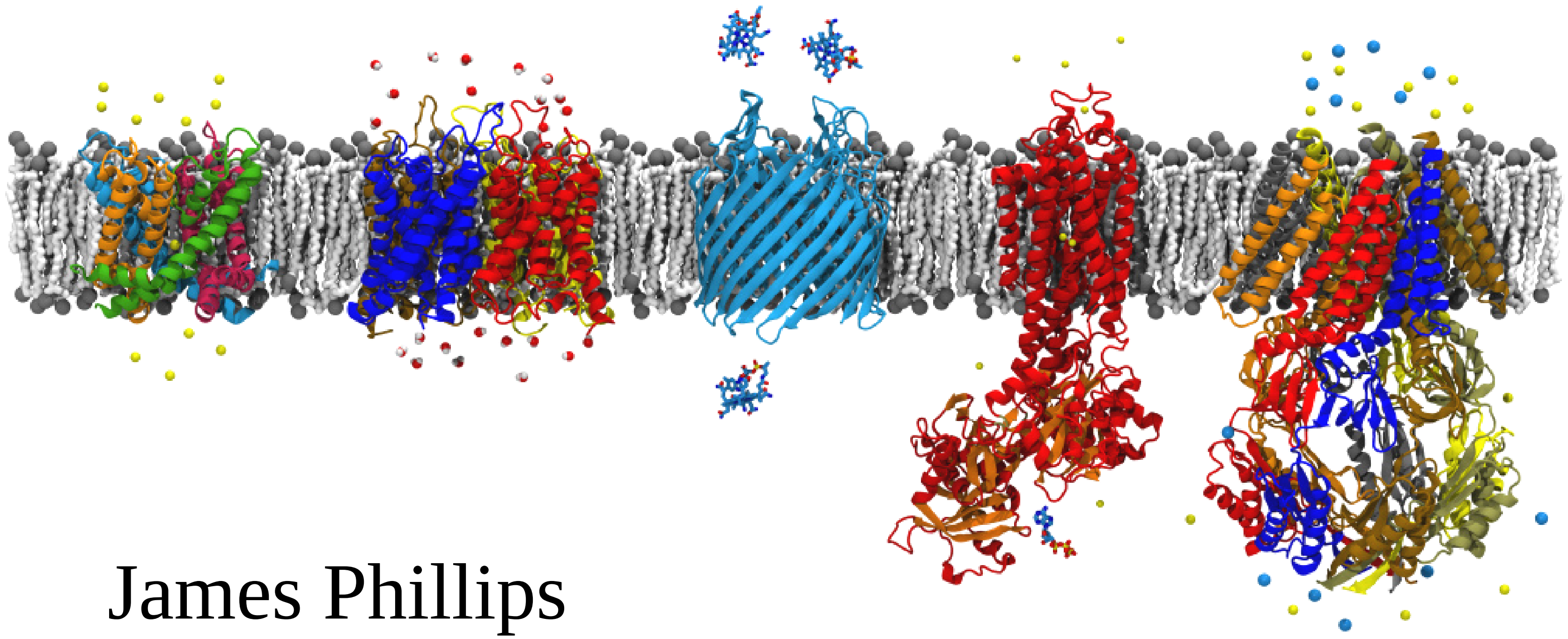
GPU Acceleration of Molecular Modeling Applications



James Phillips
John Stone

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

NAMD: Practical Supercomputing

- 25,000 users can't all be computer experts.
  - 18% are NIH-funded; many in other countries.
  - 4900 have downloaded more than one version.
- User experience is the same on all platforms.
  - No change in input, output, or configuration files.
  - Run any simulation on **any number of processors**.
  - Precompiled binaries available when possible.
- Desktops and laptops – setup and testing
  - x86 and x86-64 Windows, and Macintosh
  - Allow both shared-memory and network-based parallelism.
- Linux clusters – affordable workhorses
  - x86, x86-64, and Itanium processors
  - Gigabit ethernet, Myrinet, InfiniBand, Quadrics, Altix, etc

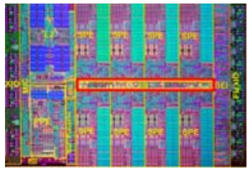
Phillips *et al.*, *J. Comp. Chem.* **26**:1781-1802, 2005.

Our Goal: Practical Acceleration

- Broadly applicable to scientific computing
  - Programmable by domain scientists
  - Scalable from small to large machines
- Broadly available to researchers
  - Price driven by commodity market
  - Low burden on system administration
- Sustainable performance advantage
  - Performance driven by Moore’s law
  - Stable market and supply chain

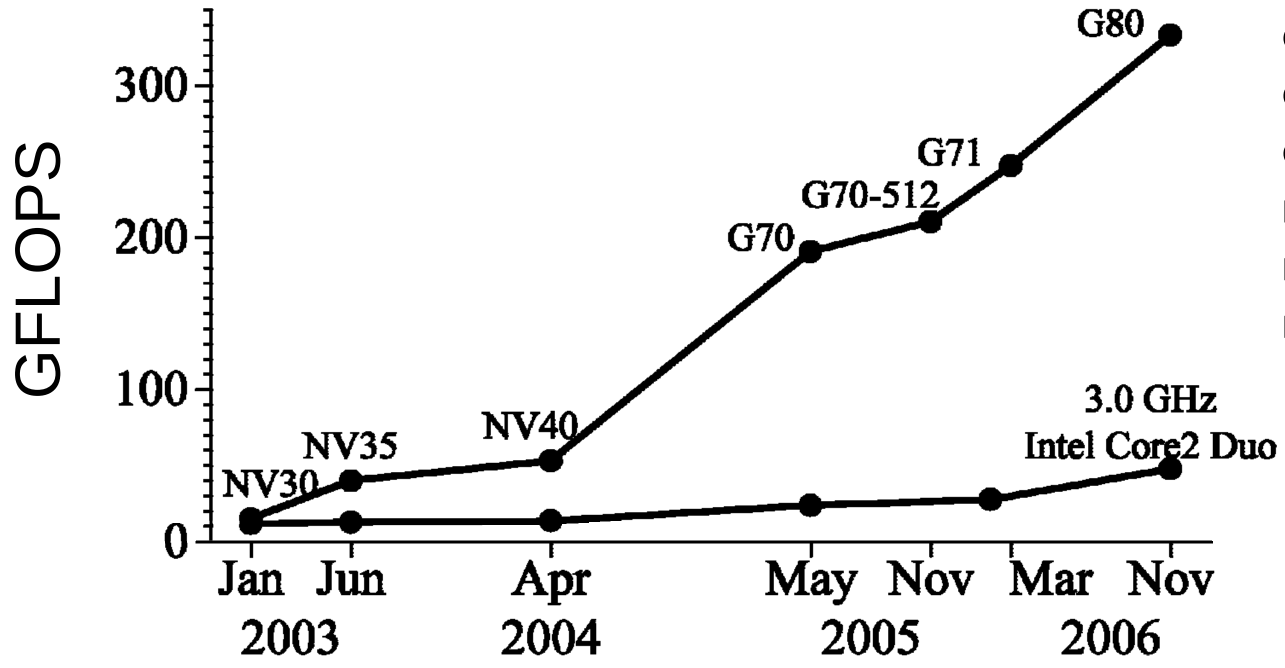
Acceleration Options for NAMD

- Outlook in 2005-2006:
  - FPGA reconfigurable computing (with NCSA)
    - Difficult to program, slow floating point, expensive
  - Cell processor (NCSA hardware)
    - Relatively easy to program, expensive
  - ClearSpeed (direct contact with company)
    - Limited memory and memory bandwidth, expensive
  - MDGRAPE
    - Inflexible and expensive
  - Graphics processor (GPU)
    - Program must be expressed as graphics operations



GPU vs CPU: Raw Performance

- Calculation: 450 GFLOPS vs 32 GFLOPS
- Memory Bandwidth: 80 GB/s vs 8.4 GB/s



G80 = GeForce 8800 GTX

G71 = GeForce 7900 GTX

G70 = GeForce 7800 GTX

NV40 = GeForce 6800 Ultra

NV35 = GeForce FX 5950 Ultra

NV30 = GeForce FX 5800

CUDA: Practical Performance

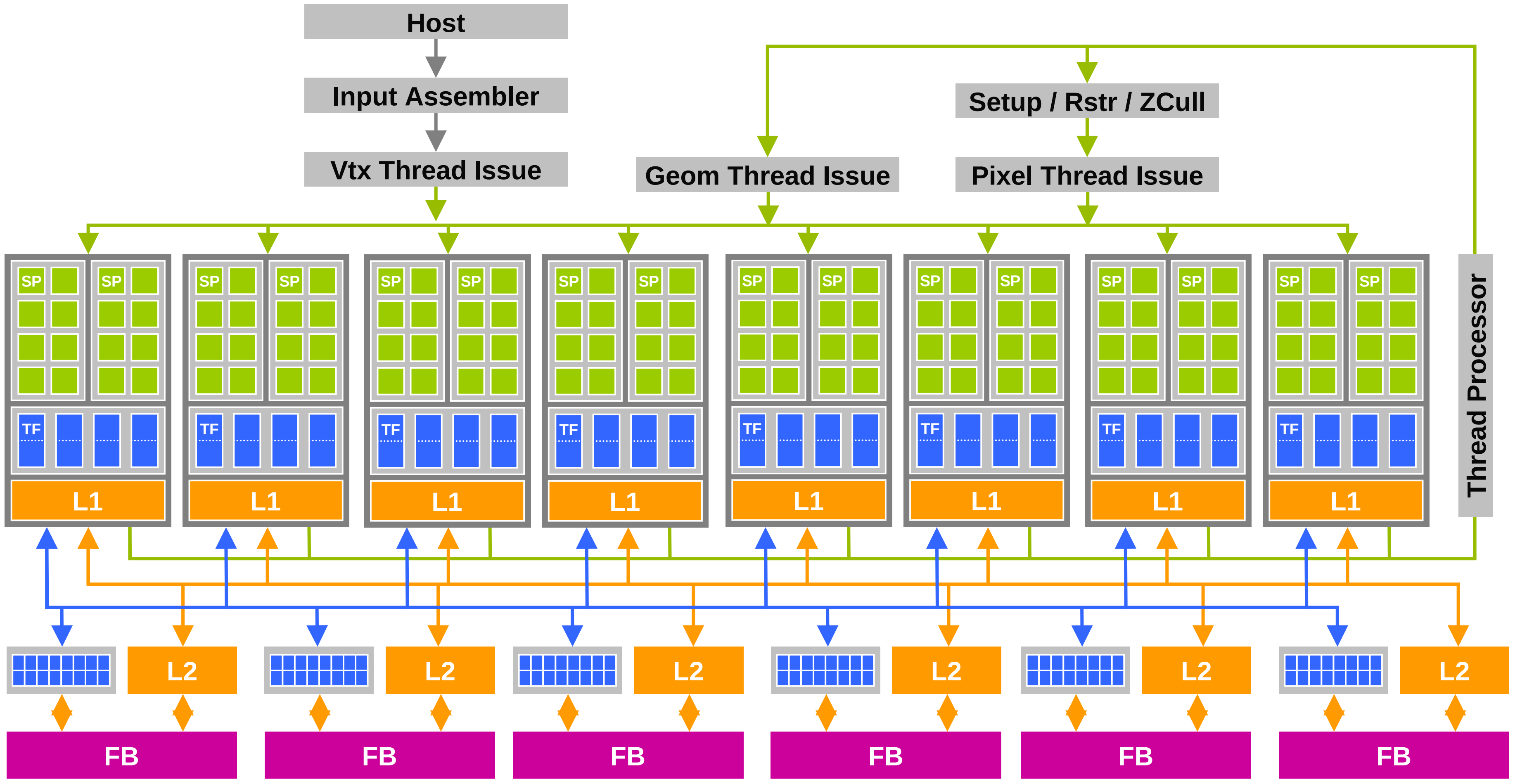
November 2006: NVIDIA announces CUDA for G80 GPU.

- CUDA makes GPU acceleration usable:
  - Developed and supported by NVIDIA.
  - No masquerading as graphics rendering.
  - New shared memory and synchronization.
  - No OpenGL or display device hassles.
  - Multiple processes per card (or vice versa).
- Resource and collaborators make it useful:
  - Experience from VMD development
  - David Kirk (Chief Scientist, NVIDIA)
  - Wen-mei Hwu (ECE Professor, UIUC)

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

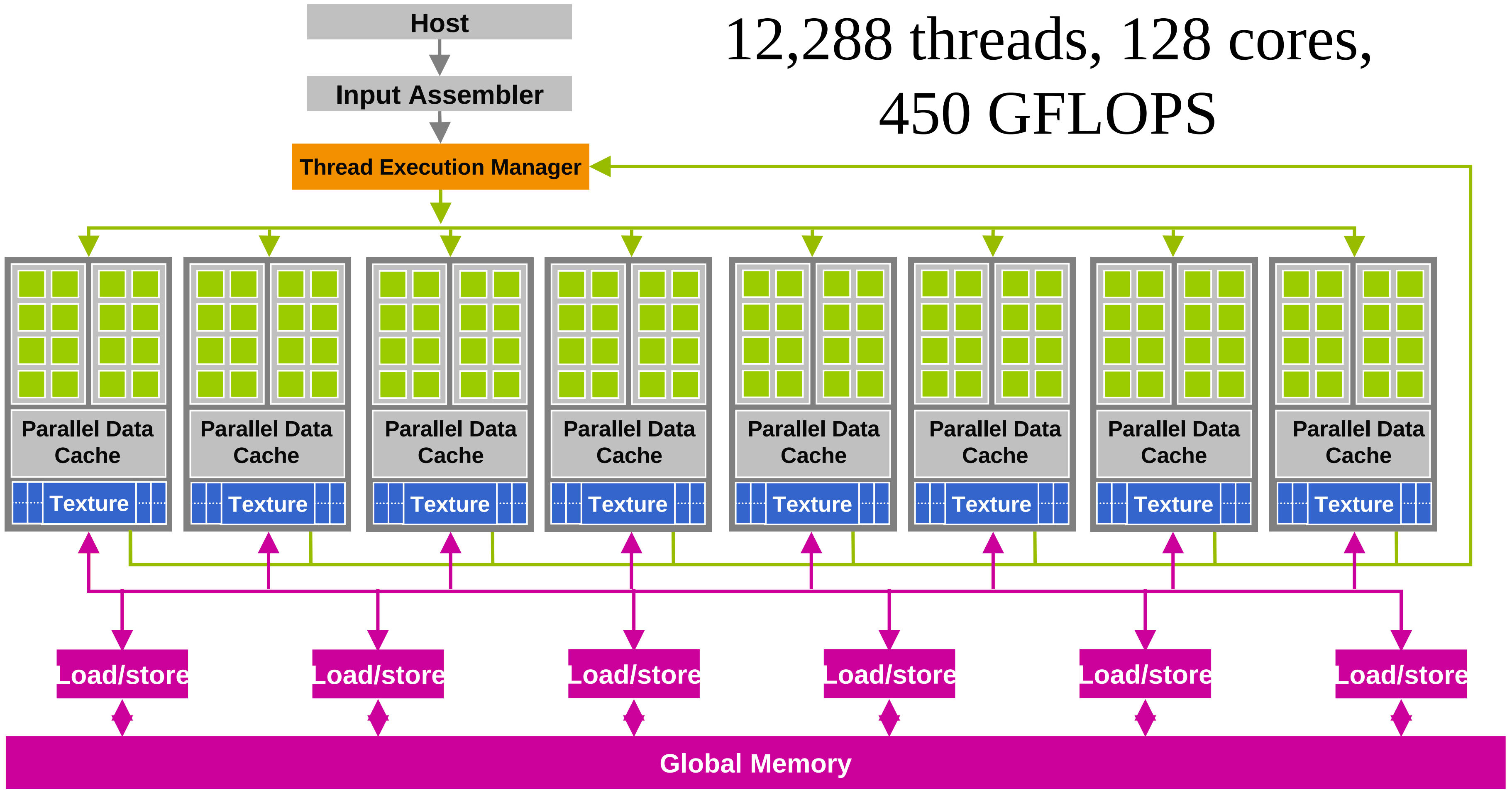
Fun to program (and drive)

GeForce 8800 Graphics Mode



GeForce 8800 General Computing

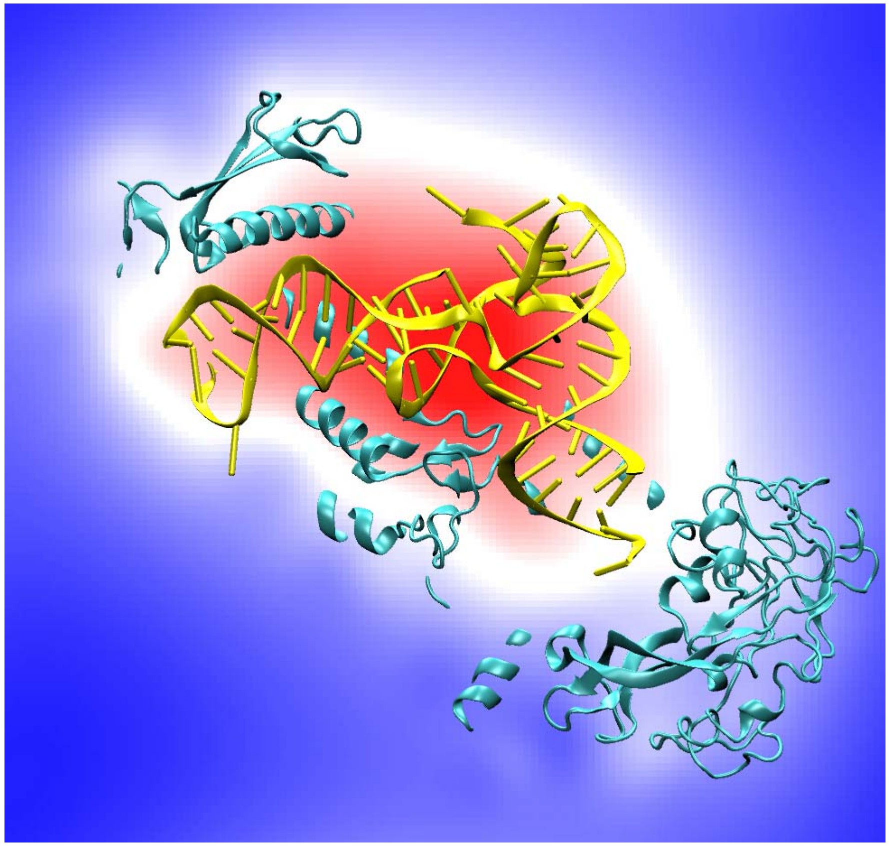
12,288 threads, 128 cores, 450 GFLOPS



768 MB DRAM, 4GB/S bandwidth to CPU

Calculating Electrostatic Potential Maps

- Used in 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



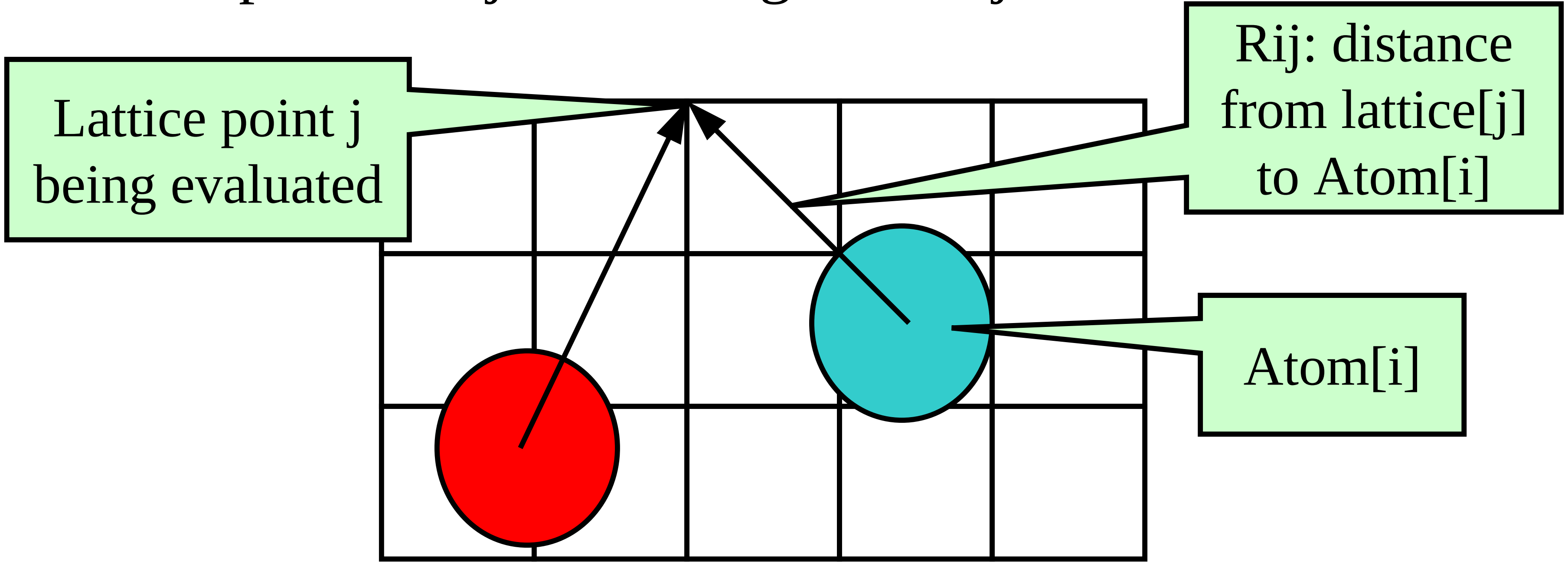
Positive potential field

Negative potential field

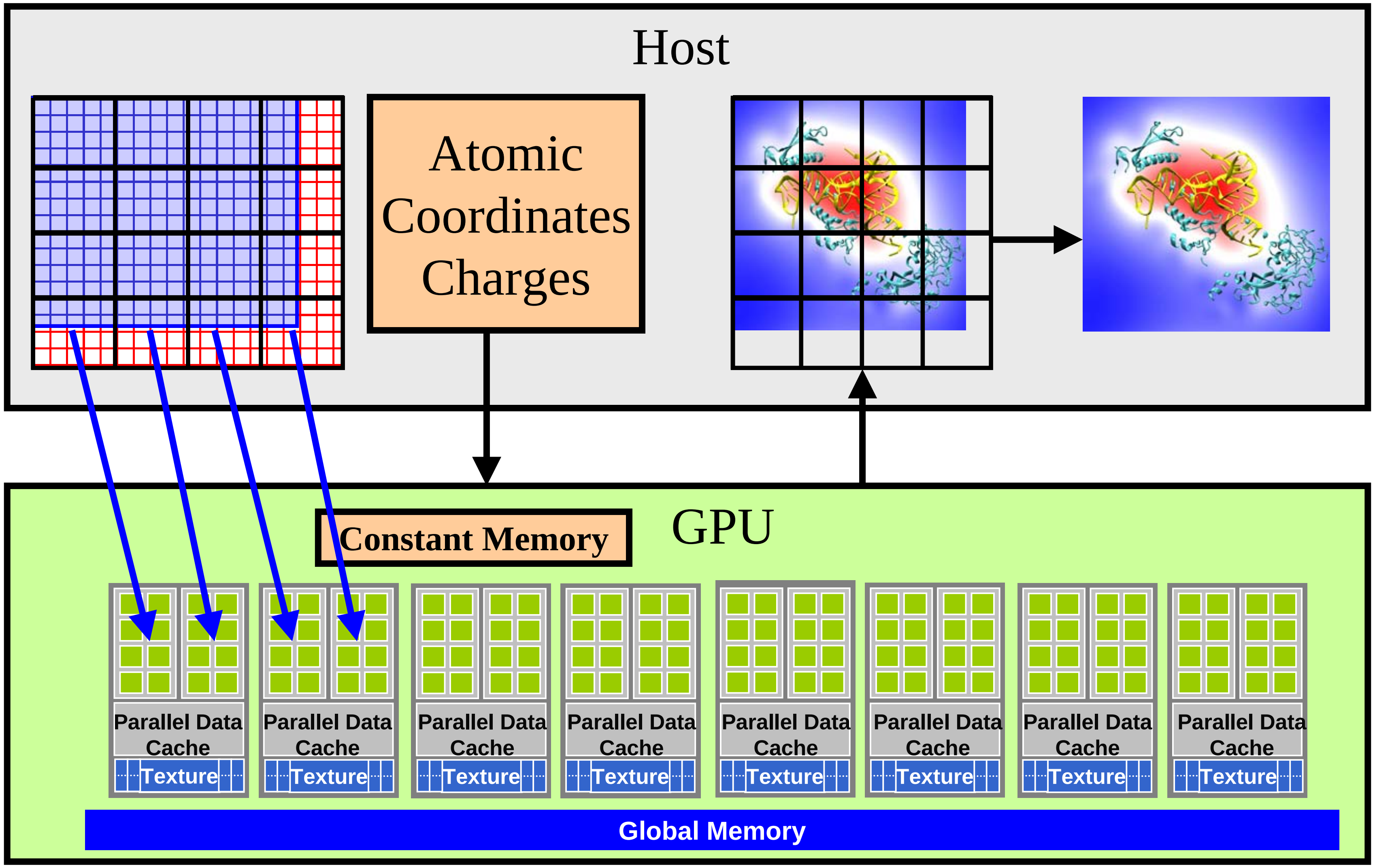
Direct Coulomb Summation

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

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


Lattice point j being evaluated
Rij: distance from lattice[j] to Atom[i]
Atom[i]

Direct Coulomb Summation on the GPU

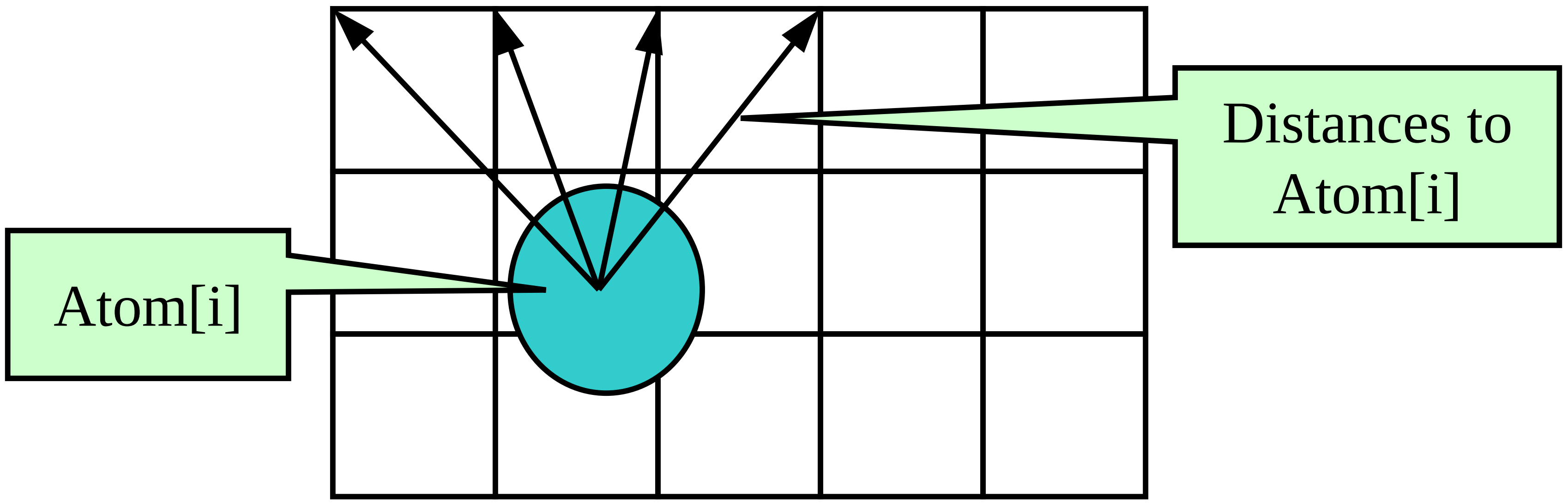


Optimizing for the GPU

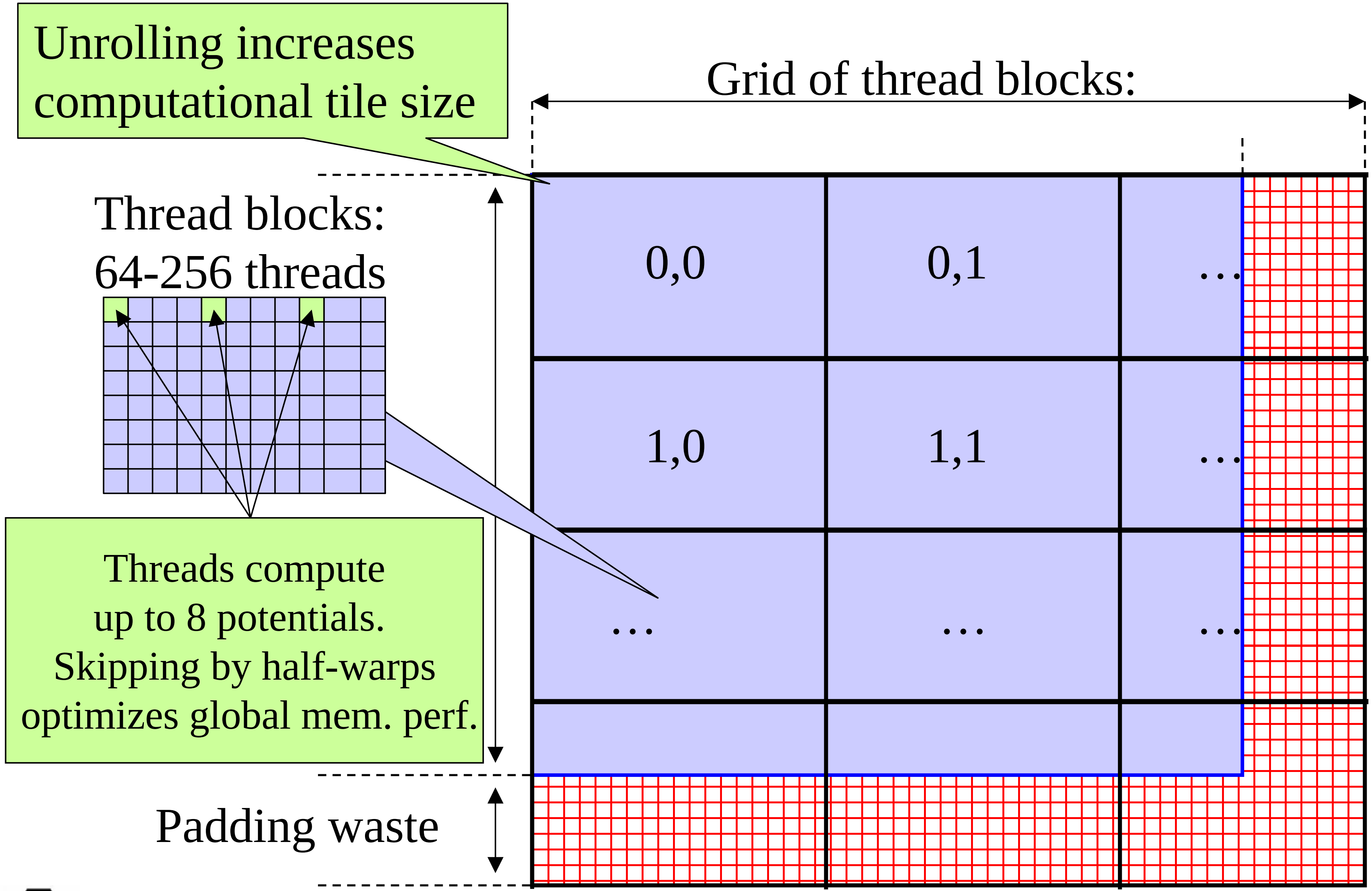
- Increase arithmetic intensity, reuse in-register data by “unrolling” lattice point computation into inner atom loop
- Each atom contributes to several lattice points, distances only differ in the X component:

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

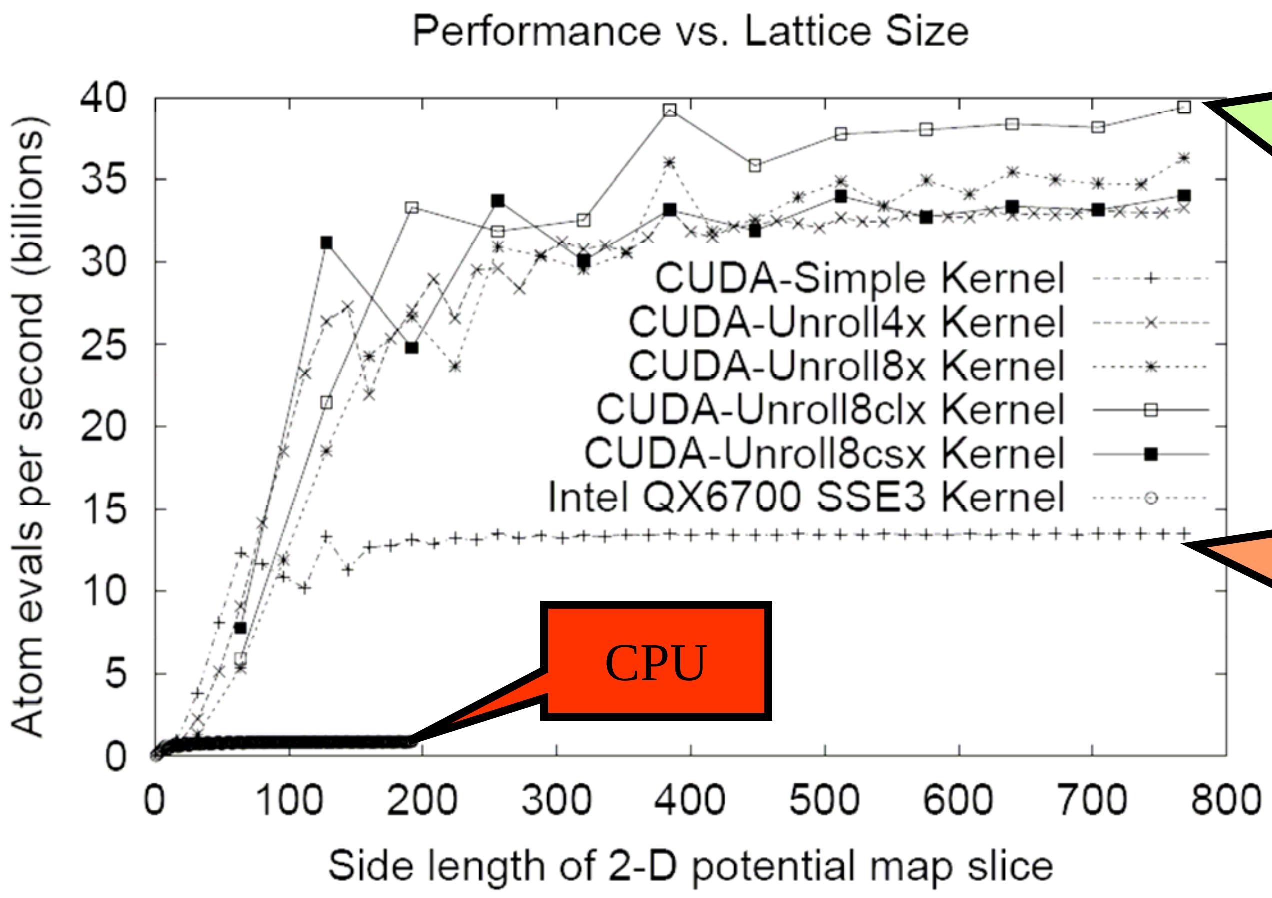
  potentialB += charge[i] / (distanceB to atom[i]) …



CUDA Block/Grid Decomposition


Unrolling increases computational tile size
Grid of thread blocks:
Thread blocks: 64-256 threads
0,0
0,1
…
1,0
1,1
…
…
…
…
Threads compute up to 8 potentials. Skipping by half-warps optimizes global mem. perf.
Padding waste

Direct Coulomb Summation Performance

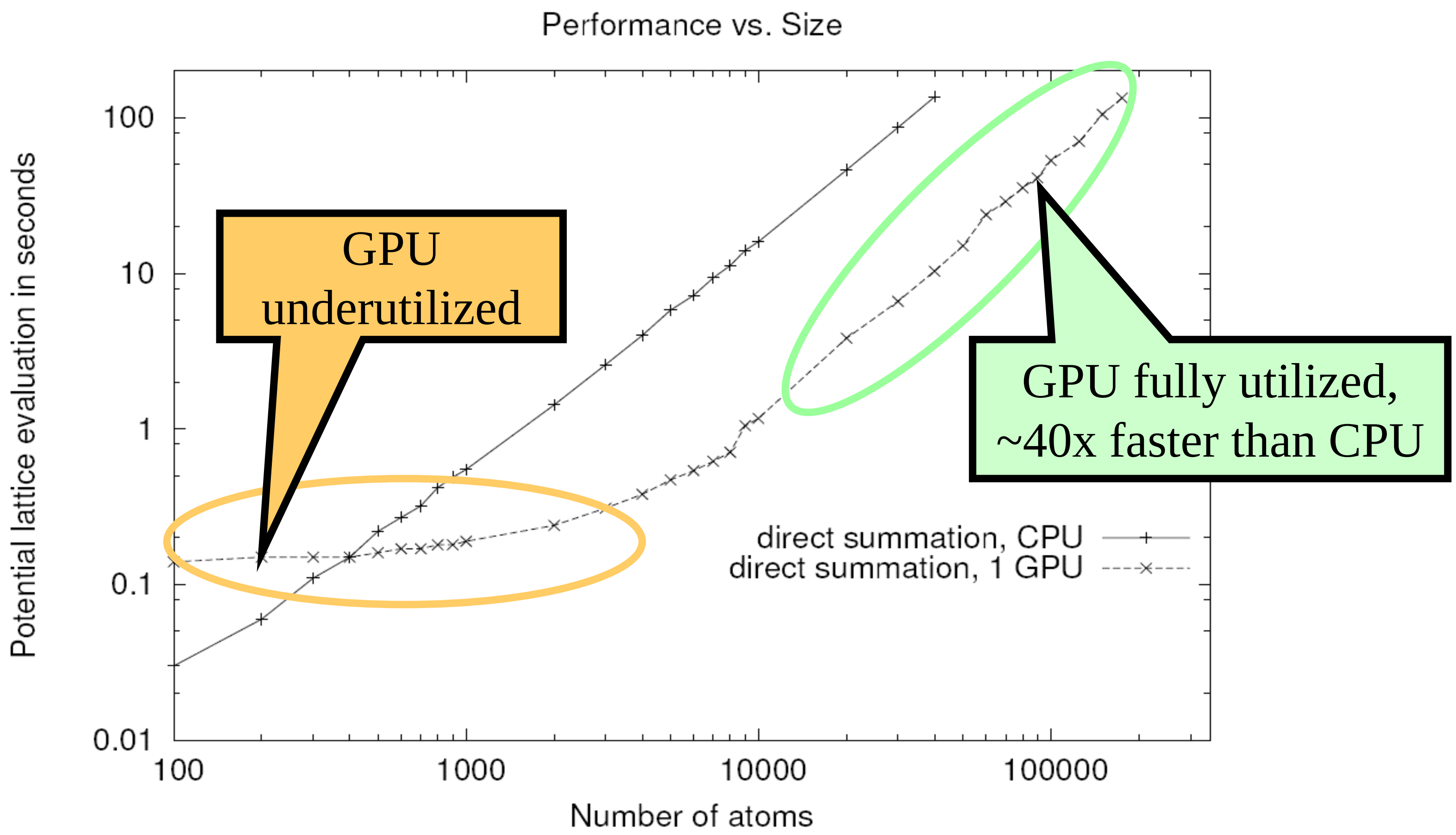


CUDA-Unroll8clx: fastest GPU kernel, 44x faster than CPU, 291 GFLOPS on GeForce 8800GTX

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.

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.

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

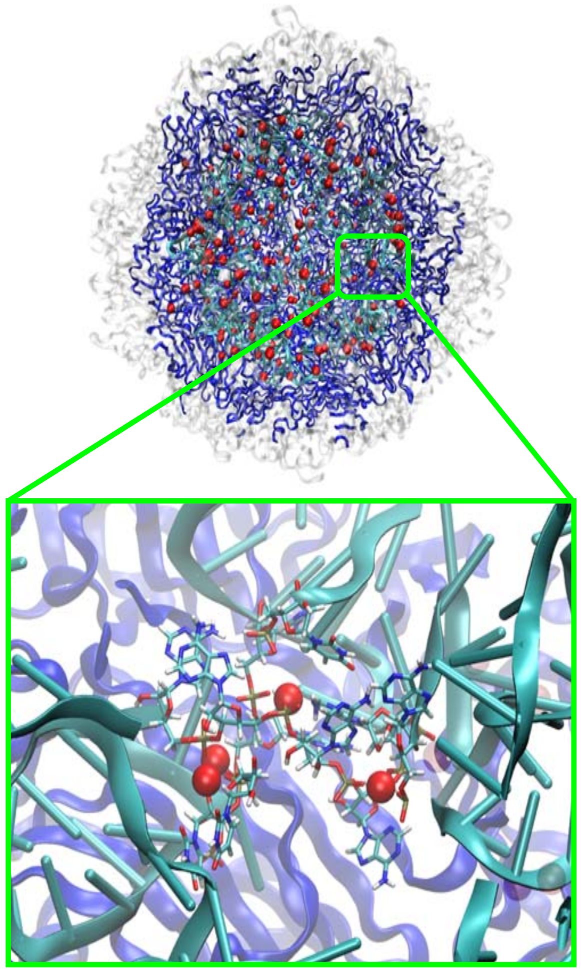
Multi-GPU Direct Coulomb Summation

- 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

GPU Application Performance
(July 2007, current kernels are 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



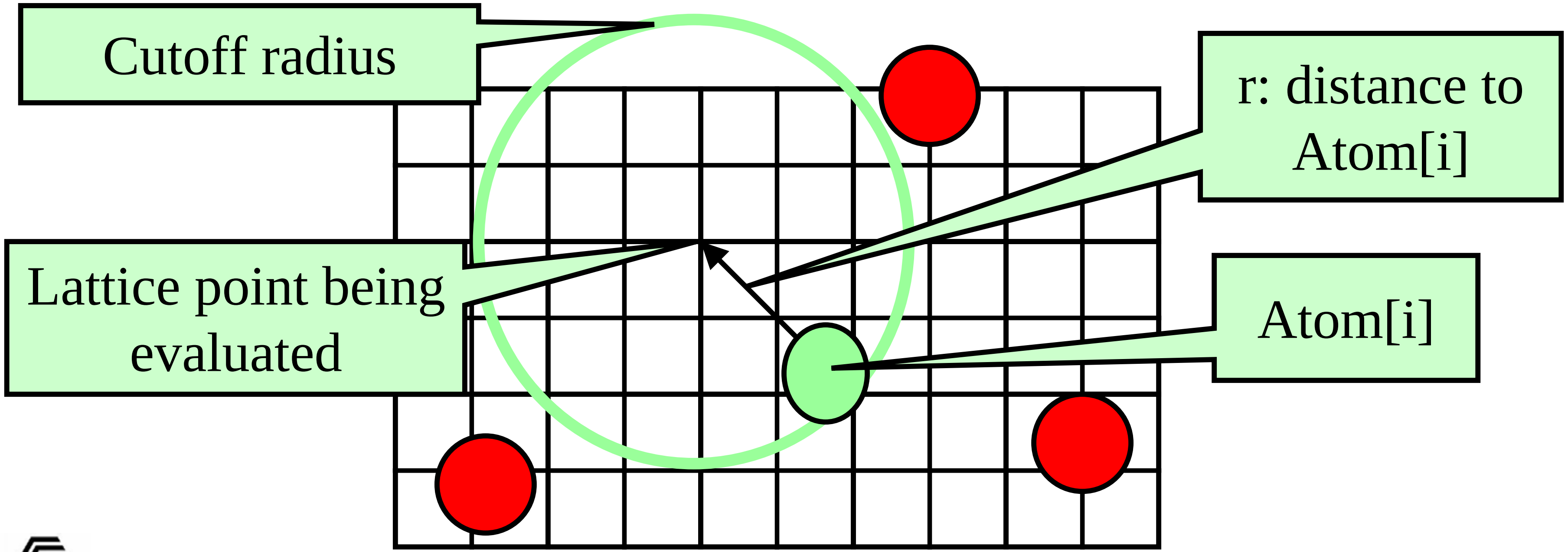
Satellite Tobacco Mosaic Virus (STMV)
Ion Placement

Cutoff Summation

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

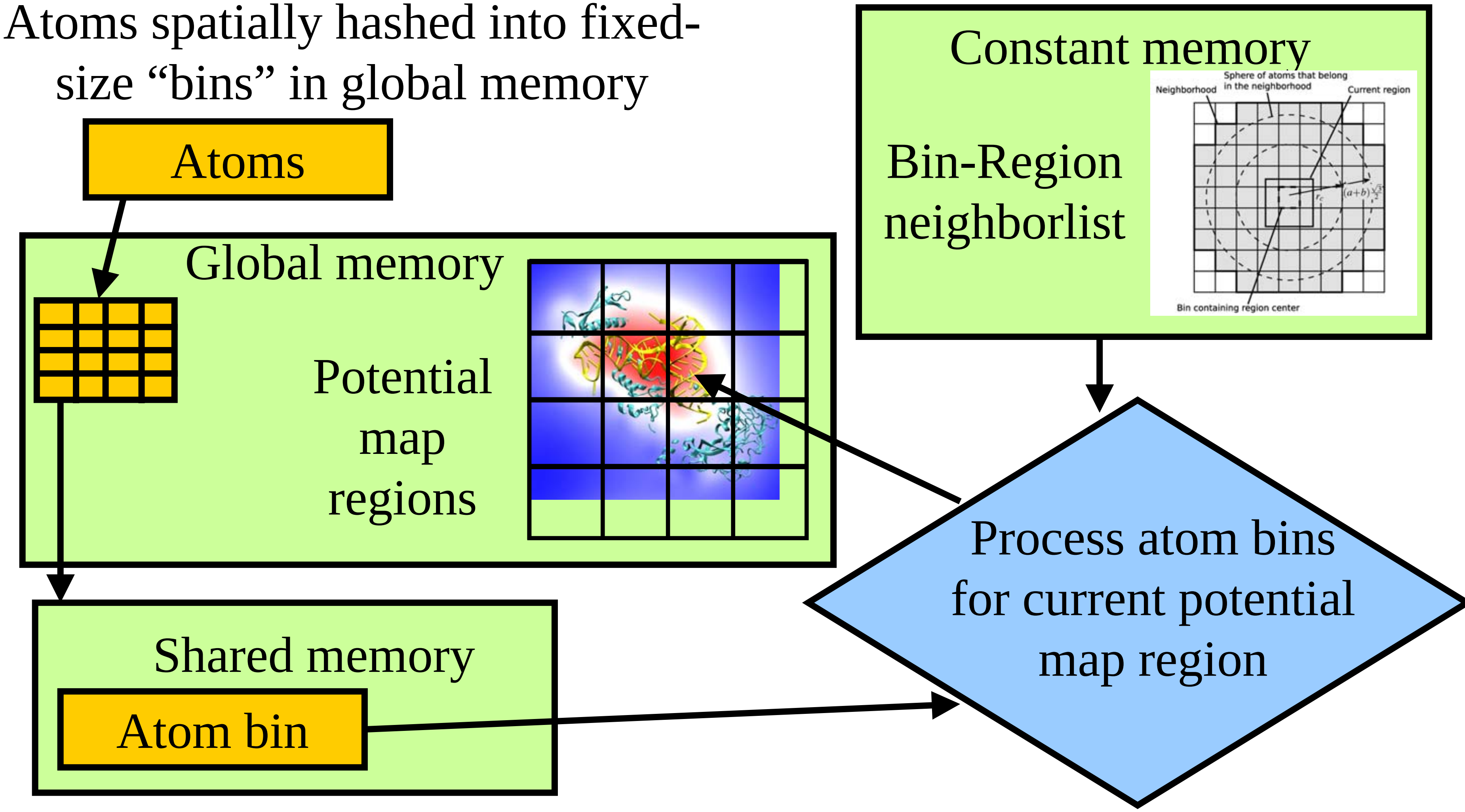
```
if (distance to atom[i] < cutoff)
  potential += (charge[i] / r) * s(r)
```

- Smoothing function s(r) is algorithm dependent

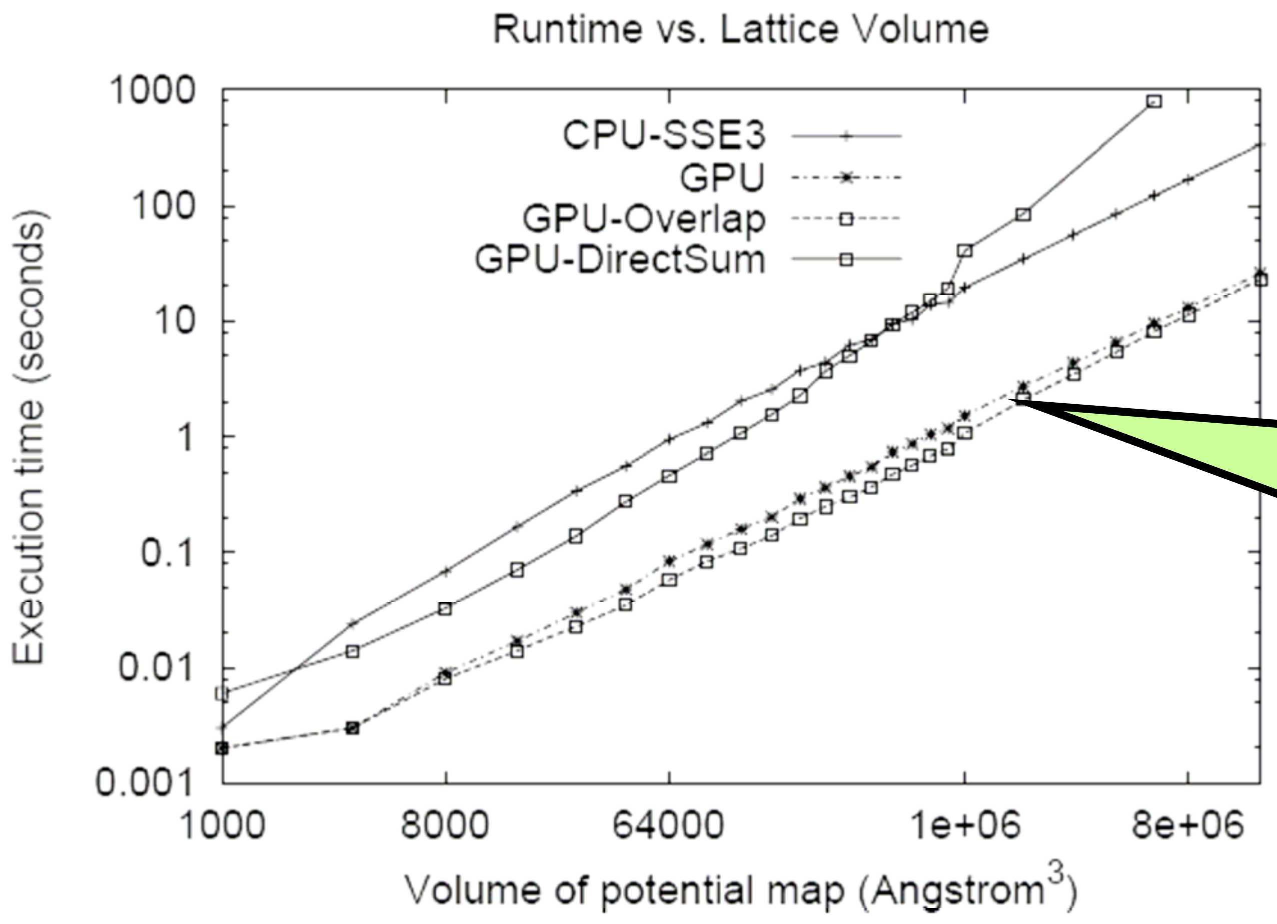


Cutoff Summation on the GPU

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
Neighborhood
Sphere of atoms that belong in the neighborhood
Current region
r_c
$(a+b)\frac{\sqrt{3}}{2}$
Bin containing region center
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 Design

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

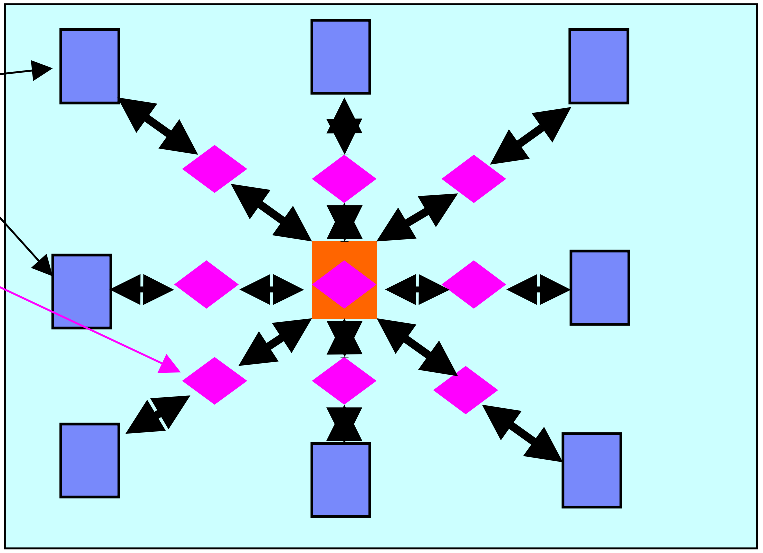
- Designed from the beginning as a parallel program
- Uses the Charm++ idea:
  - Decompose the computation into a large number of objects
  - Have an Intelligent Run-time system (of Charm++) assign objects to processors for dynamic load balancing with minimal communication

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

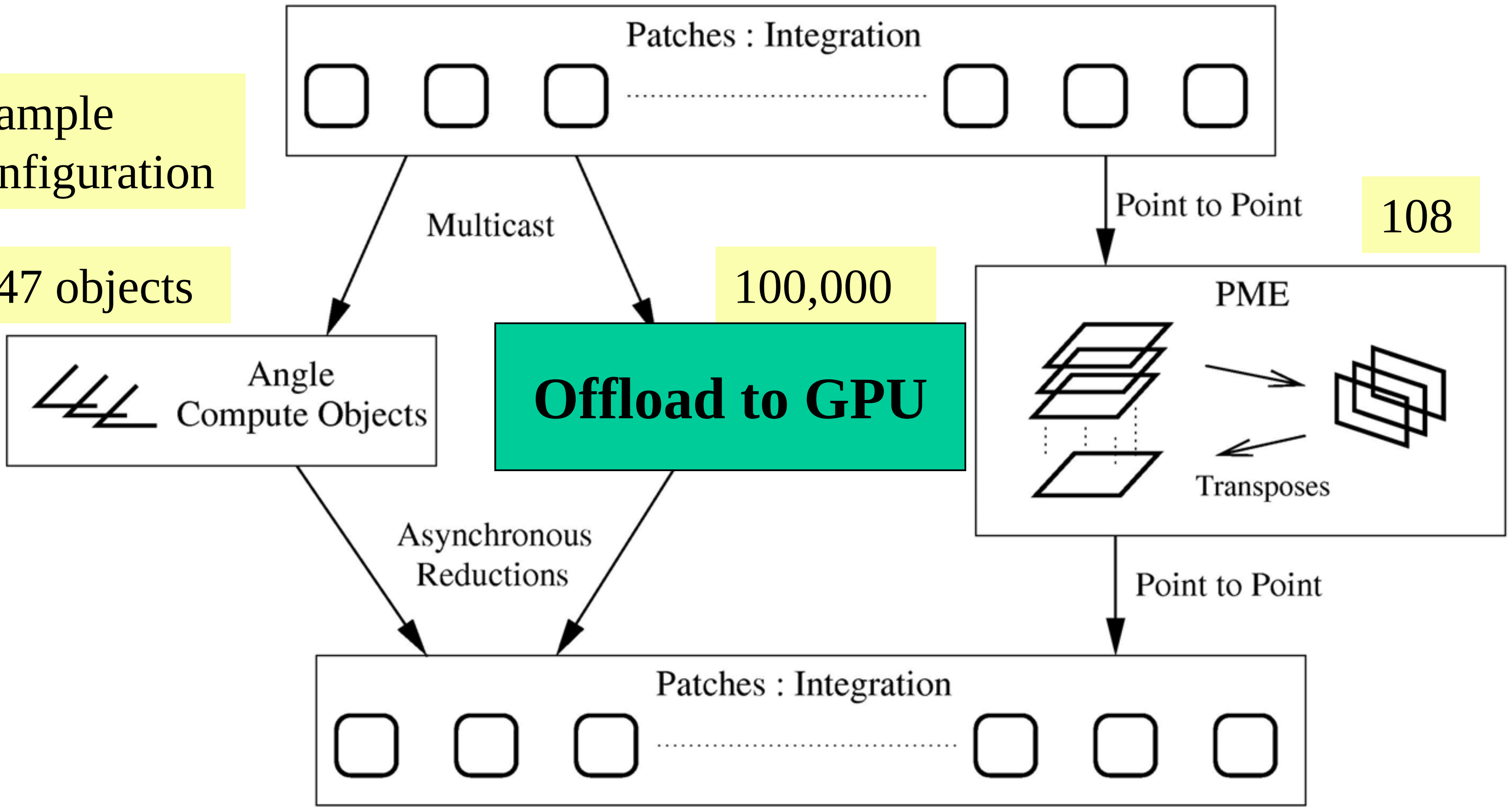
Phillips *et al.*, *SC2002*.

Example Configuration

847 objects

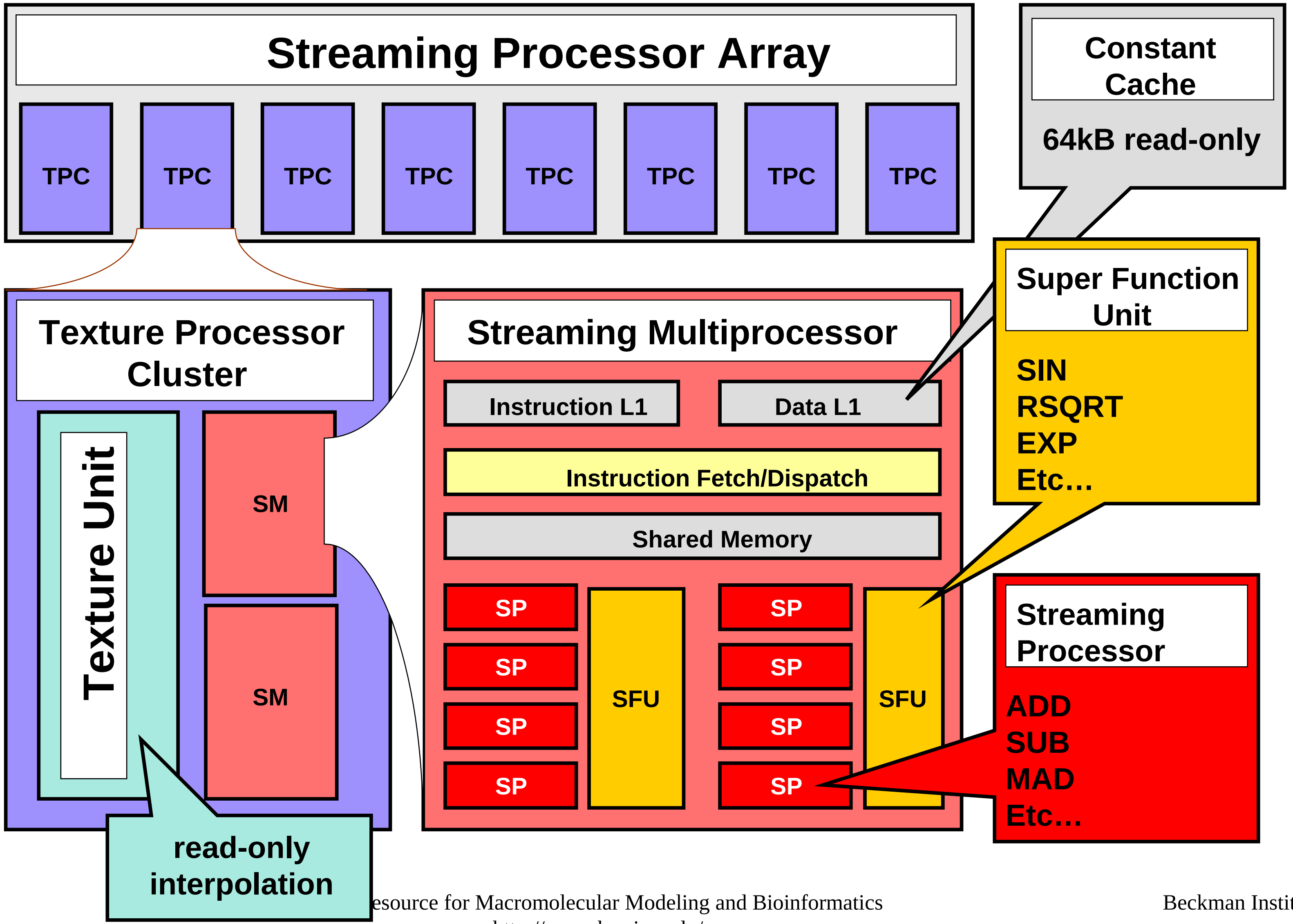
100,000

108



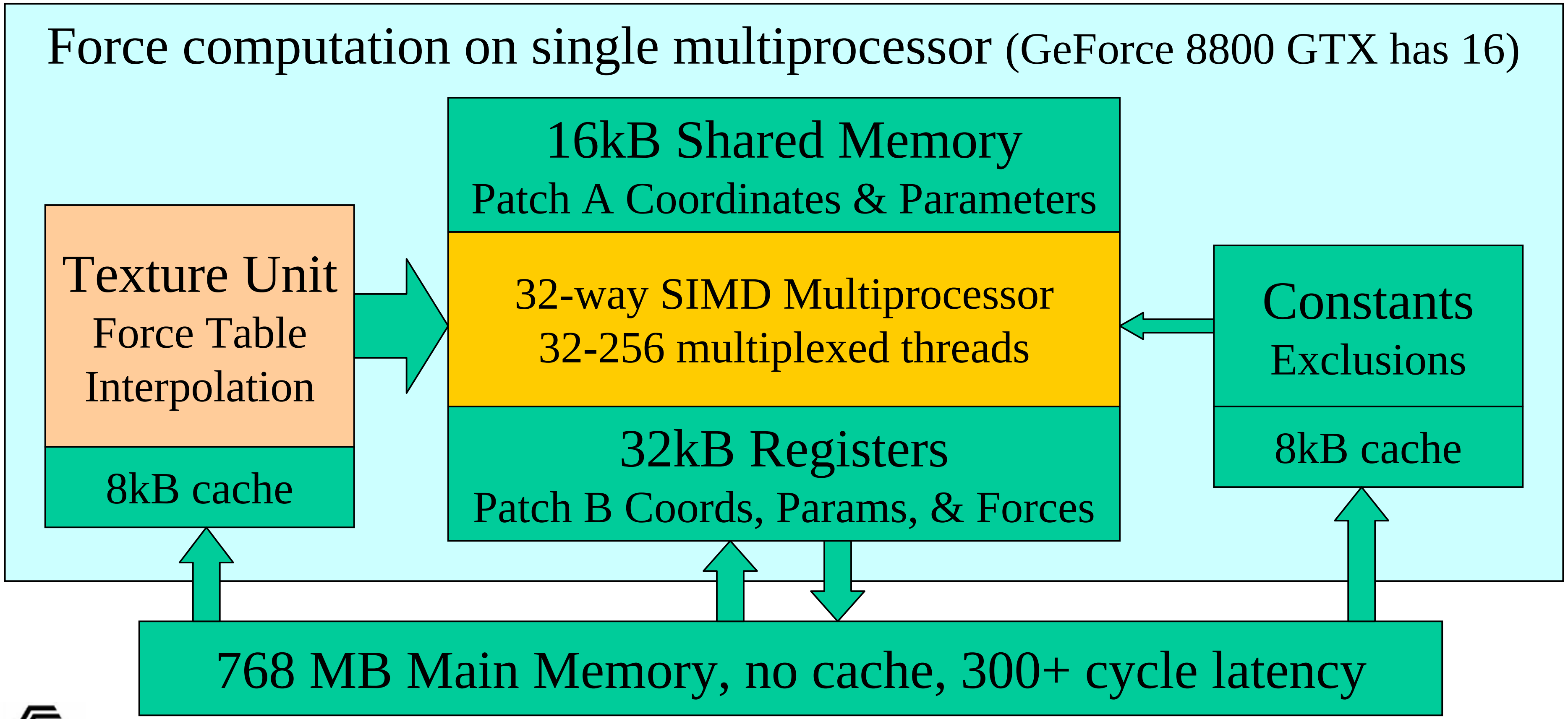
Objects are assigned to processors and queued as data arrives.

GPU Hardware Special Features



Nonbonded Forces on CUDA 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.


Force computation on single multiprocessor (GeForce 8800 GTX has 16)
16kB Shared Memory
Patch A Coordinates & Parameters
Texture Unit
Force Table
Interpolation
8kB cache
32-way SIMD Multiprocessor
32-256 multiplexed threads
Constants
Exclusions
8kB cache
32kB Registers
Patch B Coords, Params, & Forces
768 MB Main Memory, no cache, 300+ cycle latency

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.

Why Calculate Each Force Twice?

- Newton’s 3rd Law of Motion: $\mathbf{F}_{ij} = \mathbf{F}_{ji}$
  - Could calculate force once and apply to both atoms.
- Floating point operations are cheap:
  - Would save at most a factor of two.
- Almost everything else hurts performance:
  - Warp divergence
  - Memory access
  - Synchronization
  - Extra registers
  - Integer logic

What About Pairlists?

- Generation works well under CUDA
  - – Assign atoms to cells
  - – Search neighboring cells
  - – Write neighbors to lists as they are found
  - – Scatter capability essential
  - – 10x speedup relative to CPU
- Potential for significant performance boost
  - – Eliminate 90% of distance test calculations

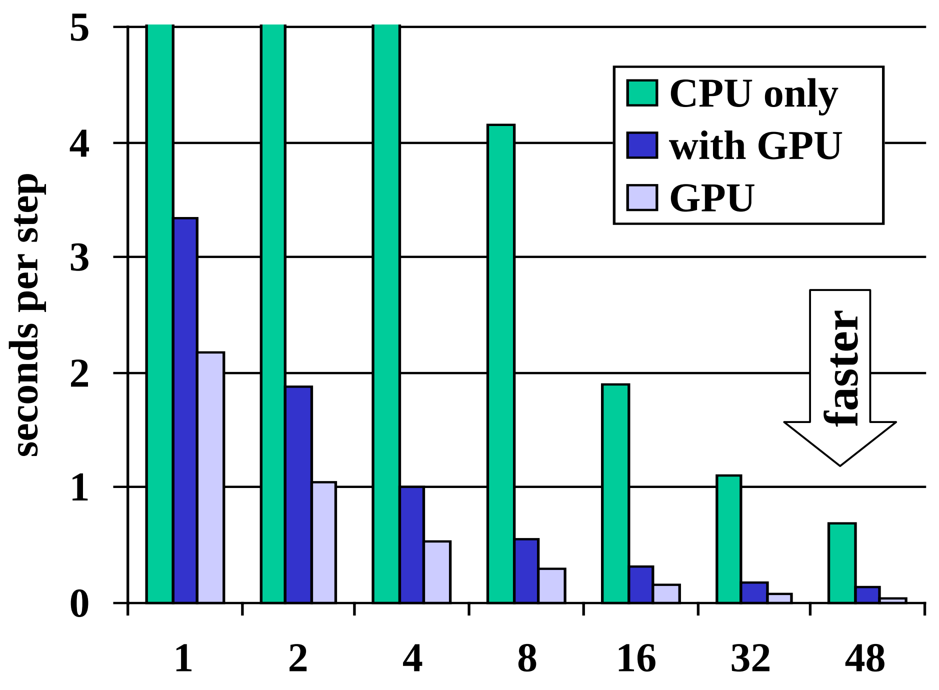
Why Not Pairlists?

- Changes FP-limited to memory limited:
  - Limited memory to hold pairlists
  - Limited bandwidth to load pairlists
  - Random access to coordinates, etc.
  - FP performance grows faster than memory
- Poor fit to NAMD parallel decomposition:
  - Number of pairs in single object varies greatly

NCSA GPU Cluster Performance

- 7x speedup
- Large system (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 (?)

STMV Performance



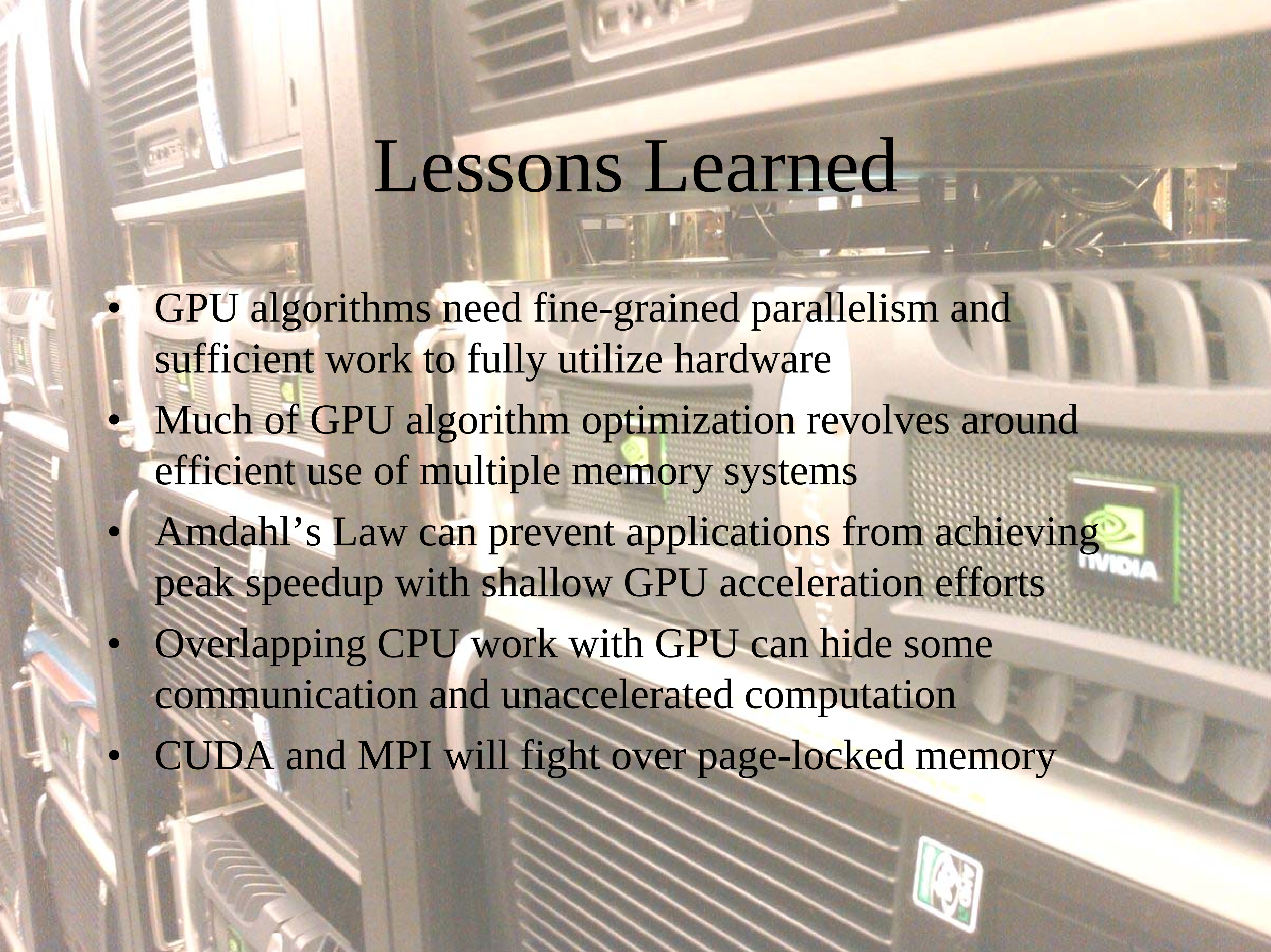
2.4 GHz Opteron + Quadro FX 5600
Thanks to NCSA and NVIDIA

GPU Performance Results, March 2008

GeForce 8800GTX w/ CUDA 1.1, Driver 169.09

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
Cutoff potential summation	Particle-grid w/ cutoff	12-21x
Direct Coulomb summation	Particle-grid	44x

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



Lessons Learned

- GPU algorithms need fine-grained parallelism and sufficient work to fully utilize hardware
- Much of GPU algorithm optimization revolves around efficient use of multiple memory systems
- 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
- CUDA and MPI will fight over page-locked memory

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
- 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.