Accelerating Scientific Applications with GPUs

John Stone

Theoretical and Computational Biophysics Group

University of Illinois at Urbana-Champaign

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

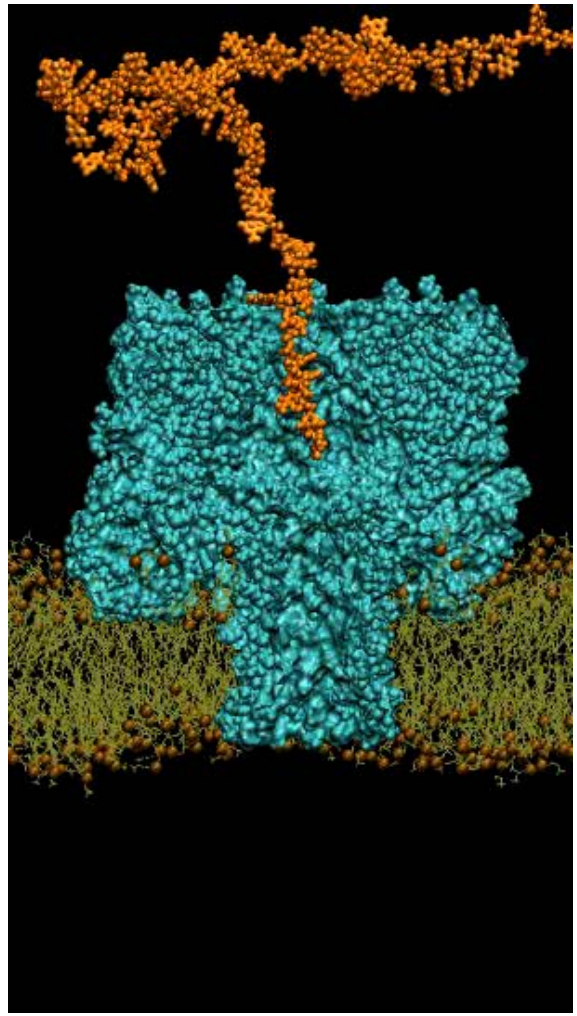
Workshop on Programming Massively Parallel Processors, July 10, 2008

GPU Computing

- Commodity devices, omnipresent in modern computers
- Massively parallel hardware, hundreds of processing units, throughput oriented design
- Support for standard integer and floating point data types, most recently double-precision floating point
- Programming tools allow software to be written in dialects of familiar C/C++ and integrated into legacy software
- 8x to 30x speedups common for data-parallel algorithms implemented as GPU kernels
- GPU algorithms are often multicore-friendly due to attention paid to data locality and work decomposition. Tools like “MCUDA” have already demonstrated the feasibility of retargeting GPU kernels to multicores with appropriate compiler/runtime toolkits.

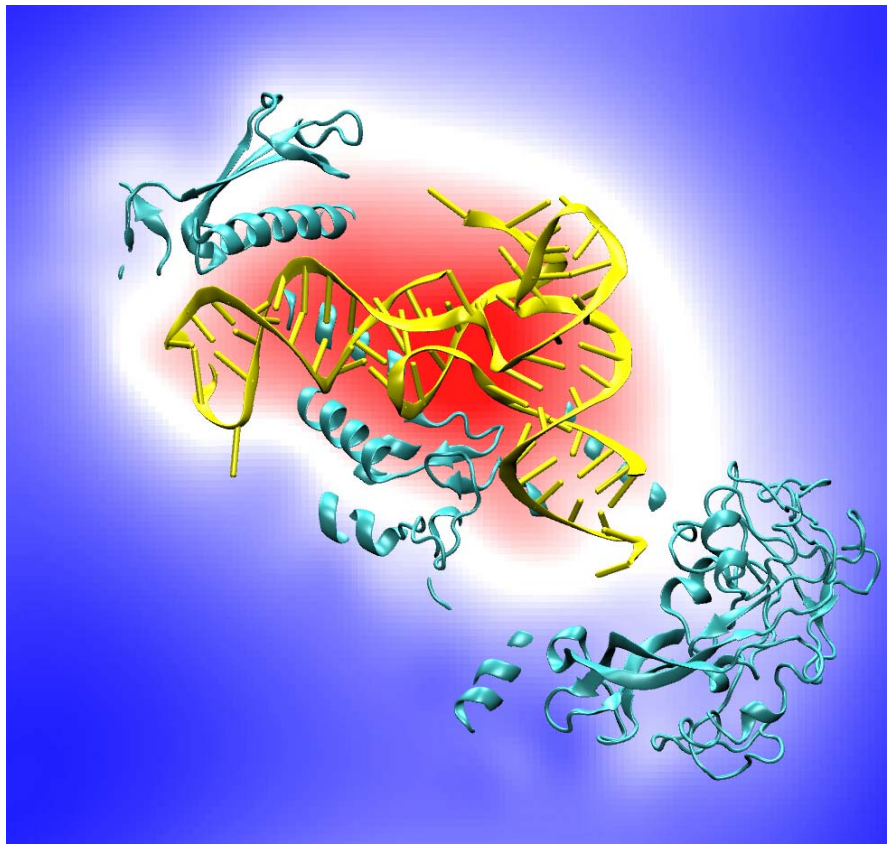
Computational Biology's Insatiable Demand for Processing Power

- Simulations still fall short of biological timescales
- Large simulations extremely difficult to prepare, analyze
- Order of magnitude increase in performance would allow use of more sophisticated models



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



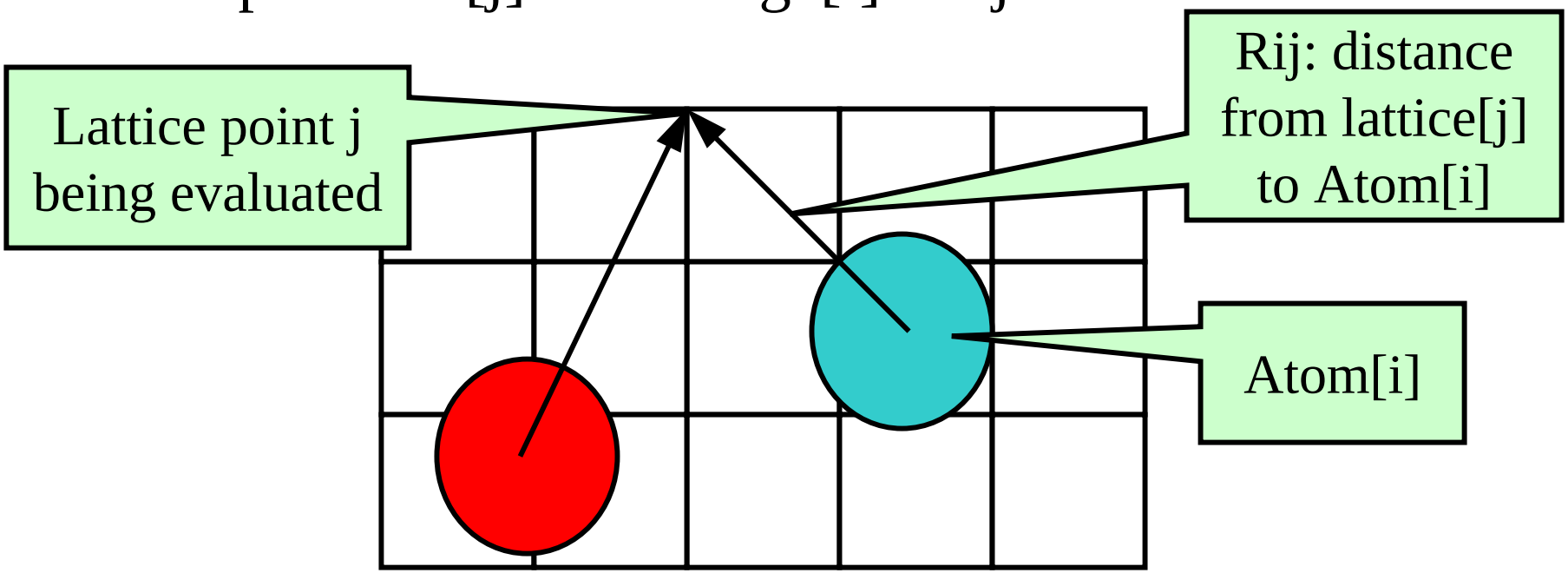
Positive potential field

Negative potential field

Direct Coulomb Summation

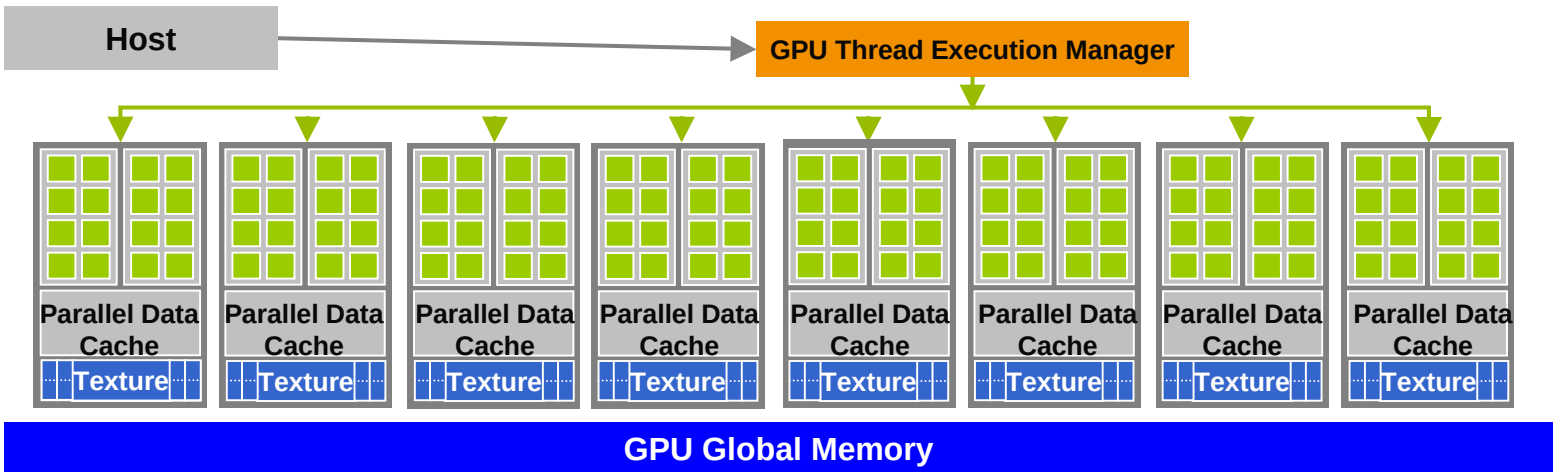
- At each lattice point, sum potential contributions for all atoms in the simulated structure:

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

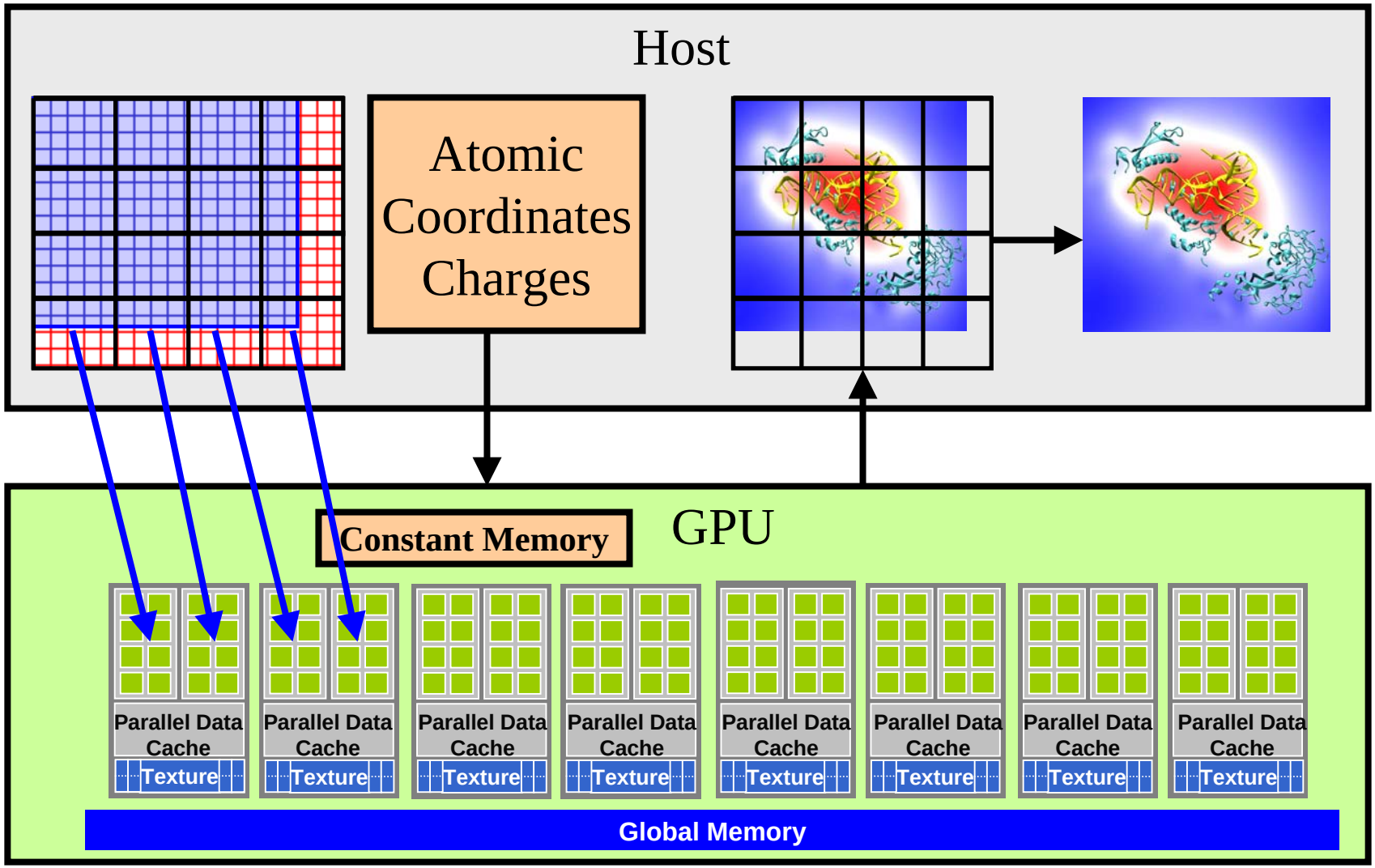


Direct Coulomb Summation on the GPU

- GPU can outrun 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
- Starting point for more sophisticated algorithms

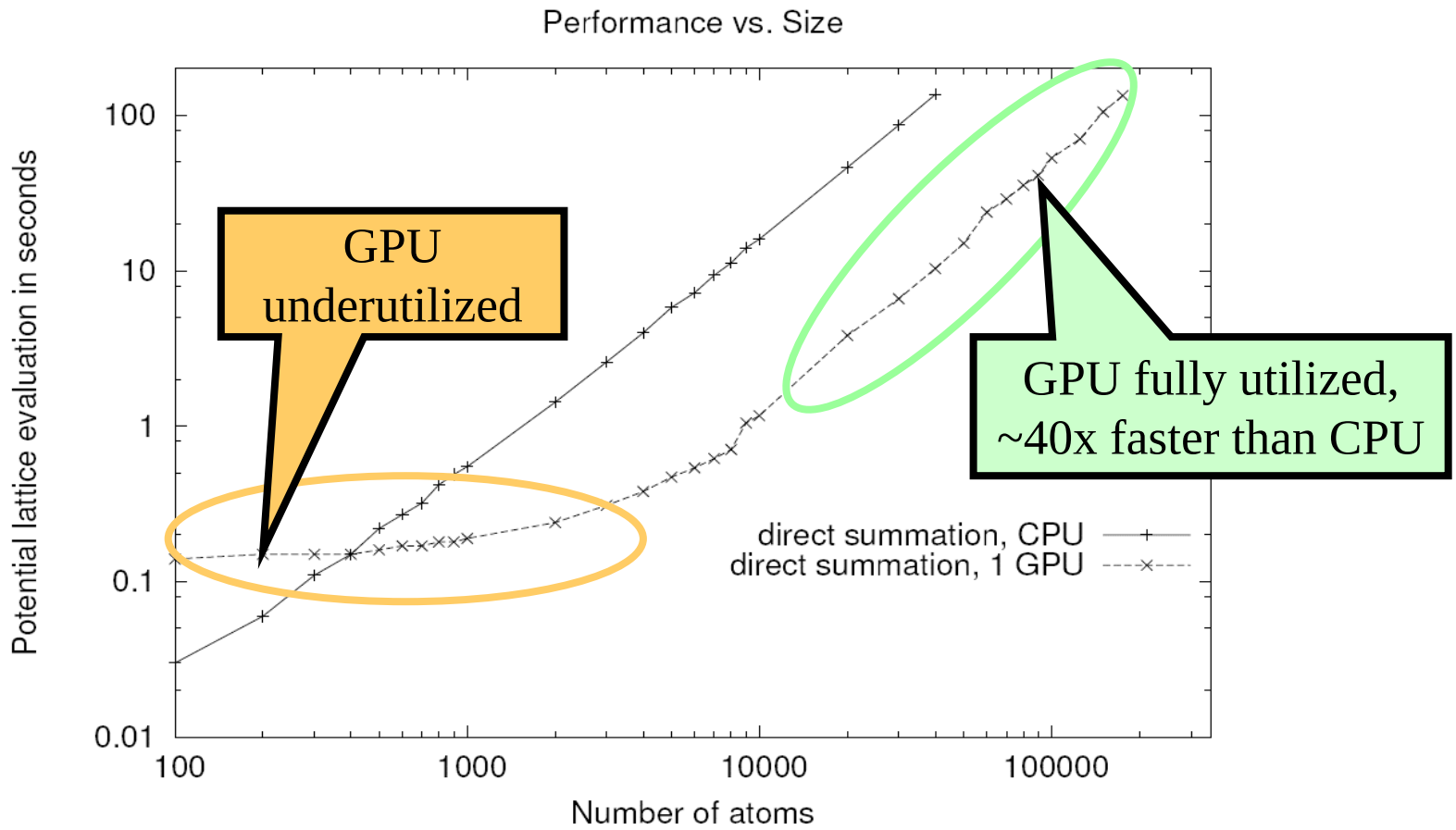


Direct Coulomb Summation on the GPU



Direct Coulomb Summation Runtime

Lower is better



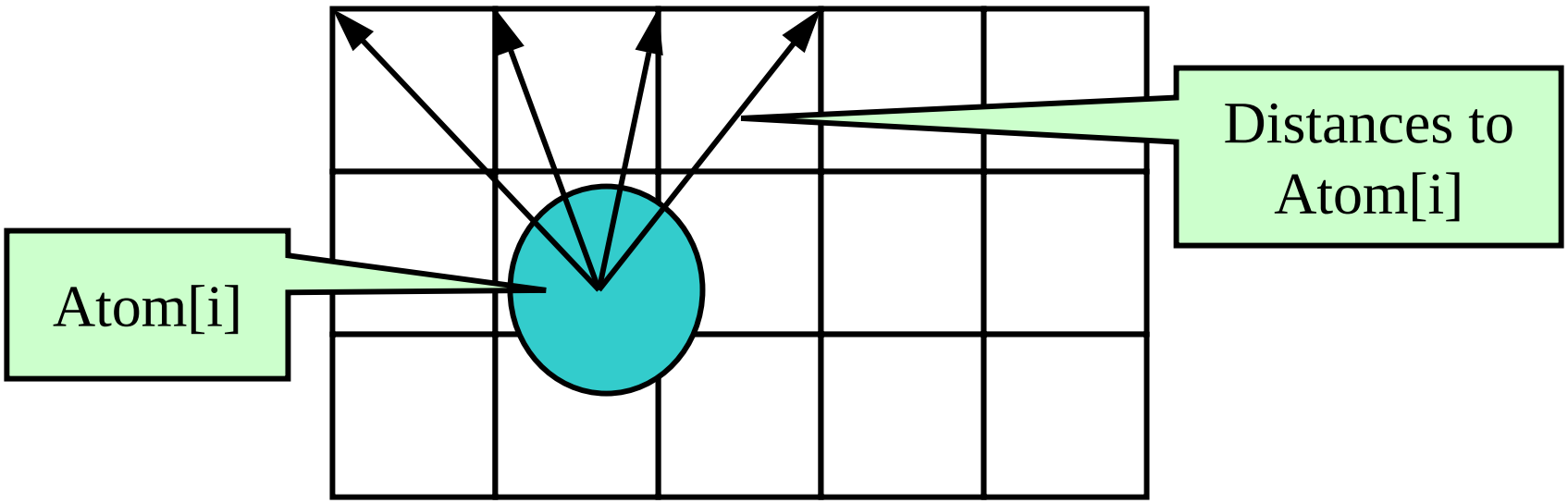
Accelerating molecular modeling applications with graphics processors.
J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, K. Schulten.
J. Comp. Chem., 28:2618-2640, 2007.

Optimizing for the GPU

- Increase arithmetic intensity, reuse in-register data by "unrolling" lattice point computation into inner atom loop
- Each atom contributes to several lattice points, distances only differ in the X component:

  potentialA += charge[i] / (distanceA to atom[i])

  potentialB += charge[i] / (distanceB to atom[i]) …

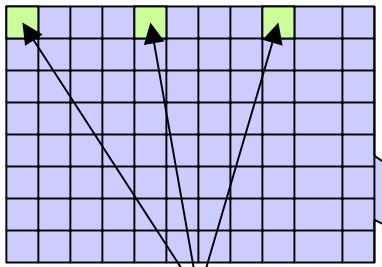


CUDA Block/Grid Decomposition

Unrolling increases computational tile size

Grid of thread blocks:

Thread blocks: 64-256 threads



Threads compute up to 8 potentials. Skipping by half-warps optimizes global mem. perf.

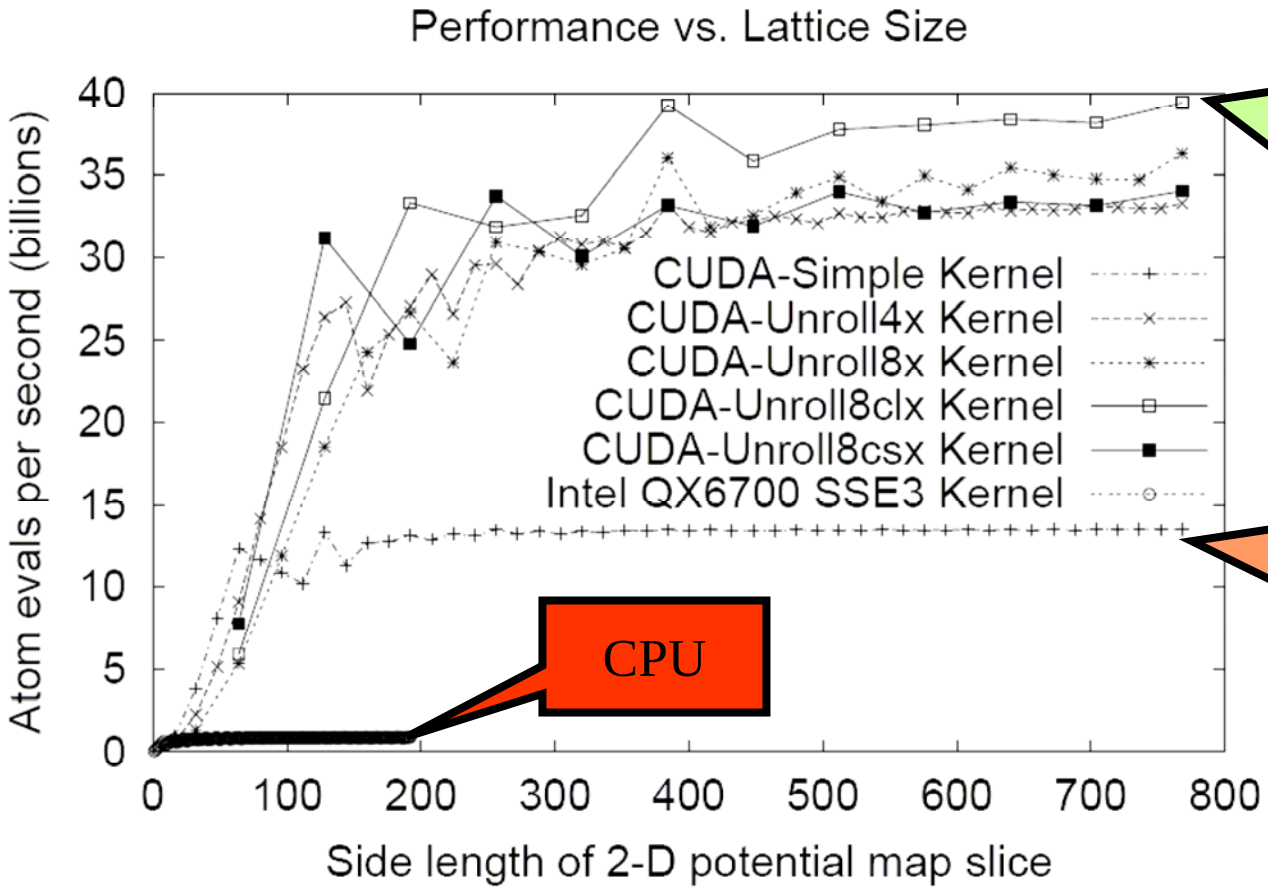
0,0 0,1 …

1,0 1,1 …

… … …

Padding waste

Direct Coulomb Summation Performance



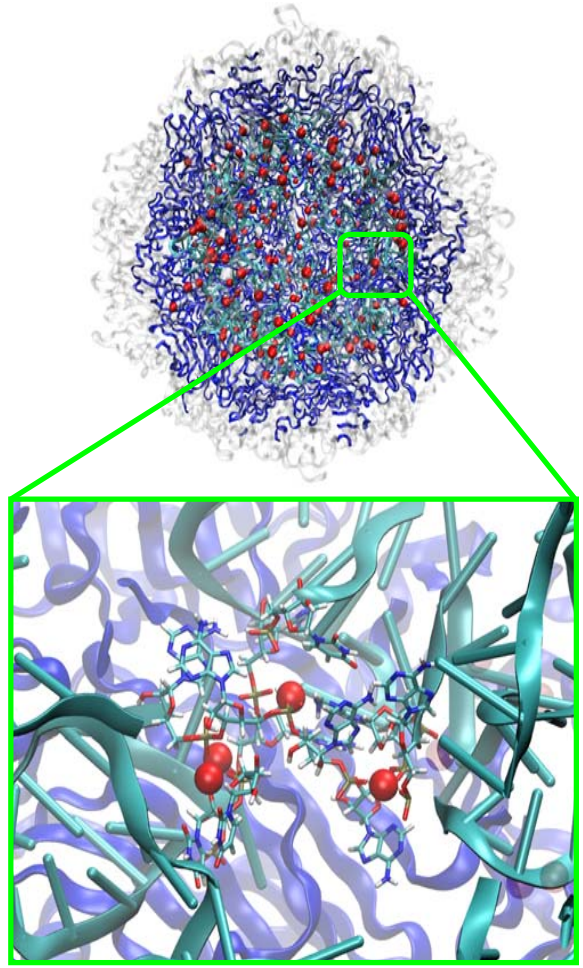
CUDA-Unroll8clx: fastest GPU kernel, 44x faster than CPU, 291 GFLOPS on GeForce 8800GTX

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*, 2008. In press.

GPU Application Performance
(July 2007, current code is 20% faster...)

- CUDA ion placement lattice calculation performance:
  - 82 times faster for virus (STMV) structure
  - 110 times faster for ribosome
- Virus ion placement: 110 CPU-hours on SGI Altix Itanium2
- Same calculation now takes 1.35 GPU-hours
- 27 minutes (wall clock) if three GPUs are used concurrently



Satellite Tobacco Mosaic Virus (STMV) Ion Placement

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

NCSA GPU Cluster

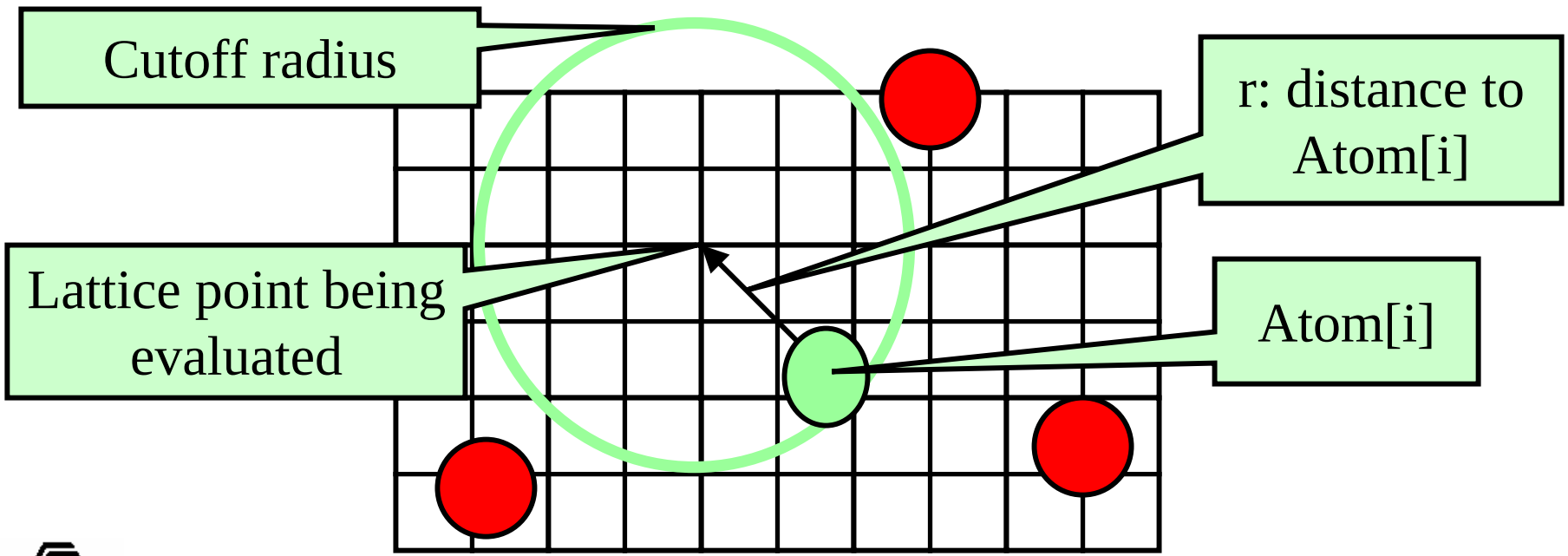
Cutoff Summation

- 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



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)

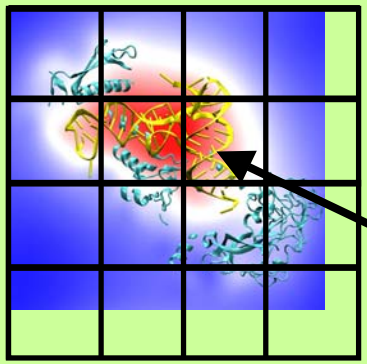
Cutoff Summation on the GPU

Atoms spatially hashed into fixed-size “bins” in global memory

Atoms

Global memory

Potential map regions

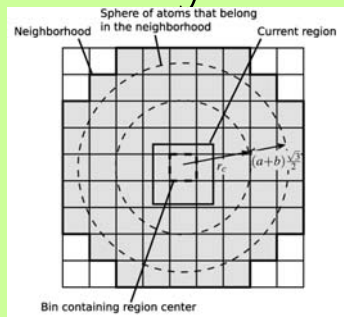


Shared memory

Atom bin

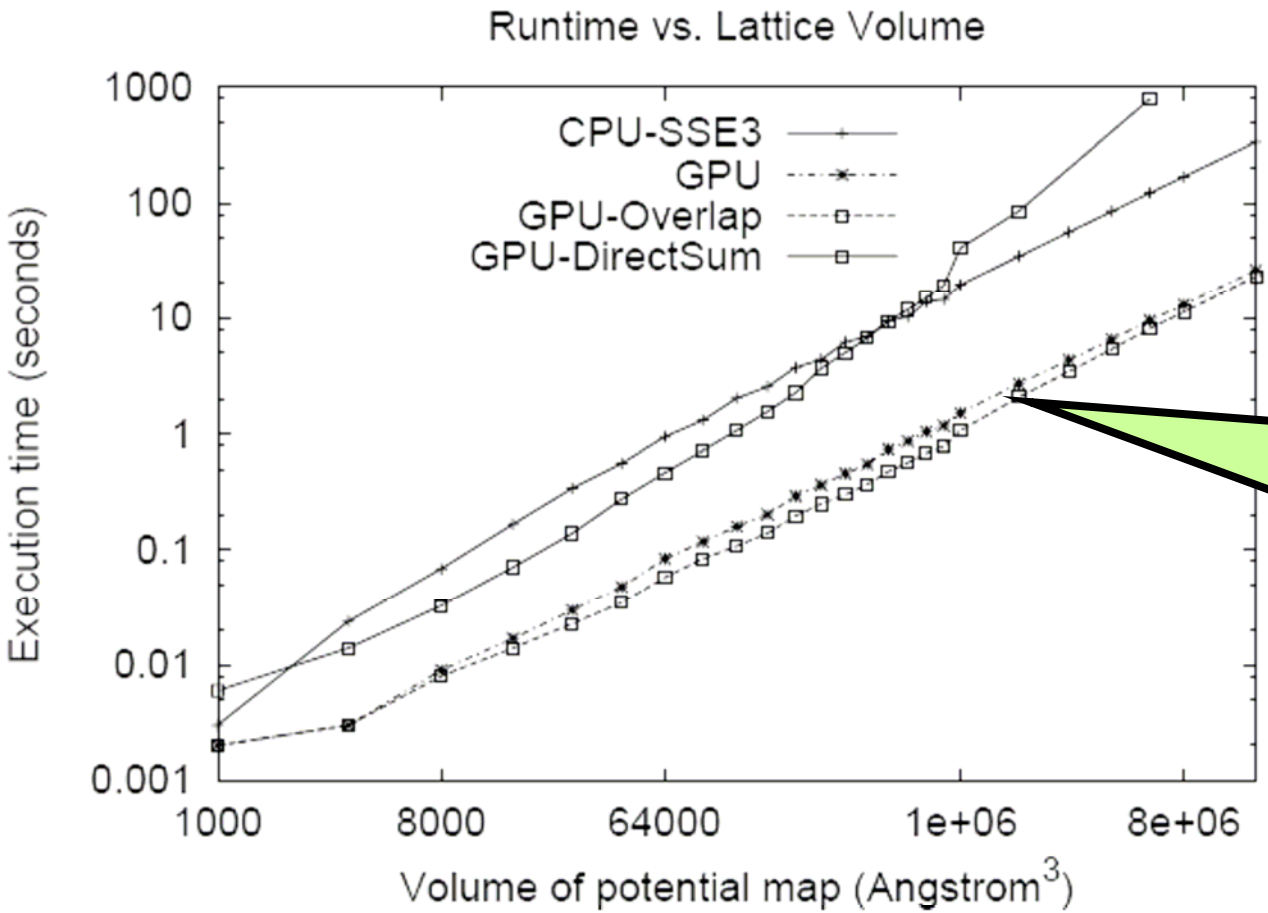
Constant memory

Bin-Region neighborlist



Process atom bins for current potential map region

Cutoff Summation Runtime



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

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*, 2008. In press.

NAMD Parallel Molecular Dynamics

Kale *et al.*, *J. Comp. Phys.* **151**:283-312, 1999.

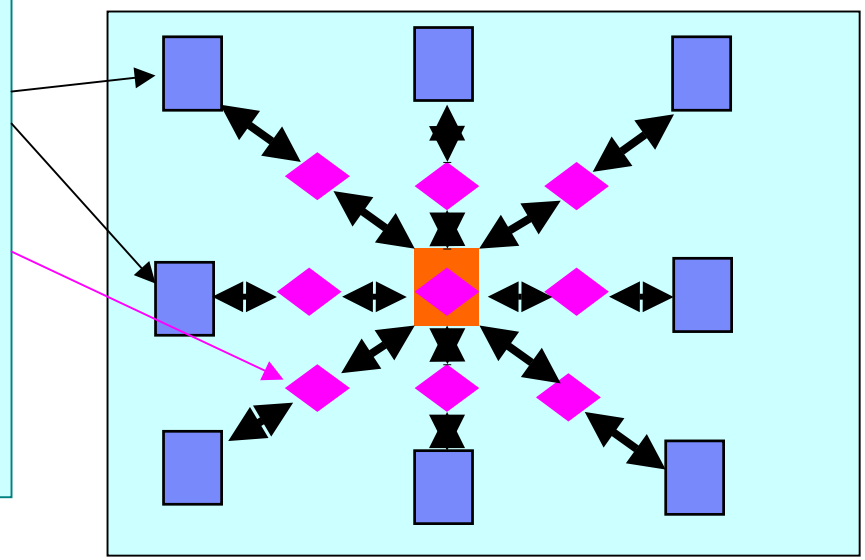
- Designed from the beginning as a parallel program
- Uses the Charm++ idea:
  - Decompose the computation into a large number of objects
  - Have an Intelligent Run-time system (of Charm++) assign objects to processors for dynamic load balancing with minimal communication

Hybrid of spatial and force decomposition:

•Spatial decomposition of atoms into cubes (called patches)

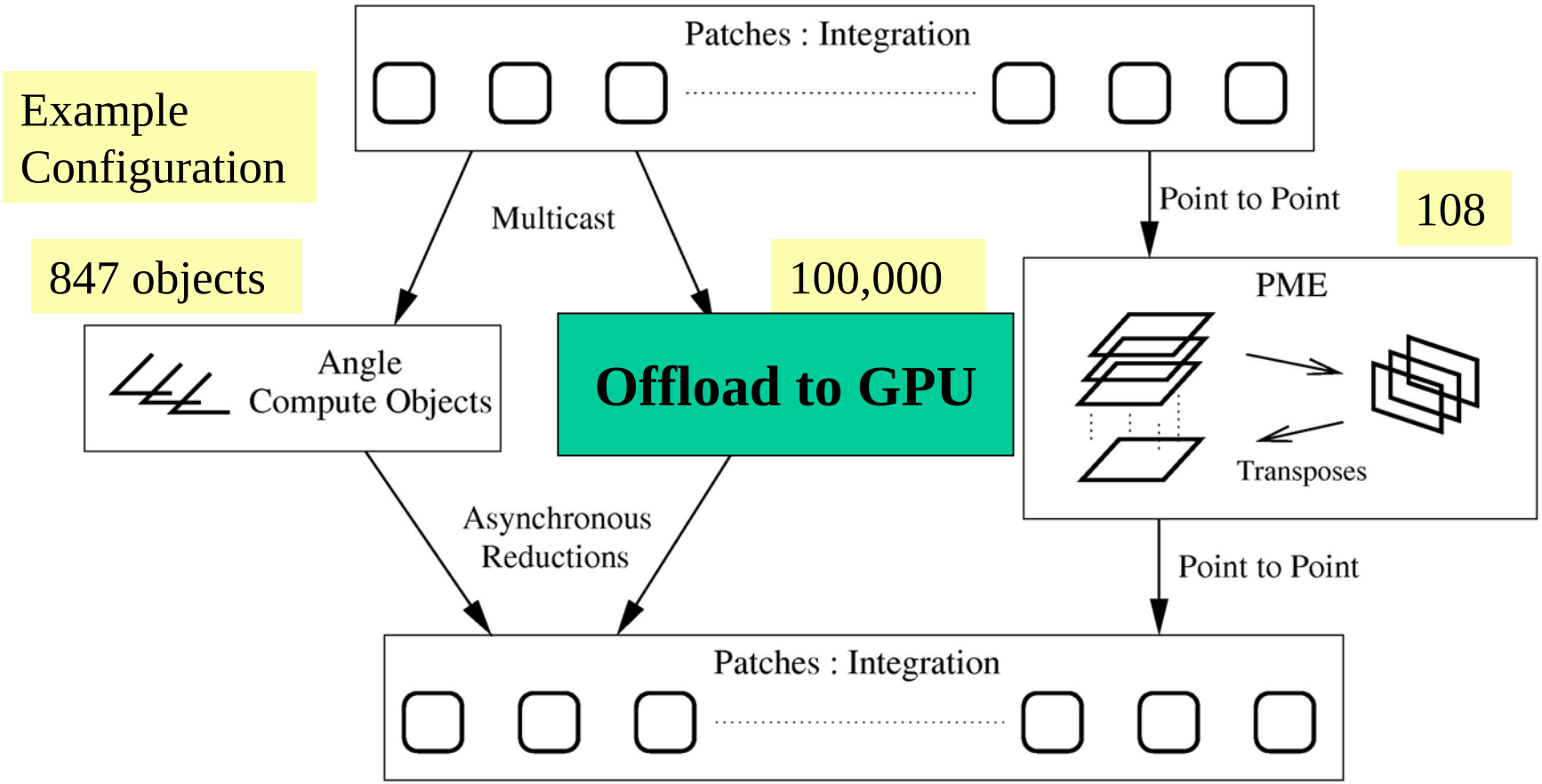
•For every pair of interacting patches, create one object for calculating electrostatic interactions

•Recent: Blue Matter, Desmond, etc. use this idea in some form



NAMD Overlapping Execution

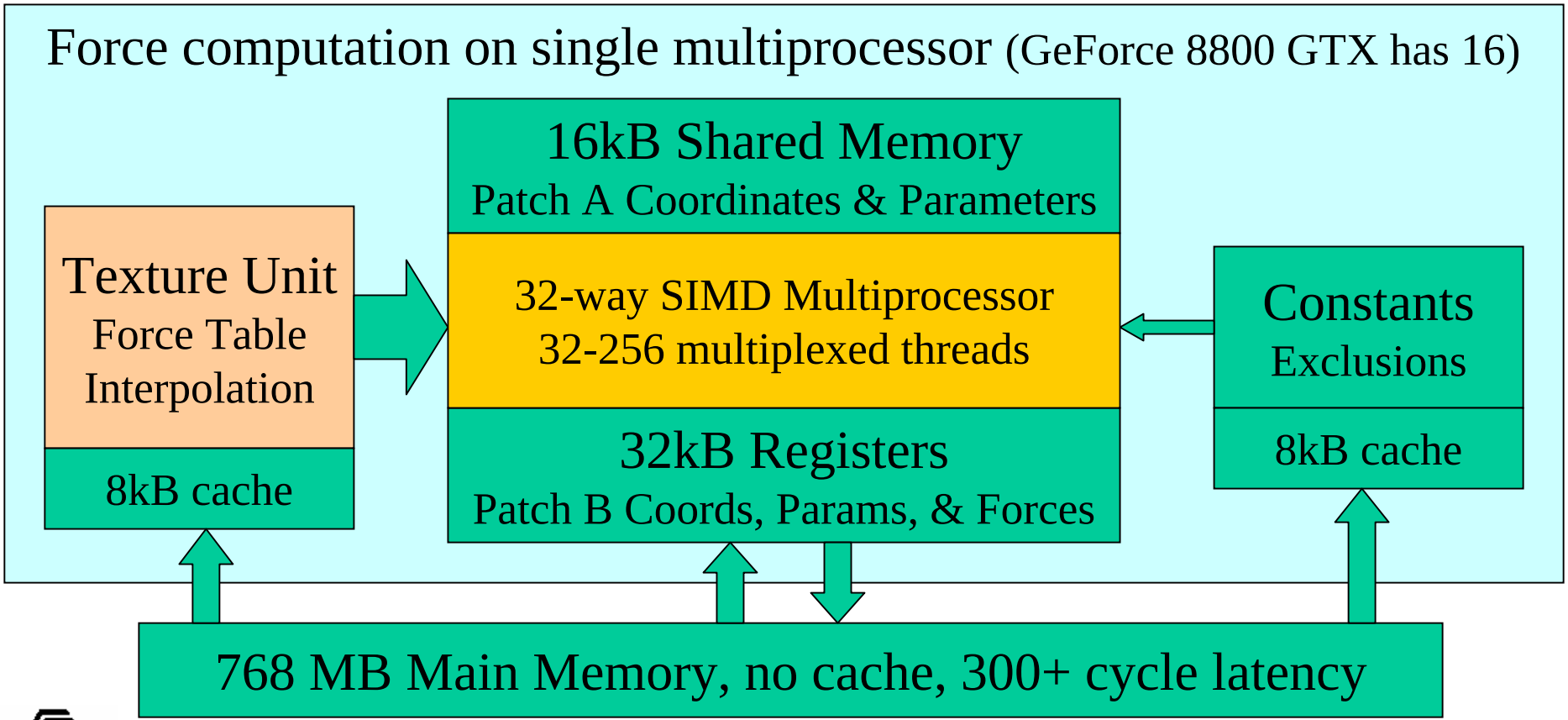
Phillips *et al.*, *SC2002*.



Objects are assigned to processors and queued as data arrives.

Nonbonded Forces on G80 GPU

- Start with most expensive calculation: direct nonbonded interactions.
- Decompose work into pairs of patches, identical to NAMD structure.
- GPU hardware assigns patch-pairs to multiprocessors dynamically.



Stone *et al.*, *J. Comp. Chem.* **28**:2618-2640, 2007.

Nonbonded Forces CUDA Code

```
texture<float4> force_table;
__constant__ unsigned int exclusions[];
__shared__ atom jatom[];
atom iatom;      // per-thread atom, stored in registers
float4 iforce;   // per-thread force, stored in registers
for ( int j = 0; j < jatom_count; ++j ) {
  float dx = jatom[j].x - iatom.x;   float dy = jatom[j].y - iatom.y;  float dz = jatom[j].z - iatom.z;
  float r2 = dx*dx + dy*dy + dz*dz;
  if ( r2 < cutoff2 ) {
    float4 ft = texfetch(force_table, 1.f/sqrt(r2));
    bool excluded = false;
    int indexdiff = iatom.index - jatom[j].index;
    if ( abs(indexdiff) <= (int) jatom[j].excl_maxdiff ) {
      indexdiff += jatom[j].excl_index;
      excluded = ((exclusions[indexdiff>>5] & (1<<(indexdiff&31))) != 0);
    }
    float f = iatom.half_sigma + jatom[j].half_sigma;  // sigma
    f *= f*f;  // sigma^3
    f *= f;  // sigma^6
    f *= ( f * ft.x + ft.y );  // sigma^12 * fi.x - sigma^6 * fi.y
    f *= iatom.sqrt_epsilon * jatom[j].sqrt_epsilon;
    float qq = iatom.charge * jatom[j].charge;
    if ( excluded ) { f = qq * ft.w; }  // PME correction
    else { f += qq * ft.z; }  // Coulomb
    iforce.x += dx * f;   iforce.y += dy * f;    iforce.z += dz * f;
    iforce.w += 1.f;  // interaction count or energy
  }
}
```

Force Interpolation

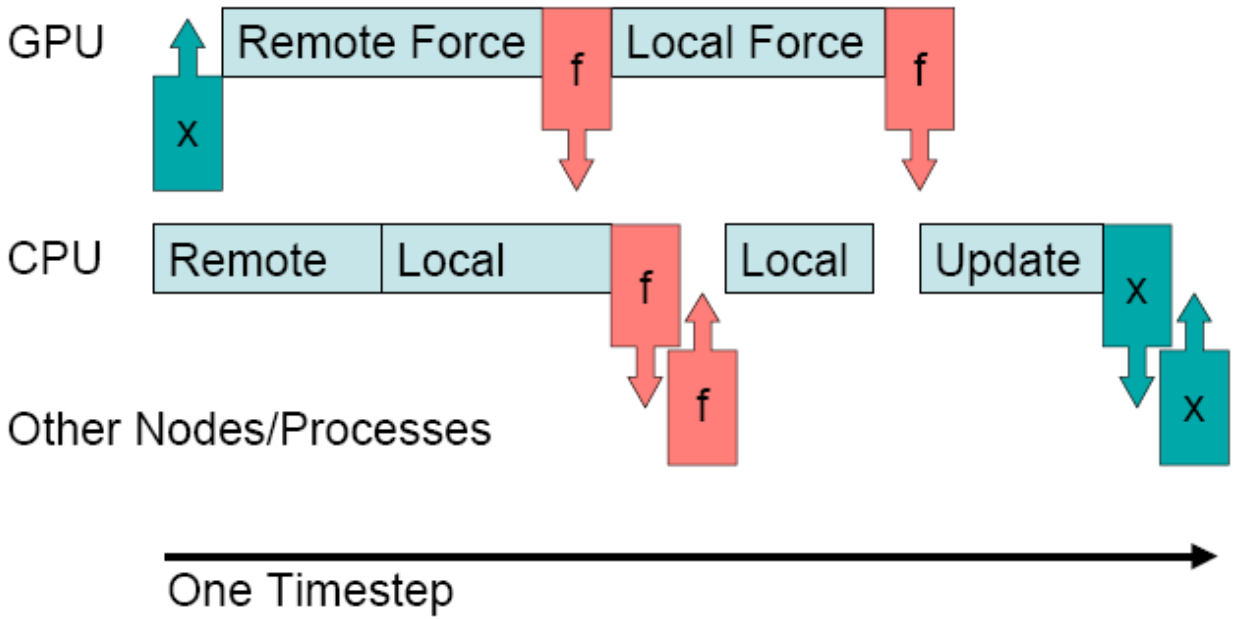
Exclusions

Parameters

Accumulation

Stone *et al., J. Comp. Chem.* **28**:2618-2640, 2007.

NAMD Overlapping Execution with Asynchronous CUDA kernels



GPU kernels are launched asynchronously, CPU continues with its own work, polling for GPU completion periodically. Forces needed by remote nodes are explicitly scheduled to be computed ASAP to improve overall performance.

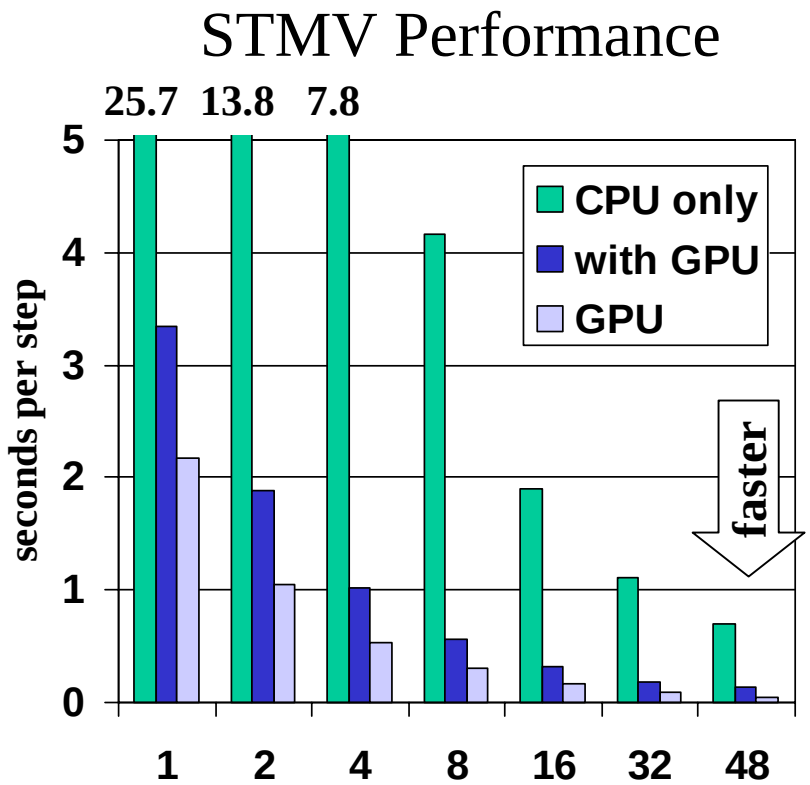
NAMD Performance on NCSA GPU Cluster, April 2008

CPU Cores & GPUs	4	8	16	32	60
GPU-accelerated performance					
Local blocks/GPU	13186	5798	2564	1174	577
Remote blocks/GPU	1644	1617	1144	680	411
GPU s/step	0.544	0.274	0.139	0.071	0.040
Total s/step	0.960	0.483	0.261	0.154	0.085
Unaccelerated performance					
Total s/step	6.76	3.33	1.737	0.980	0.471
Speedup from GPU acceleration					
Factor	7.0	6.9	6.7	6.4	5.5

STMV benchmark, 1M atoms,12A cutoff,
PME every 4 steps, running on
2.4 GHz AMD Opteron + NVIDIA Quadro FX 5600

NAMD Performance on NCSA GPU Cluster, April 2008

- 5.5-7x overall application speedup w/ G80-based GPUs
- STMV virus (1M atoms)
- Overlap with CPU
- Off-node results done first
- Infiniband scales well
- Plans for better performance
  - Tune or port remaining work
  - Balance GPU load (?)



2.4 GHz Opteron + Quadro FX 5600
Thanks to NCSA and NVIDIA

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 calculation	N-body cutoff force calculations	10x 20x (w/ pairlist)
Cutoff electron density sum	Particle-grid w/ cutoff	15-23x
MSM short-range cutoff	Particle-grid w/ cutoff	24x
MSM long-range lattice cutoff	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 hardware
- Fine-grained GPU work decompositions often compose well with the comparatively coarse-grained decompositions used for multicore or distributed memory programing
- Much of GPU algorithm optimization revolves around efficient use of multiple memory systems and latency hiding

Lessons Learned (2)

- The host CPU can potentially be used to “regularize” the computation for the GPU, yielding better overall performance
- Amdahl’s Law can prevent applications from achieving peak speedup with shallow GPU acceleration efforts
- Overlapping CPU work with GPU can hide some communication and unaccelerated computation
- Concurrent use of GPUs, Infiniband cards, or other hardware performing DMA to the same region of memory can be problematic due to the lack of OS-managed interfaces for “pinning” pages of physical memory, when accessed through high-performance OS kernel bypass software interfaces (e.g. MPI, CUDA, etc)

Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- Prof. Wen-mei Hwu, Chris Rodrigues, IMPACT Group, University of Illinois at Urbana-Champaign
- David Kirk and the CUDA team at NVIDIA
- NVIDIA, IACAT, NCSA (GPU cluster)
- NIH support: P41-RR05969

Publications

- http://www.ks.uiuc.edu/Research/gpu/
- 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.
- GPU computing. J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips. *Proceedings of the IEEE*, 2008. 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*, 2008. In press.