Multilevel Summation of Electrostatic Potentials Using GPUs

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/~dhardy/

Purdue University, September 9, 2009

GPU Computing

- Ubiquitous, commodity devices

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

- Massively parallel hardware, hundreds of processing units, throughput oriented architecture
- Incorporates fully programmable processors
- Supports standard integer and floating point types
- Programming tools 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

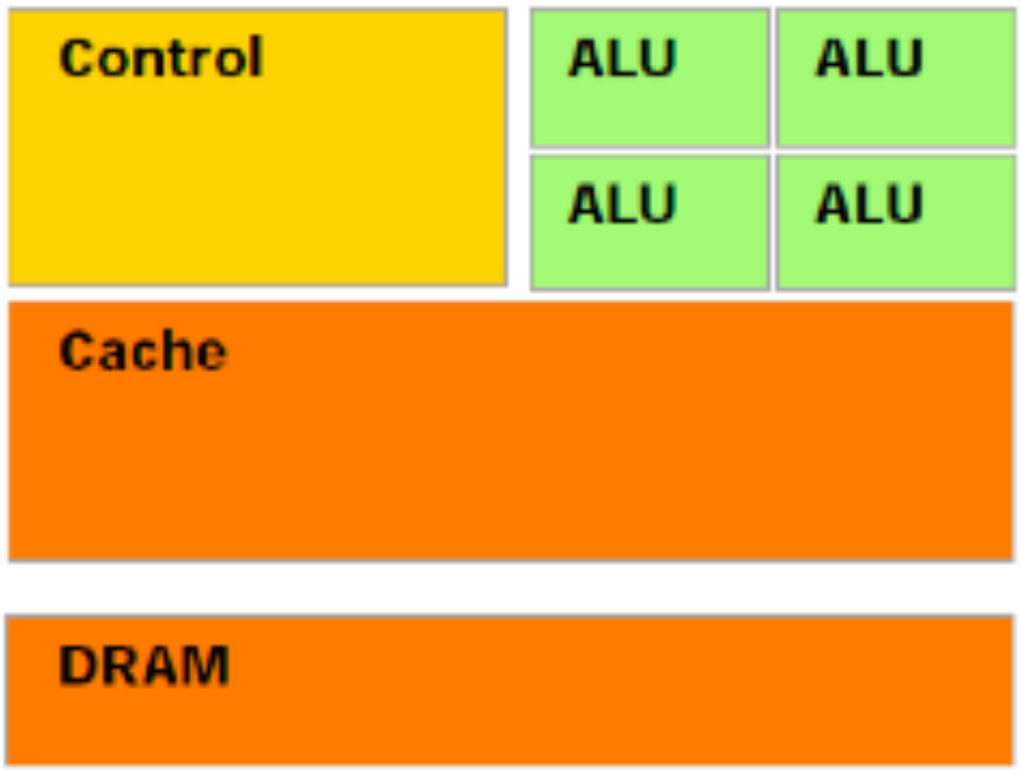
What speedups can GPUs achieve?

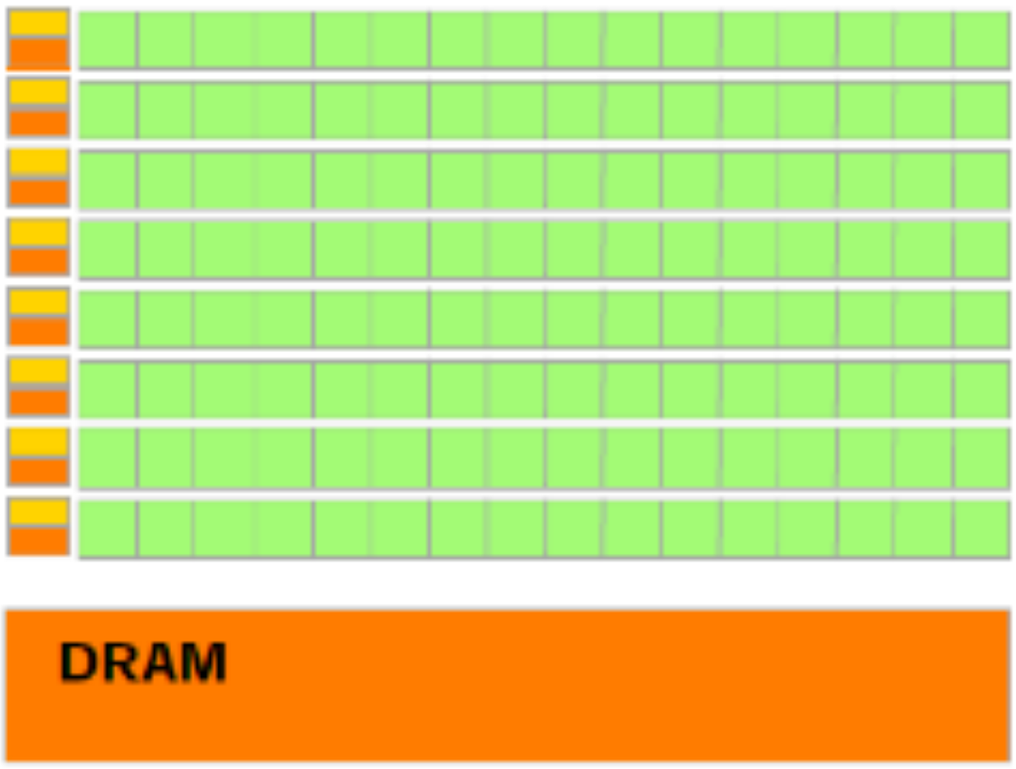
- Single-GPU speedups of 10x to 30x over one CPU core are common
- Best speedups of 100x or more attained for codes dominated by floating point arithmetic, especially for native GPU machine instructions, e.g. expf(), rsqrtf()
- Legacy codes employing "shallow" efforts at GPU acceleration do not exhibit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

Comparison of CPU and GPU Hardware Architecture

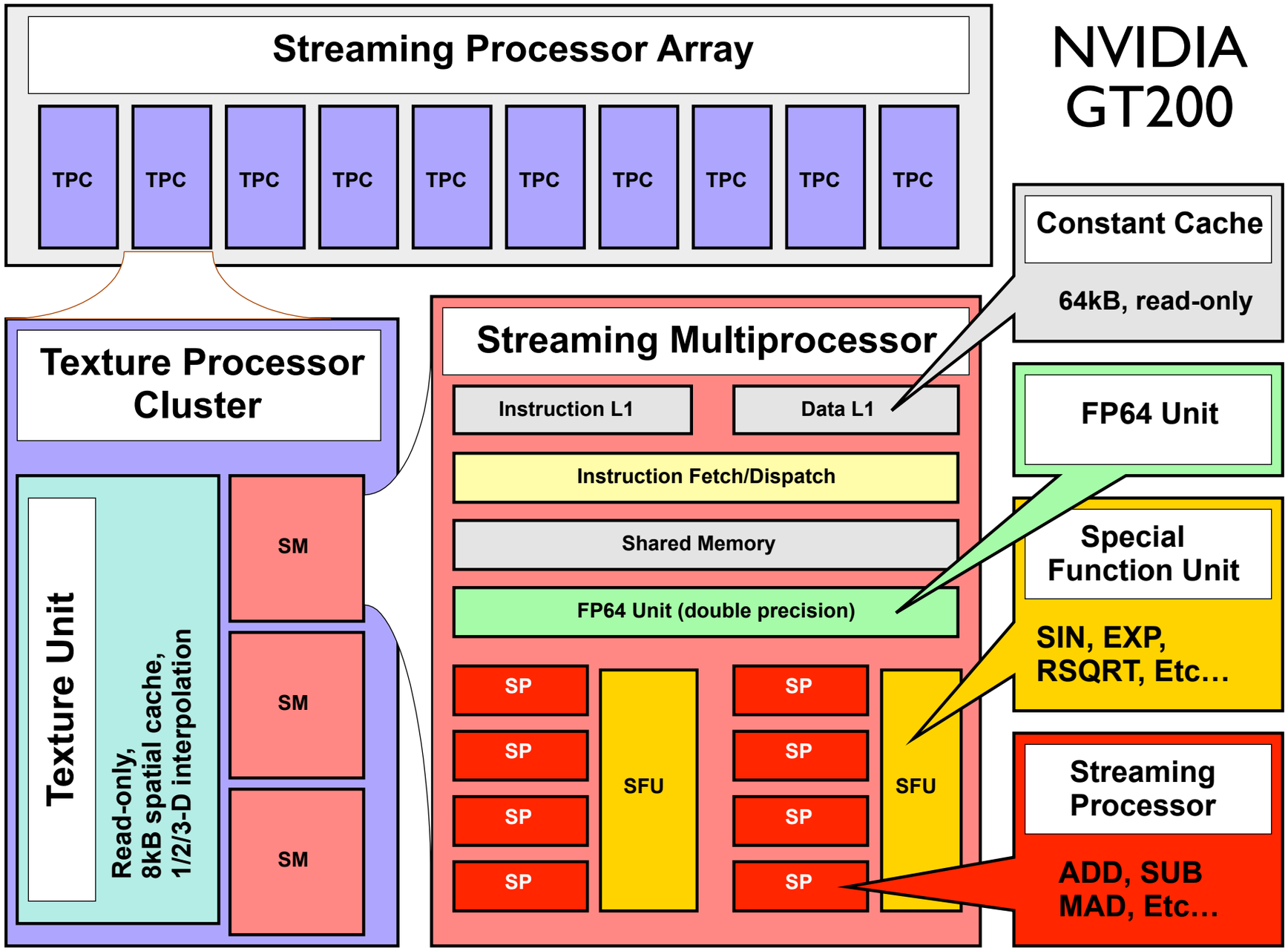
CPU: Cache heavy, focused on individual thread performance

GPU: ALU heavy, massively parallel, throughput oriented

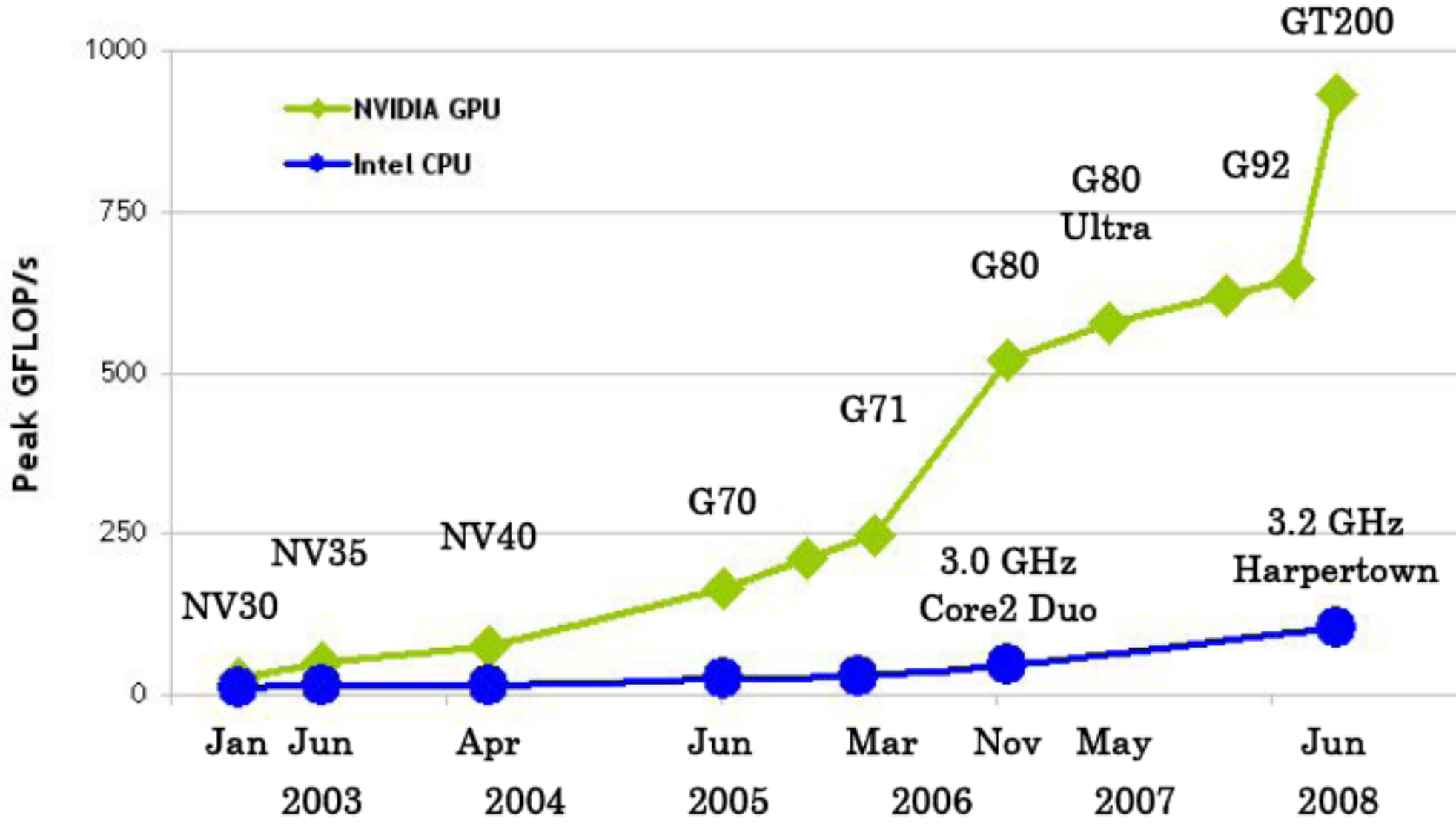




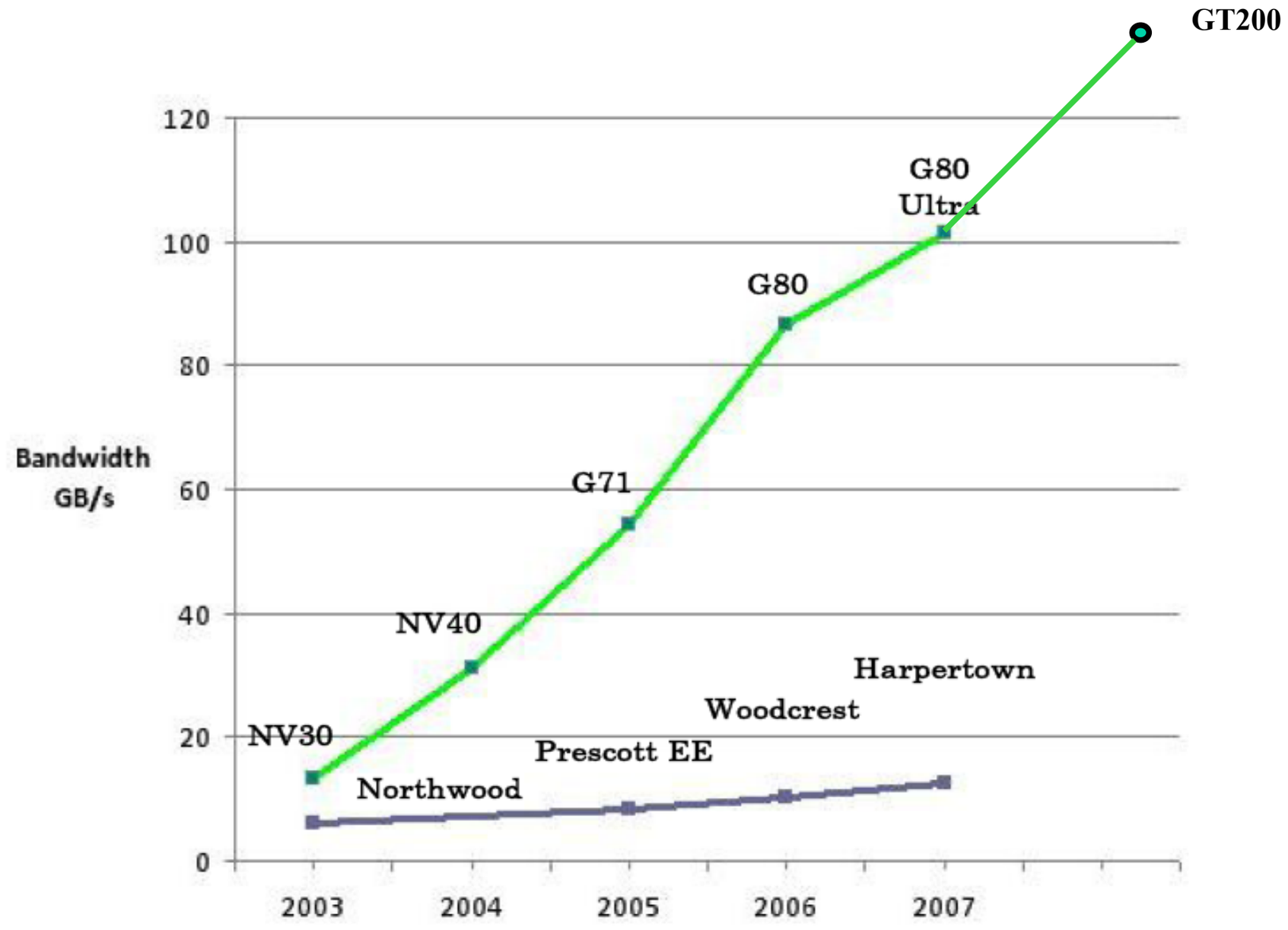
NVIDIA GPU Architecture



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



GPU Peak Memory Bandwidth: **Linear** Trend



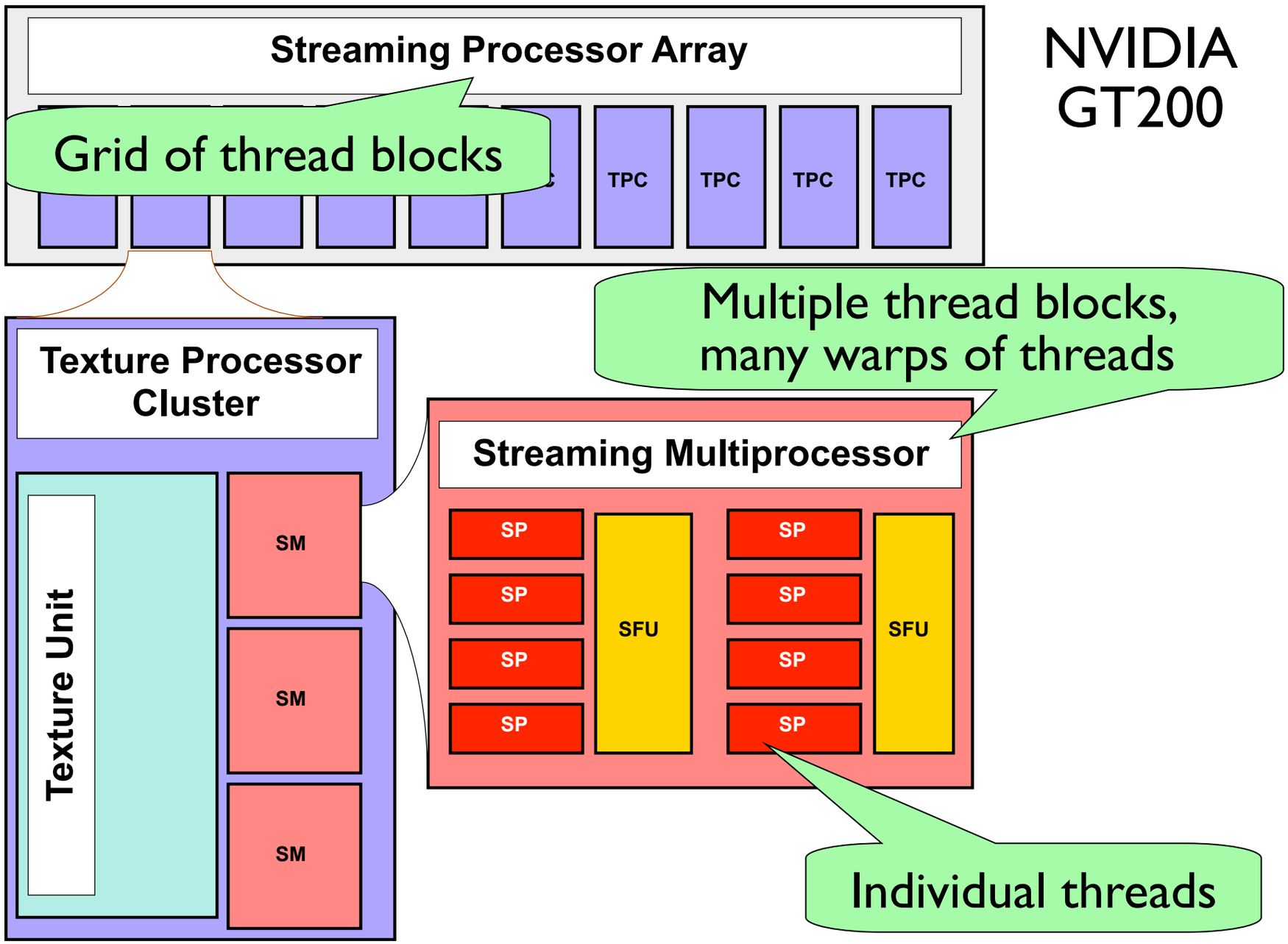
NVIDIA CUDA Overview

- Hardware and software architecture for GPU computing, foundation for building higher level programming libraries, toolkits
- C for 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*
- **Grids** of hundreds of **thread blocks** are 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)

Mapping CUDA Abstractions onto GPU



NVIDIA GT200

GPU Memory Accessible in CUDA

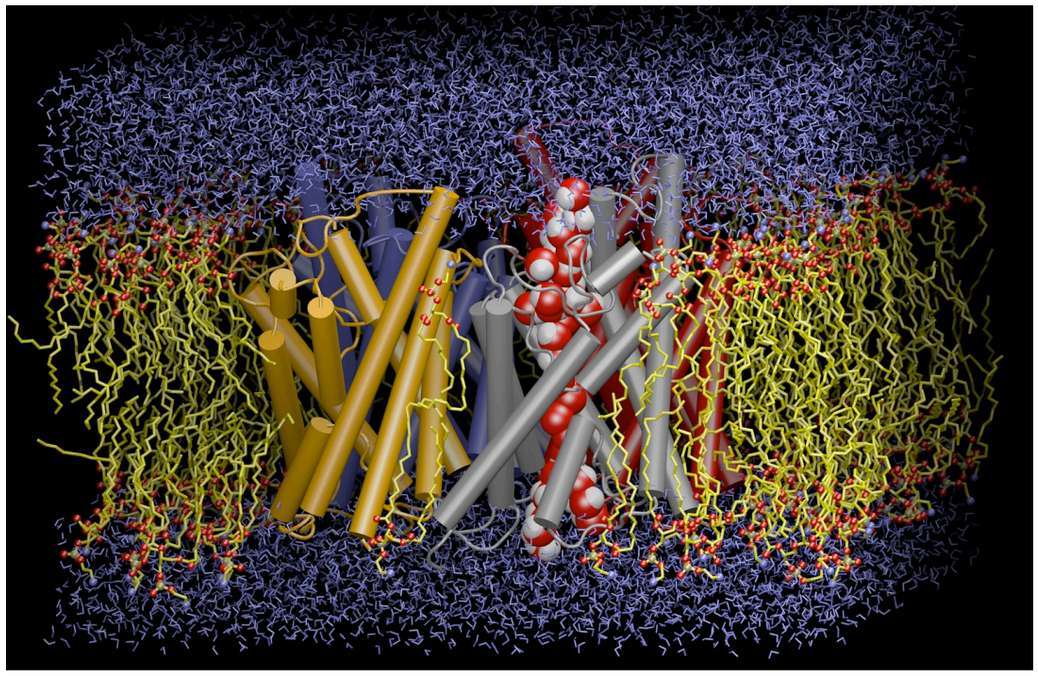
- Mapped host memory: up to 4GB, ~5.7GB/sec bandwidth (PCIe), accessible by multiple GPUs
- Global memory: up to 4GB, high latency (~600 clock cycles), 140GB/sec bandwidth, accessible by all threads; also supports slow atomic operations
- Texture memory: read-only, cached, and interpolated/filtered access 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

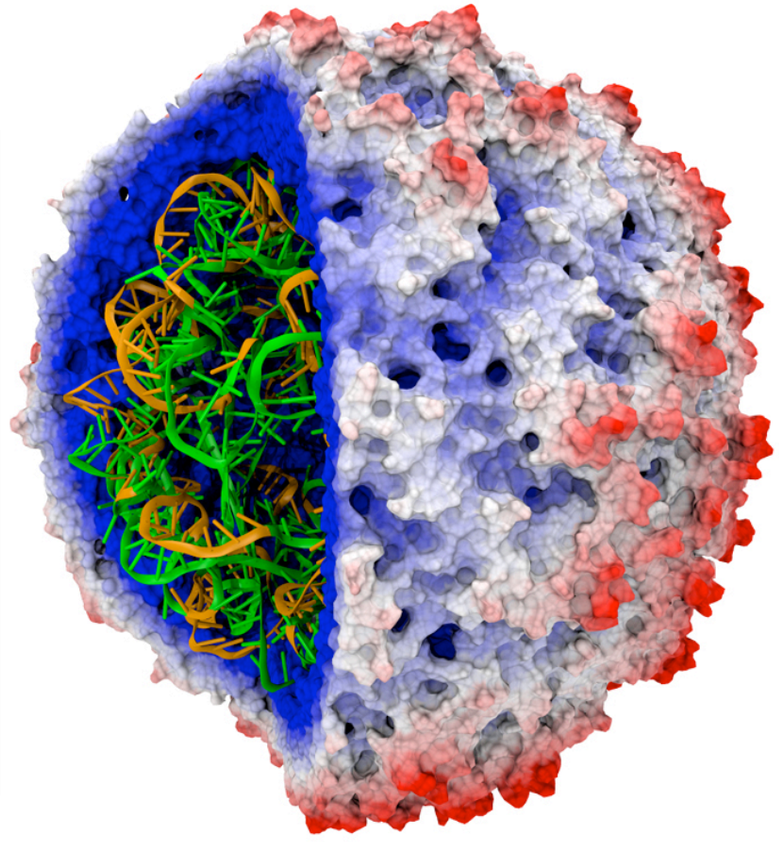
An Approach to Writing CUDA Kernels

- 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

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/

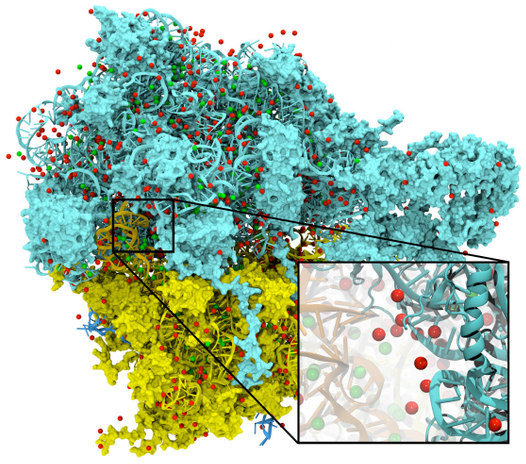




Range of VMD Usage Scenarios

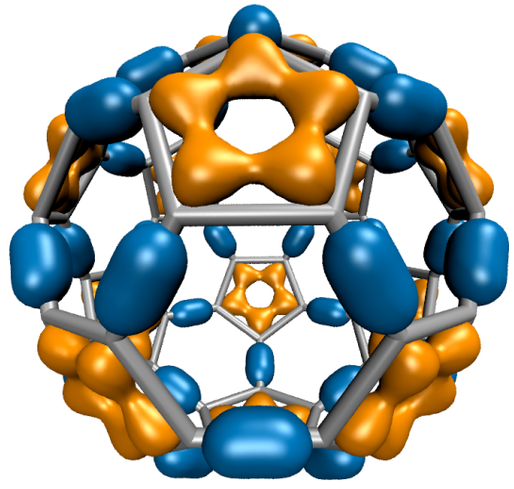
- Users run VMD on a diverse range of hardware: laptops, desktops, clusters, and supercomputers
- Typically used as a desktop application for interactive 3D molecular graphics and analysis
- Can also be run in pure text mode for numerically intensive analysis tasks, batch mode movie rendering, etc.
-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

CUDA Acceleration in VMD



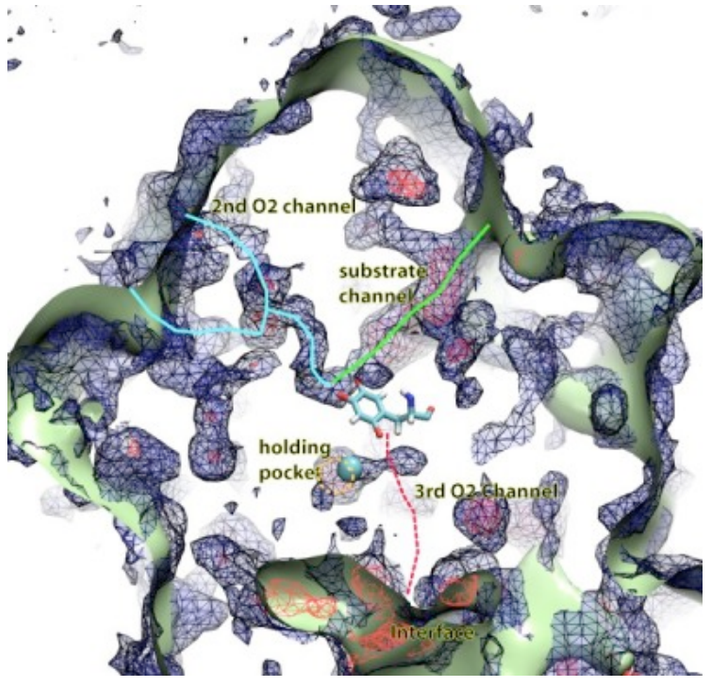
Electrostatic field calculation, ion placement:

factor of 20x to 44x faster



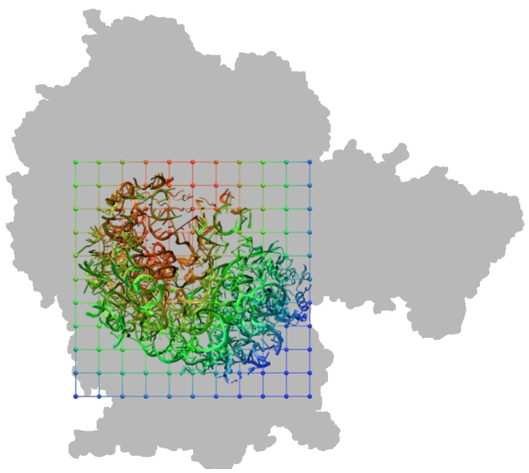
Molecular orbital calculation and display:

factor of 120x faster



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

factor of 20x to 30x faster

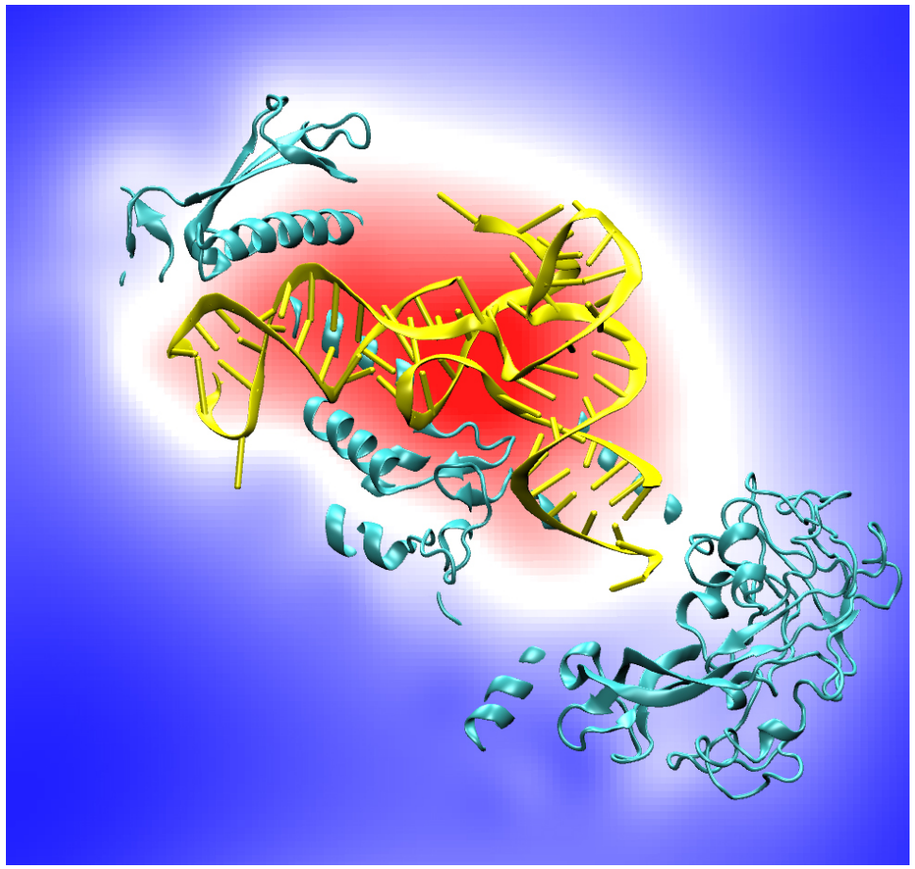


Electrostatic Potential Maps

- 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



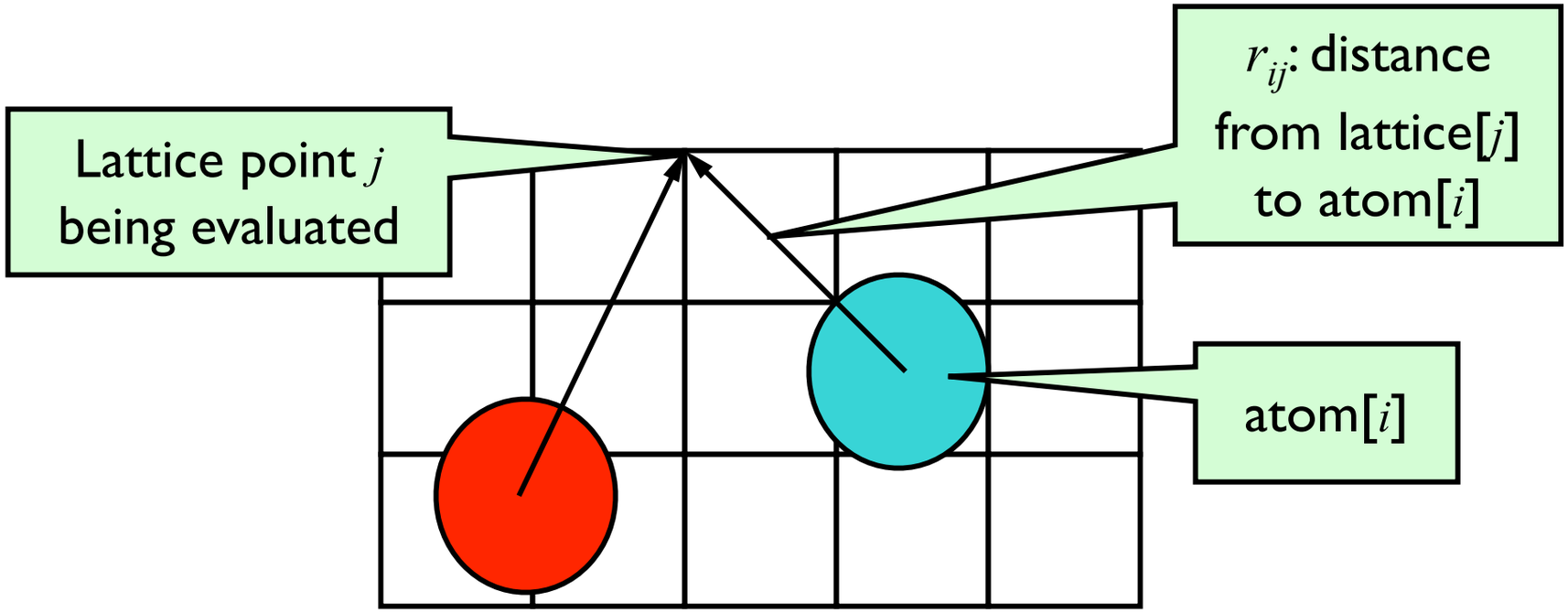
Isoleucine tRNA synthetase

Direct Coulomb Summation

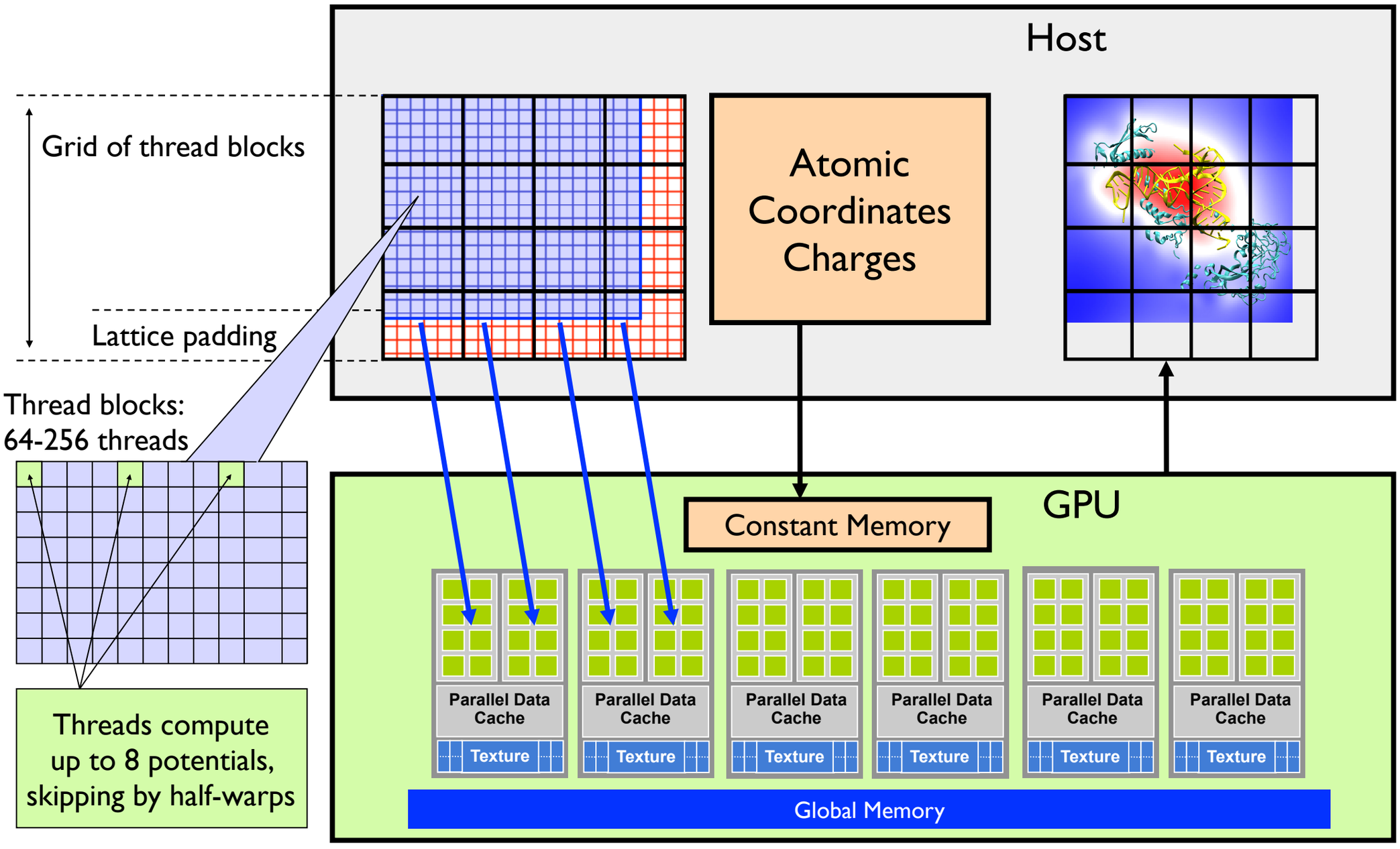
(naive approach)

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

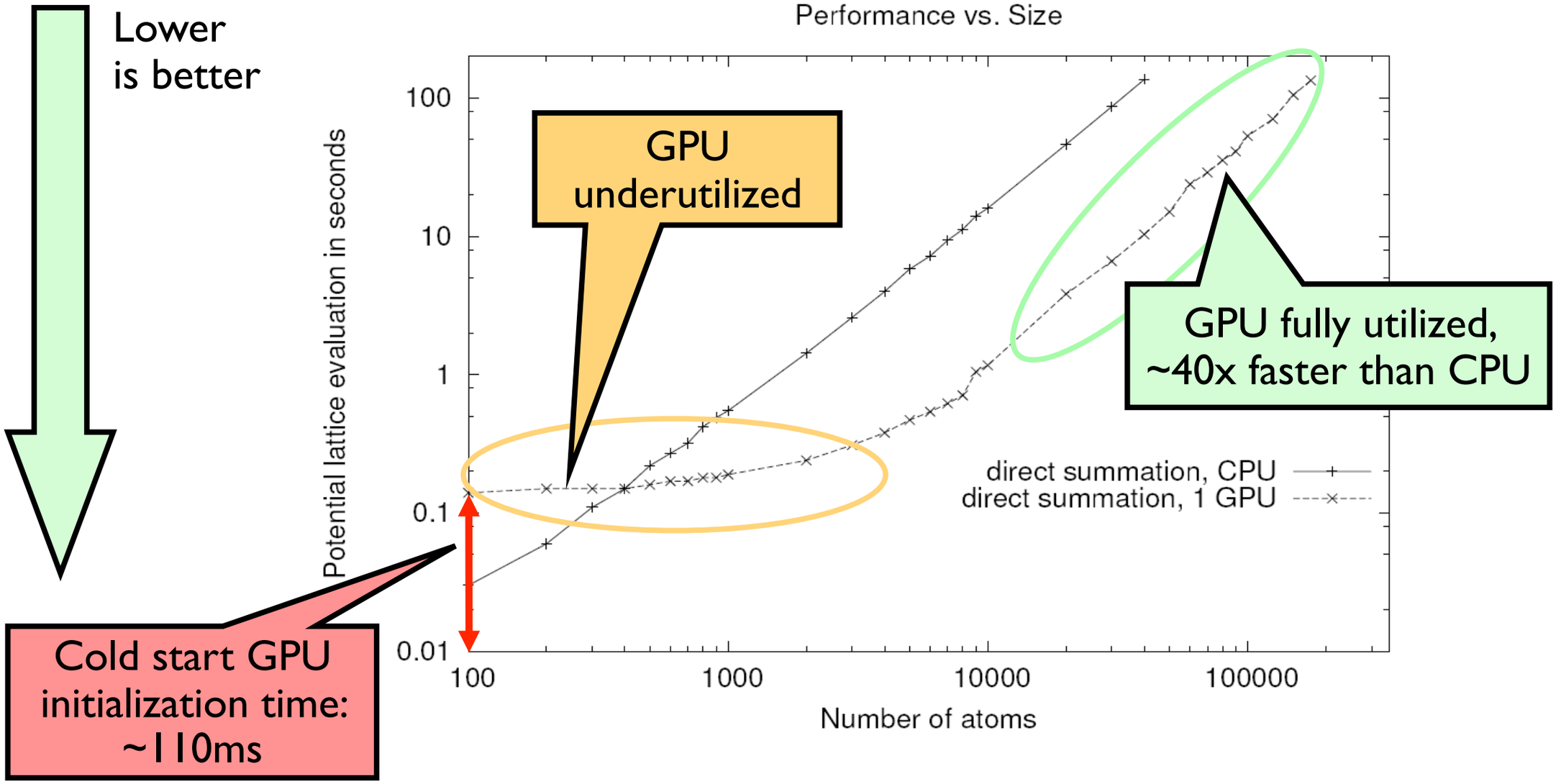
  potential[j] += charge[i] / r_{ij}



Direct Coulomb Summation on GPU

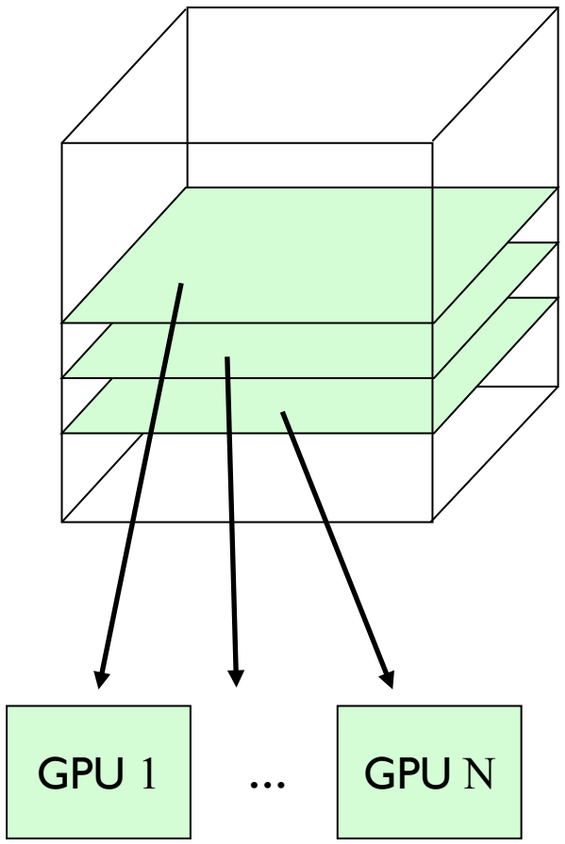


Direct Coulomb Summation Performance



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



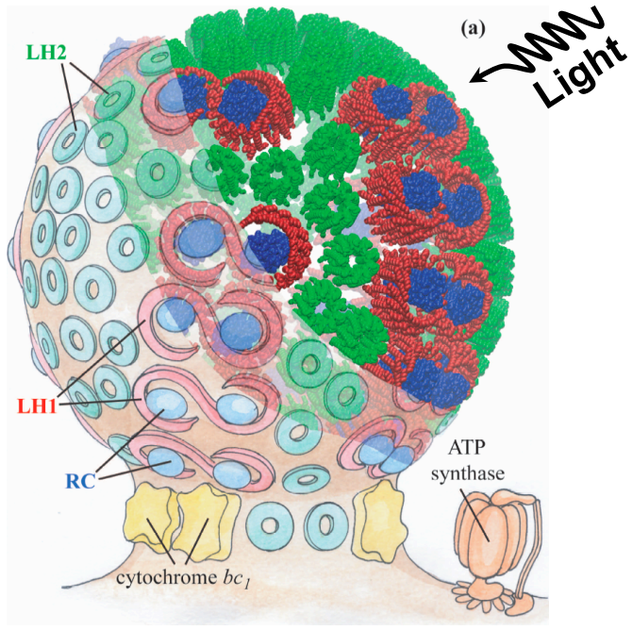
NCSA GPU Cluster
http://www.ncsa.uiuc.edu/Projects/GPUcluster/

	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

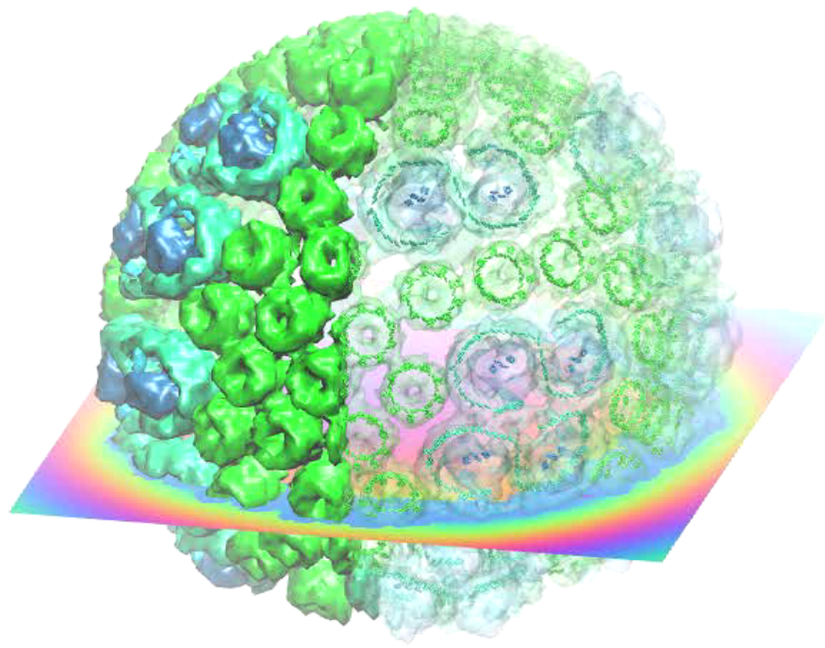
*Speedups relative to Intel QX6700 CPU core w/ SSE

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

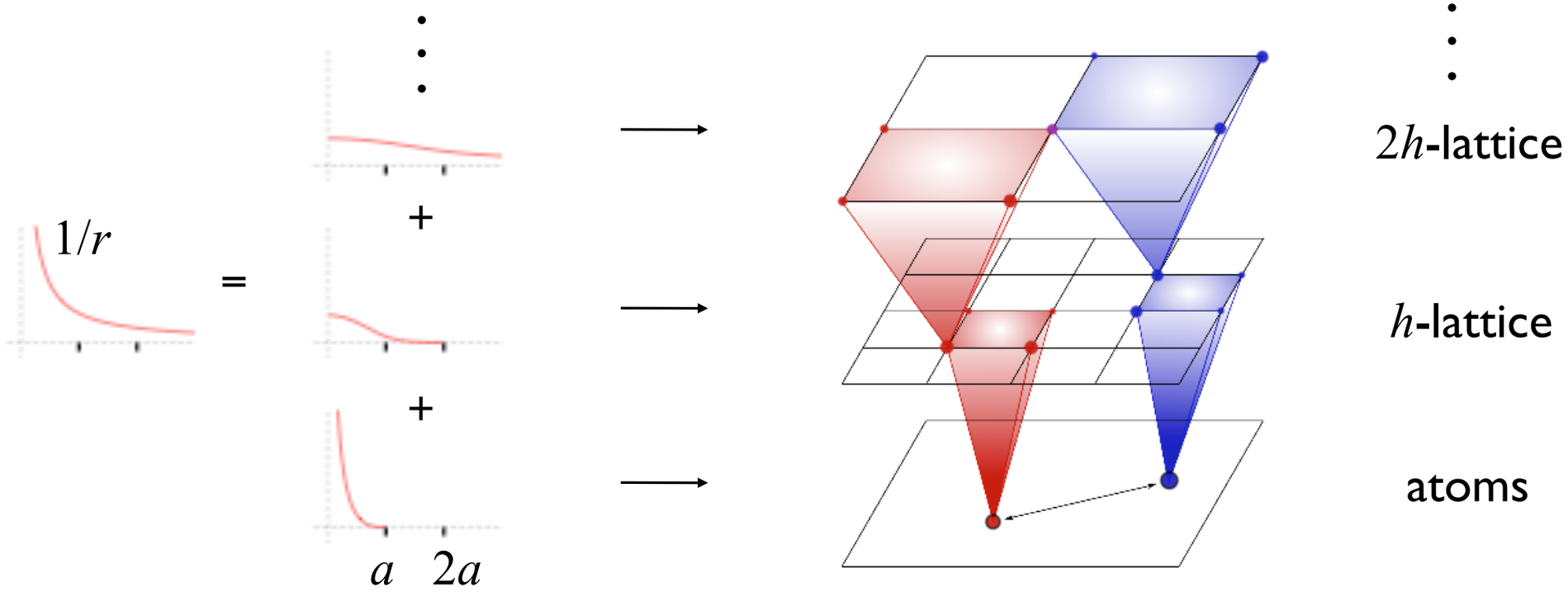
- Approximates full electrostatic potential
- Calculates sum of smoothed pairwise potentials interpolated from a hierarchy of lattices
- Advantages over PME (particle-mesh Ewald) and/or FMM (fast multipole method):
  - Algorithm has linear time complexity
  - Permits non-periodic and periodic boundaries
  - 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

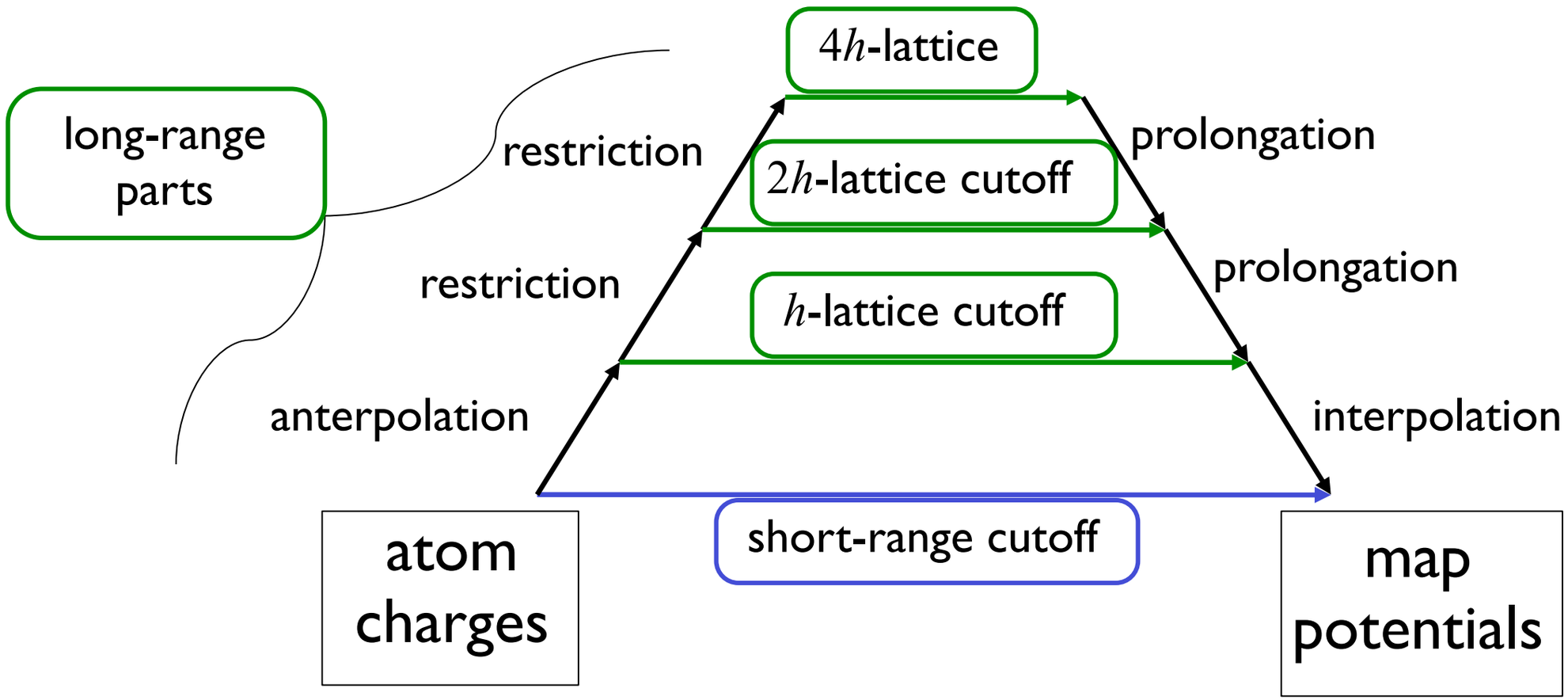
Interpolate the smoothed potentials



Multilevel Summation Calculation

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

Computational Steps

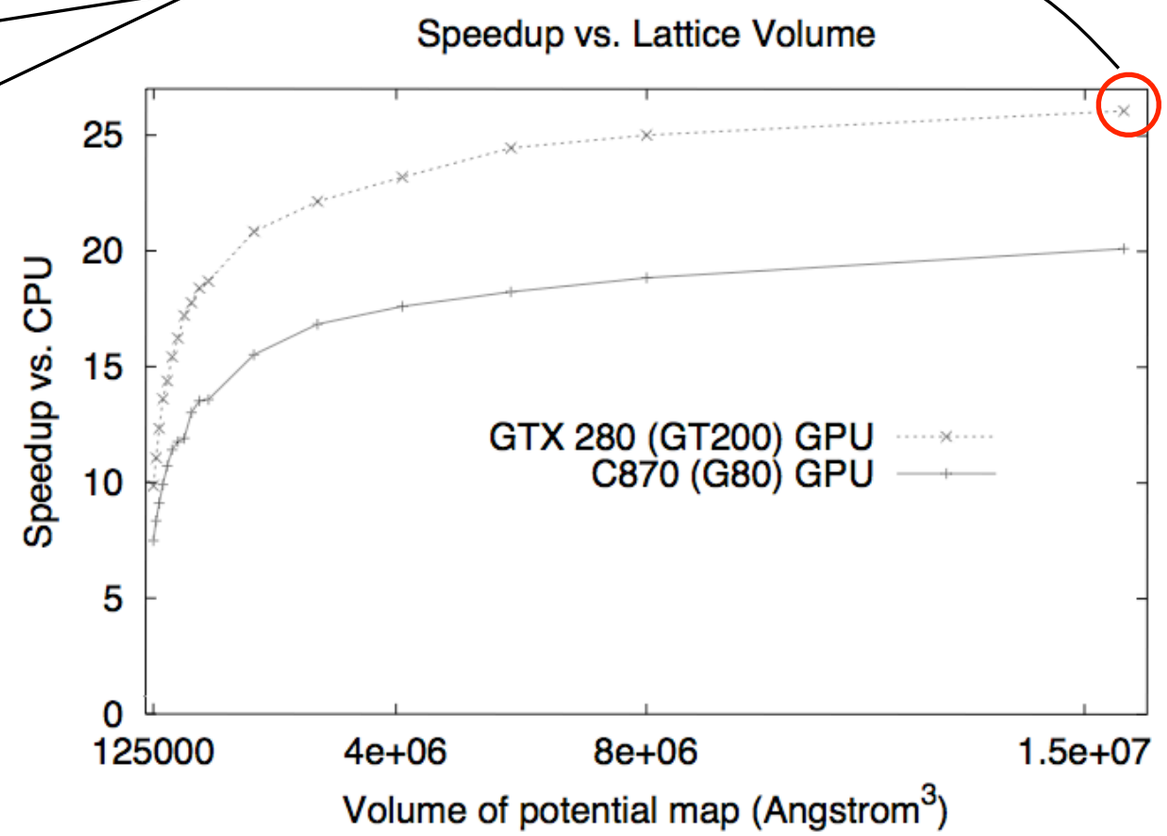


Multilevel Summation on the GPU

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



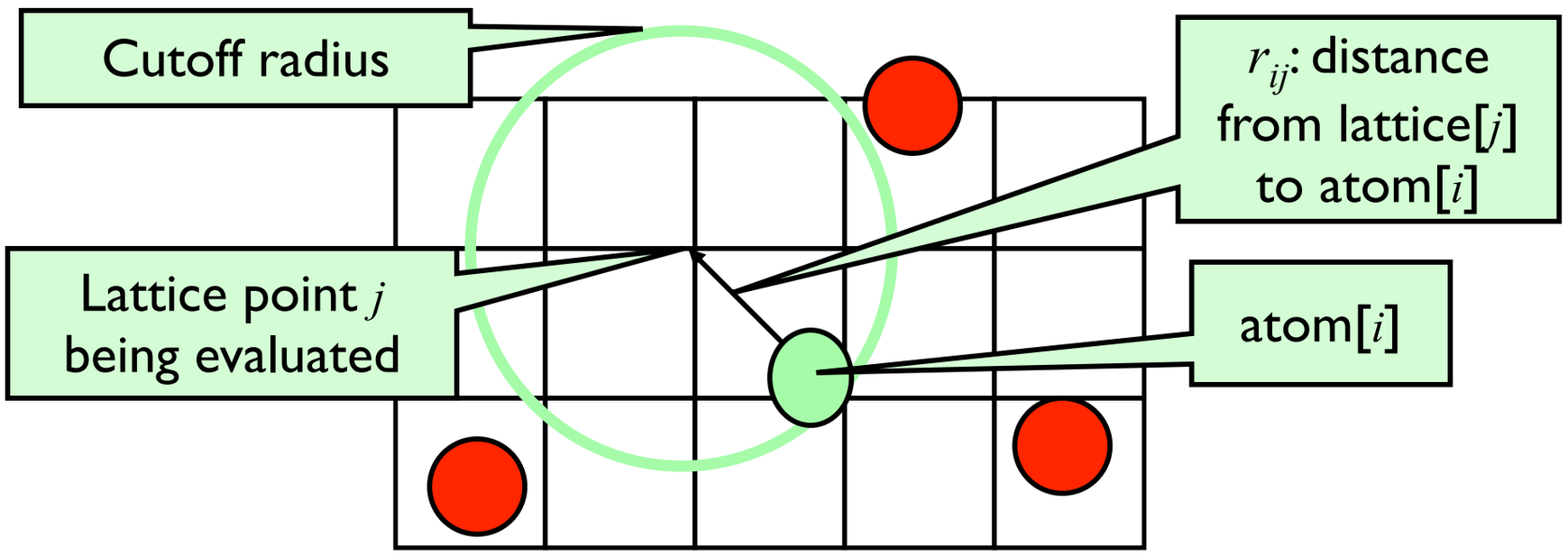
Multilevel summation of electrostatic potentials using graphics processing units.
D. Hardy, J. Stone, K. Schulten. *J. Parallel Computing*, 35:164-177, 2009.

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 Summation on the GPU

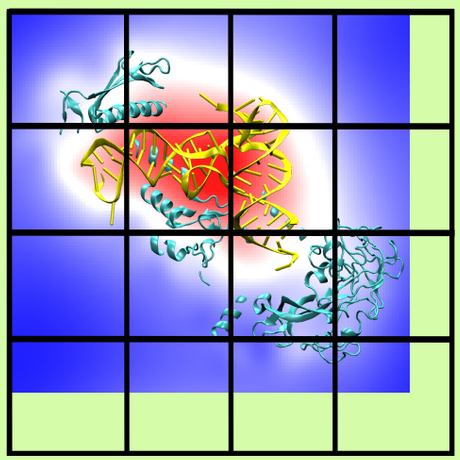
- Atoms are spatially hashed into fixed-size bins
- CPU handles overflowed bins (so GPU kernel can be aggressive)
- GPU thread block calculates corresponding region of potential map
- Solve costly bin/region neighbor checks with lookup table of offsets

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

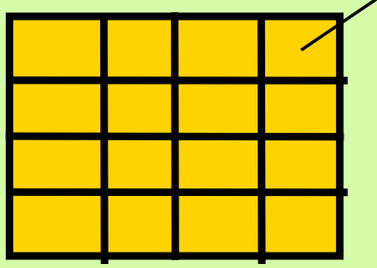
Shared memory

atom bin

Global memory



Potential map regions

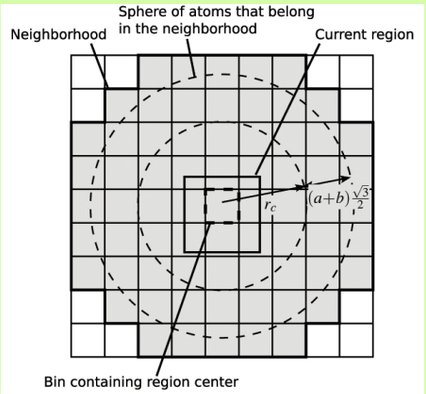


Bins of atoms

Constant memory

Offsets for bin neighborhood

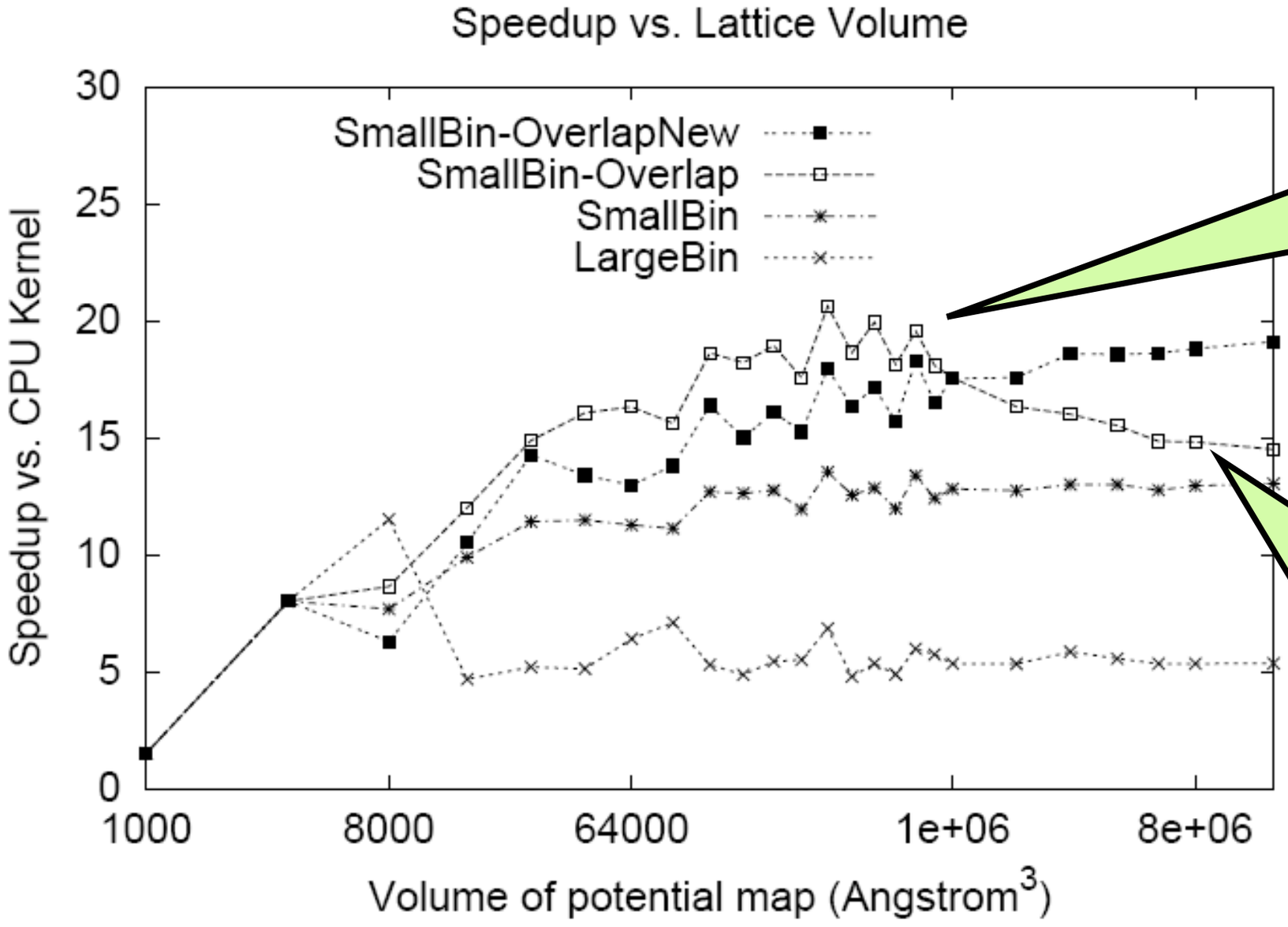
Lookup table encodes "logic" of spatial geometry



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…

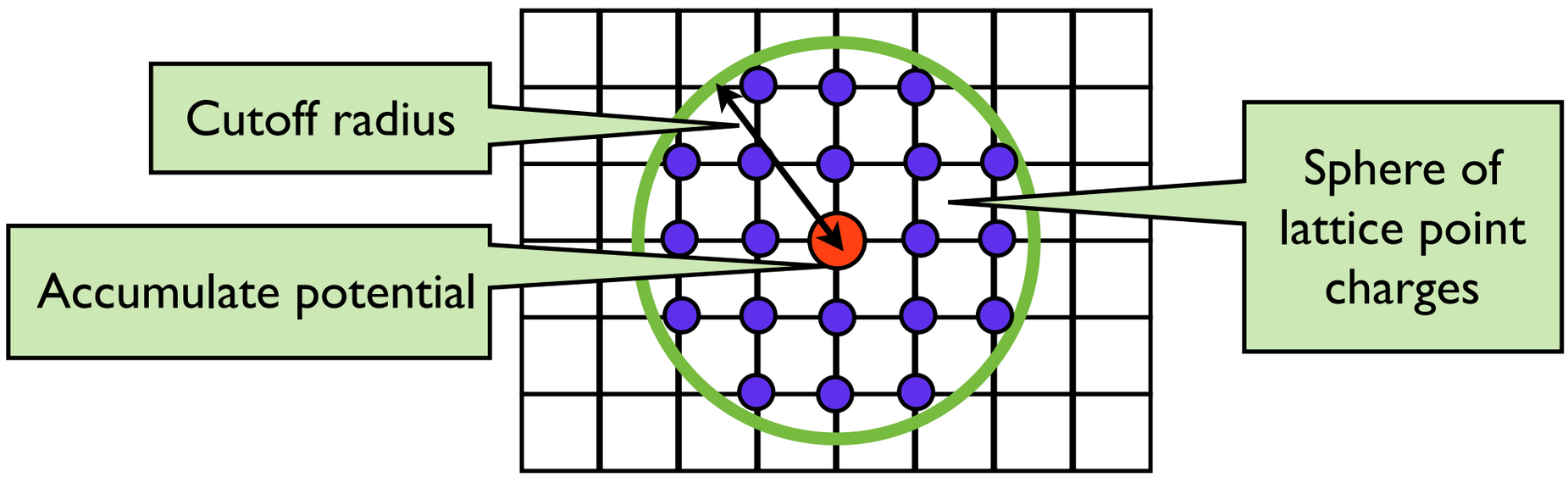
GPU acceleration of cutoff pair potentials for molecular modeling applications. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.

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

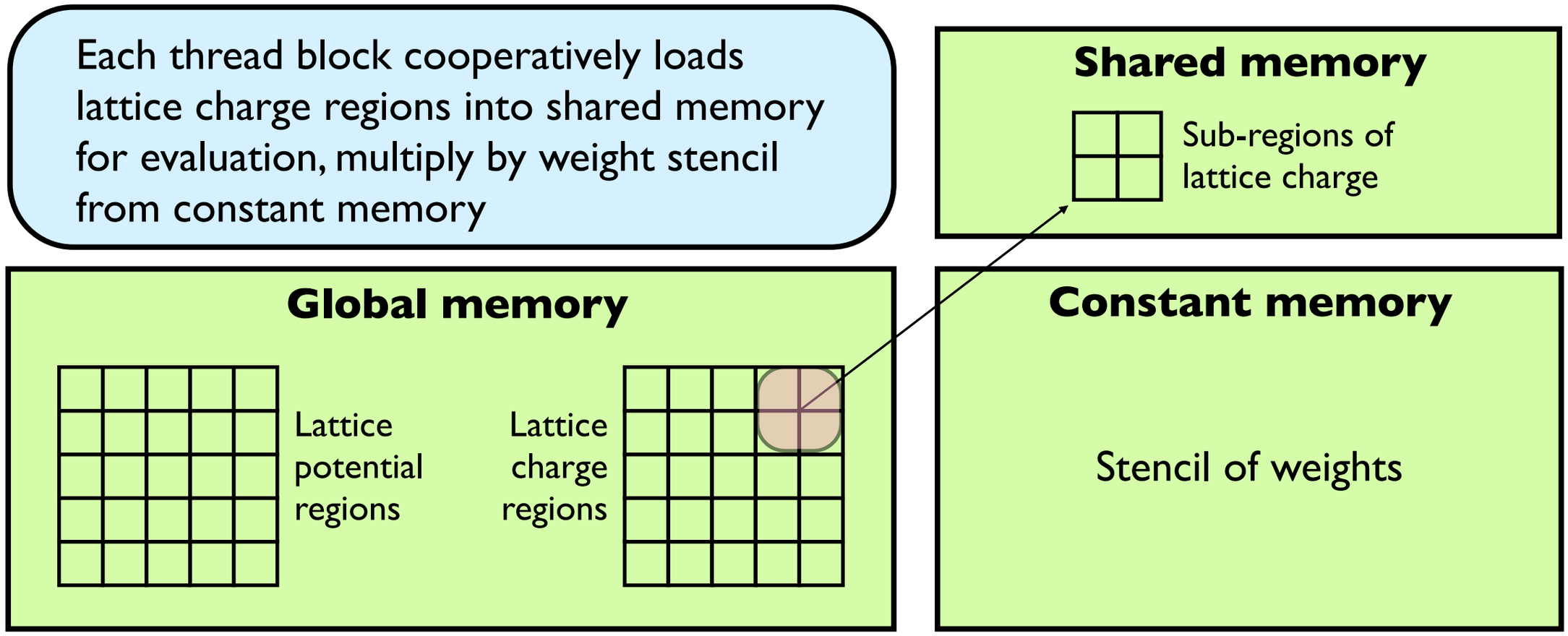
Lattice Cutoff Summation

- 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



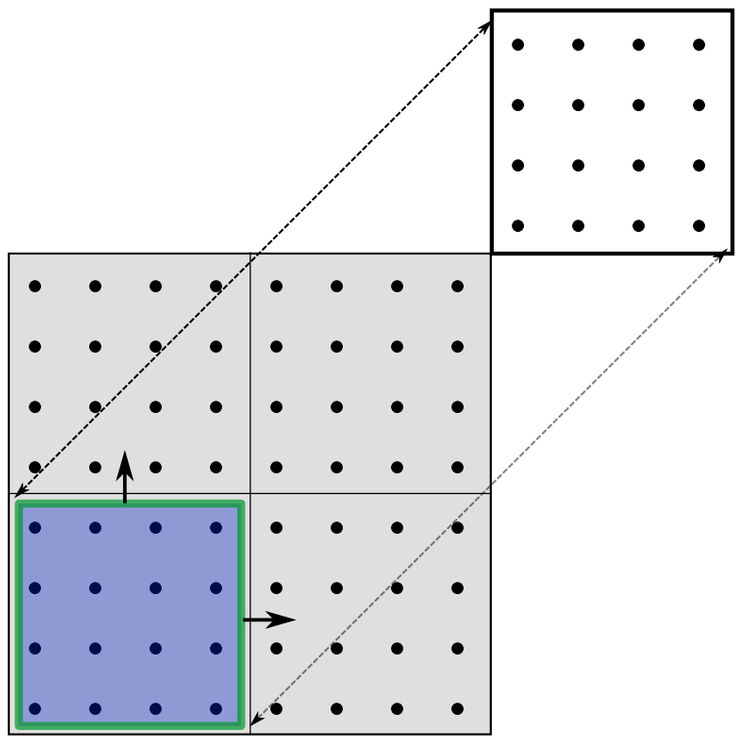
Lattice Cutoff Summation on GPU

- Store stencil of weights in constant memory
- Assign GPU thread block to calculate 4x4x4 region of lattice potentials
- 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)



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

Acknowledgments

- Prof. Klaus Schulten, John Stone, and the Theoretical and Computational Biophysics Group at the Beckman Institute, University of Illinois at Urbana-Champaign
- Prof. Wen-mei Hwu, Chris Rodrigues, and 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

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.