Using GPU Computing to Accelerate Molecular Modeling Applications

David J. Hardy

Theoretical and Computational Biophysics Group
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
http://www.ks.uiuc.edu/Research/gpu/

CECAM Workshop, "Algorithmic re-engineering..."
Lugano, Switzerland, October 2, 2009

Outline

- Overview of GPU computing
    - NVIDIA CUDA programming model
- GPU acceleration of NAMD and VMD
- Case study: GPU acceleration of multilevel summation of electrostatic potentials

Why GPU Computing?

Over a **million** sold per **week**!

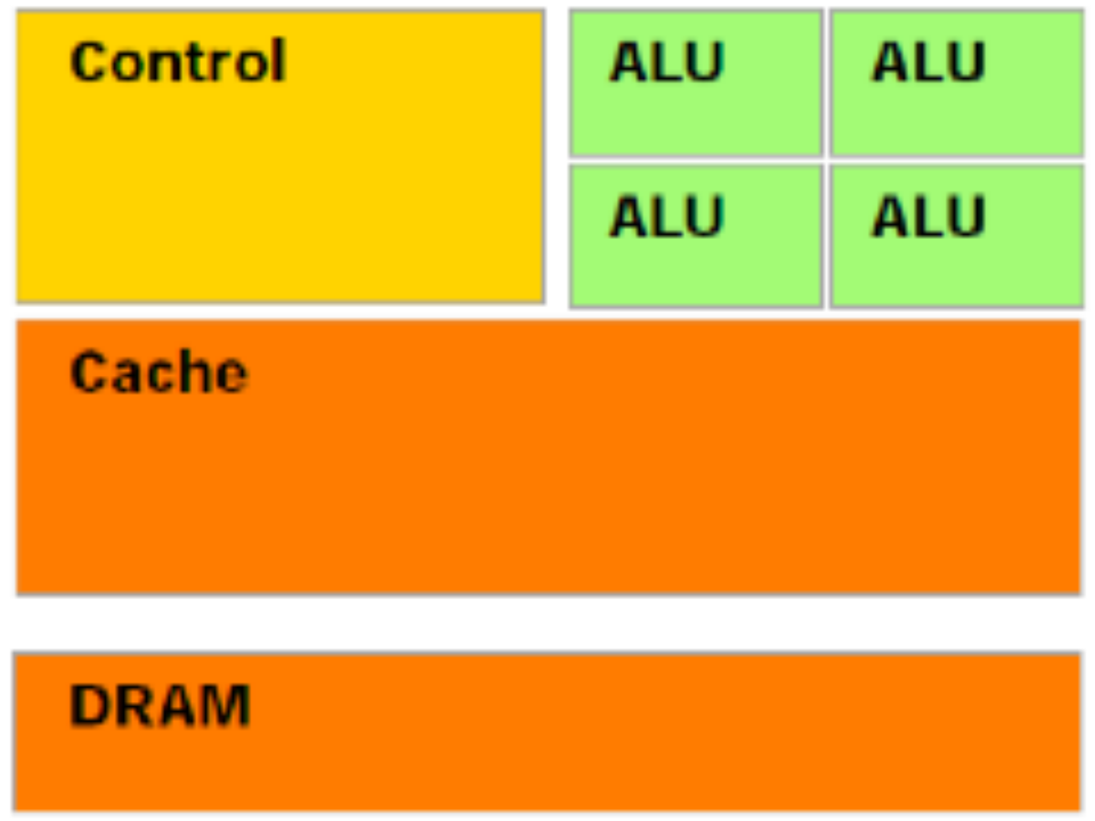
- Cost effective, commodity devices
- Massively parallel hardware with hundreds of processing units offering substantial floating point performance over CPUs
- Fully programmable processors that support standard integer and floating point types
- Programming toolkits allow software to be written in dialects of C/C++ and integrated into legacy software
- GPU algorithms are generally multicore friendly due to data locality and data-parallel work decomposition
- We need more processing power, but CPU core speeds aren't getting faster! We must exploit GPU/multicore technologies!

What speedups can GPUs achieve?

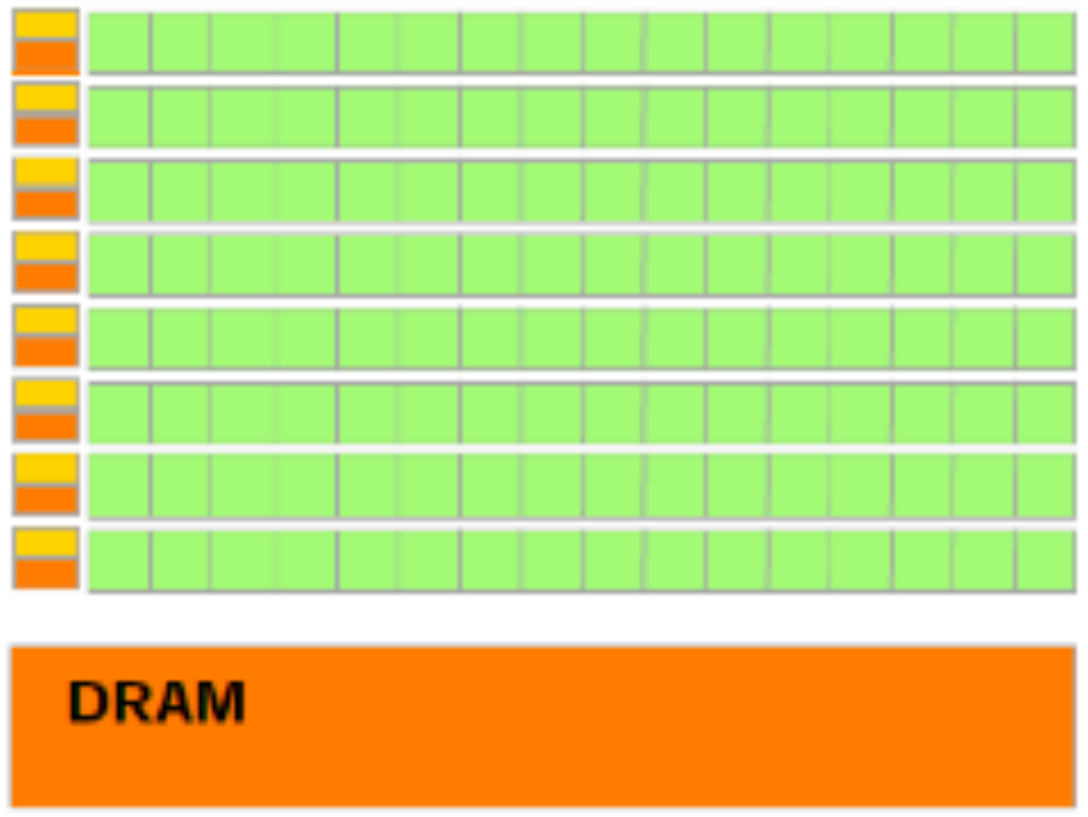
- Single GPU speedups of 10x to 30x over one CPU core are common
- Best speedups of 100x or more attained for codes dominated by floating point arithmetic, especially for native GPU machine instructions, e.g. expf(), rsqrtf()
- Legacy codes employing "shallow" efforts at GPU acceleration might not exhibit these peak speedups due to Amdahl's Law
- GPU programming toolkits require the programmer to balance architectural tradeoffs for best performance

Comparison of CPU and GPU Hardware Architecture

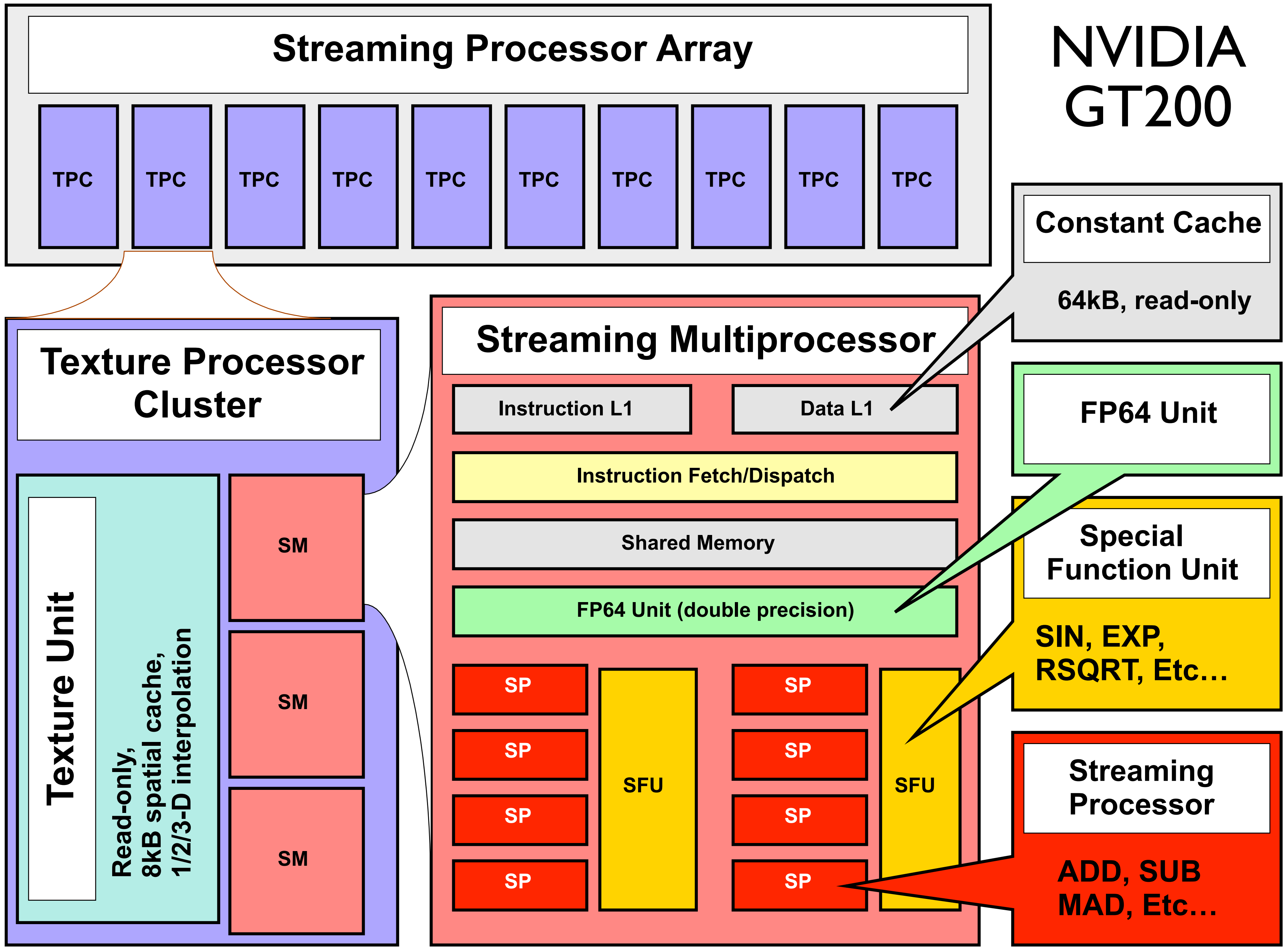
CPU: Cache heavy, focused on individual thread performance



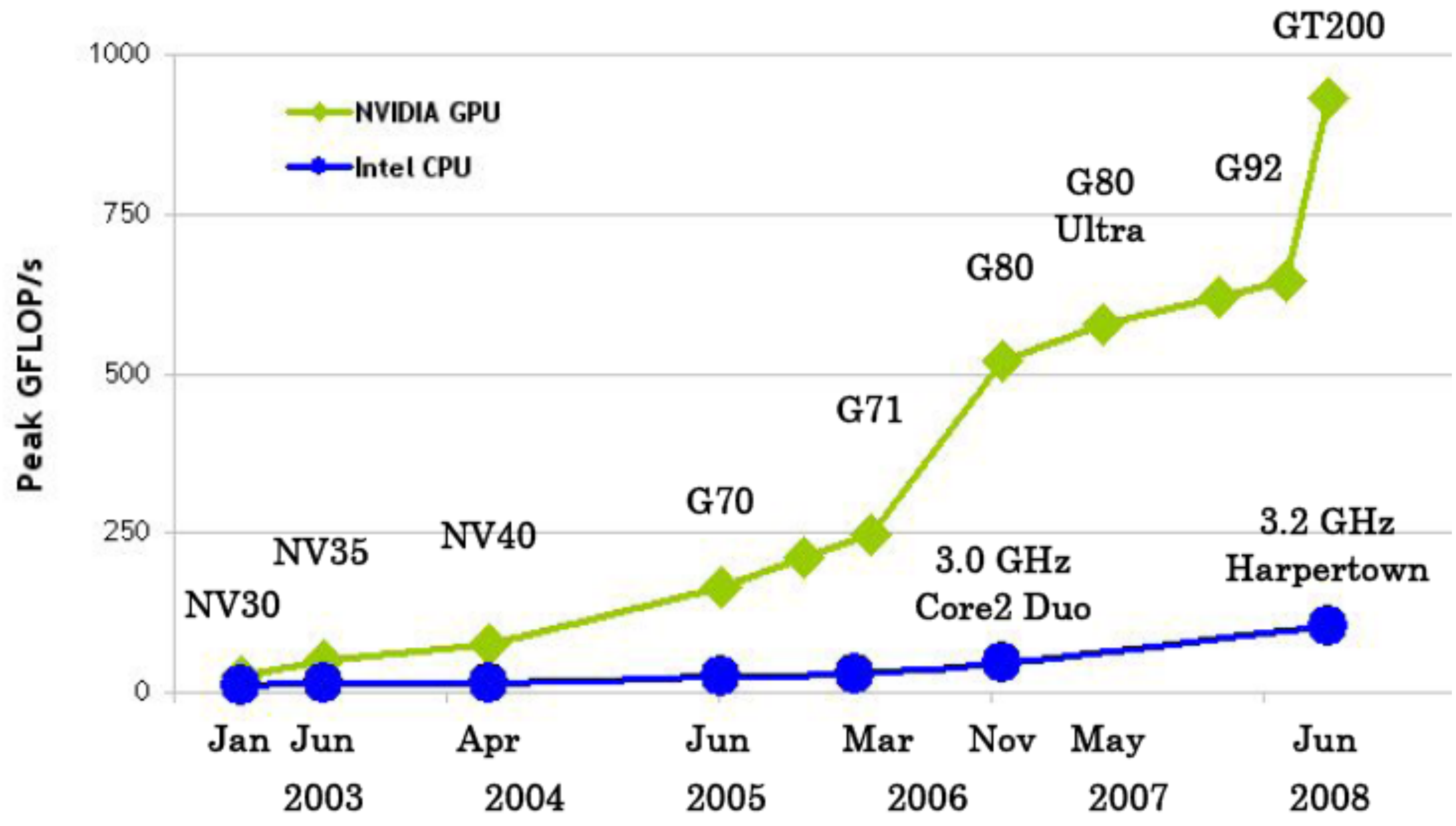
GPU: ALU heavy, massively parallel, throughput oriented



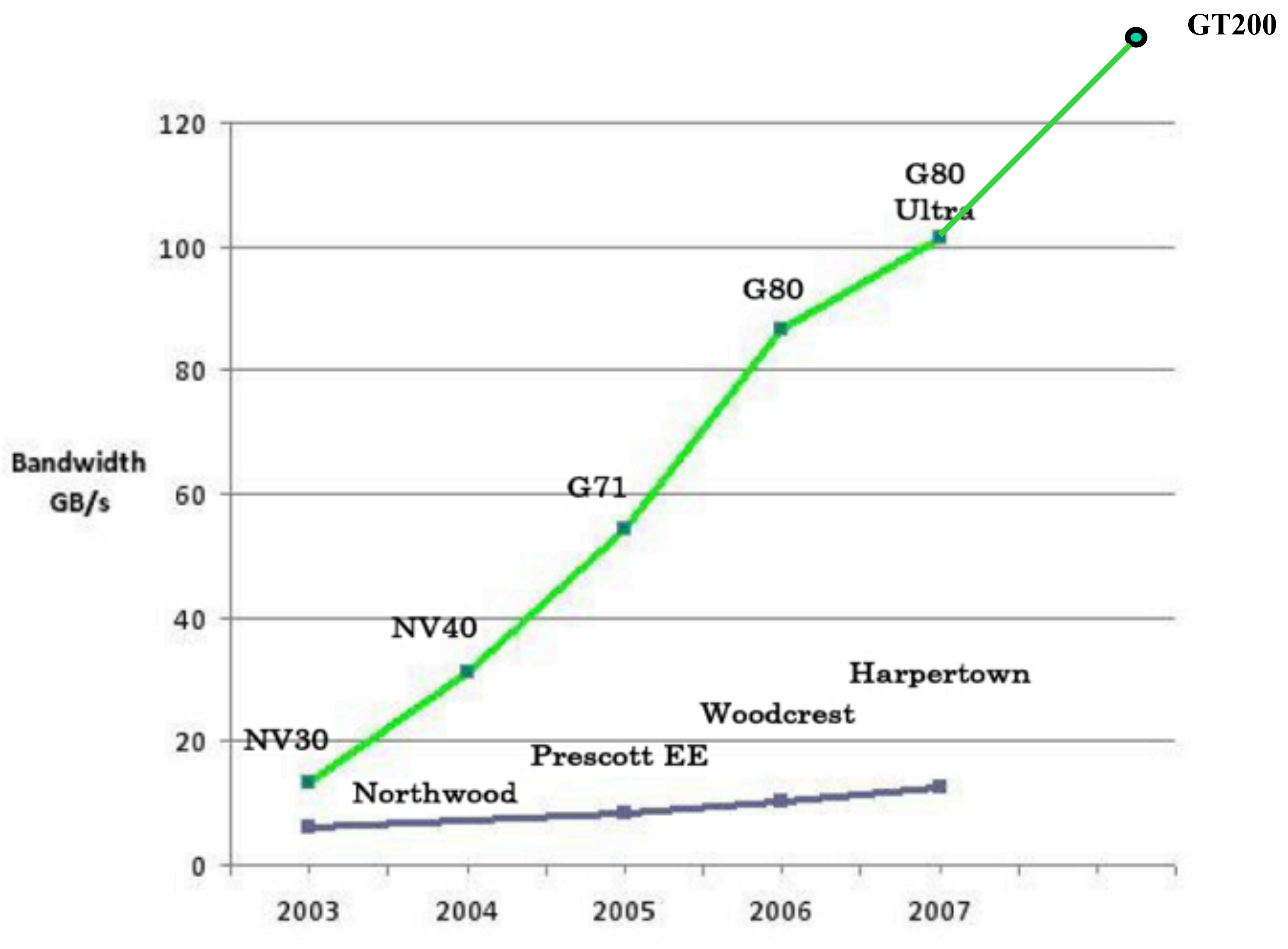
NVIDIA GPU Architecture



GPU Peak Single-precision Performance: **Exponential** Trend



GPU Peak Memory Bandwidth: **Linear** Trend



GPU Future Performance Trends

- We expect the ratio between floating point performance and memory bandwidth will continue to increase
- Algorithms with linear time complexity O(N) will be increasingly memory bound
- Implications for GPU algorithm design:
  - Use shared memory and constant memory caches to amplify the effective GPU memory bandwidth
  - We can benefit from tradeoffs that increase computational density while either decreasing memory access or increasing parallel scheduling (e.g. don't use Newton's 3rd Law to evaluate particle-particle interactions)

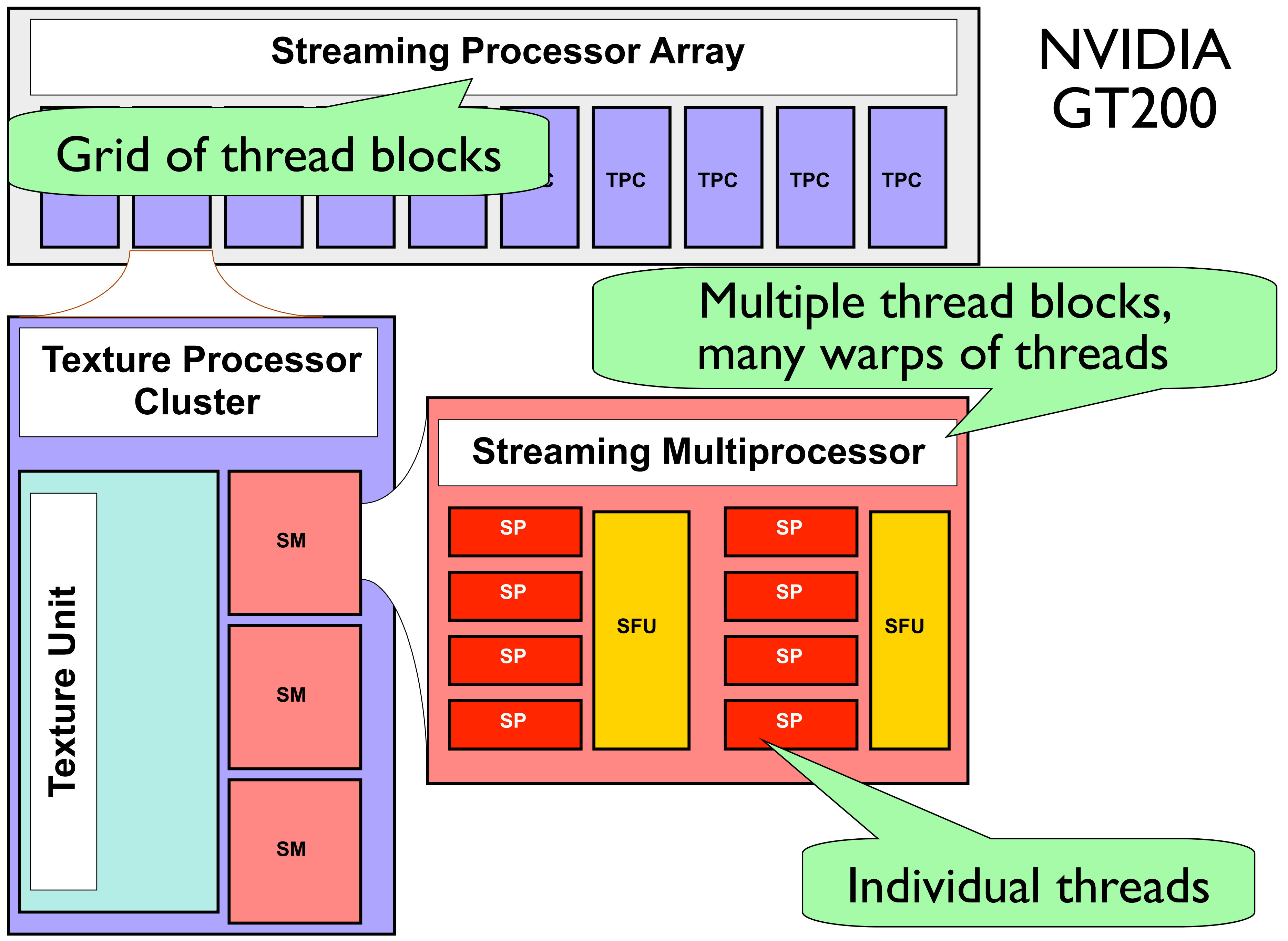
NVIDIA CUDA Overview

- Hardware and software architecture for GPU computing, foundation for building higher level programming libraries and toolkits
- CUDA released in 2007:
  - Data-parallel programming model
  - Work is decomposed into ***grids*** of ***blocks*** containing ***warps*** of ***threads***, multiplexed onto massively parallel GPU hardware
  - Light-weight, low level of abstraction, exposes many GPU architecture features to enable development of high performance GPU kernels

CUDA Threads, Blocks, Grids

- GPUs use hardware multithreading to hide latency and achieve high ALU utilization
- For high performance, a GPU must be *saturated* with concurrent work: *at least 10,000 threads*
- A ***grid*** of hundreds of ***thread blocks*** is scheduled onto a large array of ***SIMT*** cores
- Each core executes several thread blocks of 64–512 threads each, switching among them to hide latencies for slow memory accesses, etc.
- 32-thread ***warps*** are executed in lock-step (e.g. in SIMD-like fashion)
- Conditionals are serialized over the *if* and *else* branches

Mapping CUDA Abstractions onto GPU



NVIDIA GT200

GPU Memory Accessible in CUDA

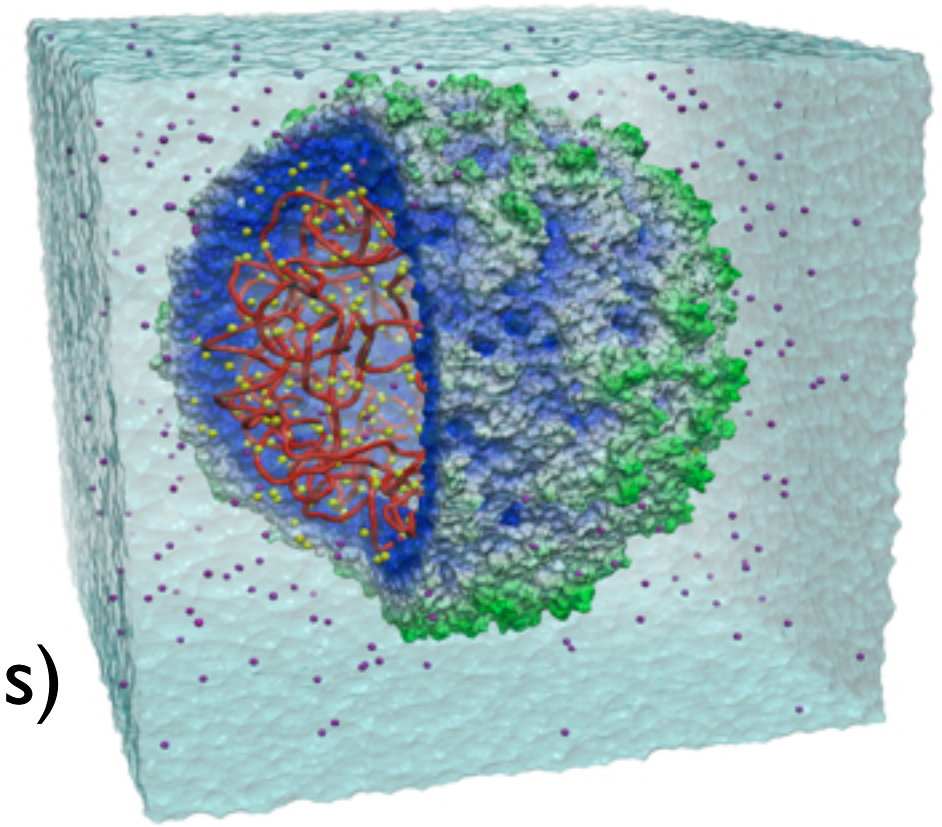
- Mapped host memory: up to 4GB, ~5.7GB/sec bandwidth (PCIe), accessible by multiple GPUs
- Global memory: up to 4GB, high latency (~600 clock cycles), 140GB/sec bandwidth, accessible by all threads; also supports slow atomic operations
- Texture memory: read-only, cached, and interpolated/filtered access to global memory
- Constant memory: 64KB, read-only, cached, fast/low-latency if data elements are accessed in unison by peer threads
- Shared memory: 16KB, low-latency, accessible among threads in the same block, fast if accessed without bank conflicts

An Approach to Writing CUDA Kernels

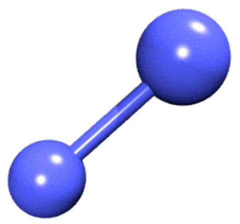
- Find an algorithm that exposes substantial parallelism, thousands of independent threads
- Identify appropriate GPU memory subsystems for storage of data used by kernel
- Are there tradeoffs that can be made to exchange computation for more parallelism?
  - Although counterintuitive, this strategy has resulted in past success
  - "Brute force" methods that expose significant parallelism do surprisingly well on current GPUs
- Analyze the real-world use case for the problem and optimize the kernel for problem size and characteristics that will be heavily used

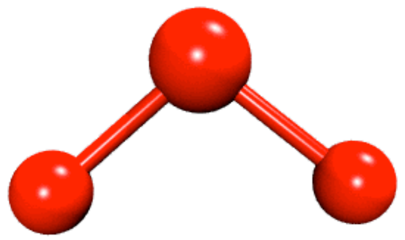
NAMD — "Nanoscale Molecular Dynamics"

- CHARMM, AMBER, OPLS, force fields
- Efficient PME full electrostatics
- Conjugate gradient minimization
- Temperature and pressure controls
- Steered molecular dynamics (many methods)
- Interactive molecular dynamics (with VMD)
- Locally enhanced sampling
- Alchemical free energy perturbation
- User extensible in Tcl for forces and algorithms
- *All features run in parallel and scale to millions of atoms!*

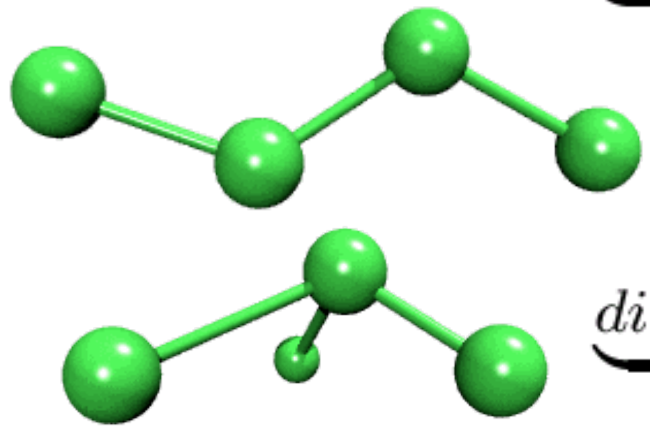


Molecular Mechanics Force Field





$$U(\vec{R}) = \underbrace{\sum_{bonds} k_i^{bond}(r_i - r_0)^2}_{U_{bond}} + \underbrace{\sum_{angles} k_i^{angle}(\theta_i - \theta_0)^2}_{U_{angle}} +$$

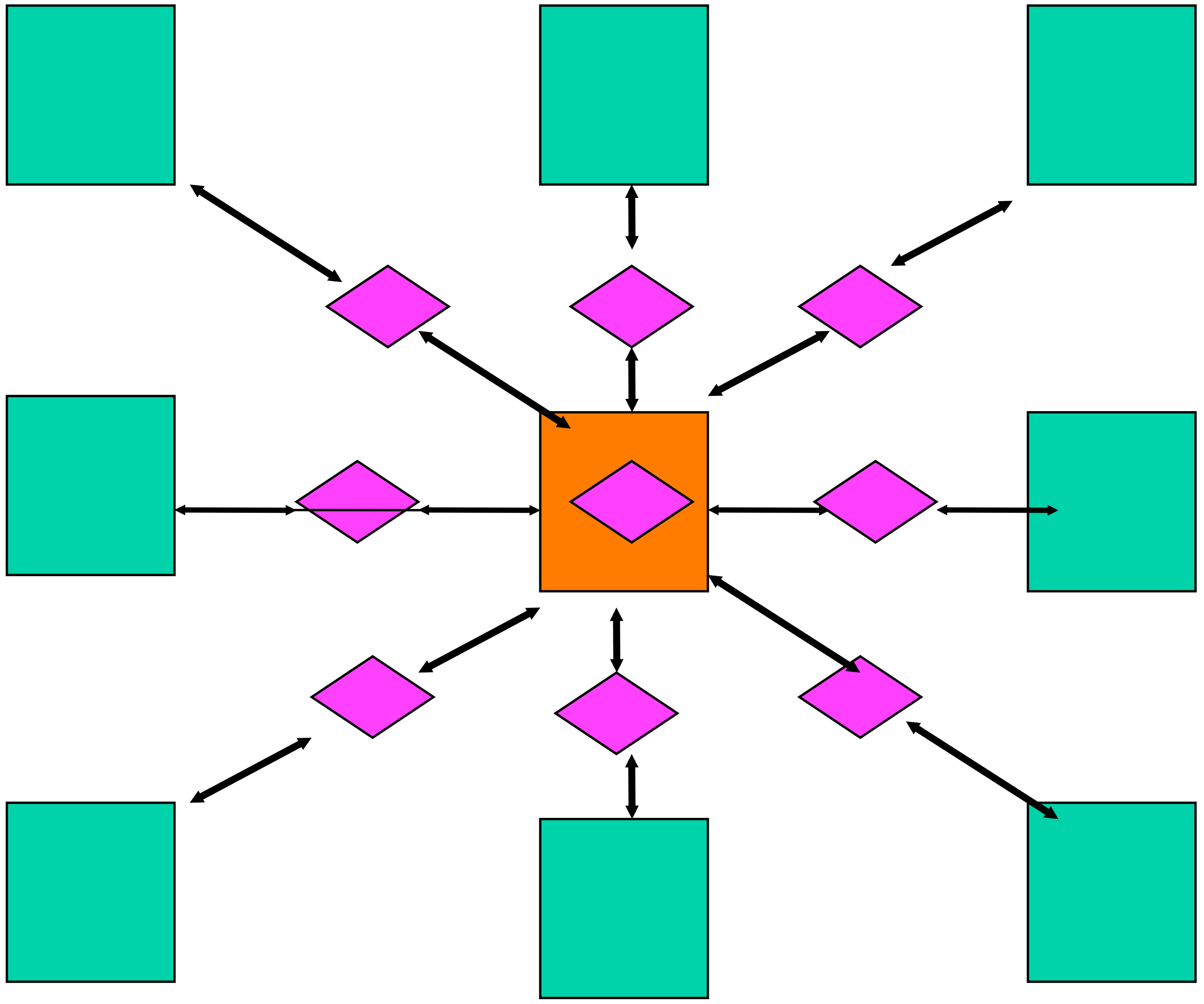


$$\underbrace{\sum_{dihedrals} k_i^{dihe}[1 + \cos(n_i\phi_i + \delta_i)]}_{U_{dihedral}} +$$

$$\underbrace{\sum_i \sum_{j \neq i} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] + \sum_i \sum_{j \neq i} \frac{q_i q_j}{\epsilon r_{ij}}}_{U_{nonbond}}$$

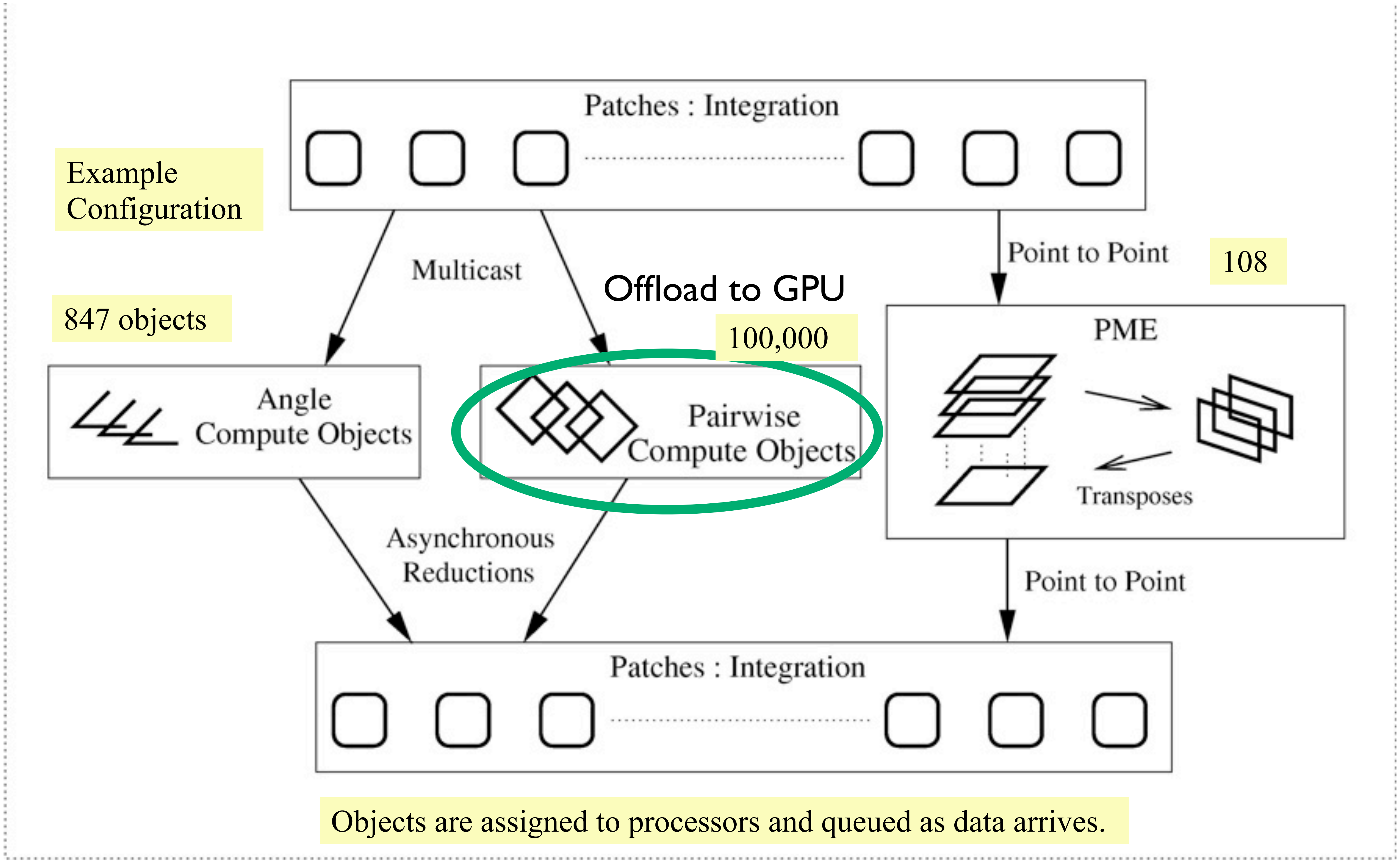
NAMD Hybrid Decomposition

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

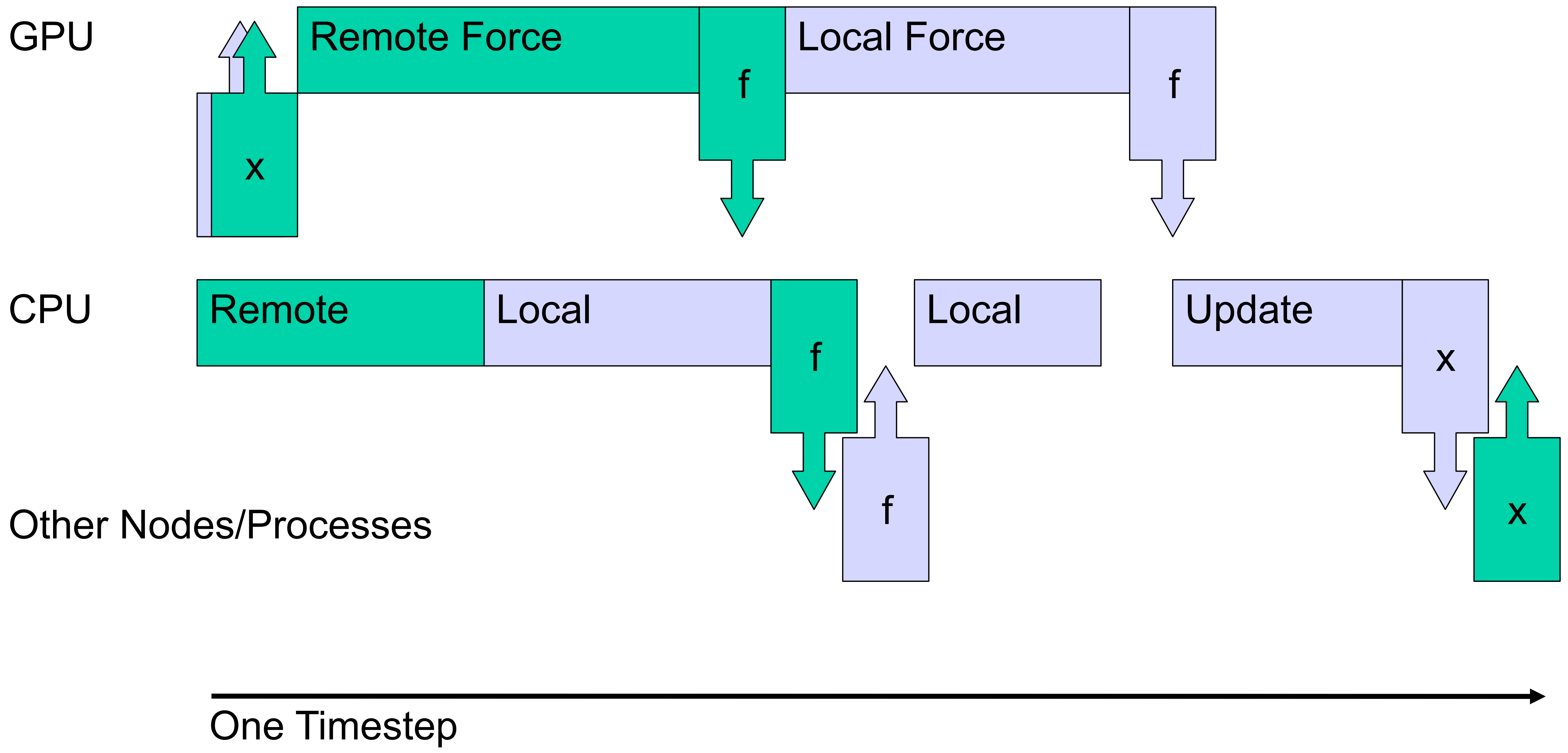


- Spatially decompose data and communication
- Separate but related work decomposition
- "Compute objects" facilitate iterative, measurement-based load balancing system

NAMD Overlapping Execution

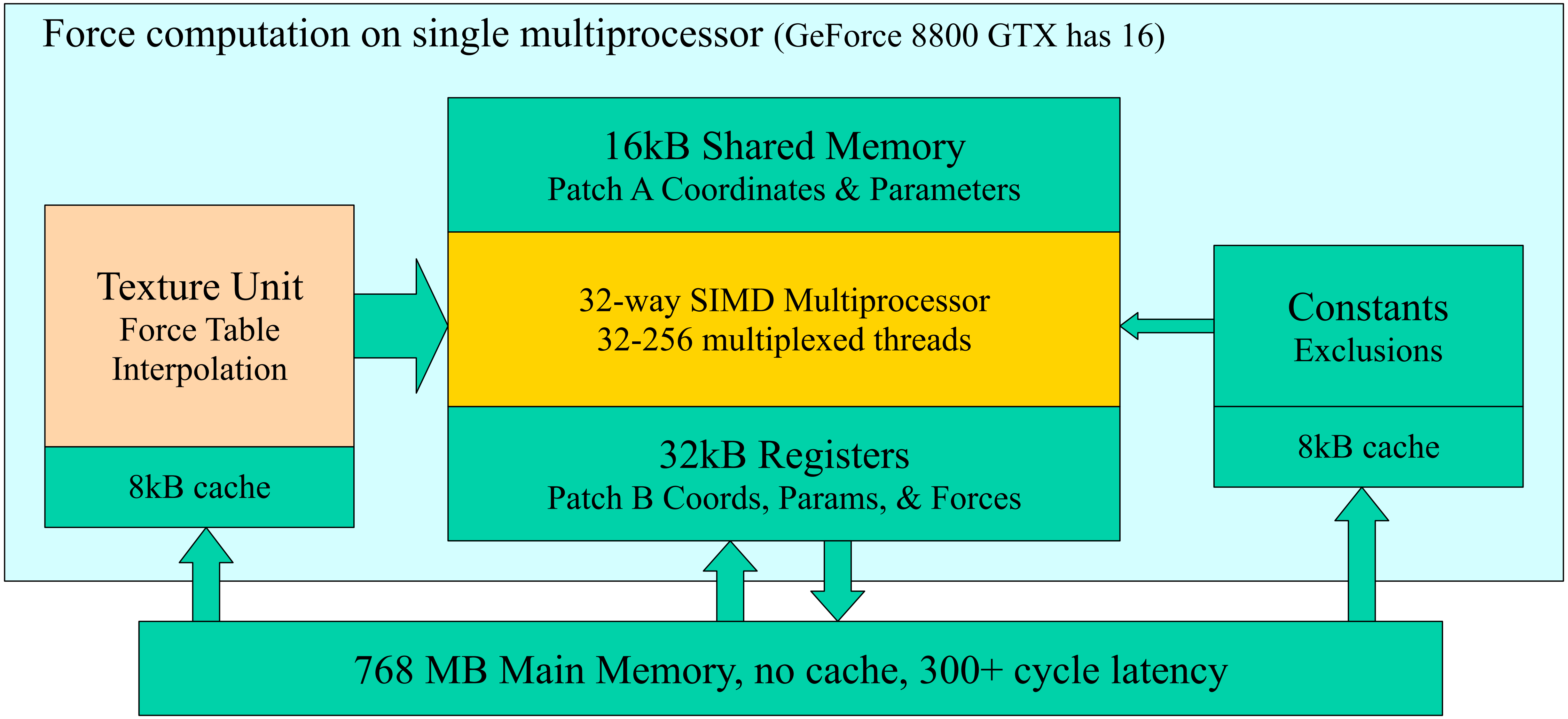


Overlapping GPU and CPU with Communication



Nonbonded Forces with CUDA

- Decompose work into pairs of patches, identical to NAMD structure
- Each thread block is assigned to a pair of patches (replacing a nonbonded compute object).


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

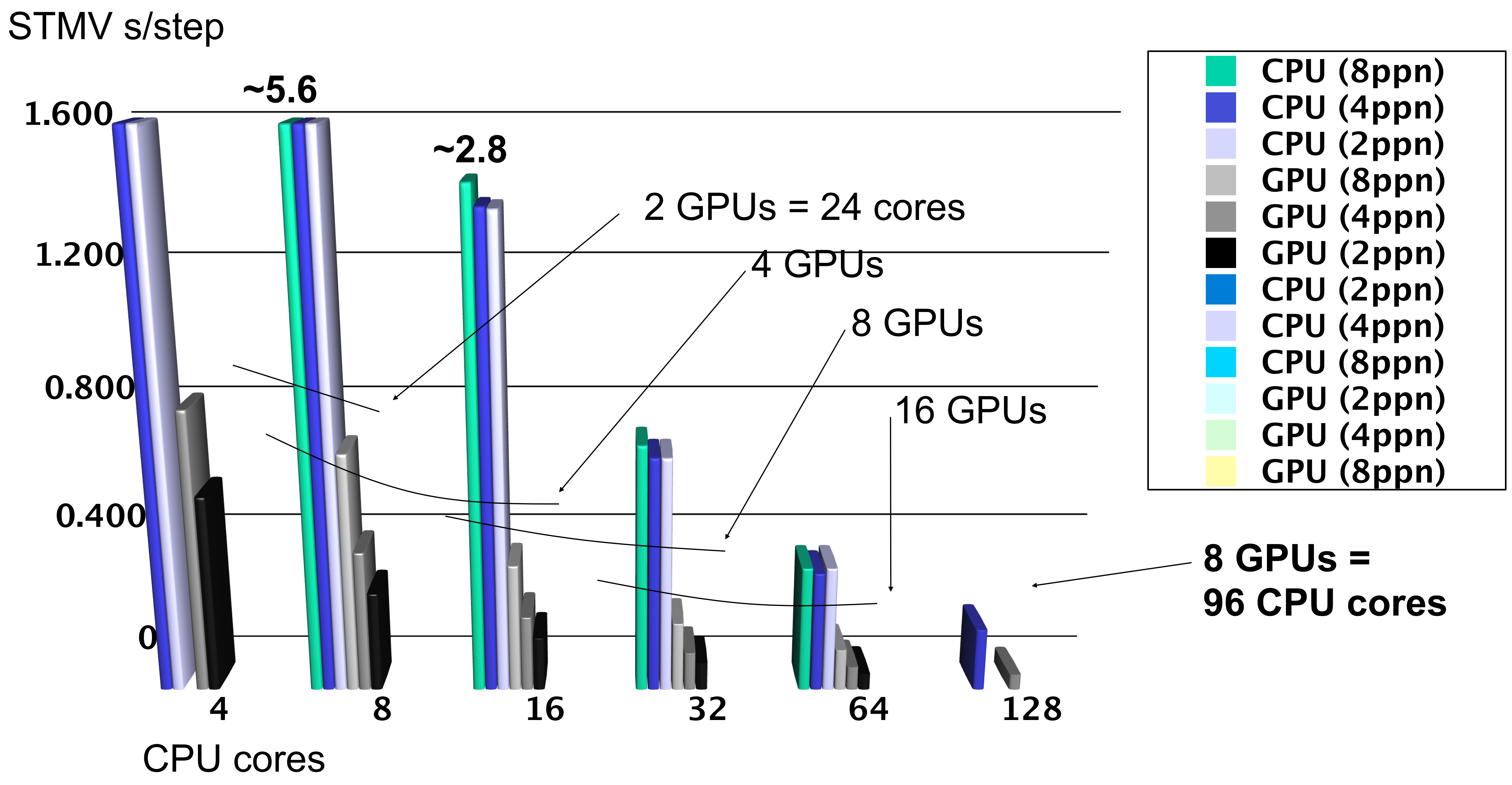
Stone, et al., *J. Comp. Chem.* **28**:2618-2640, 2007

New NCSA "8+2" Lincoln Cluster

- CPU: 2 Intel E5410 Quad-Core 2.33 GHz
- GPU: 2 NVIDIA C1060
  - Actually S1070 shared by two nodes
- How to share a GPU among 4 CPU cores?
  - Send all GPU work to one process?
  - Coordinate via messages to avoid conflict?
  - Or just hope for the best?

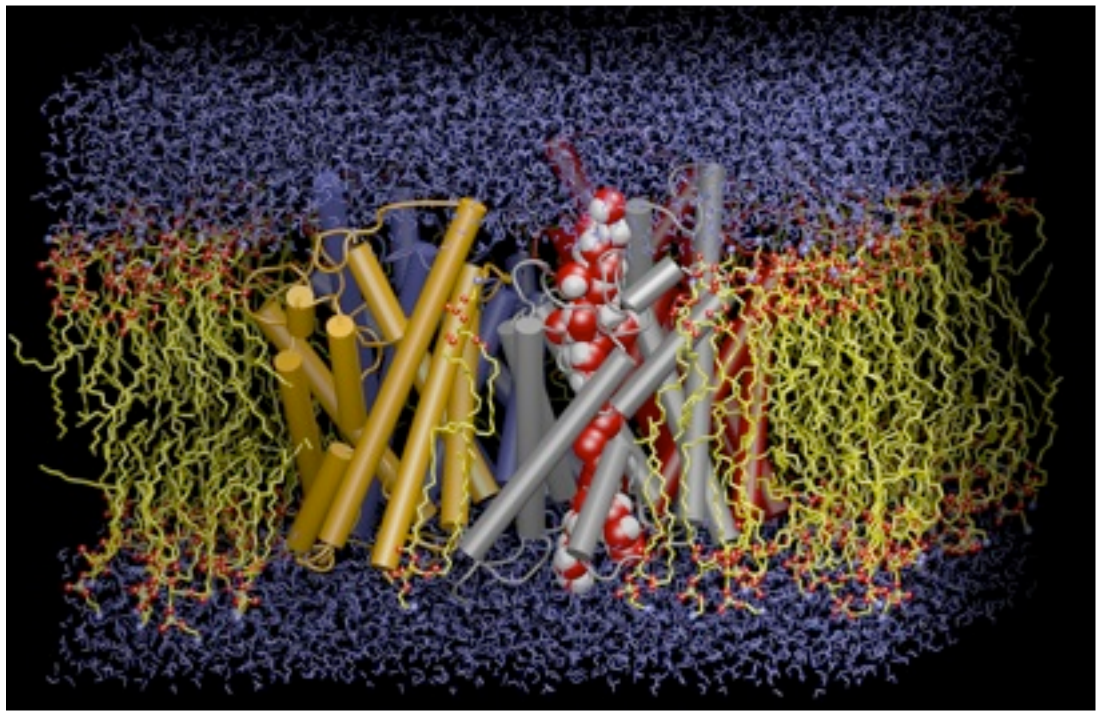
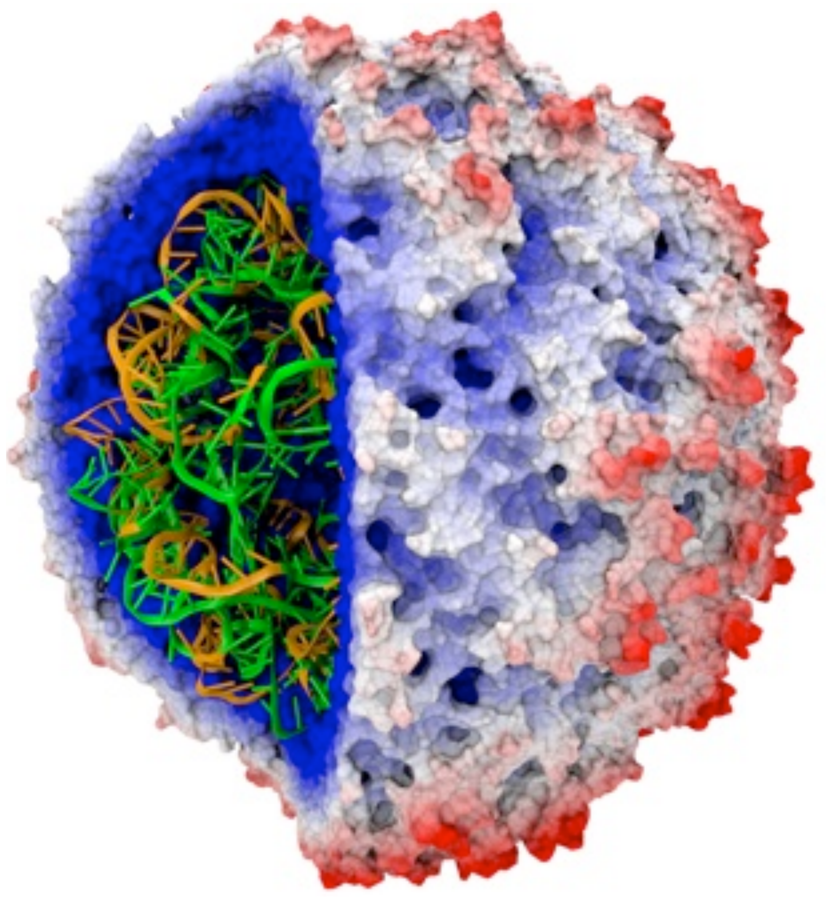
NCSA Lincoln Cluster Performance

(8 cores and 2 GPUs per node)



VMD — "Visual Molecular Dynamics"

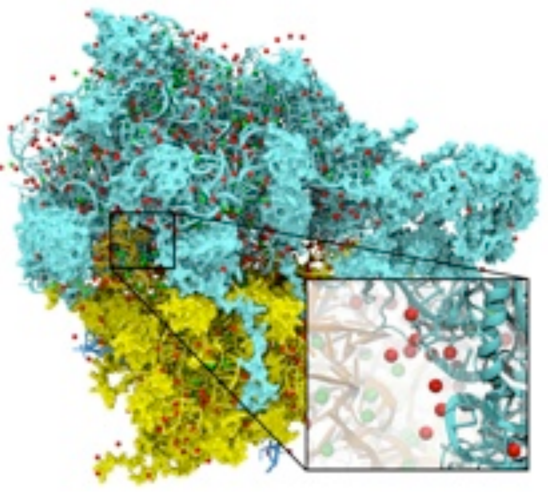
- Visualization and analysis of molecular dynamics simulations, sequence data, volumetric data, quantum chemistry simulations, particle systems, ...
- User extensible with scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/

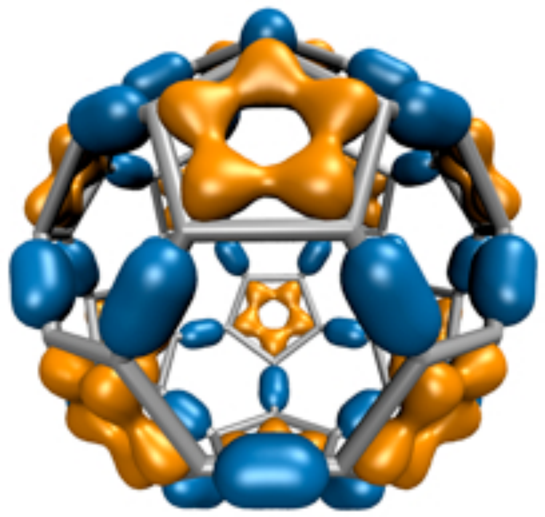
Range of VMD Usage Scenarios

- Users run VMD on a diverse range of hardware: laptops, desktops, clusters, and supercomputers
- Typically used as a desktop application for interactive 3D molecular graphics and analysis
- Can also be run in pure text mode for numerically intensive analysis tasks, batch mode movie rendering, etc.
- GPU acceleration provides an opportunity to make some **slow, or batch,** calculations capable of being run **interactively, or on-demand...**

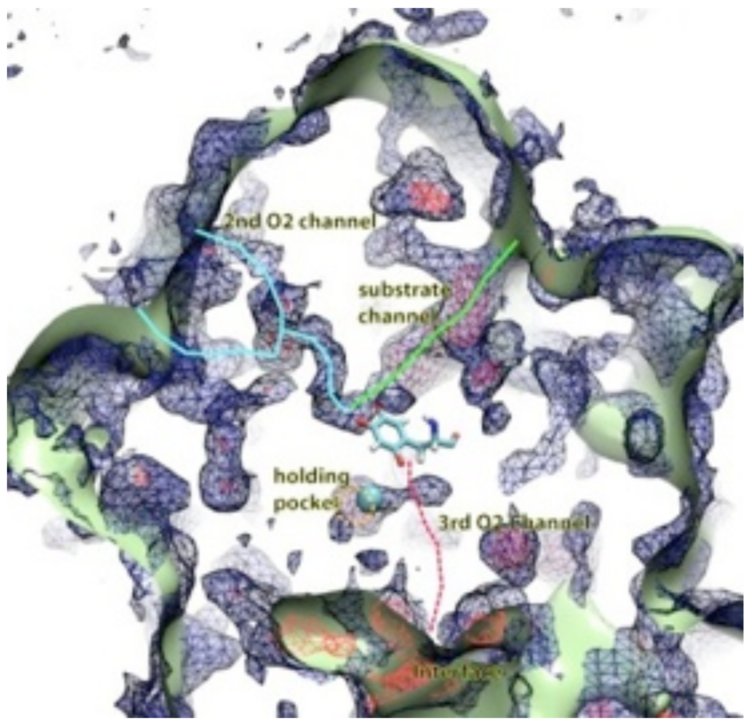
CUDA Acceleration in VMD



Electrostatic field calculation, ion placement: factor of 20x to 44x faster

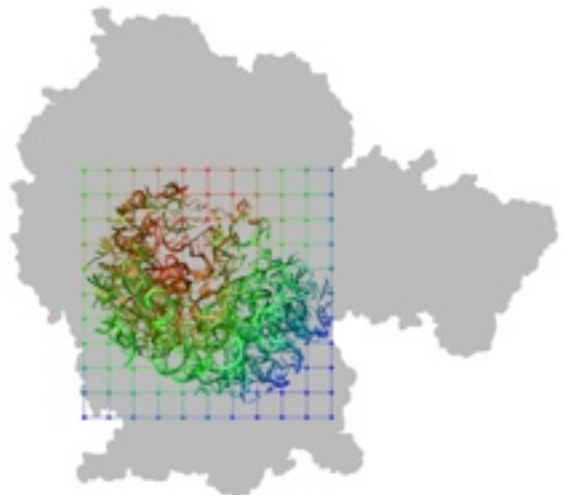


Molecular orbital calculation and display: factor of 120x faster



Imaging of gas migration pathways in proteins with implicit ligand sampling:

factor of 20x to 30x faster

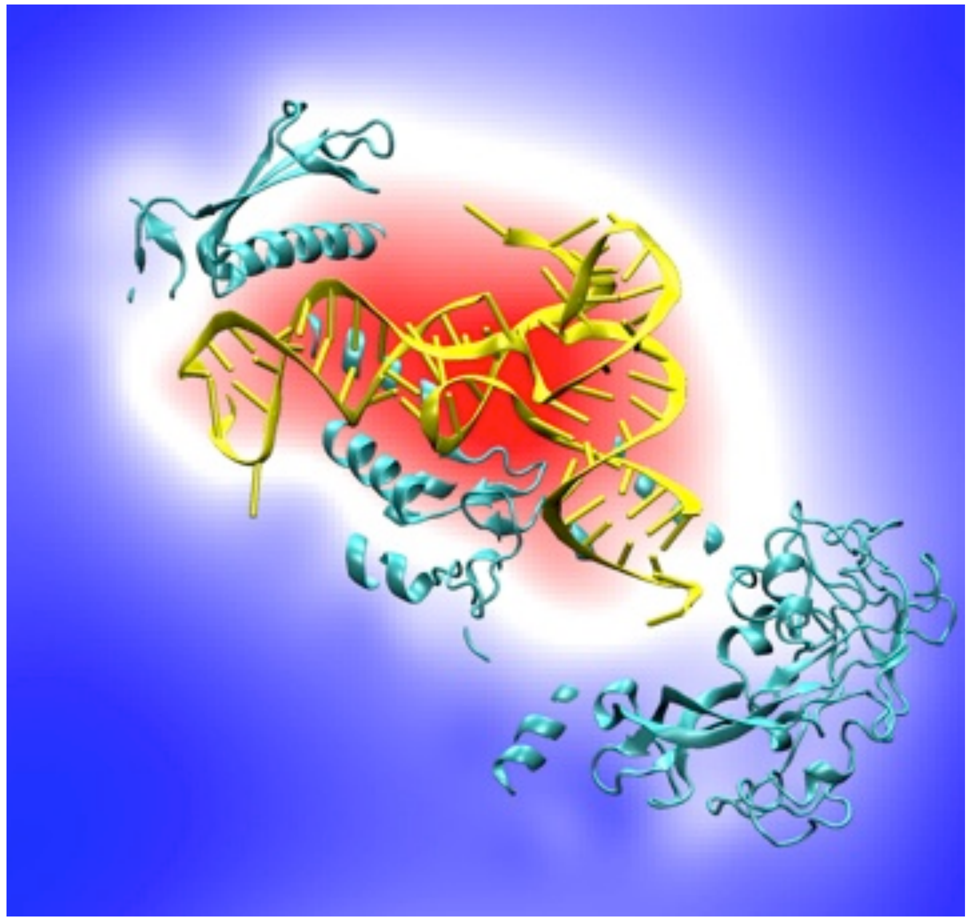


Electrostatic Potential Maps

- Electrostatic potentials evaluated on 3D lattice:

$$V_i = \sum_j \frac{q_j}{4\pi\epsilon_0 |\mathbf{r}_j - \mathbf{r}_i|}$$

- Applications include:
  - Ion placement for structure building
  - Time-averaged potentials for simulation
  - Visualization and analysis



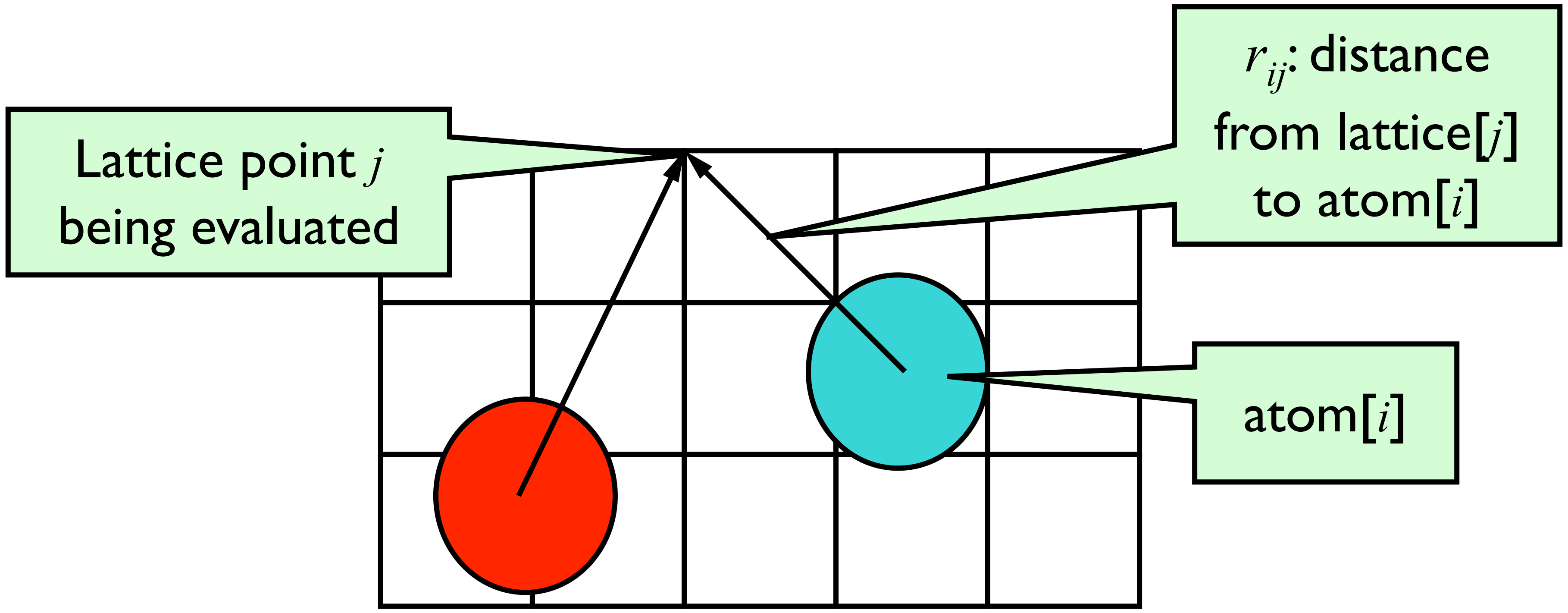
Isoleucine tRNA synthetase

Direct Coulomb Summation

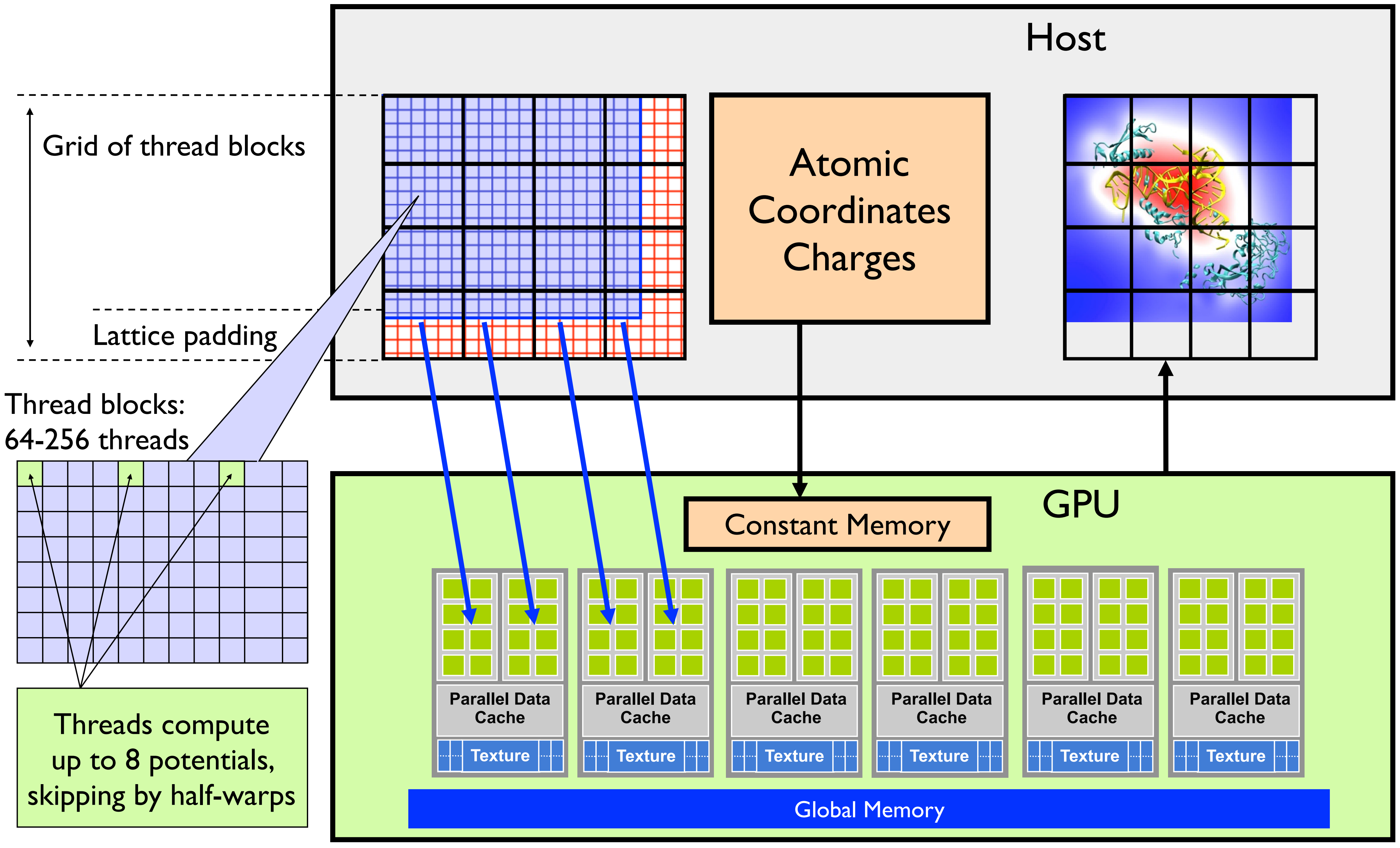
(naïve approach)

- Each lattice point accumulates electrostatic potential contribution from all atoms:

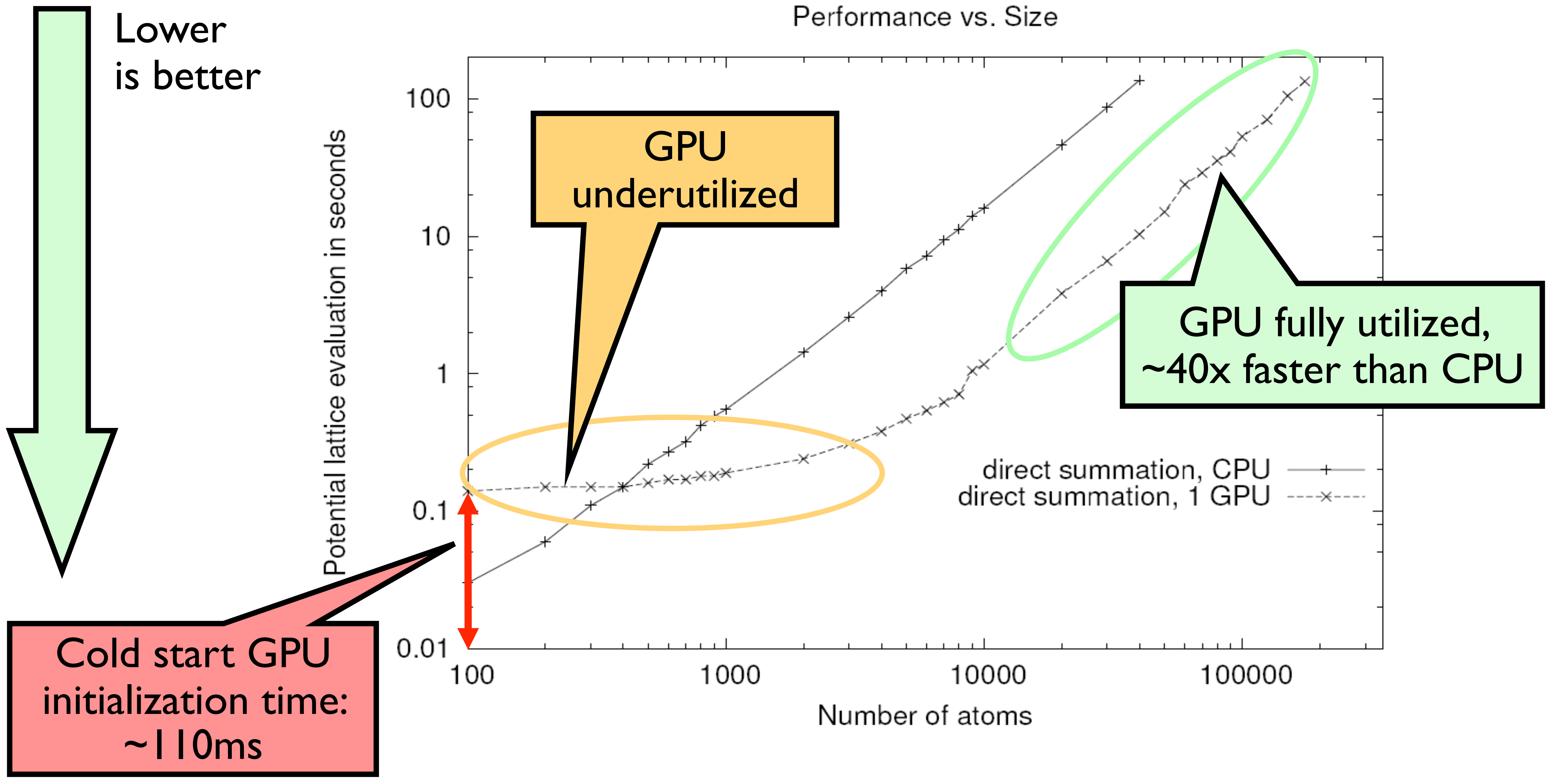
  potential[j] += charge[i] / r_{ij}


Lattice point j being evaluated
r_{ij}: distance from lattice[j] to atom[i]
atom[i]

Direct Coulomb Summation on GPU

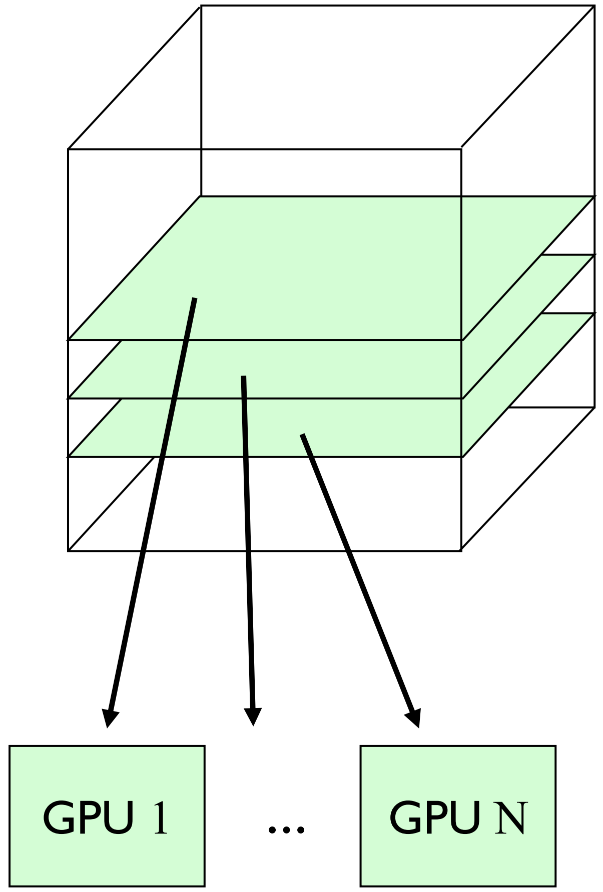

Host
Grid of thread blocks
Lattice padding
Thread blocks: 64-256 threads
Atomic Coordinates Charges
Threads compute up to 8 potentials, skipping by half-warps
GPU
Constant Memory
Parallel Data Cache
Texture
Global Memory

Direct Coulomb Summation Performance



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.

Using Multiple GPUs for Direct Coulomb Summation



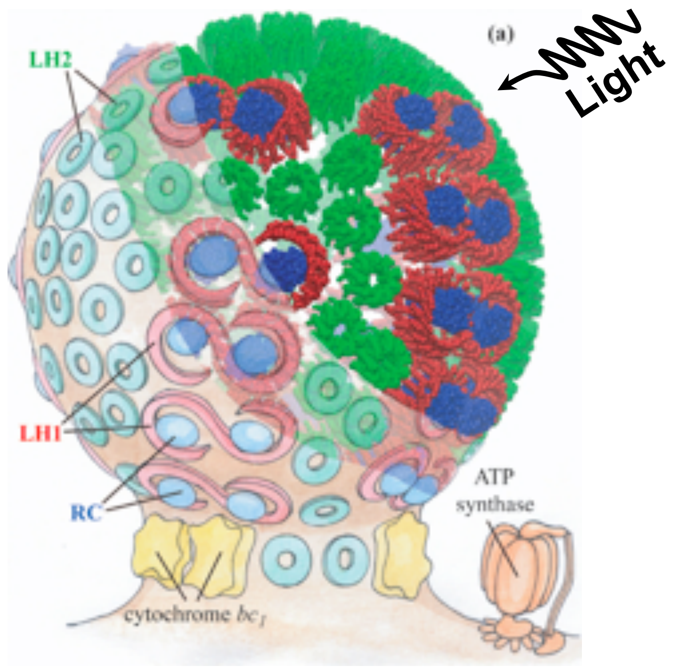
NCSA GPU Cluster
http://www.ncsa.uiuc.edu/Projects/GPUcluster/

	Evals/sec	TFLOPS	Speedup*
4-GPU (2 Quadroplex) Opteron node at NCSA	157 billion	1.16	176
4-GPU GTX 280 (GT200)	241 billion	1.78	271

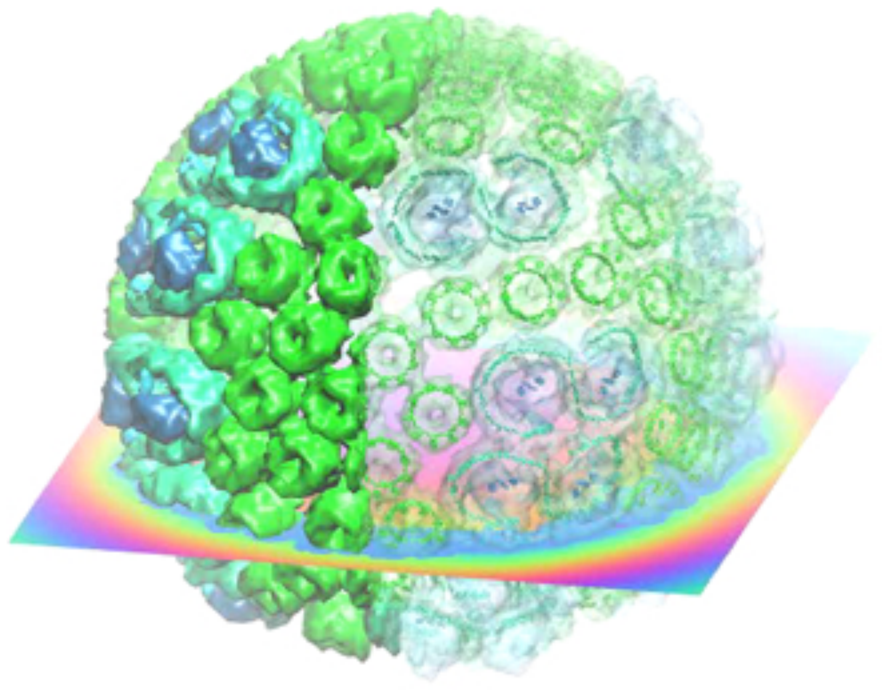
*Speedups relative to Intel QX6700 CPU core w/ SSE

Photobiology of Vision and Photosynthesis

Investigations of the chromatophore, a photosynthetic organelle



Partial model: ~10M atoms



Electrostatics needed to build full structural model, place ions, study macroscopic properties

Electrostatic field of chromatophore model from **multilevel summation method**: computed with 3 GPUs (G80) in ~90 seconds, 46x faster than single CPU core

Full chromatophore model will permit structural, chemical and kinetic investigations at a structural systems biology level

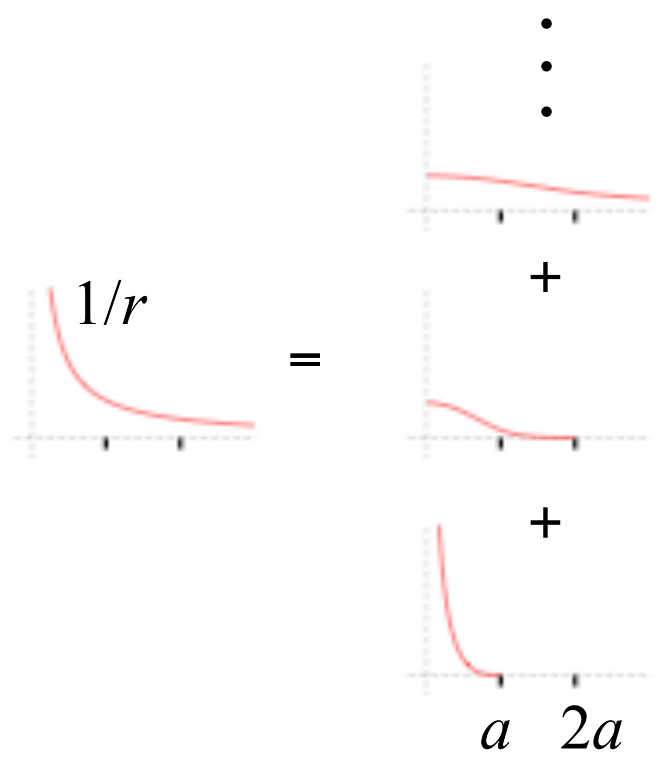
Multilevel Summation Method

- Approximates full electrostatic potential
- Calculates sum of smoothed pairwise potentials interpolated from a hierarchy of lattices
- Advantages over PME (particle-mesh Ewald) and/or FMM (fast multipole method):
  - Algorithm has linear time complexity
  - Permits non-periodic and periodic boundaries
  - Produces continuous forces for dynamics (advantage over FMM)
  - Avoids 3D FFTs for better parallel scaling (advantage over PME)
  - Spatial separation allows use of multiple time steps
  - Can be extended to other types of pairwise interactions

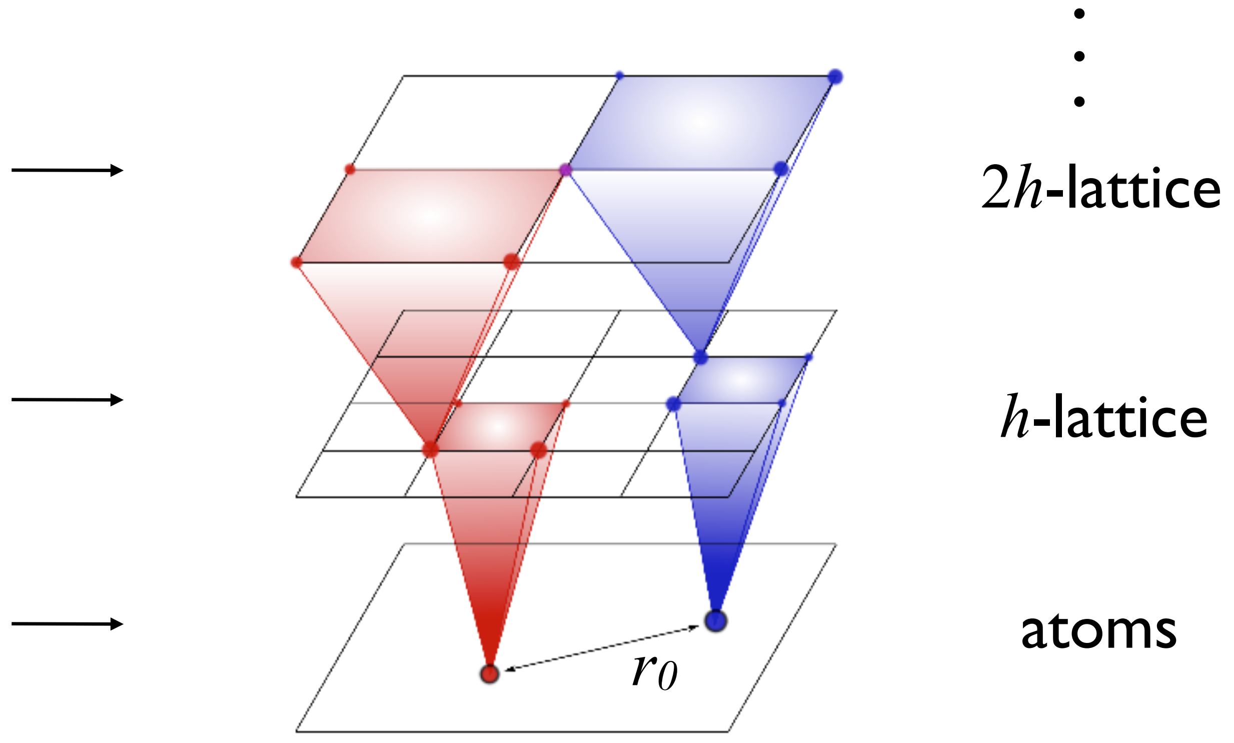
Multilevel Summation Main Ideas

- Split the $1/r$ potential into a short-range cutoff part plus smoothed parts that are successively more slowly varying. All but the top level potential are cut off.
- Smoothed potentials are interpolated from successively coarser lattices.
- Finest lattice spacing h and smallest cutoff distance a are doubled at each successive level.

Split the $1/r$ potential



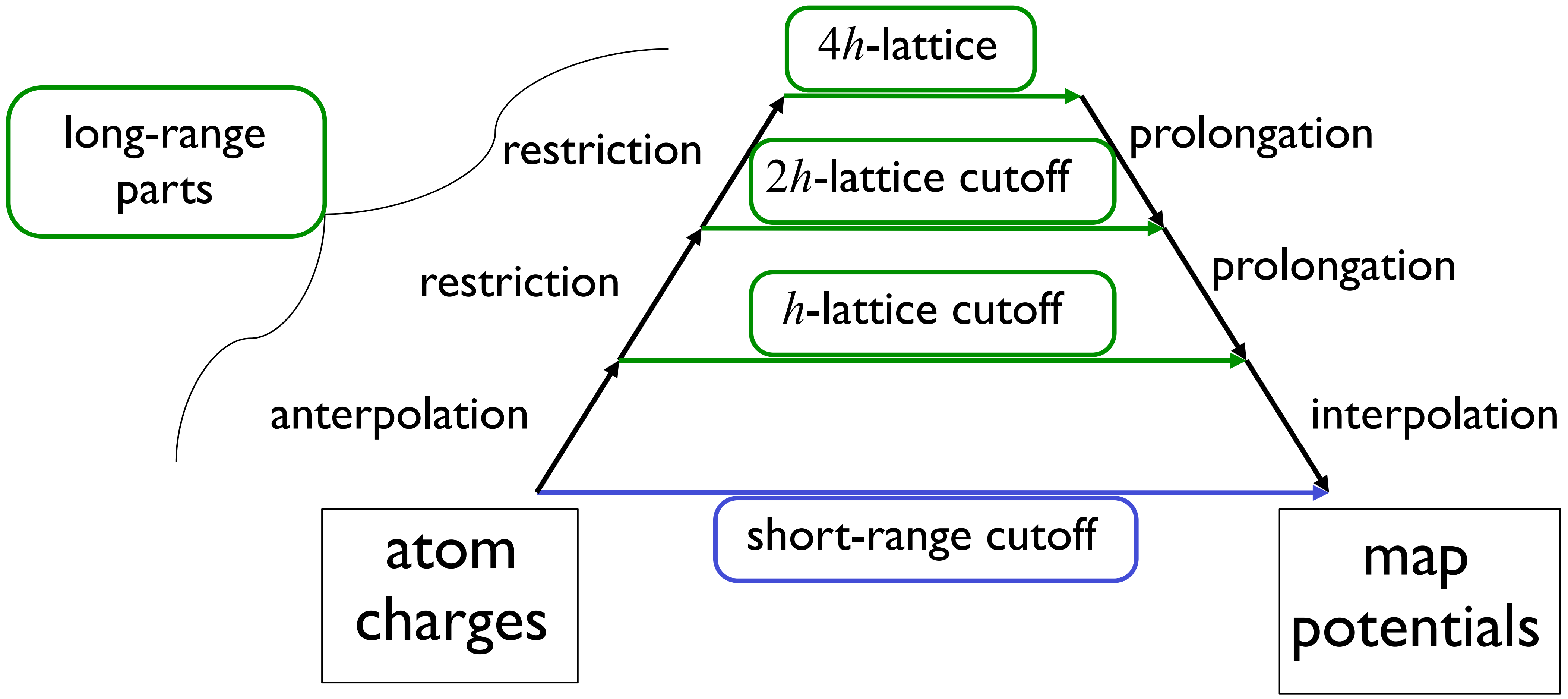
Interpolate the smoothed potentials



Multilevel Summation Calculation

map potential	=	**exact short-range interactions**	+	**interpolated long-range interactions**

Computational Steps

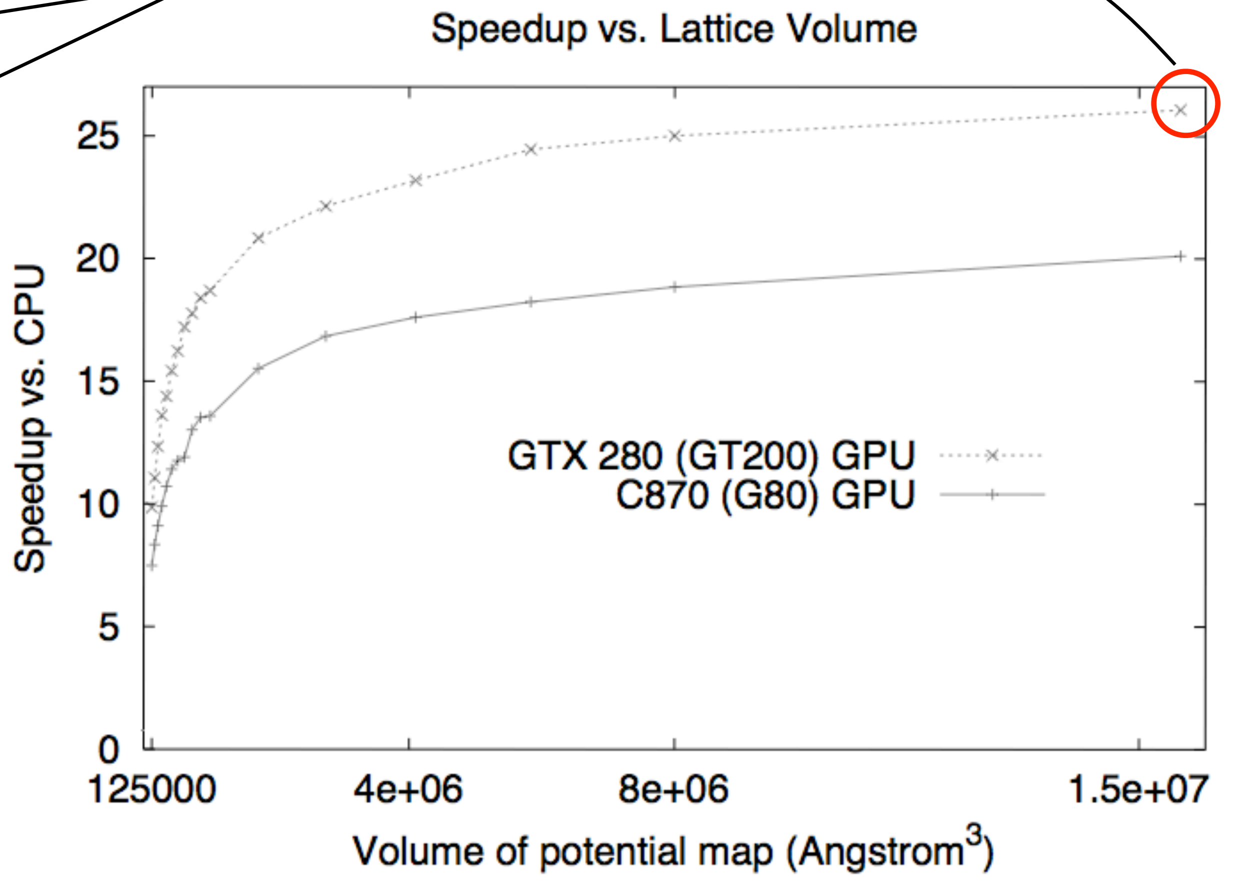


Multilevel Summation on the GPU

Accelerate **short-range cutoff** and **lattice cutoff** parts

Performance profile for 0.5 Å map of potential for 1.5 M atoms.
Hardware platform is Intel QX6700 CPU and NVIDIA GTX 280.

Computational steps	CPU (s)	w/ GPU (s)	Speedup
Short-range cutoff	480.07	14.87	32.3
Long-range anterpolation	0.18		
restriction	0.16		
lattice cutoff	49.47	1.36	36.4
prolongation	0.17		
interpolation	3.47		
Total	533.52	20.21	26.4



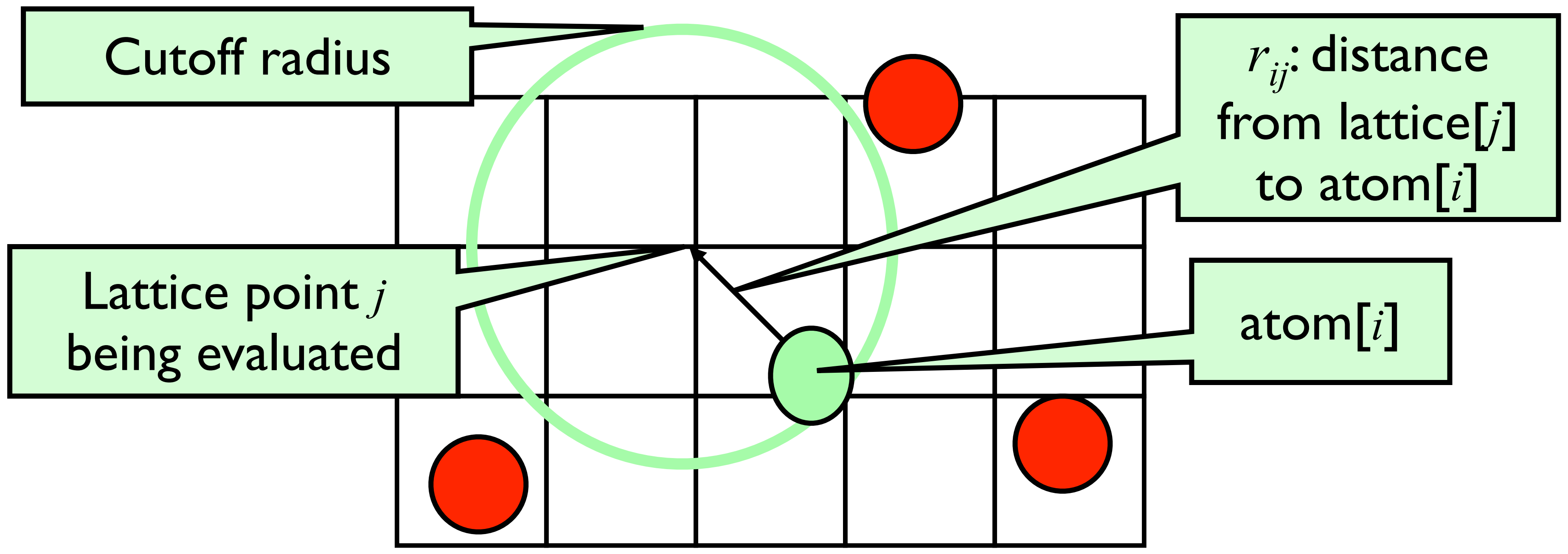
Multilevel summation of electrostatic potentials using graphics processing units.
D. Hardy, J. Stone, K. Schulten. *J. Parallel Computing*, 35:164-177, 2009.

Short-range Cutoff Summation

- Each lattice point accumulates electrostatic potential contribution from atoms within cutoff distance:

```
if ($r_{ij}$ < cutoff)
    potential[$j$] += (charge[$i$] / $r_{ij}$) * $s(r_{ij})$
```

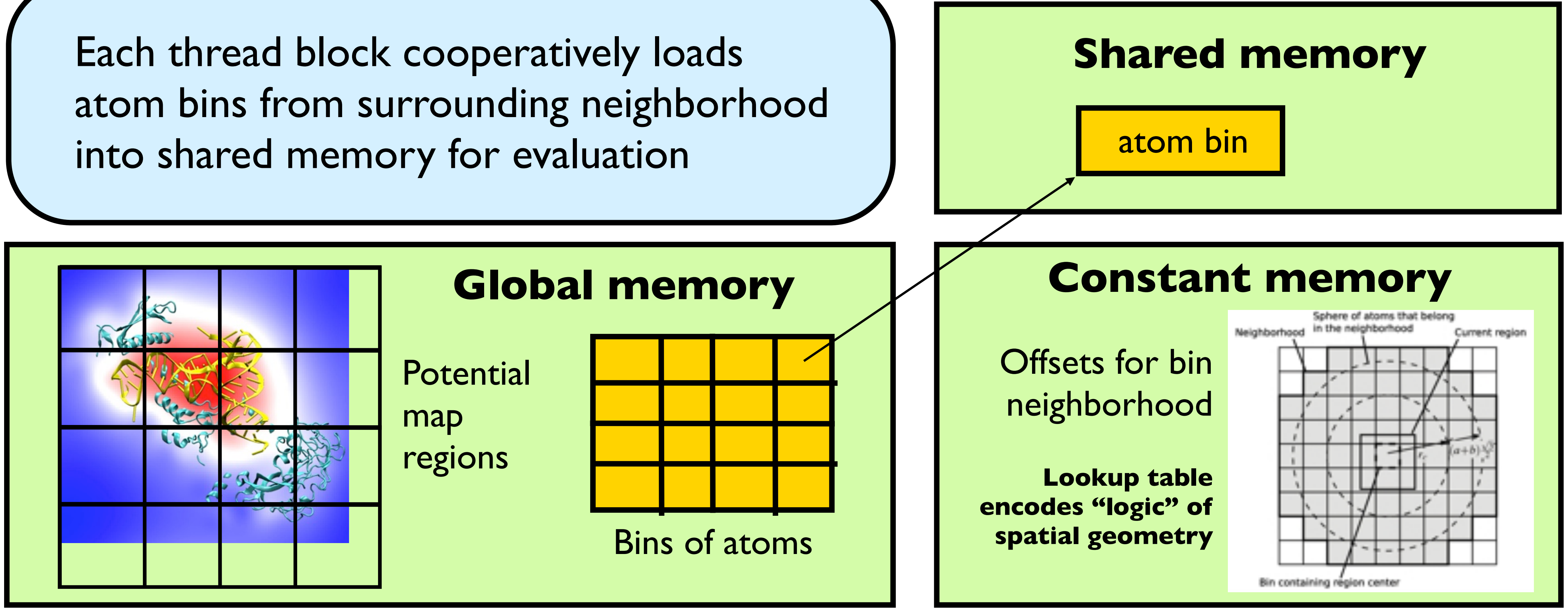
- Smoothing function $s(r)$ is algorithm dependent


Cutoff radius
r_{ij}: distance from lattice[j] to atom[i]
Lattice point j being evaluated
atom[i]

Cutoff Summation on the GPU

- Atoms are spatially hashed into fixed-size bins (8 deep, stored x/y/z/q)
- CPU handles overflowed bins, so GPU kernel can be aggressive (choosing 4Å bin length works well in practice)
- GPU thread block calculates corresponding region of potential map
- Solve costly bin/region neighbor checks with lookup table of offsets

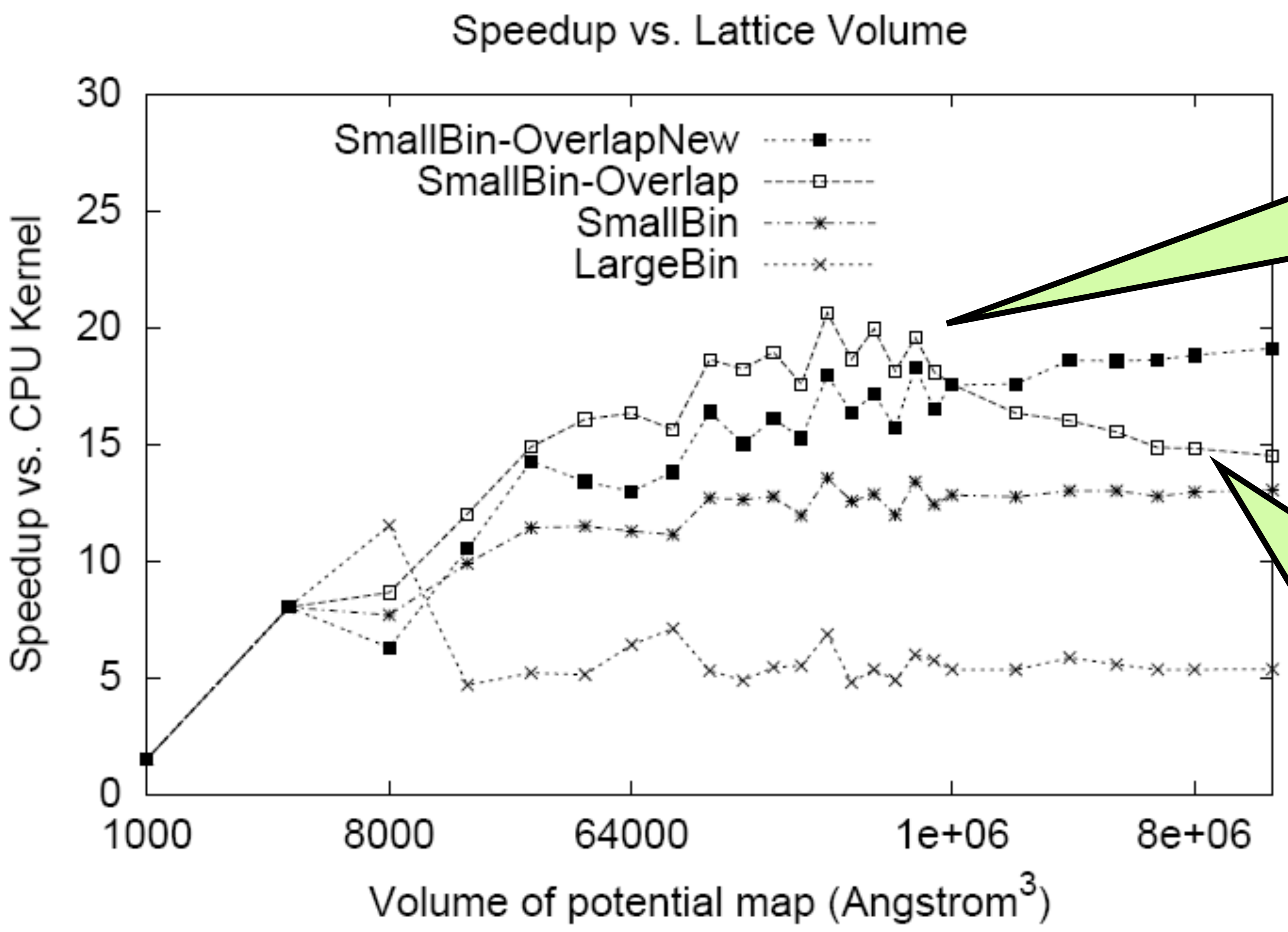
Each thread block cooperatively loads atom bins from surrounding neighborhood into shared memory for evaluation


Shared memory
atom bin
Global memory
Potential map regions
Bins of atoms
Constant memory
Offsets for bin neighborhood
Lookup table encodes "logic" of spatial geometry
Neighborhood
Sphere of atoms that belong in the neighborhood
Current region
Bin containing region center

Using 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 to attain good fraction of peak performance

Cutoff Summation Performance



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

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

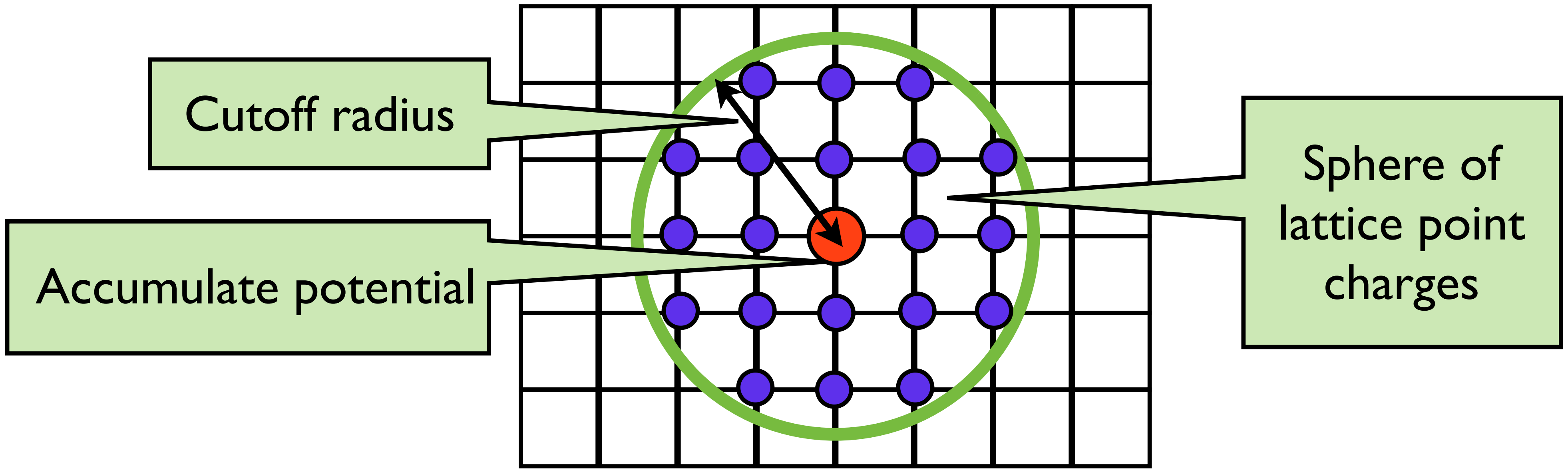
GPU acceleration of cutoff pair potentials for molecular modeling applications. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.

Cutoff Summation Observations

- Use of CPU to handle overflowed bins is very effective, overlaps well with GPU work
- Caveat when using streaming API to invoke GPU kernel: avoid overfilling stream queue with work so as not to trigger blocking behavior (improved in current drivers)
- Increasing floating point precision with compensated summation (all GPUs) or double-precision (GT200 only) for potential accumulation results in just ~10% performance penalty versus pure single-precision arithmetic

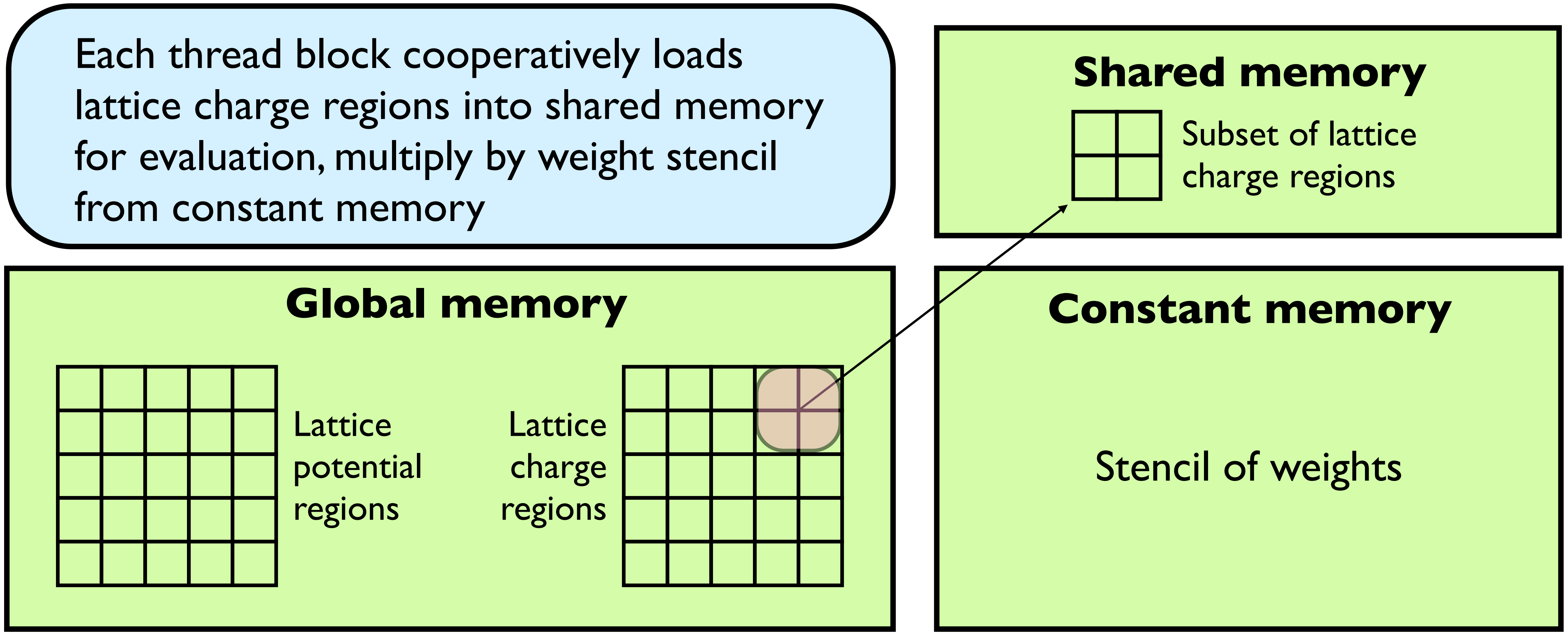
Lattice Cutoff Summation

- Each lattice point accumulates electrostatic potential contribution from all lattice point charges within cutoff distance
- Relative distances are the same between points on a uniform lattice, multiplication by a precomputed stencil of "weights"
- Weights at each level are identical up to a scaling factor (due to choice of splitting and doubling of lattice spacing and cutoff)
- Calculate as 3D convolution of sub-cube of lattice point charges with enclosing cube of weights, size determined by cutoff *a* and grid spacing *h*
  - Cube length is $2\times\lceil 2a/h \rceil - 1$; we use a=12Å and h=2Å for cube stencil size 23x23x23



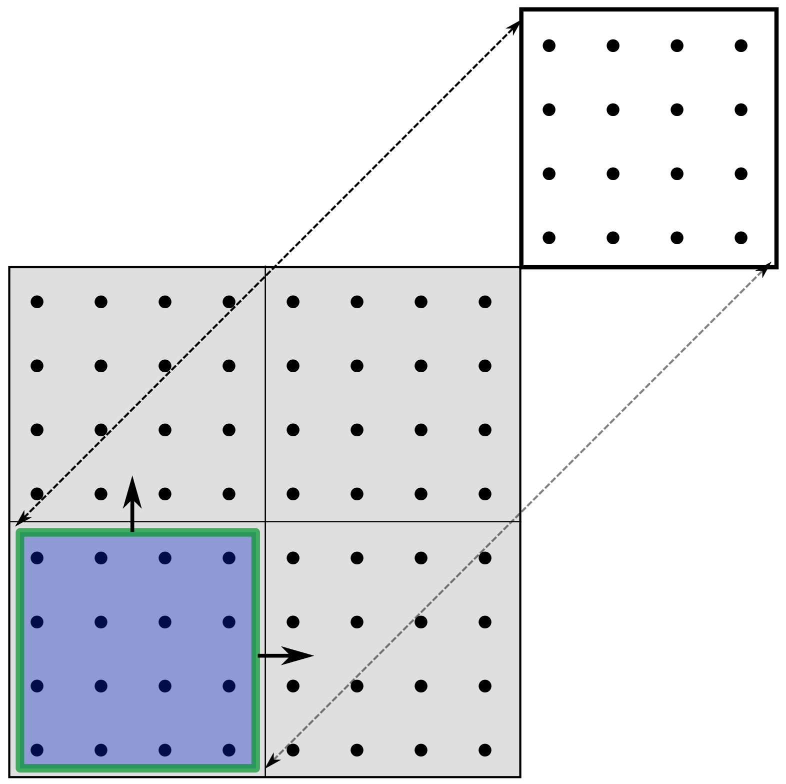
Lattice Cutoff Summation on GPU

- Store stencil of weights in constant memory (padded up to next multiple of 4)
- Assign GPU thread block to calculate 4x4x4 region of lattice potentials, stored contiguously
- Regions stored in flattened array, each level padded with zero charge region, levels stored contiguously, constant memory stores mapping of 3D region level sets into flattened array
- Load nearby regions of lattice point charges into shared memory (analogous to loading atom bins for short-range cutoff)
- Evaluate all lattice levels concurrently, scaling by level factor (keeps GPU from running out of work at upper lattice levels)


Each thread block cooperatively loads lattice charge regions into shared memory for evaluation, multiply by weight stencil from constant memory
Shared memory
Subset of lattice charge regions
Global memory
Lattice potential regions
Lattice charge regions
Constant memory
Stencil of weights

Evaluation Using Sliding Window

- Every thread in block needs to simultaneously read and use the same weight from constant memory
- Read into shared memory an 8x8x8 block (8 regions) of lattice point charges
- Slide a window of size 4x4x4 by 4 shifts along each dimension



Lessons Learned

- GPU algorithms need fine-grained parallelism and sufficient work to fully utilize the hardware
- Efficient use of GPU multiple memory systems and latency hiding is essential for good performance
- CPU can be used to "regularize" computation for GPU, handling exceptional cases for overall better 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 improve overall numerical precision at low cost to performance

Related Publications

- Multilevel summation of electrostatic potentials using graphics processing units. D. Hardy, J. Stone, K. Schulten. *J. Parallel Computing*, 35:164-177, 2009.
- 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.
- 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.
- Multilevel Summation for the Fast Evaluation of Forces for the Simulation of Biomolecules. David J. Hardy. Ph.D. thesis, University of Illinois at Urbana-Champaign, 2006.
- Multiple grid methods for classical molecular dynamics. R. Skeel, I. Tezcan, D. Hardy. *J. Comp. Chem.*, 23:673-684, 2002.

See GPU development at http://www.ks.uiuc.edu/Research/gpu/

Acknowledgments

- Prof. Klaus Schulten, John Stone, and Jim Phillips of the Theoretical and Computational Biophysics Group at the Beckman Institute, University of Illinois at Urbana-Champaign
- Prof. Wen-mei Hwu and Chris Rodrigues of the IMPACT group, University of Illinois at Urbana-Champaign
- Prof. Robert Skeel, Purdue University
- NVIDIA Center of Excellence, University of Illinois at Urbana-Champaign
- NCSA Innovative Systems Lab
- The CUDA team at NVIDIA
- NIH Grant P41-RR05969

Looking Forward to NVIDIA Fermi

- New architecture "Fermi" expected in 2010, features include:
  - ECC memory (GDDR5)
  - L1 (64 KB, together with shared memory) and L2 (768 KB) caches
  - 512 cores
  - Improved double precision performance (single to double ratio 1:2)
  - Improved floating point accuracy
  - Faster thread context switching (factor of 10)
  - Concurrent kernel execution
  - Second DMA engine to overlap two memory transfers
  - Extended C++ support (e.g. virtual functions, exception handling)
  - 40 nm process technology with 3 billion transistors
  - Similar power consumption to current models

(from HPCwire, "NVIDIA Takes GPU Computing to the Next Level," Setemper 29, 2009)