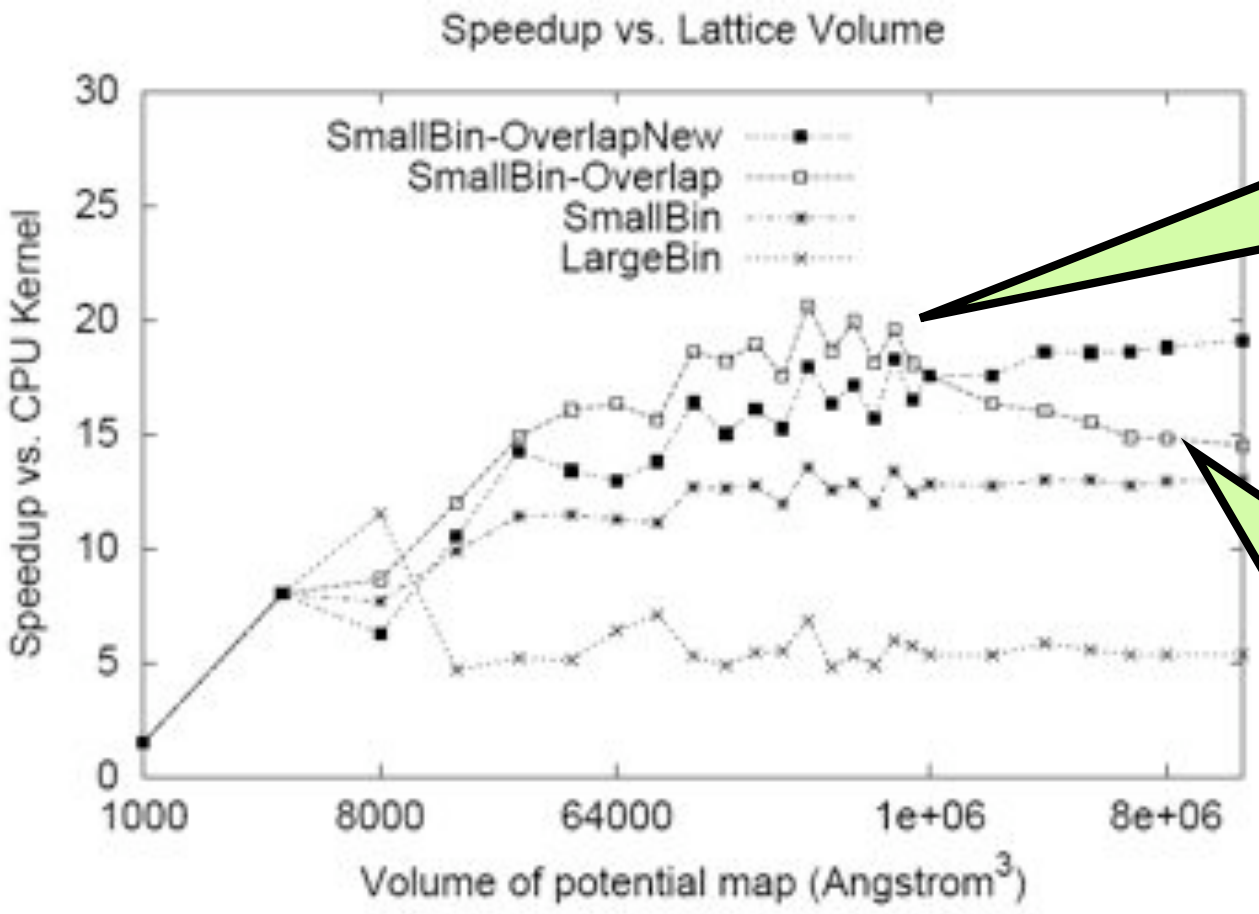
Cutoff Summation on the GPU



Using the CPU to Improve GPU Performance

- GPU performs best when the work evenly divides into the number of threads/processing units
- Optimization strategy:
  - Use the CPU to “regularize” the GPU workload
  - Use fixed size bin data structures, with “empty” slots skipped or producing zeroed out results
  - Handle exceptional or irregular work units on the CPU while the GPU processes the bulk of the work
  - On average, the GPU is kept highly occupied, attaining a much higher fraction of peak performance

Cutoff Summation Runtime



GPU cutoff with CPU overlap: 17x-21x faster than CPU core

If asynchronous stream blocks due to queue filling, performance will degrade from peak…

GPU acceleration of cutoff pair potentials for molecular modeling applications. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.

Cutoff Summation Observations

- Use of CPU to handle overflowed bins is very effective, overlaps completely with GPU work
- One caveat when using streaming API is to avoid overfilling the stream queue with work, as doing so can trigger blocking behavior (greatly improved in current drivers)
- The use of compensated summation (all GPUs) or double-precision (GT200 only) for potential accumulation resulted in only a ~10% performance penalty vs. pure single-precision arithmetic, while reducing the effects of floating point truncation

GPU Kernel Performance, May 2008

GeForce 8800GTX w/ CUDA 1.1, Driver 169.09

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

Calculation / Algorithm	Algorithm class	Speedup vs. Intel QX6700 CPU core
Fluorescence microphotolysis	Iterative matrix / stencil	12x
Pairlist calculation	Particle pair distance test	10-11x
Pairlist update	Particle pair distance test	5-15x
Molecular dynamics non-bonded force calc.	N-body cutoff force calculations	10x (w/ pairlist) 20x
Cutoff electron density sum	Particle-grid w/ cutoff	15-23x
MSM short-range	Particle-grid w/ cutoff	24x
MSM long-range	Grid-grid w/ cutoff	22x
Direct Coulomb summation	Particle-grid	44x

Lessons Learned

- GPU algorithms need fine-grained parallelism and sufficient work to fully utilize the hardware
- Much of per-thread GPU algorithm optimization revolves around efficient use of multiple memory systems and latency hiding
- Concurrency can often be traded for per-thread performance, in combination with increased use of registers or shared memory
- Fine-grained GPU work decompositions often compose well with the comparatively coarse-grained decompositions used for multicore or distributed memory programing

Lessons Learned (2)

- The host CPU can potentially be used to "regularize" the computation for the GPU, yielding better overall performance
- Overlapping CPU work with GPU can hide some communication and unaccelerated computation
- Targeted use of double-precision floating point arithmetic, or compensated summation can reduce the effects of floating point truncation at low cost to performance

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, (in press)*
- GPU acceleration of cutoff pair potentials for molecular modeling applications. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, 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.