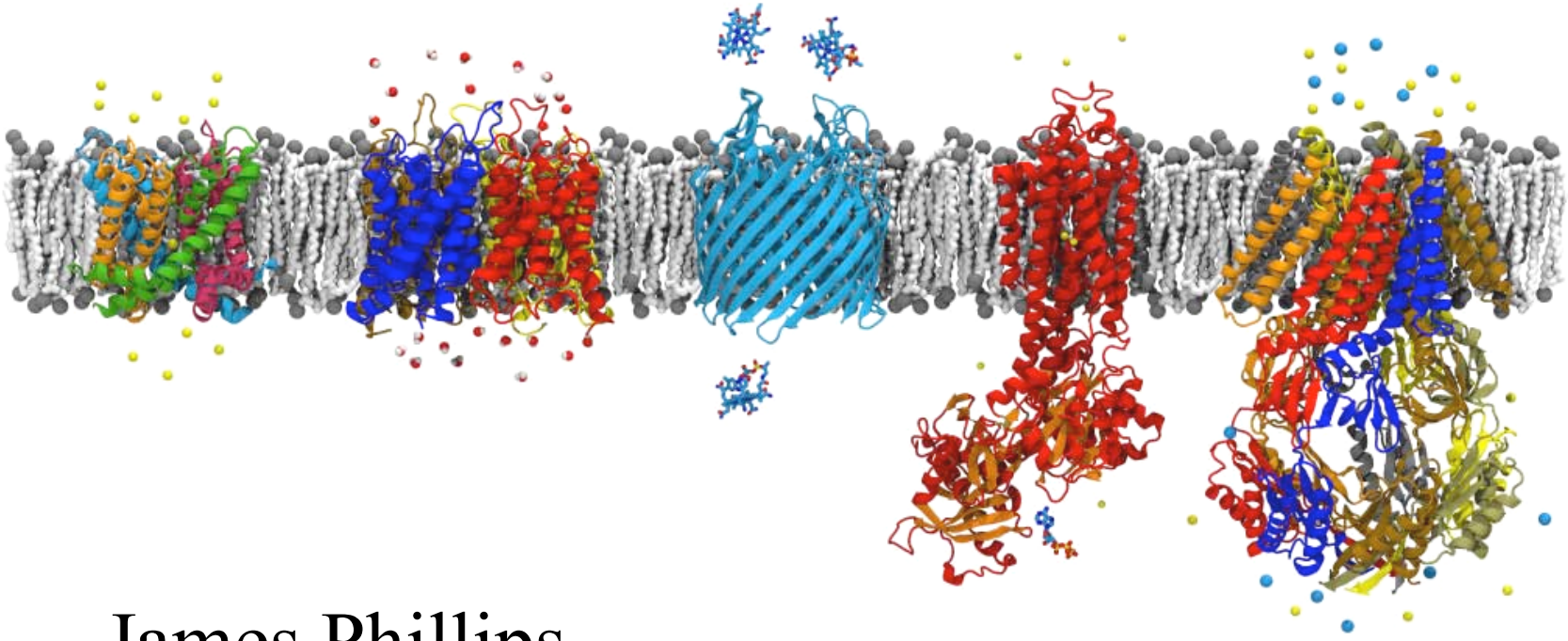
NAMD: Molecular Dynamics on GPU Clusters

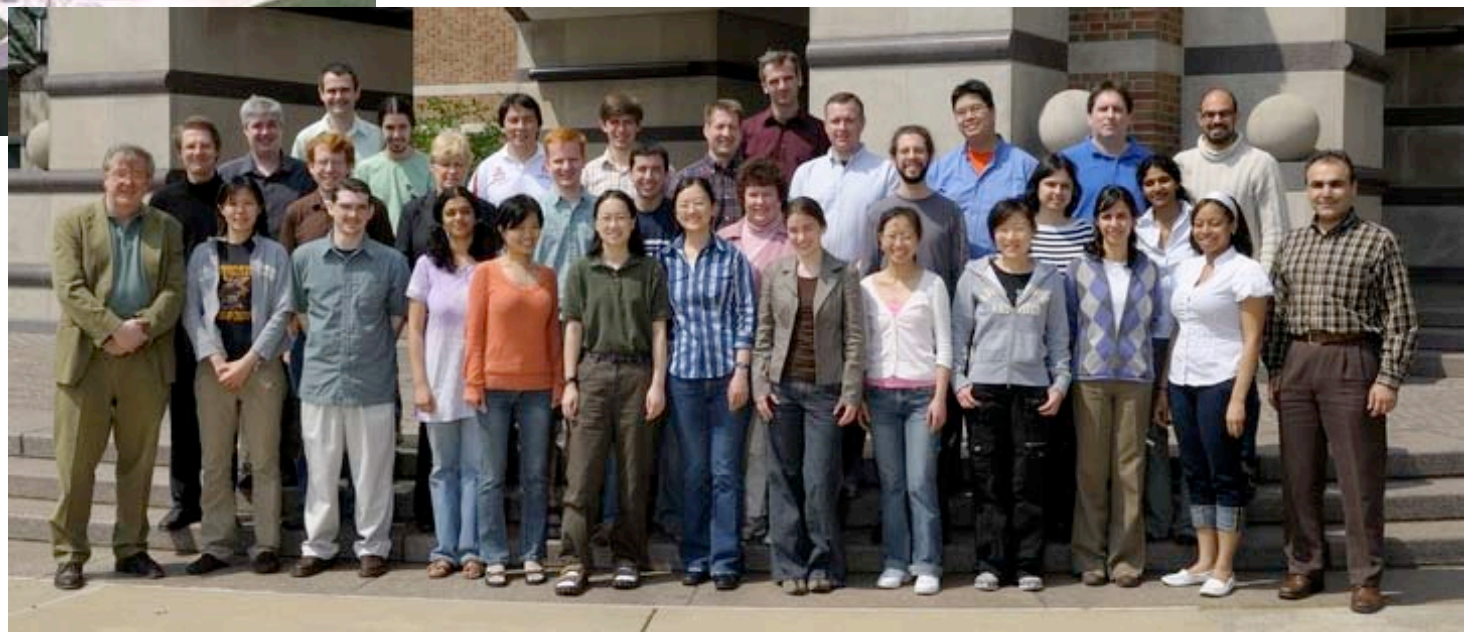


James Phillips

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

Beckman Institute
University of Illinois at Urbana-Champaign

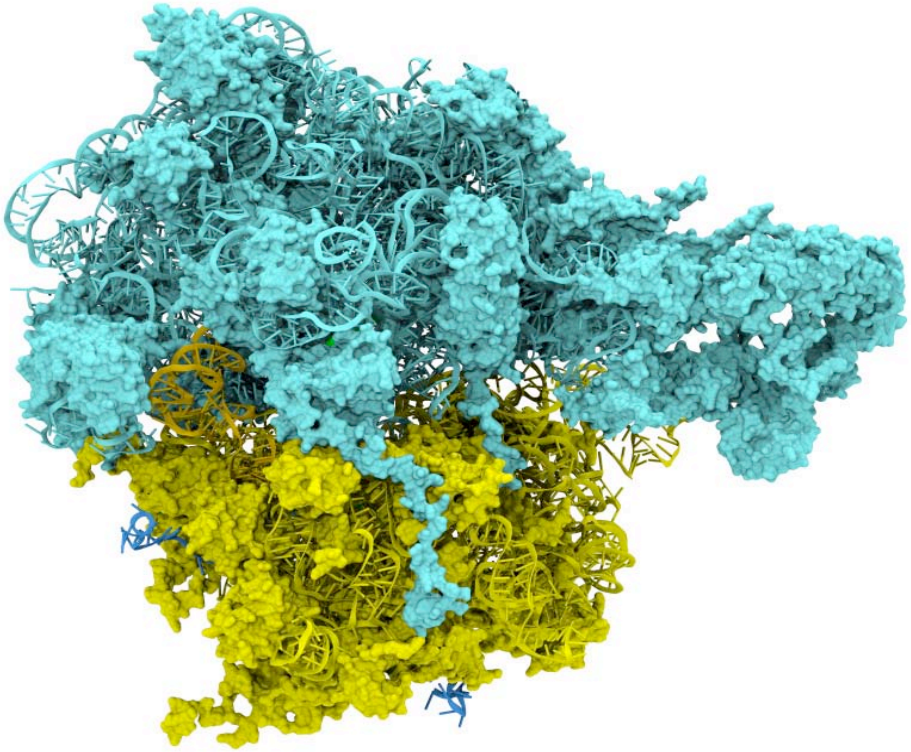
Theoretical and Computational Biophysics Group



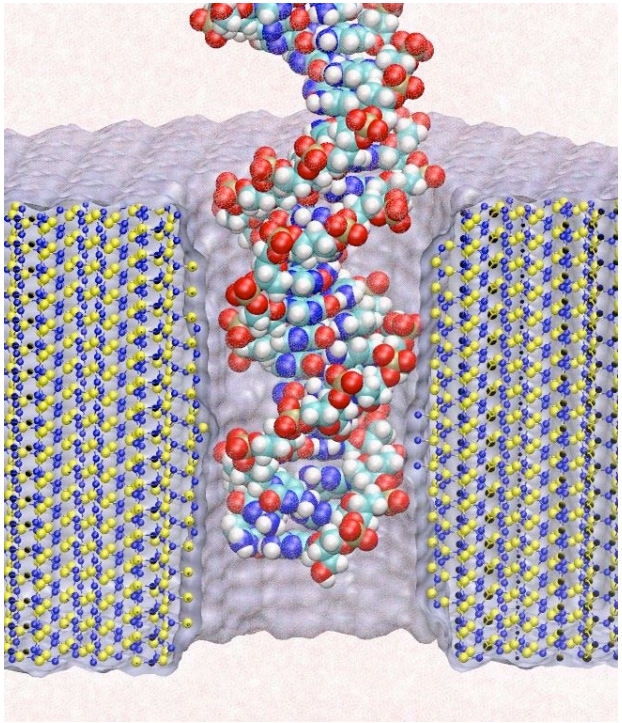
National Center for Research Resources

Computational Microscopy

Ribosome: synthesizes proteins from genetic information, target for antibiotics



Silicon nanopore: bionanodevice for sequencing DNA efficiently



NAMD: Practical Supercomputing

- 35,000 users can't all be computer experts.
  - 18% are NIH-funded; many in other countries.
  - 8200 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

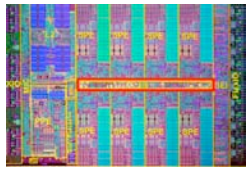
Phillips *et al.*, *J. Comp. Chem.* **26**:1781-1802, 2005.

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

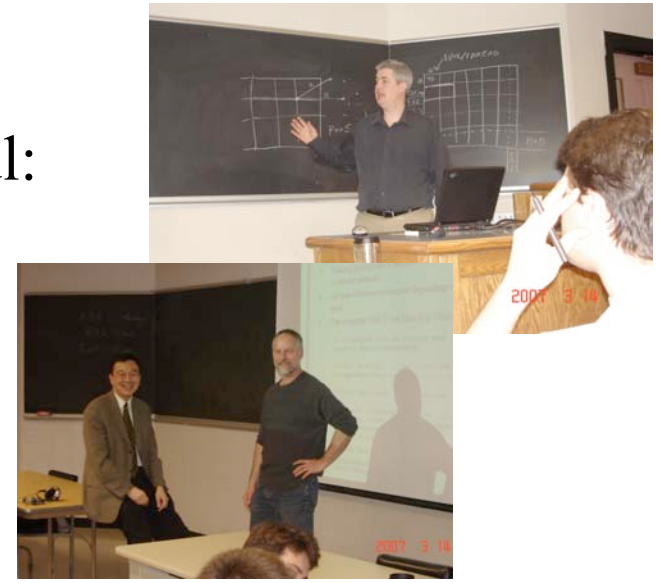


CUDA: Practical Performance

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)

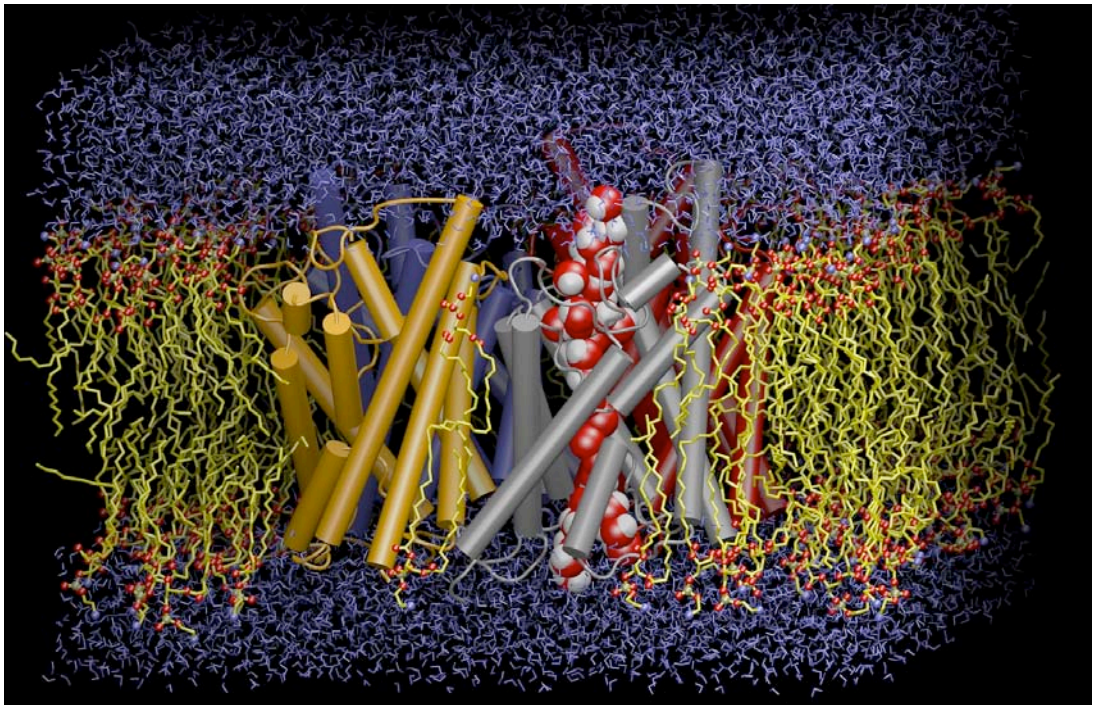
Fun to program (and drive)

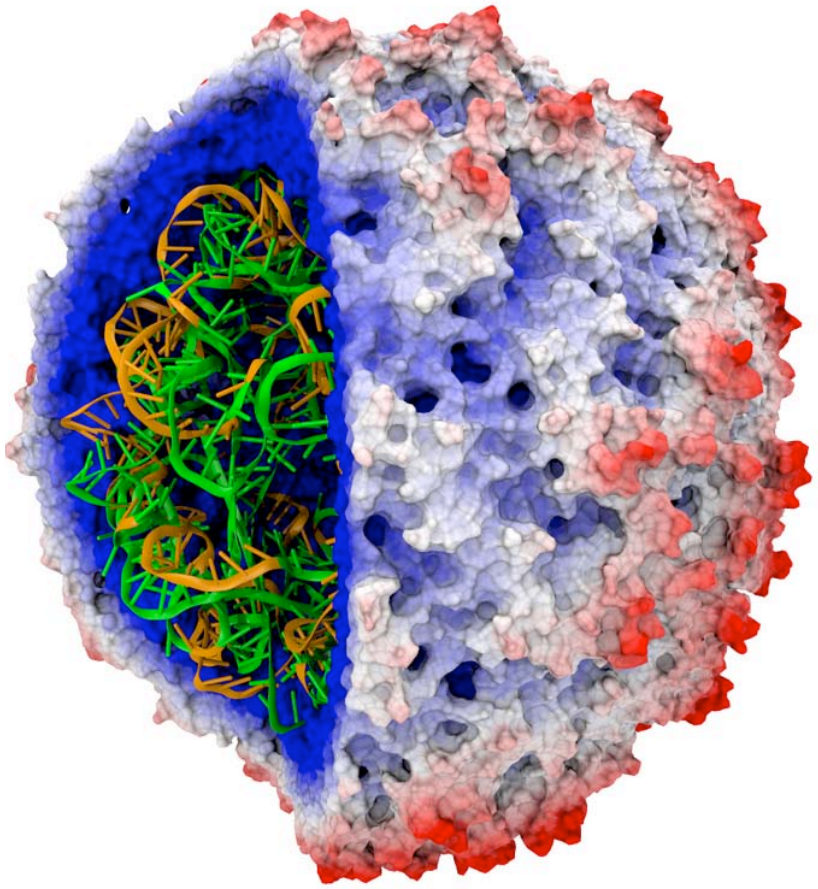


Stone *et al.*, *J. Comp. Chem.* **28**:2618-2640, 2007.

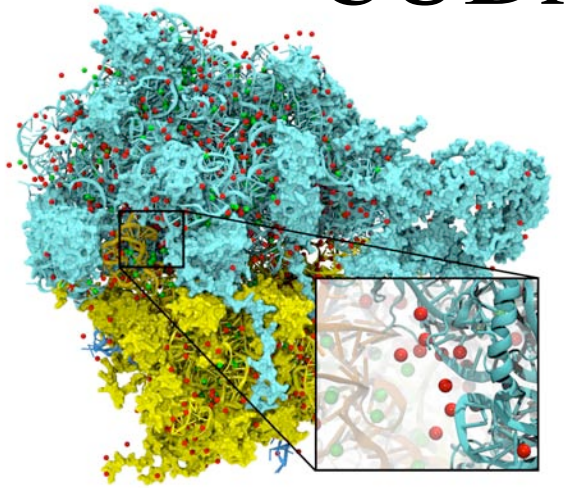
VMD – "Visual Molecular Dynamics"

- 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/



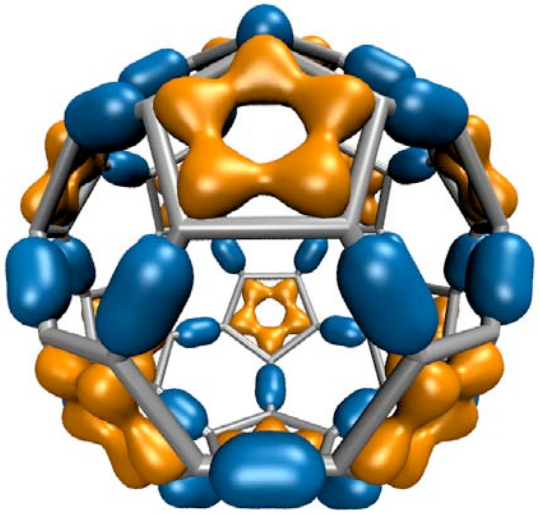


CUDA Acceleration in VMD



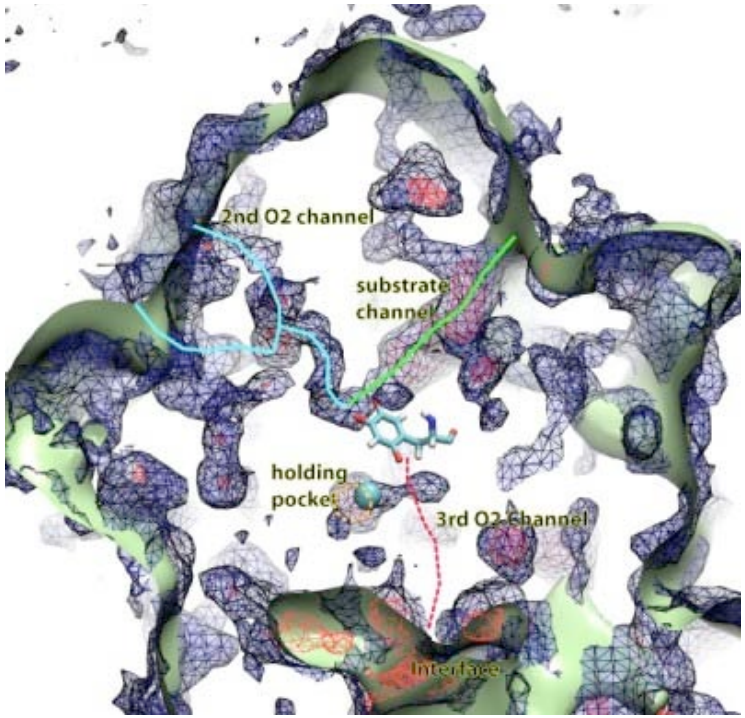
Electrostatic field calculation, ion placement

20x to 44x faster



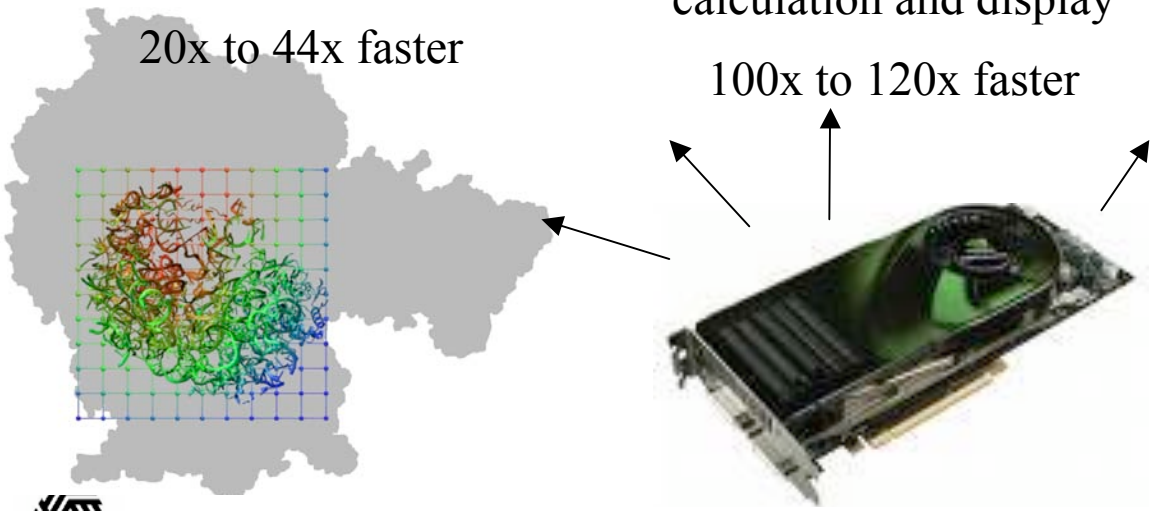
Molecular orbital calculation and display

100x to 120x faster



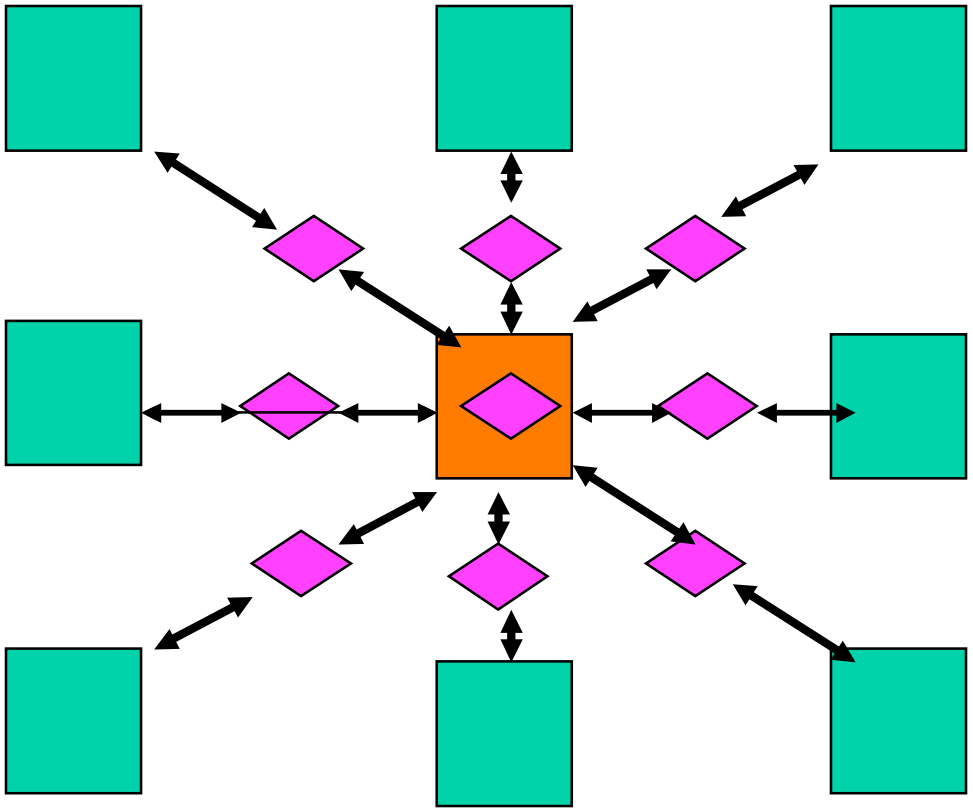
Imaging of gas migration pathways in proteins with implicit ligand sampling

20x to 30x faster



NAMD Hybrid Decomposition

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



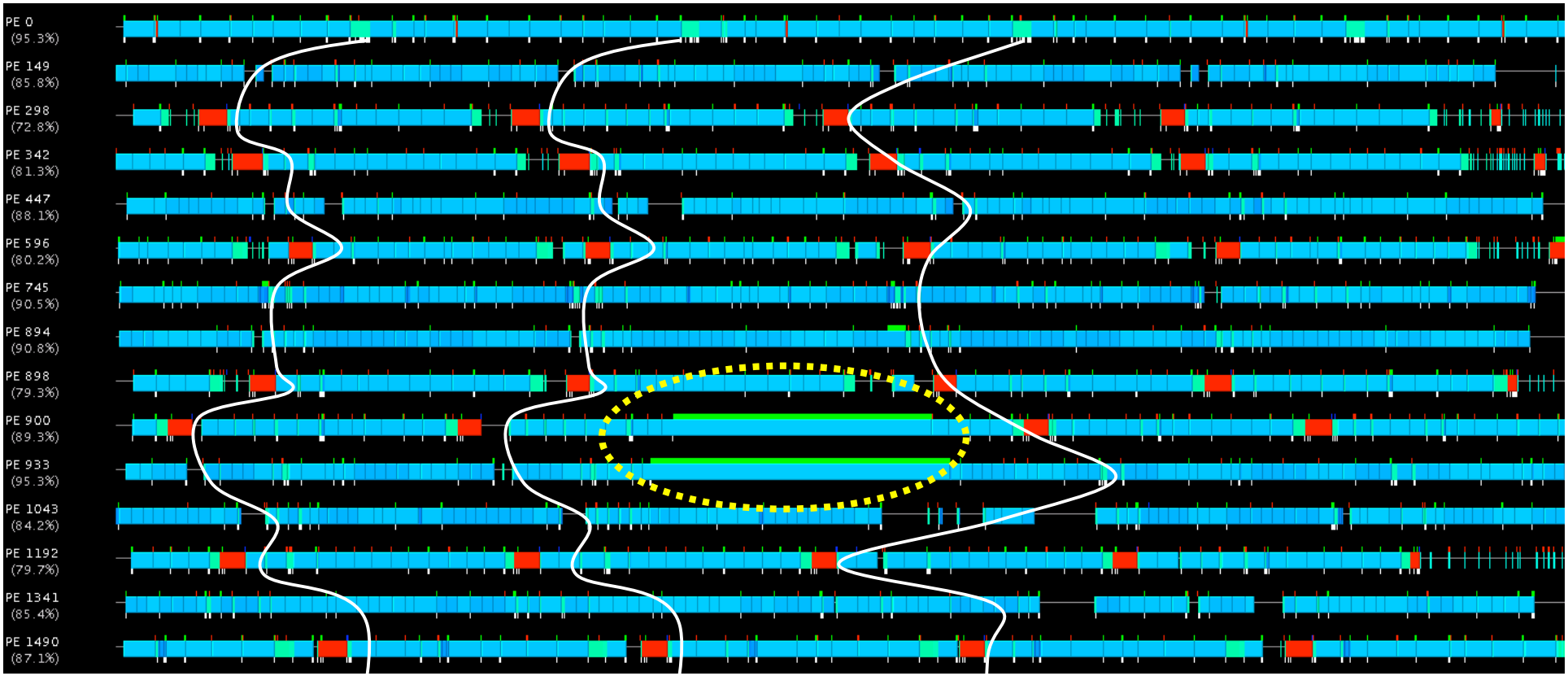
- Spatially decompose data and communication.
- Separate but related work decomposition.
- "Compute objects" facilitate iterative, measurement-based load balancing system.

NAMD Code is Message-Driven

- No receive calls as in “message passing”
- Messages sent to object “entry points”
- Incoming messages placed in queue
  - Priorities are necessary for performance
- Execution generates new messages
- Implemented in Charm++ on top of MPI
  - Can be emulated in MPI alone
  - Charm++ provides tools and idioms
  - Parallel Programming Lab: http://charm.cs.uiuc.edu/

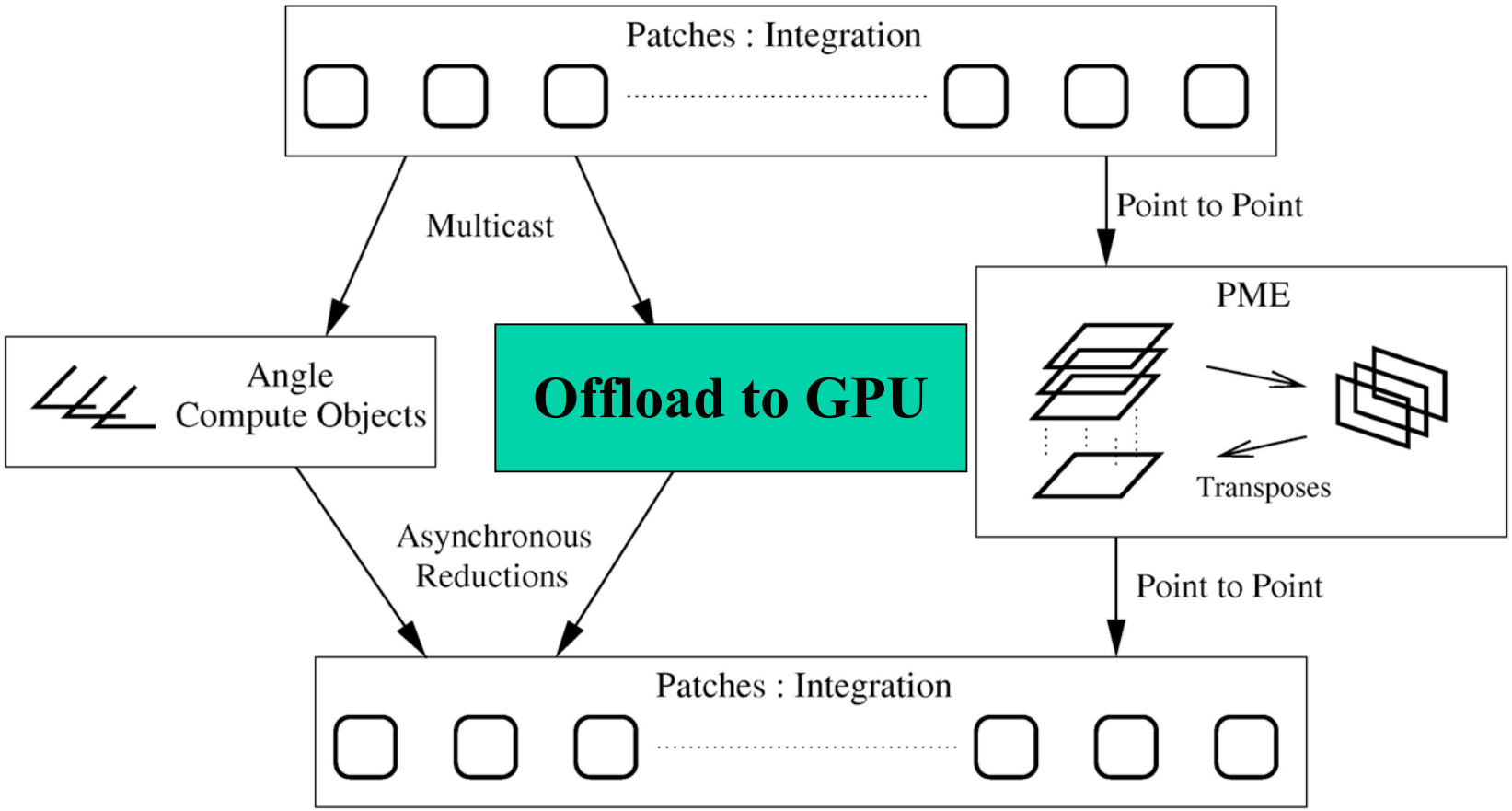
System Noise Example

Timeline from Charm++ tool “Projections” http://charm.cs.uiuc.edu/



NAMD Overlapping Execution

Phillips *et al., SC2002*.

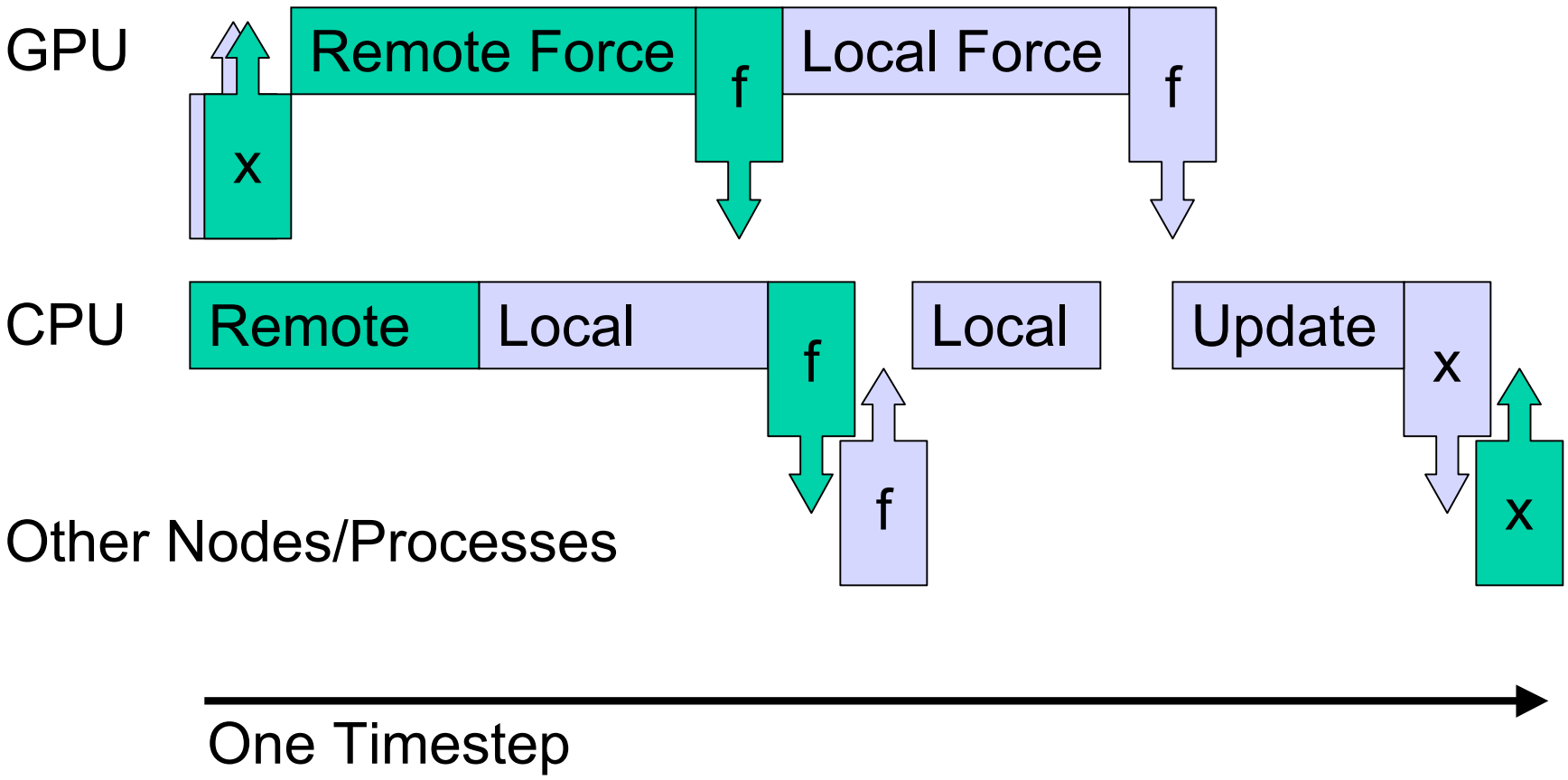


Objects are assigned to processors and queued as data arrives.

Message-Driven CUDA?

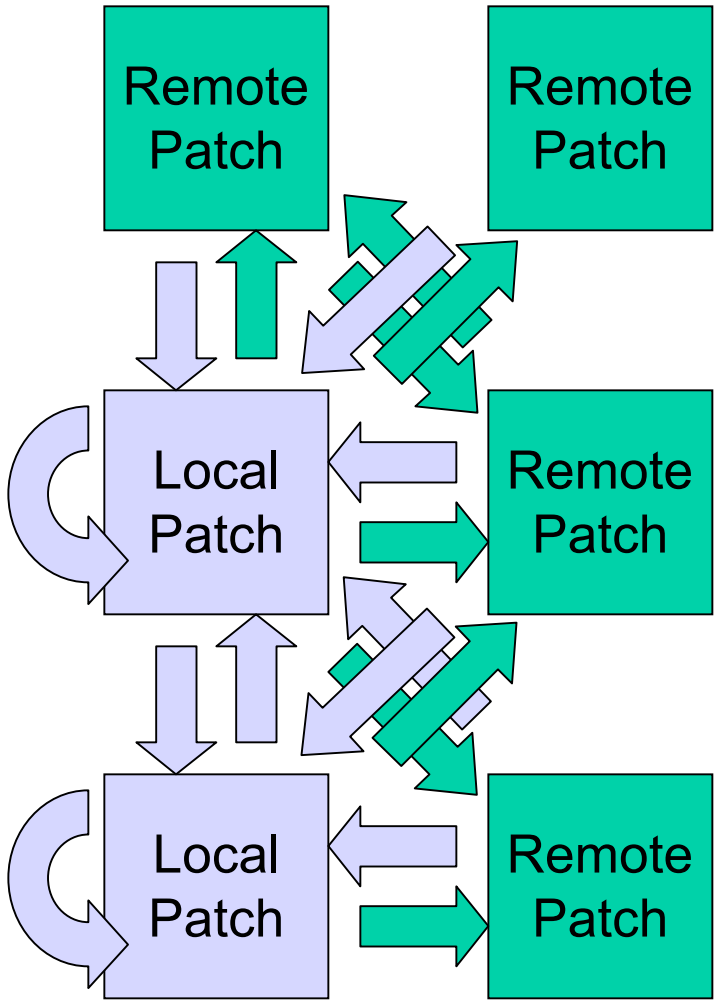
- No, CUDA is too coarse-grained.
  - CPU needs fine-grained work to interleave and pipeline.
  - GPU needs large numbers of tasks submitted all at once.
- No, CUDA lacks priorities.
  - FIFO isn't enough.
- Perhaps in a future interface:
  - Stream data to GPU.
  - Append blocks to a running kernel invocation.
  - Stream data out as blocks complete.
- Fermi looks very promising!

Overlapping GPU and CPU with Communication



“Remote Forces”

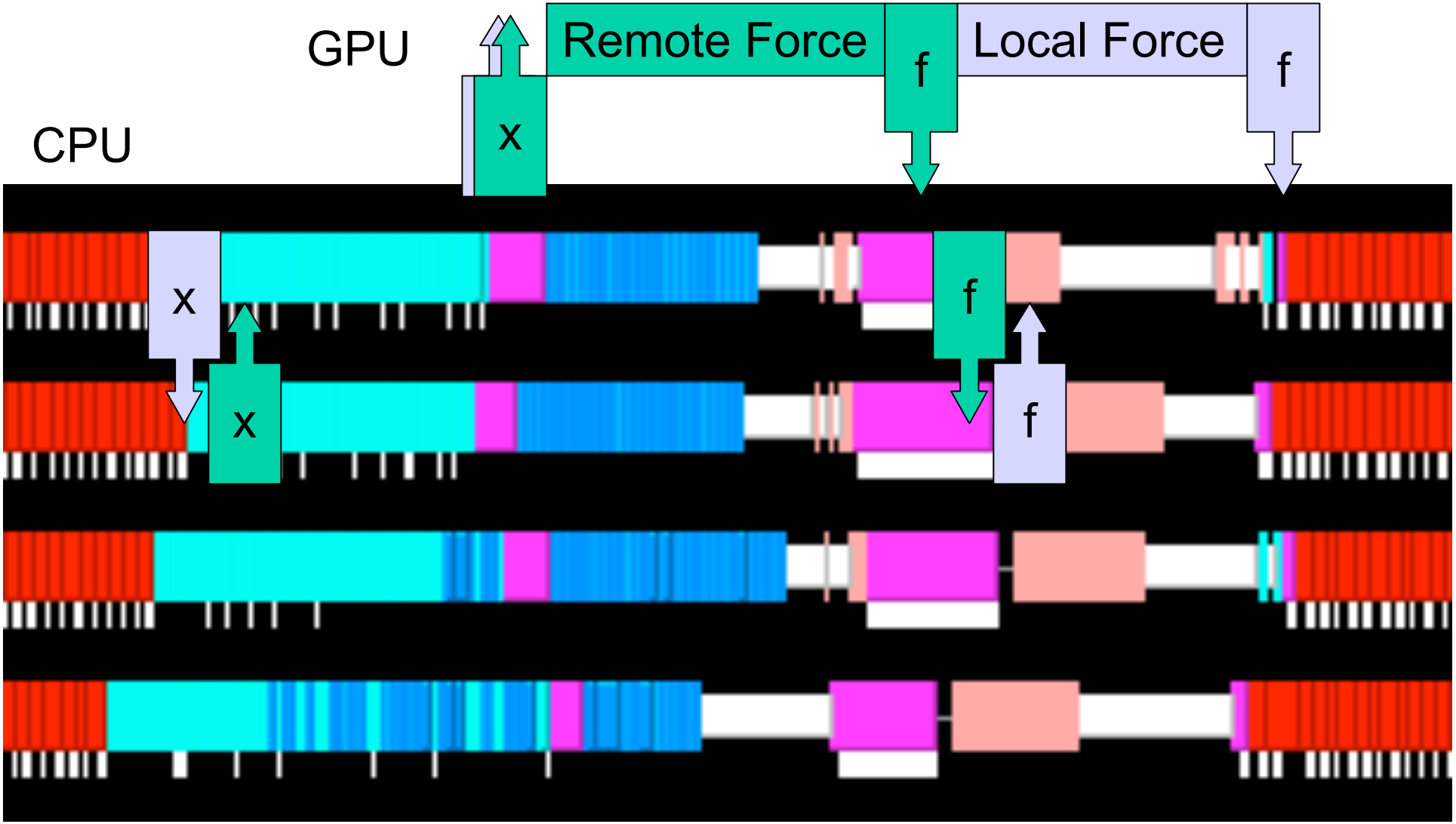
- Forces on atoms in a local patch are “local”
- Forces on atoms in a remote patch are “remote”
- Calculate remote forces first to overlap force communication with local force calculation
- Not enough work to overlap with position communication



Work done by **one** processor

Actual Timelines from NAMD

Generated using Charm++ tool “Projections” http://charm.cs.uiuc.edu/

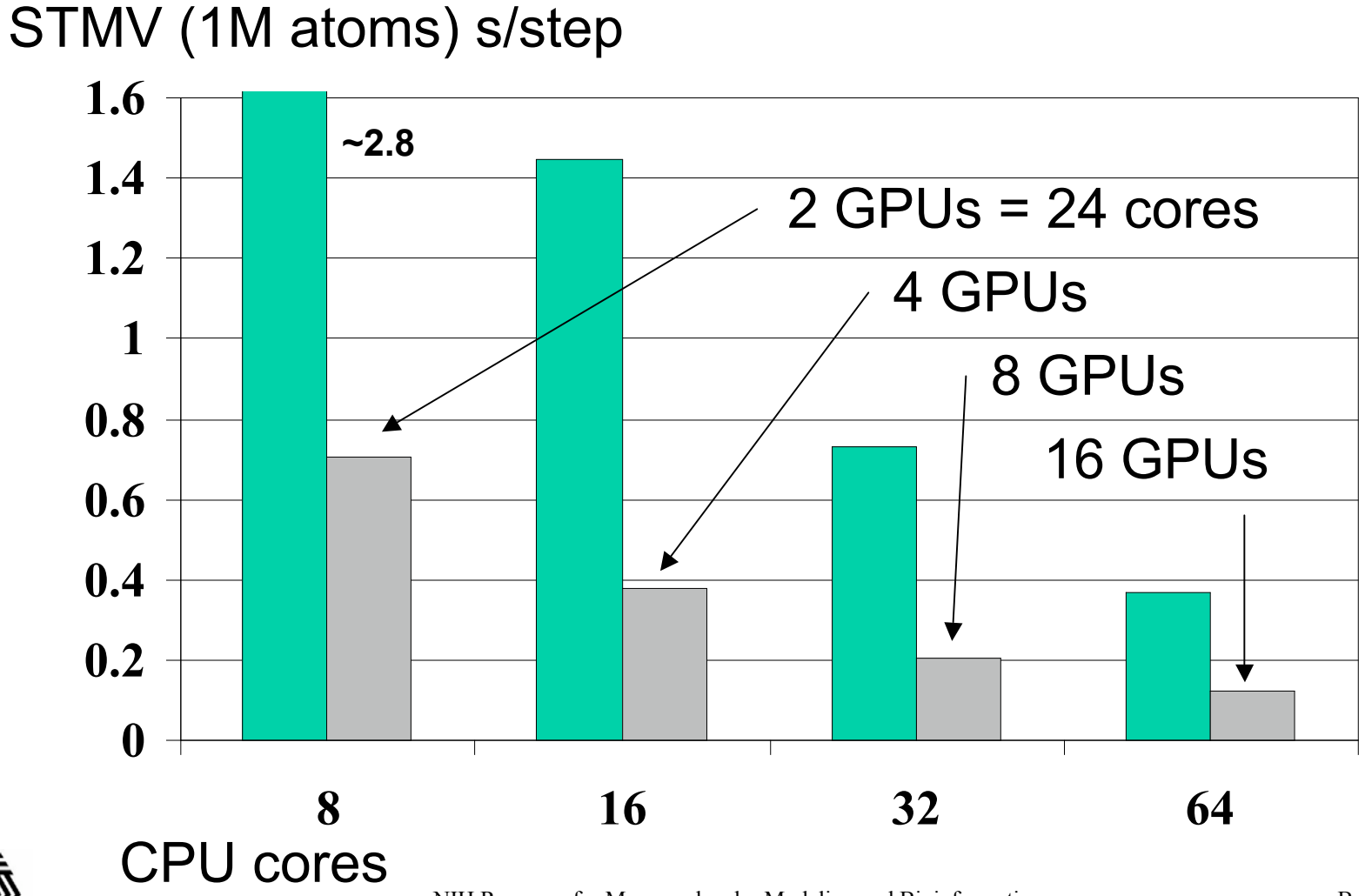


NCSA “8+2” Lincoln Cluster

- 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?

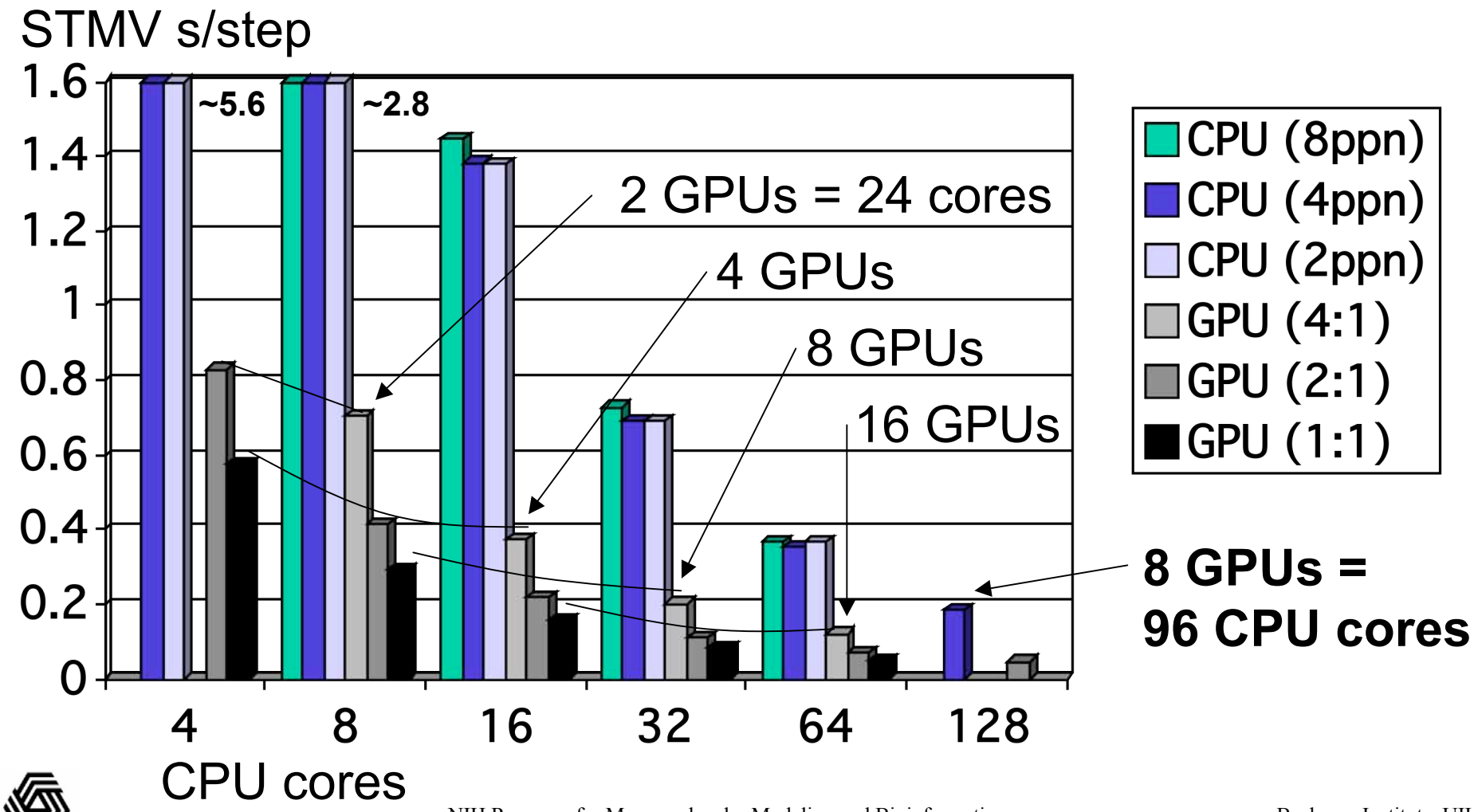
NCSA Lincoln Cluster Performance

(8 Intel cores and 2 NVIDIA Telsa GPUs per node)

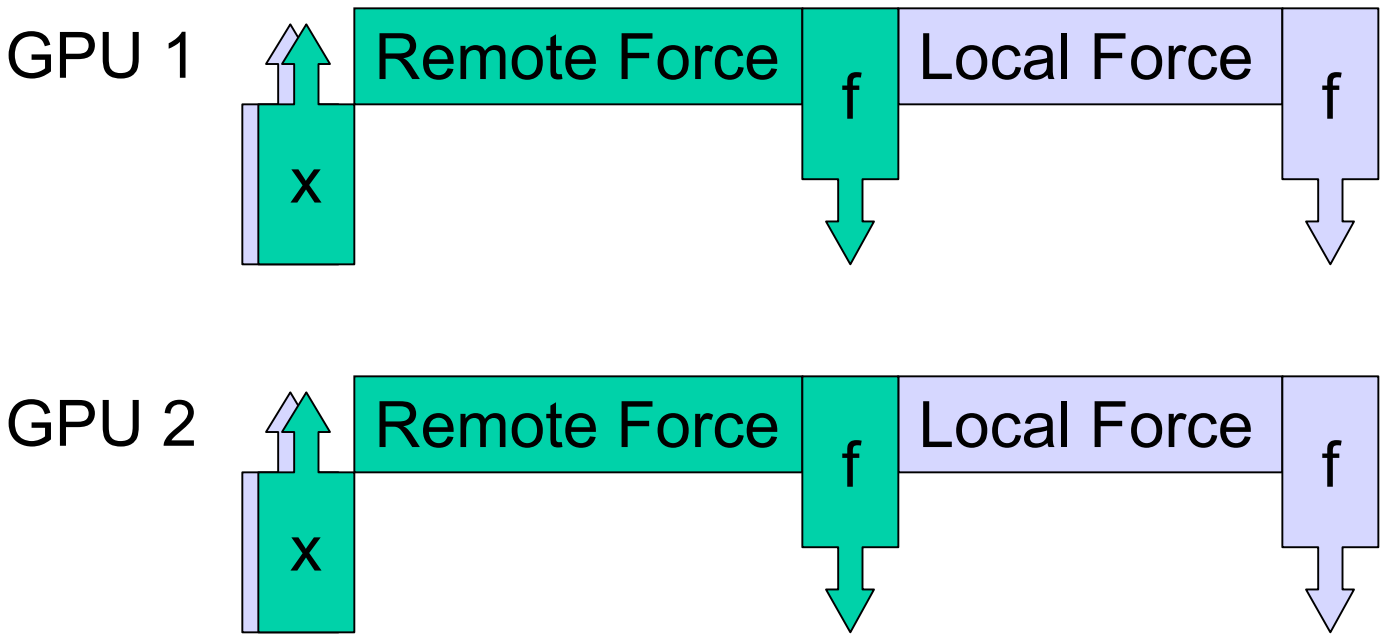


NCSA Lincoln Cluster Performance

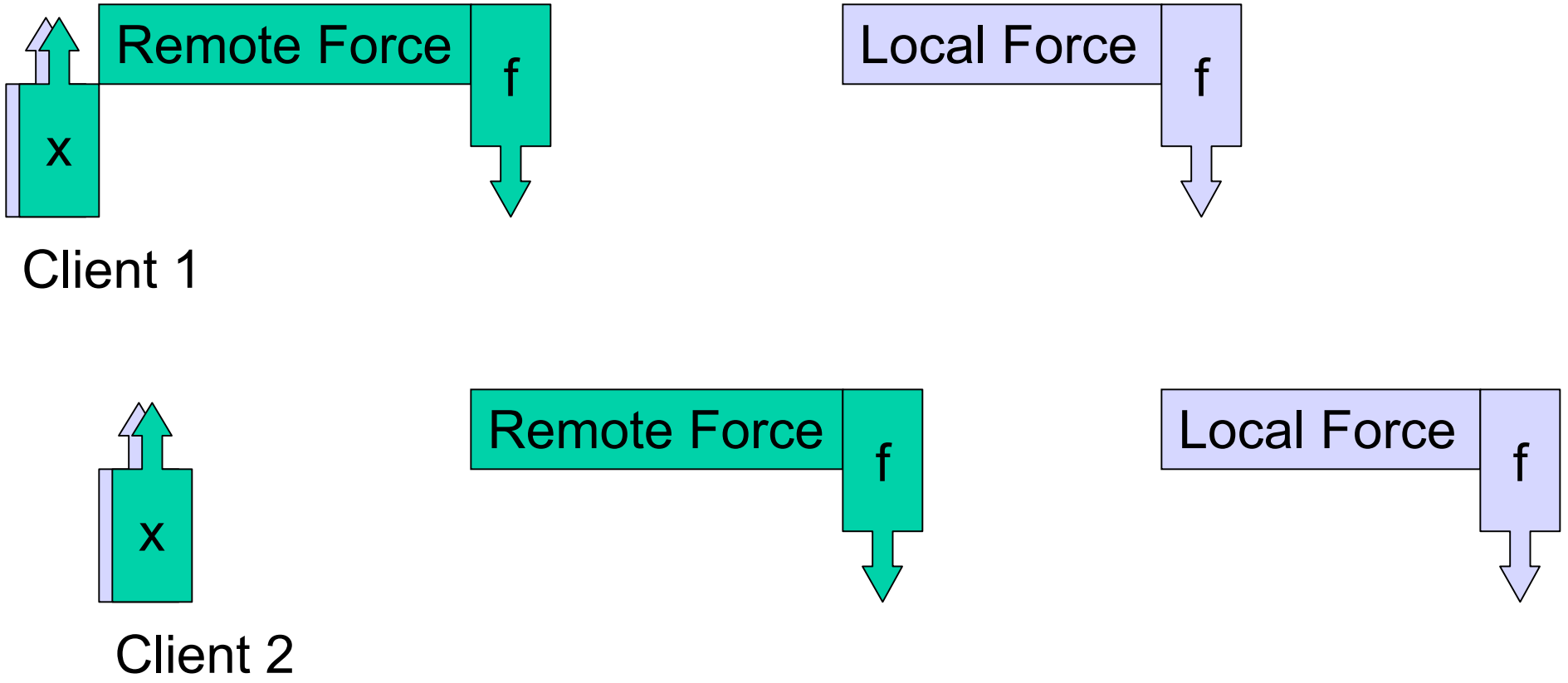
(8 cores and 2 GPUs per node)



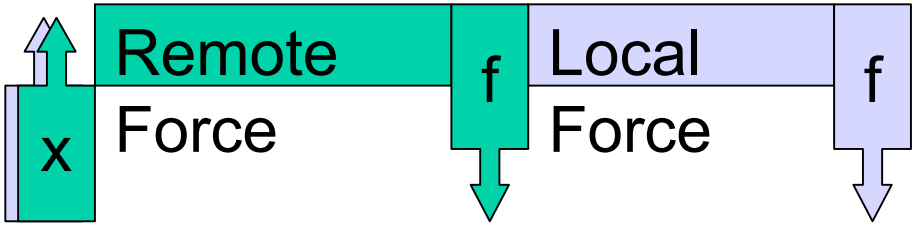
No GPU Sharing (Ideal World)



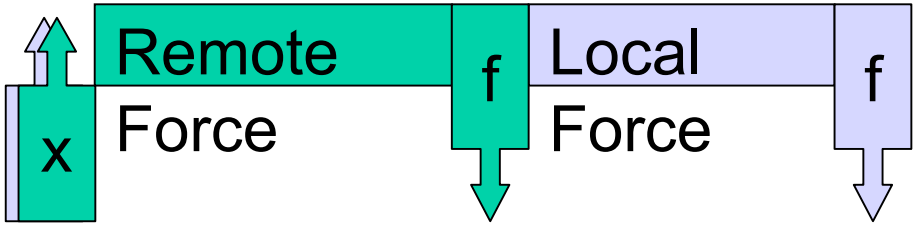
GPU Sharing (Desired)



GPU Sharing (Feared)

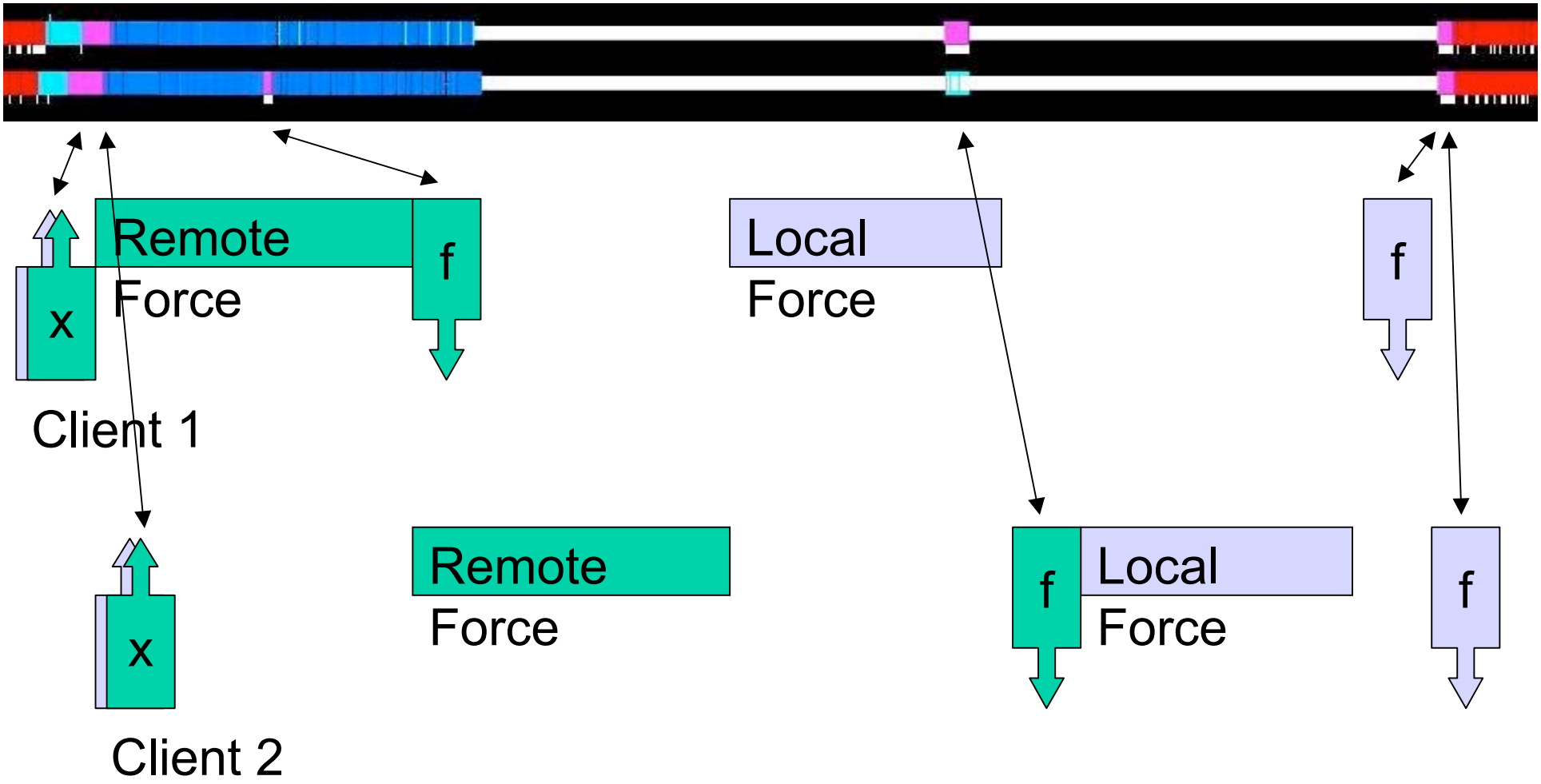


Client 1



Client 2

GPU Sharing (Observed)



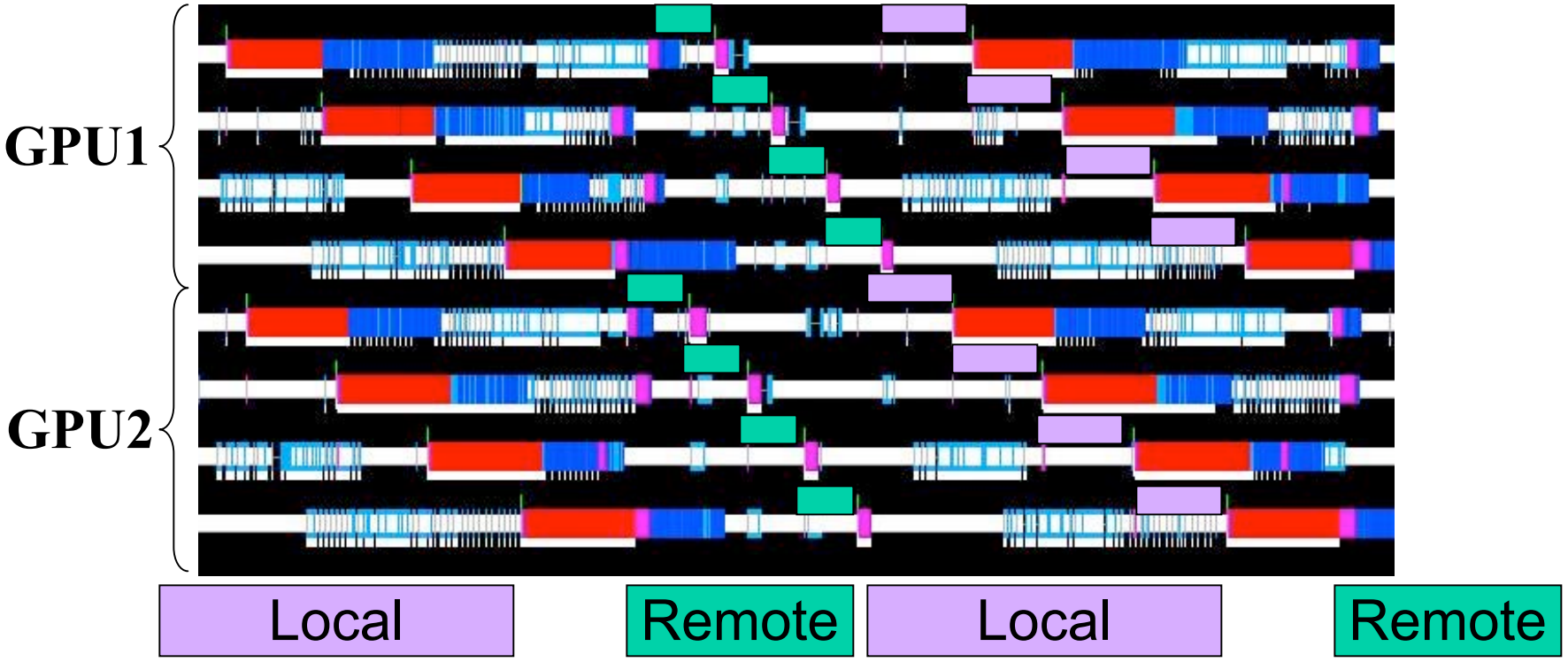
GPU Sharing (Explained)

- CUDA is behaving reasonably, but
- Force calculation is actually two kernels
  - Longer kernel writes to multiple arrays
  - Shorter kernel combines output
- Possible solutions:
  - Modify CUDA to be less “fair” (please!)
  - Use locks (atomics) to merge kernels (not G80)
  - Explicit inter-client coordination

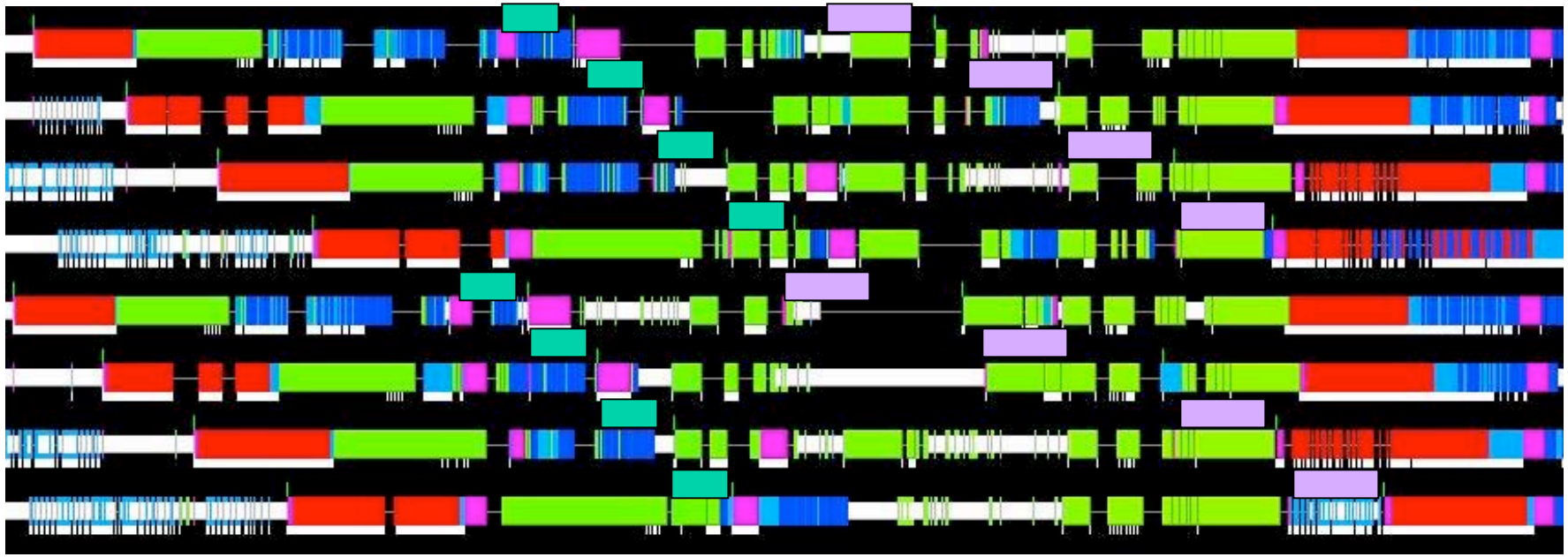
Inter-client Communication

- First identify which processes share a GPU
  - Need to know physical node for each process
  - GPU-assignment must reveal real device ID
  - Threads don't eliminate the problem
  - Production code can't make assumptions
- Token-passing is simple and predictable
  - Rotate clients in fixed order
  - High-priority, yield, low-priority, yield, …

Token-Passing GPU-Sharing



GPU-Sharing with PME



Weakness of Token-Passing

- GPU is idle while token is being passed
  - Busy client delays itself and others
- Next strategy requires threads:
  - One process per GPU, one thread per core
  - Funnel CUDA calls through a single stream
  - No local work until all remote work is queued
  - Typically funnels MPI as well

Recent NAMD GPU Developments

- Production features in 2.7b3 release:
  - Full electrostatics with PME
  - 1-4 exclusions
  - Constant-pressure simulation
  - Improved force accuracy:
    - Patch-centered atom coordinates
    - Increased precision of force interpolation
- Performance enhancements (in progress):
  - Recursive bisection within patch on 32-atom boundaries
  - Block-based pairlists based on sorted atoms
  - Sort blocks in order of decreasing work

GPU-Accelerated NAMD Issues

- Serial performance
  - Kernel developed in 2007 for G80 GPU, CUDA 1.0
  - Strict memory coalescing rules, no atomic operations
  - Remaining CPU code limits performance
- Parallel scaling
  - Distributing work to GPU – reuse CPU decomposition
  - Overlapping CPU, GPU, and network – SC08 paper
  - Sharing GPU between multiple CPU cores – unresolved
- Some advanced simulation methods not supported
  - GPU register memory restricts algorithm complexity

GPU-Accelerated NAMD Plans

- Serial performance
  - Target NVIDIA Fermi architecture
  - Revisit GPU kernel design decisions made in 2007
  - Improve performance of remaining CPU code
- Parallel scaling
  - Target NSF Track 2D Keeneland cluster at ORNL
  - Finer-grained work units on GPU (feature of Fermi)
  - One process per GPU, one thread per CPU core
  - Dynamic load balancing of GPU work
- Wider range of simulation options and features

Conclusions and Outlook

- CUDA today is sufficient for
  - Single-GPU acceleration (the mass market)
  - Coarse-grained multi-GPU parallelism
    - Enough work per call to spin up all multiprocessors
- Improvements in CUDA are needed for
  - Assigning GPUs to processes
  - Sharing GPUs between processes
  - Fine-grained multi-GPU parallelism
    - Fewer blocks per call than chip has multiprocessors
  - Moving data between GPUs (same or different node)
- Fermi addresses some but not all of these

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
- Mike Showerman, Jeremy Enos, NCSA
- David Kirk, Massimiliano Fatica, others at NVIDIA
- UIUC NVIDIA CUDA Center of Excellence
- NIH support: P41-RR05969

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