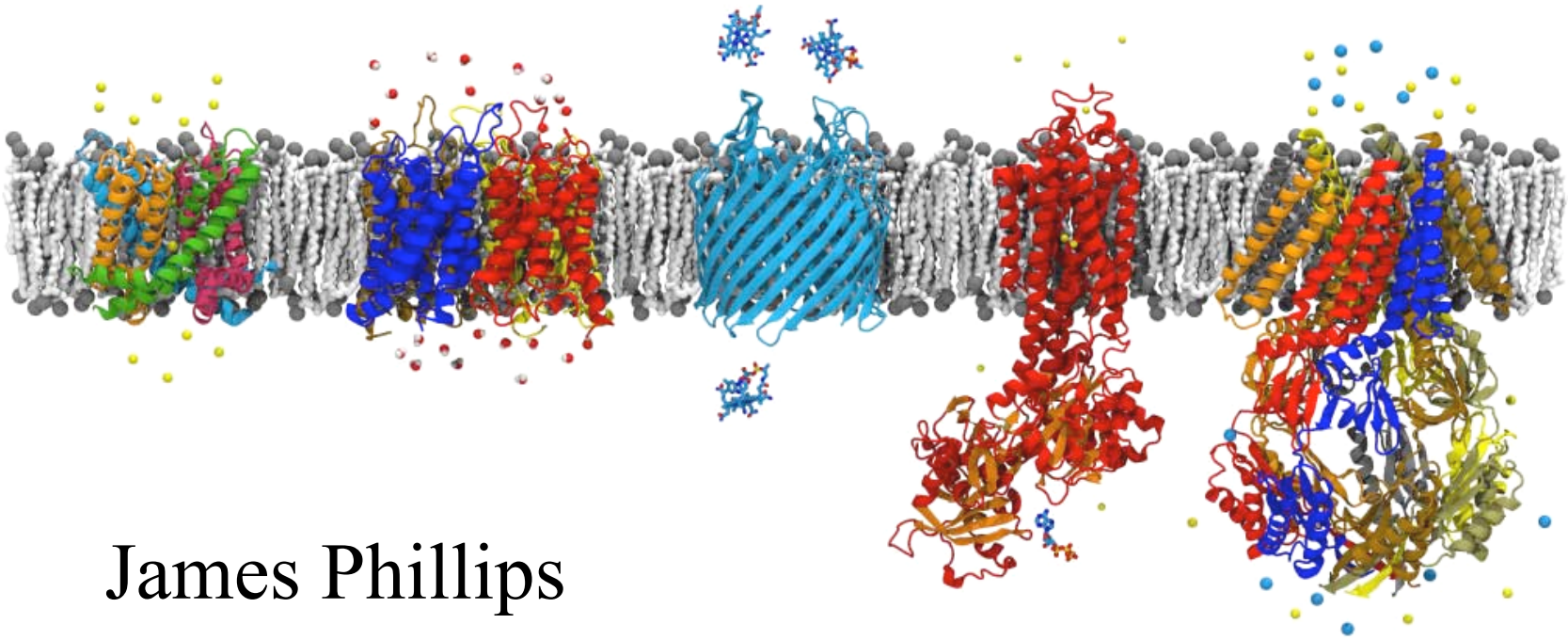
NAMD, CUDA, and Clusters: Taking GPU Molecular Dynamics Beyond the Deskop

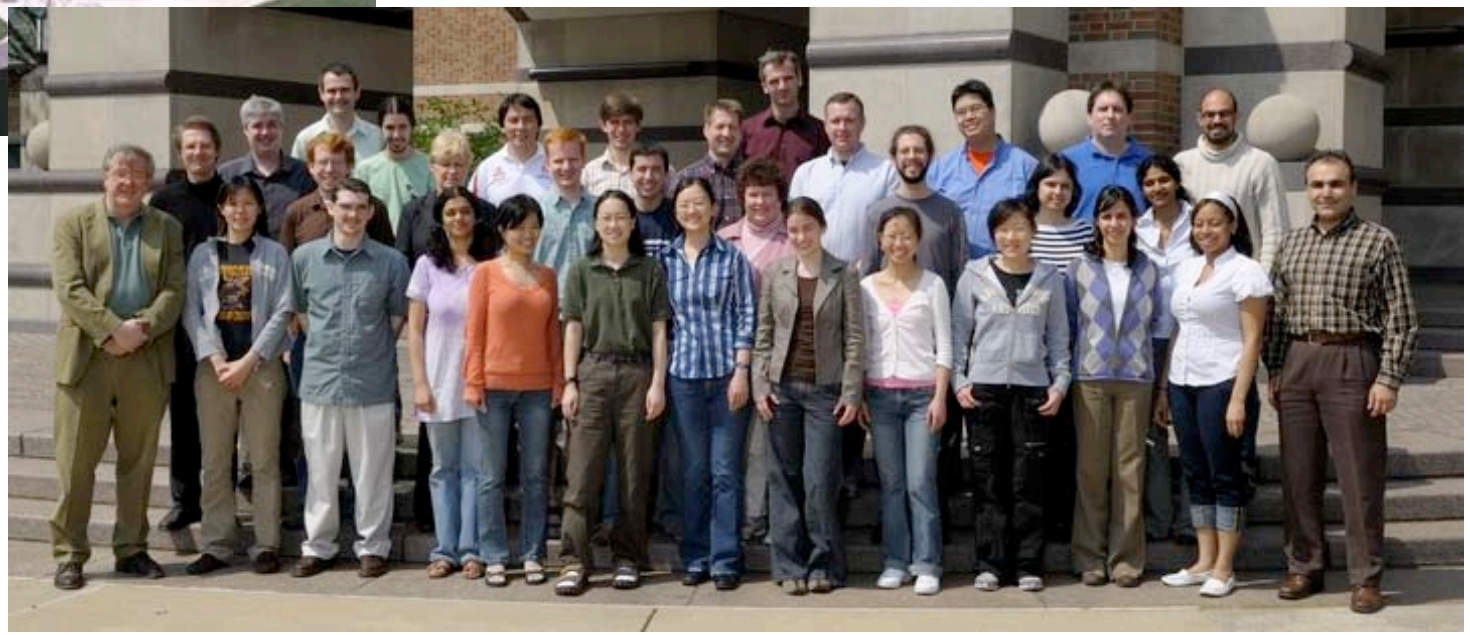


James Phillips

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

Beckman Institute
University of Illinois at Urbana-Champaign

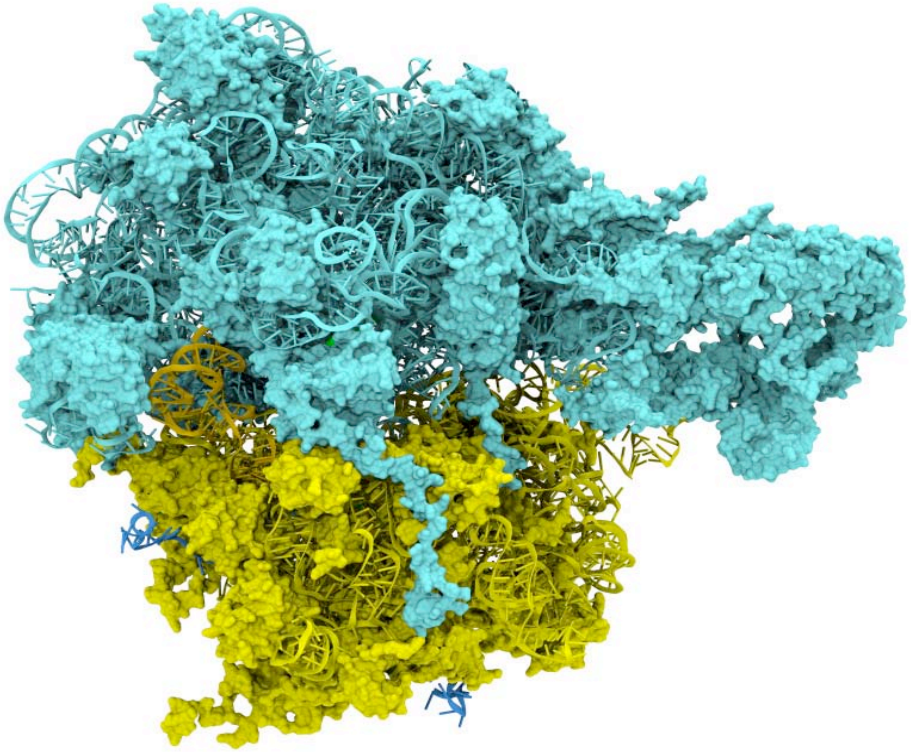
Theoretical and Computational Biophysics Group



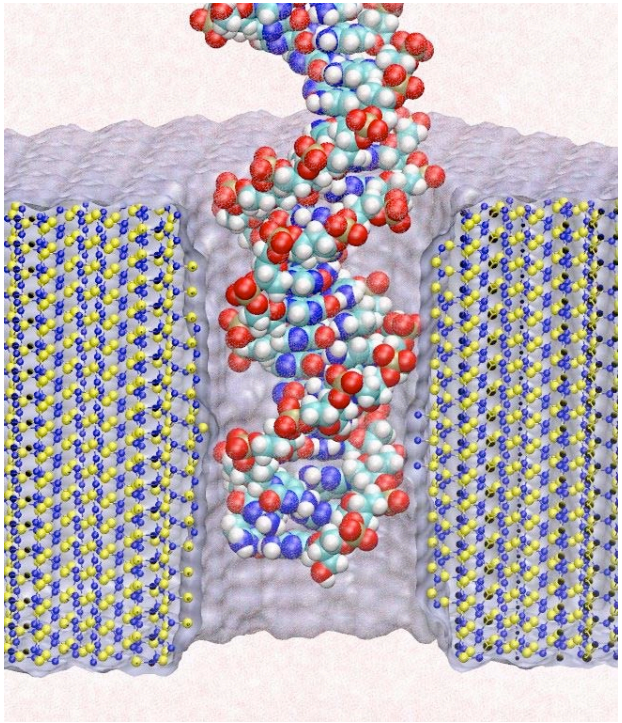
National Center for Research Resources

Computational Microscopy

Ribosome: synthesizes proteins from genetic information, target for antibiotics

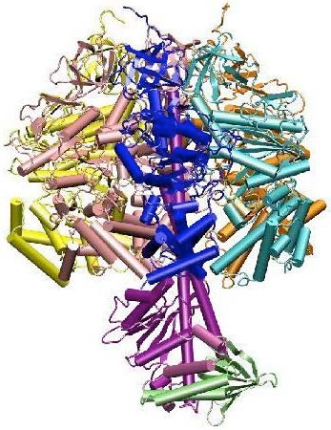


Silicon nanopore: bionanodevice for sequencing DNA efficiently



NAMD: Scalable Molecular Dynamics

2002 Gordon Bell Award



ATP synthase

PSC Lemieux

40,000 Users, 1700 Citations

Computational Biophysics Summer School

Blue Waters Target Application

Illinois Petascale Computing Facility

GPU Acceleration

NVIDIA Tesla

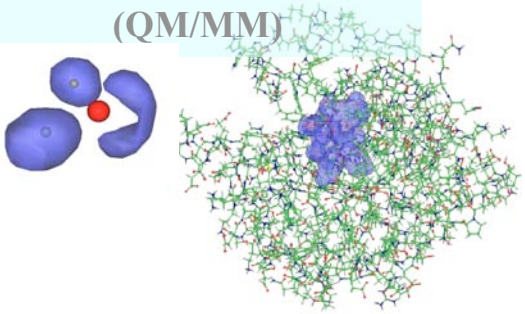
NCSA Lincoln

Parallel Programming Lab
University of Illinois at Urbana-Champaign

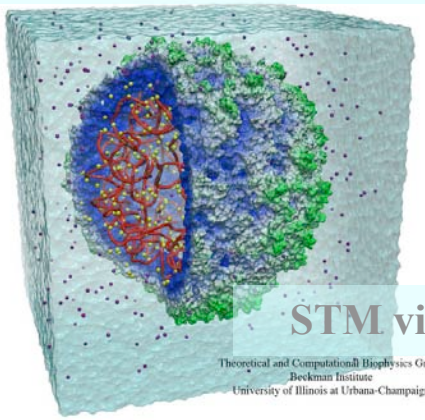
Siebel Center for Computer Science

Develop abstractions in context of full-scale applications

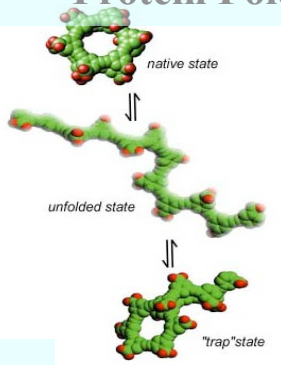
Quantum Chemistry (QM/MM)



NAMD: Molecular Dynamics

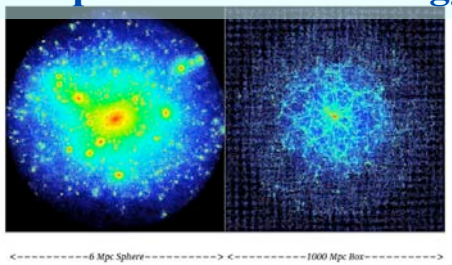


Protein Folding



STM virus simulation

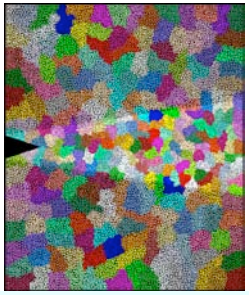
Computational Cosmology



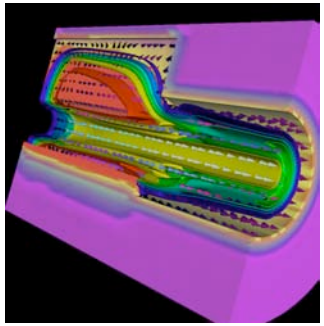
Parallel Objects,

Adaptive Runtime System

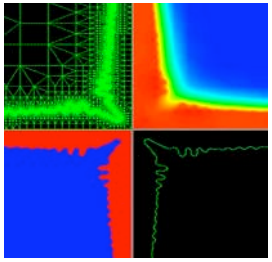
Libraries and Tools



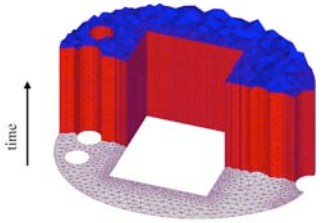
Crack Propagation



Rocket Simulation



Dendritic Growth

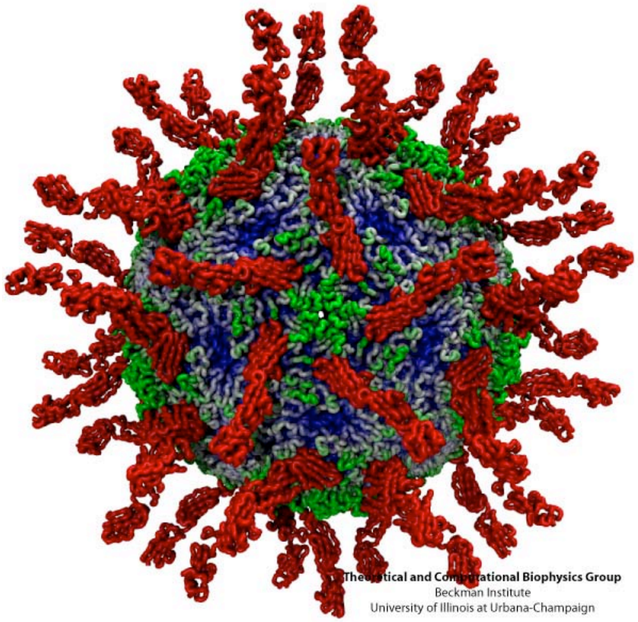


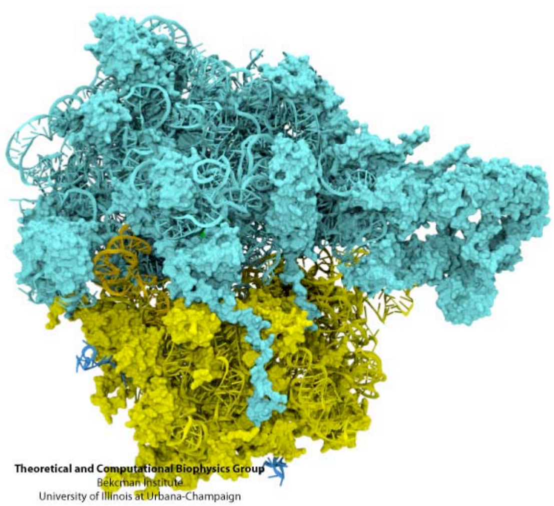
Space-time meshes

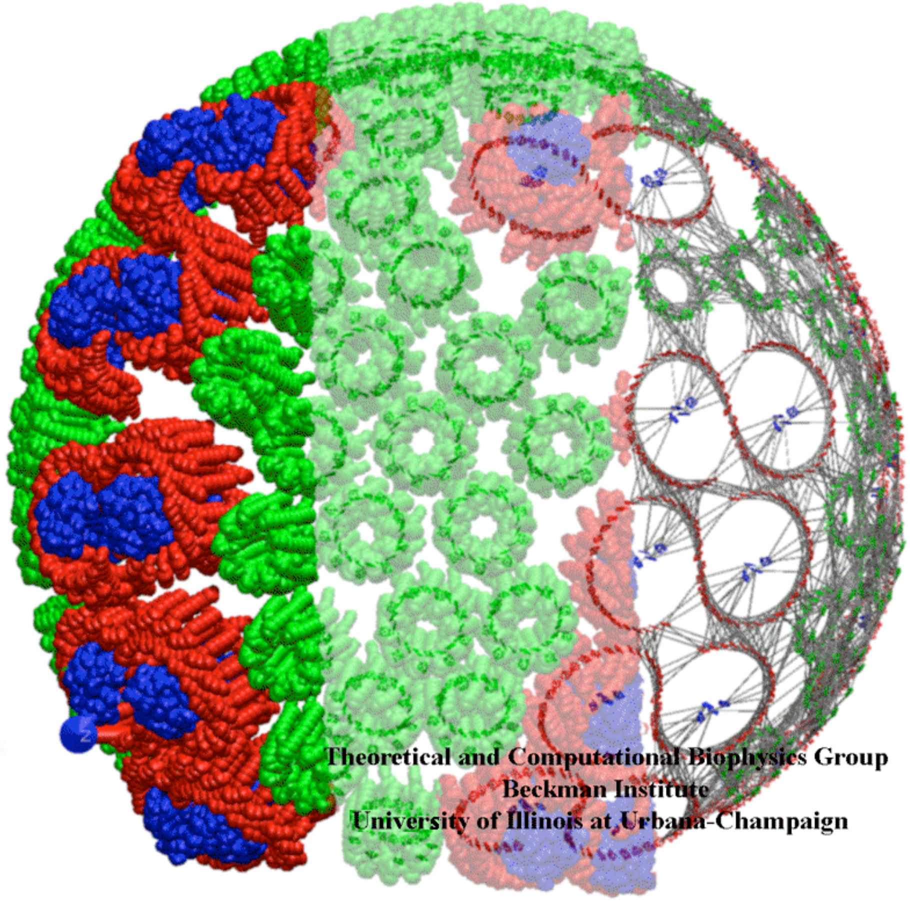
The enabling CS technology of parallel objects and intelligent runtime systems has led to several collaborative applications in CSE

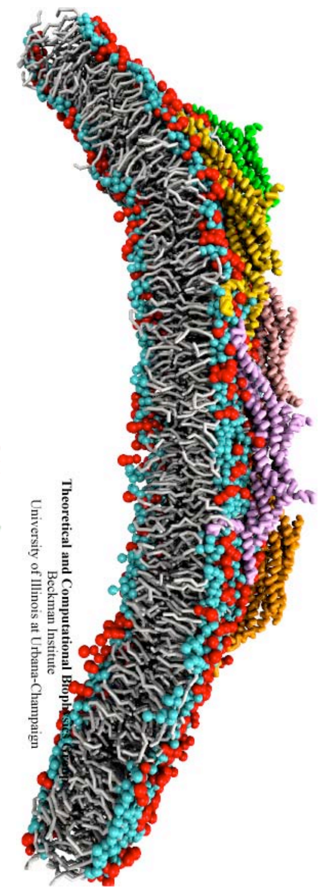
NAMD Petascale Preparations

Planned Petascale Simulations









NAMD: Practical Supercomputing

- 40,000 users can't all be computer experts.
  - 18% are NIH-funded; many in other countries.
  - 10,000 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

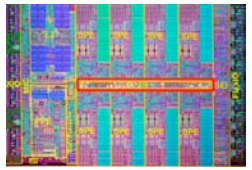
Phillips *et al.*, *J. Comp. Chem.* **26**:1781-1802, 2005.

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

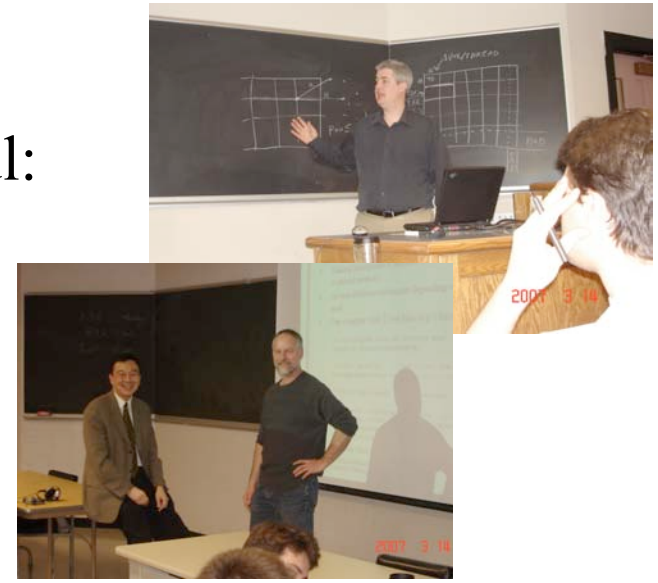


CUDA: Practical Performance

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)

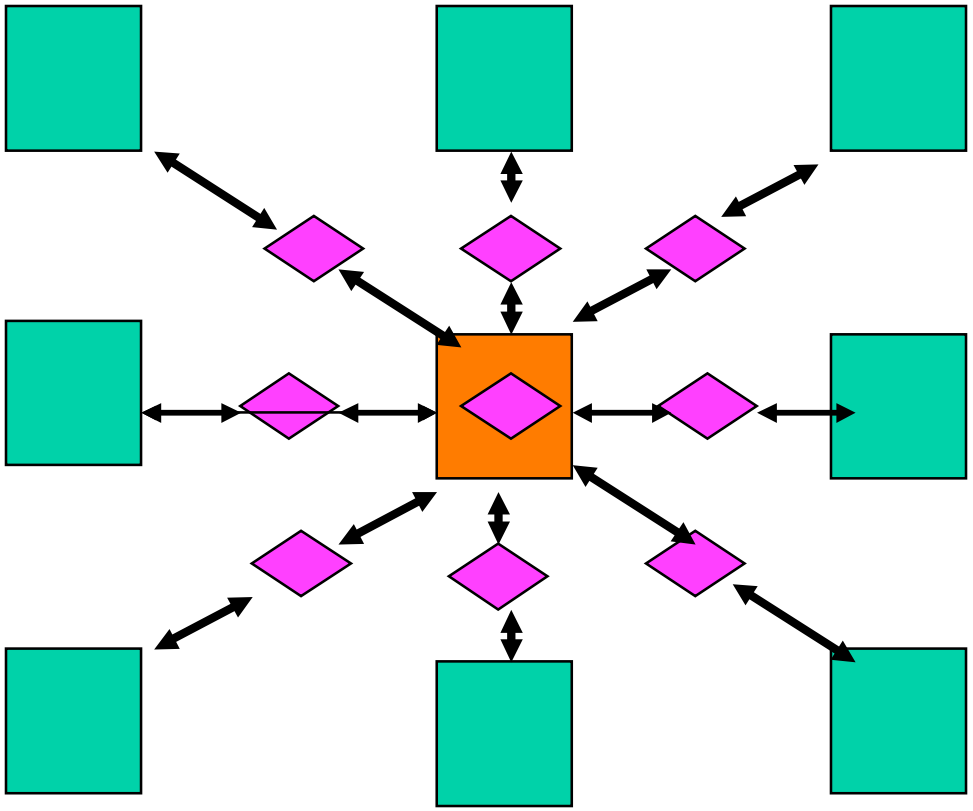
Fun to program (and drive)



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

NAMD Hybrid Decomposition

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



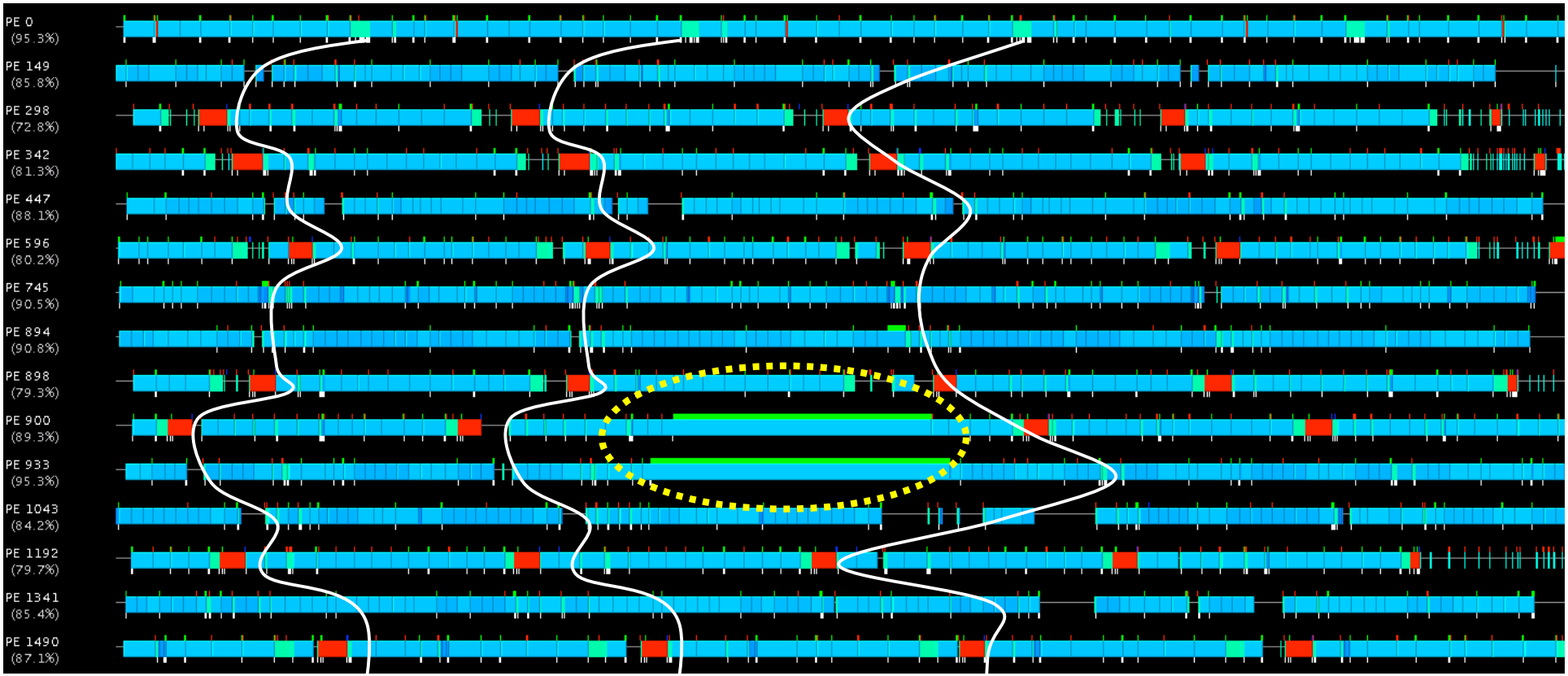
- Spatially decompose data and communication.
- Separate but related work decomposition.
- "Compute objects" facilitate iterative, measurement-based load balancing system.

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++
  - Can be emulated in MPI
  - Charm++ provides tools and idioms
  - Parallel Programming Lab: http://charm.cs.uiuc.edu/

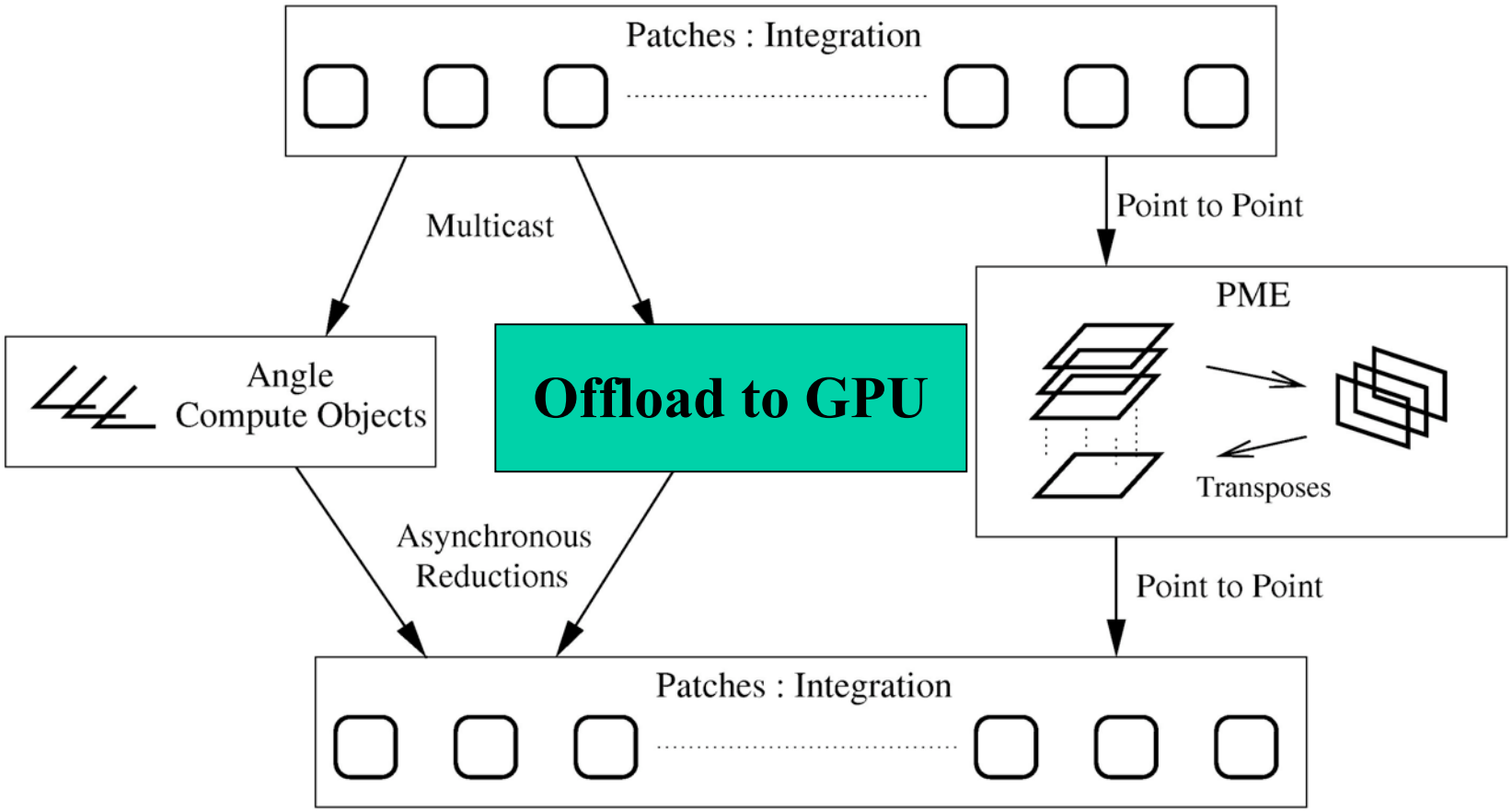
System Noise Example

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



NAMD Overlapping Execution

Phillips *et al., SC2002*.



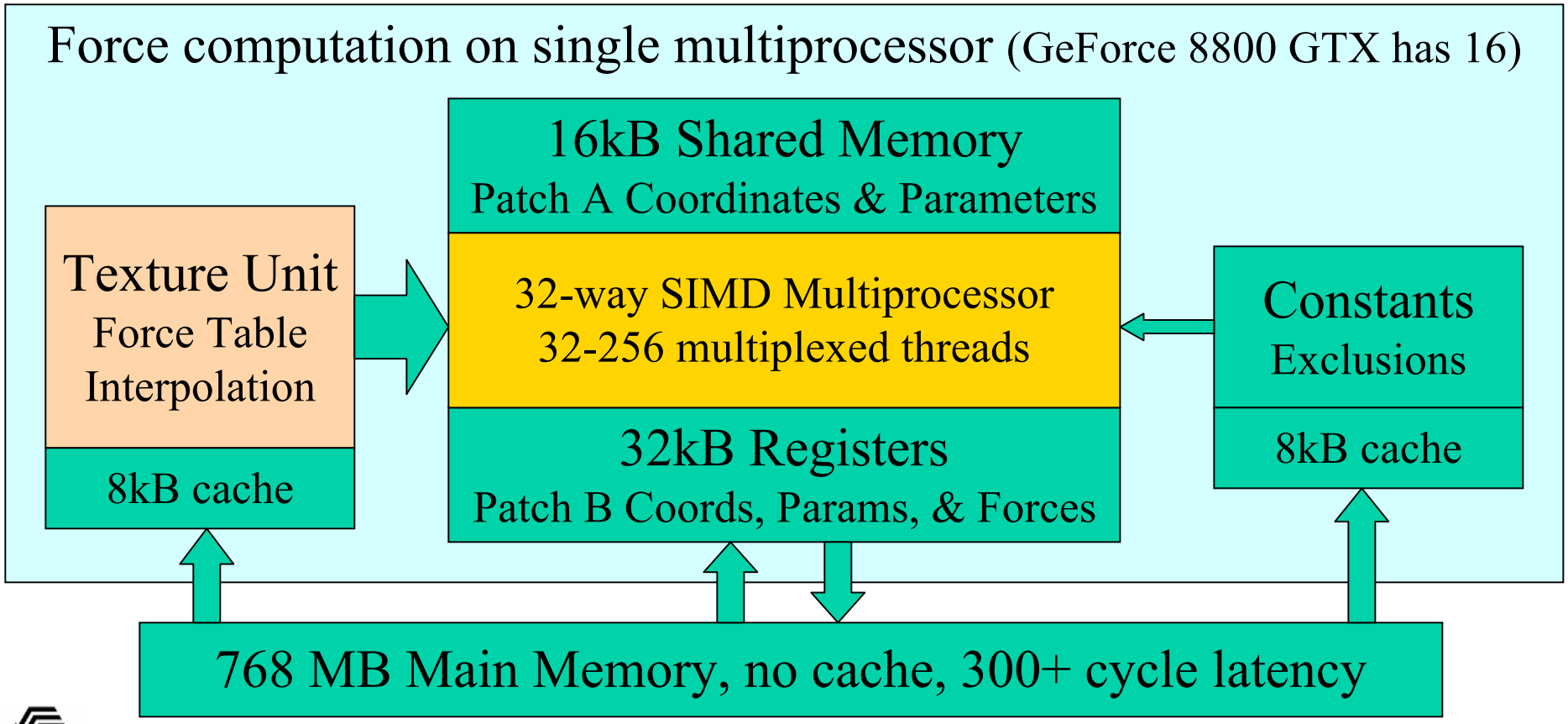
Objects are assigned to processors and queued as data arrives.

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.
- May be possible to implement on Fermi

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.



National Center for Research Resources

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

Nonbonded Forces CUDA Code

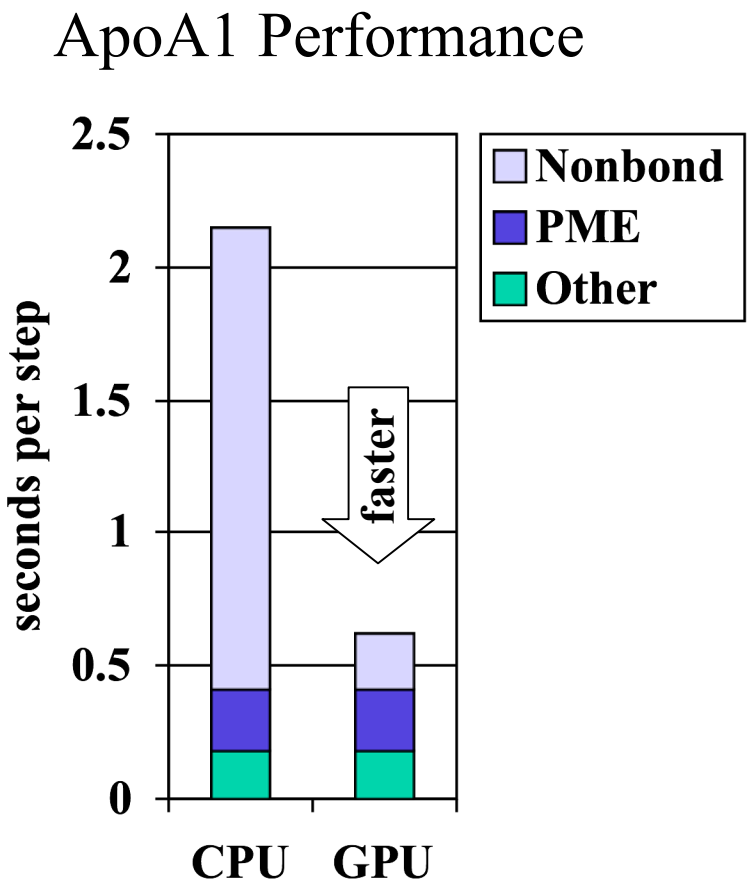
```
texture<float4> force_table;
__constant__ unsigned int exclusions[];
__shared__ atom jatom[];
atom iatom;      // per-thread atom, stored in registers
float4 iforce;   // per-thread force, stored in registers
for ( int j = 0; j < jatom_count; ++j ) {
  float dx = jatom[j].x - iatom.x;   float dy = 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));                          // Force Interpolation
    bool excluded = false;                                                    // Exclusions
    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                 // Parameters
    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;          // Accumulation
    iforce.w += 1.f;  // interaction count or energy
  }
}
```

CUDA Kernel Evolution

- Original - minimize main memory access
  - Enough threads to load all atoms in patch
  - Needed two atoms per thread to fit
  - Swap atoms between shared and registers
- Revised - multiple blocks for concurrency
  - 64 threads/atoms per block (now 128 for Fermi)
  - Loop over shared memory atoms in sets of 16
  - Two blocks for each patch pair

Initial GPU Performance (2007)

- Full NAMD, not test harness
- Useful performance boost
  - 8x speedup for nonbonded
  - 5x speedup overall w/o PME
  - 3.5x speedup overall w/ PME
  - GPU = quad-core CPU
- Plans for better performance
  - Overlap GPU and CPU work.
  - Tune or port remaining work.
    - PME, bonded, integration, etc.

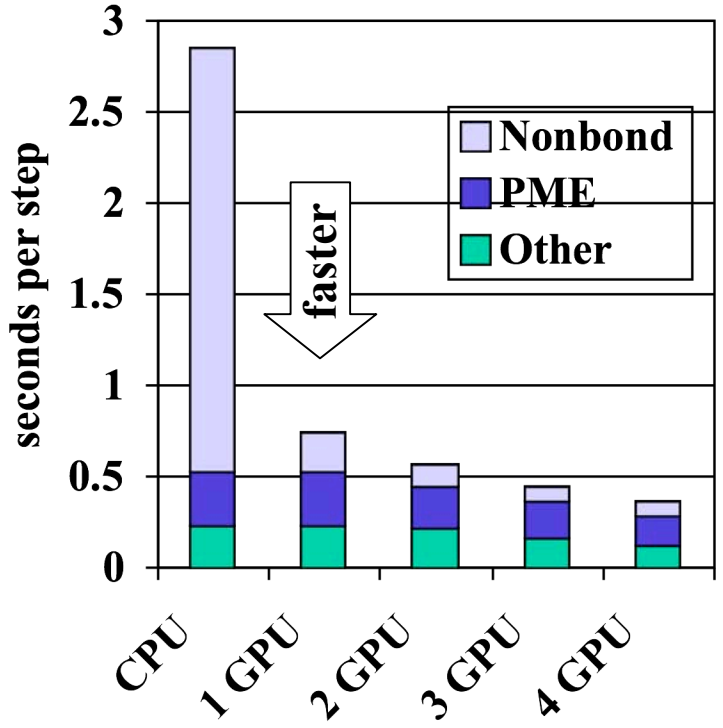


2.67 GHz Core 2 Quad Extreme + GeForce 8800 GTX

2007 GPU Cluster Performance

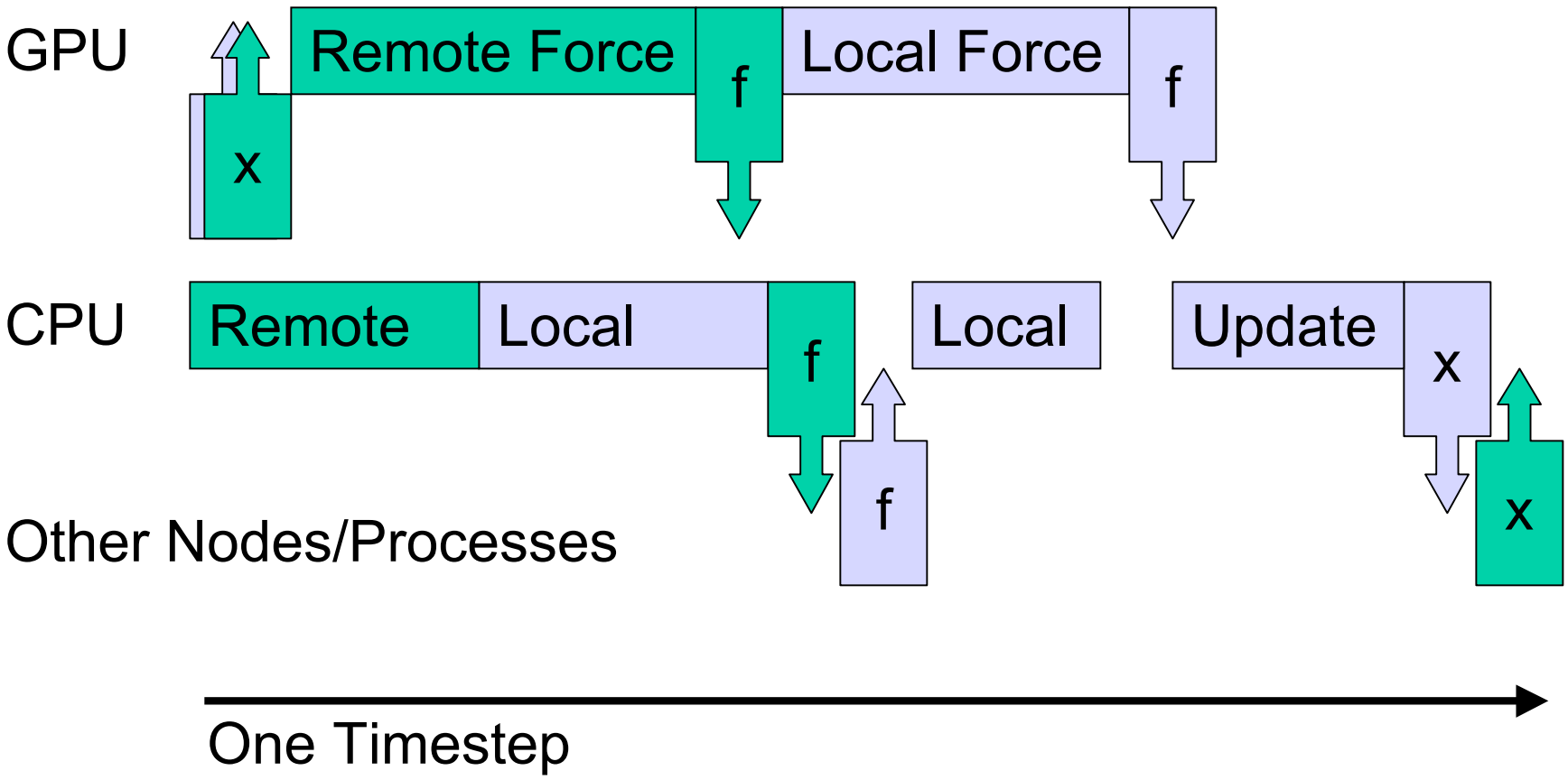
- Poor scaling unsurprising
  - 2x speedup on 4 GPUs
  - Gigabit ethernet
  - Load balancer disabled
- Plans for better scaling
  - InfiniBand network
  - Tune parallel overhead
  - Load balancer changes
    - Balance GPU load.
    - Minimize communication.

ApoA1 Performance



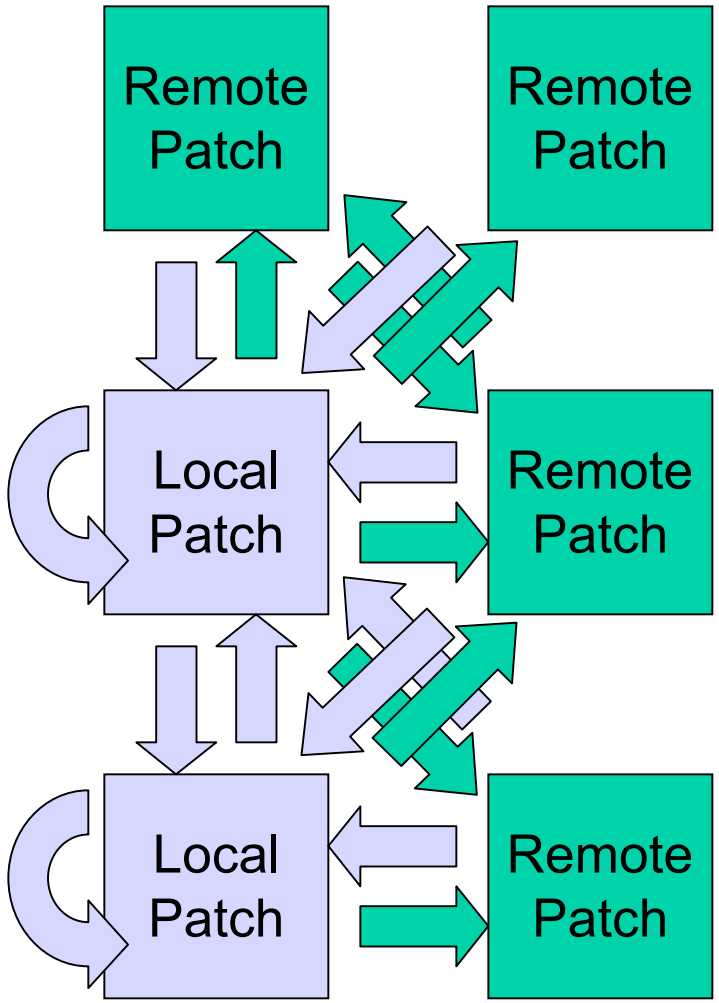
2.2 GHz Opteron + GeForce 8800 GTX

Overlapping GPU and CPU with Communication



“Remote Forces”

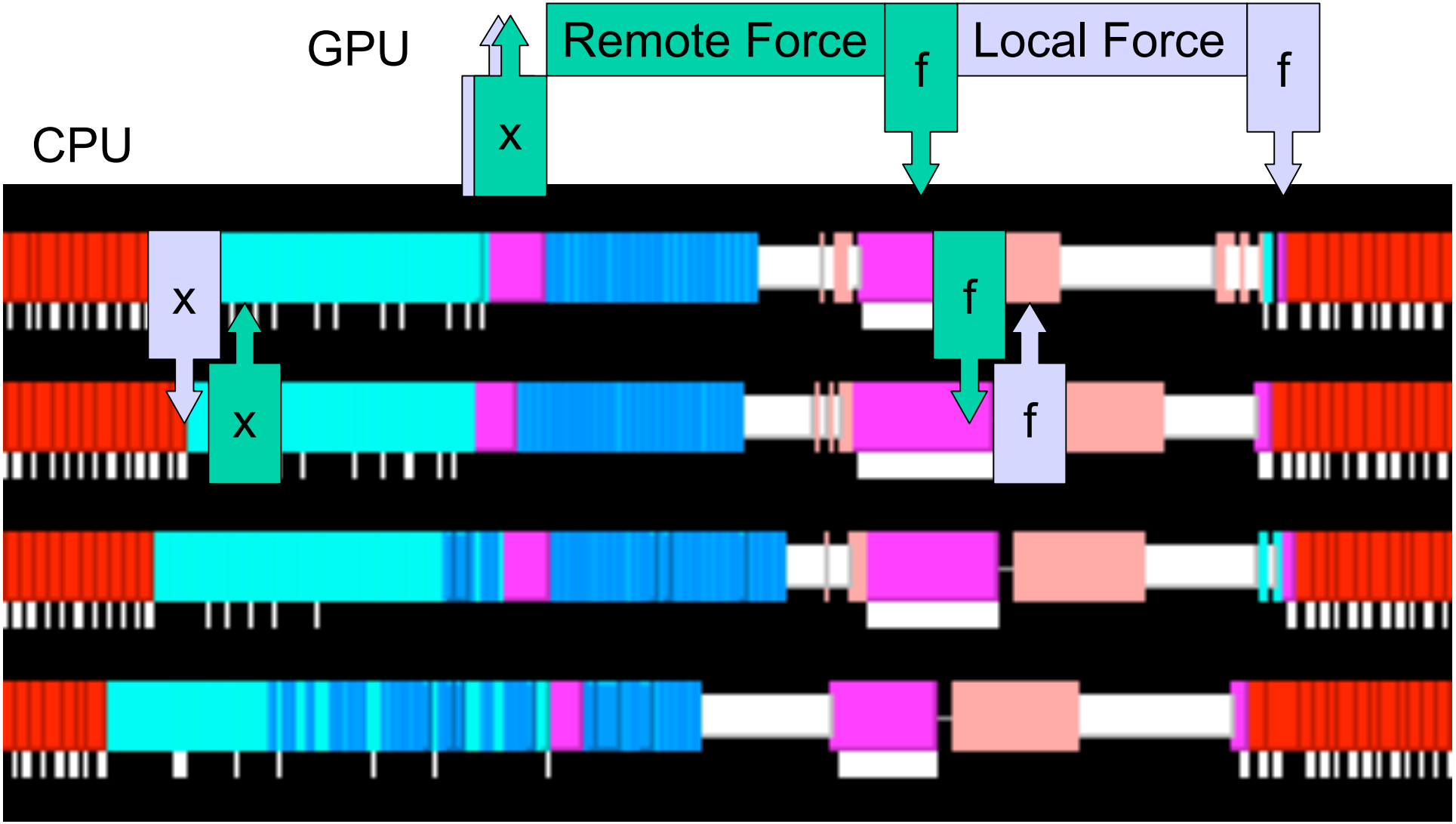
- Forces on atoms in a local patch are “local”
- Forces on atoms in a remote patch are “remote”
- Calculate remote forces first to overlap force communication with local force calculation
- Not enough work to overlap with position communication



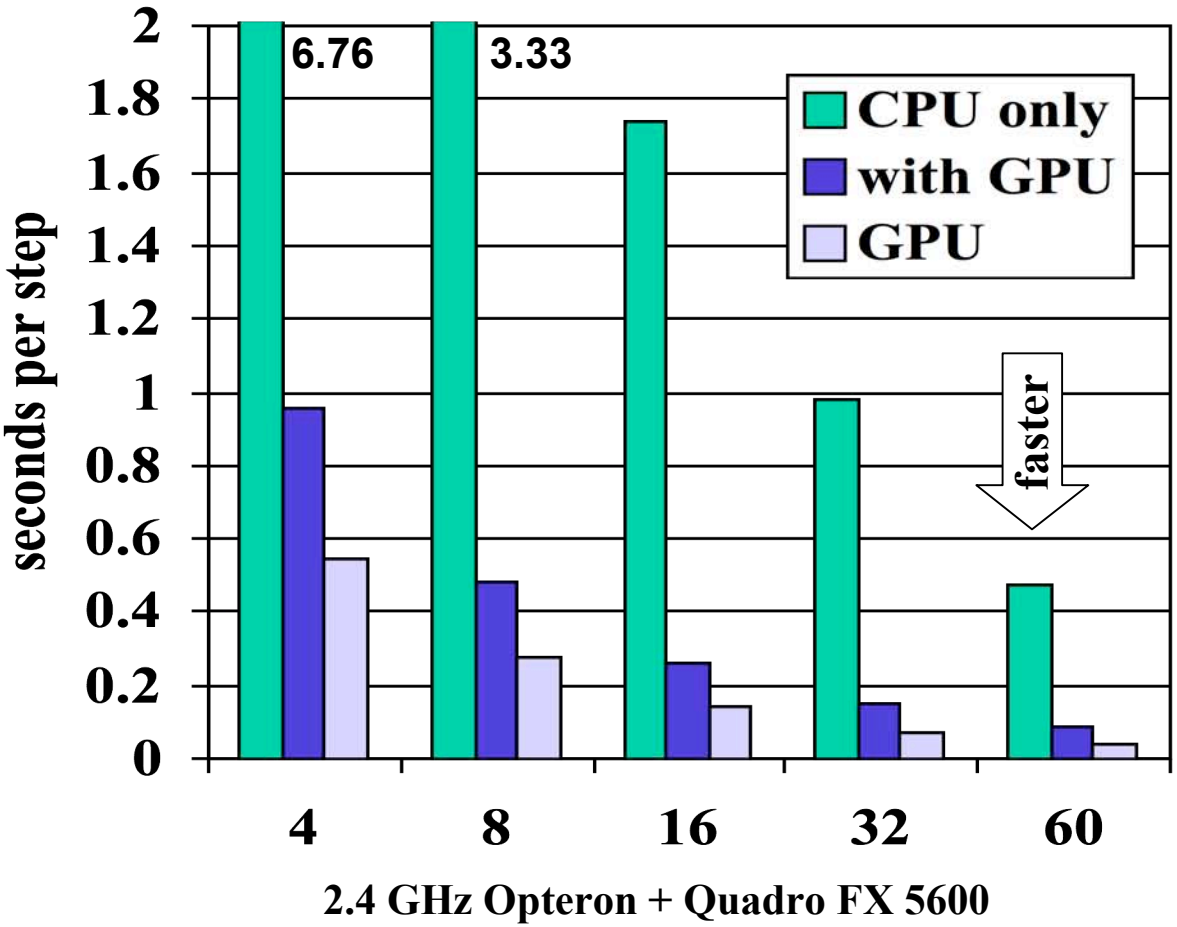
Work done by **one** processor

Actual Timelines from NAMD

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



NCSA “4+4” QP Cluster

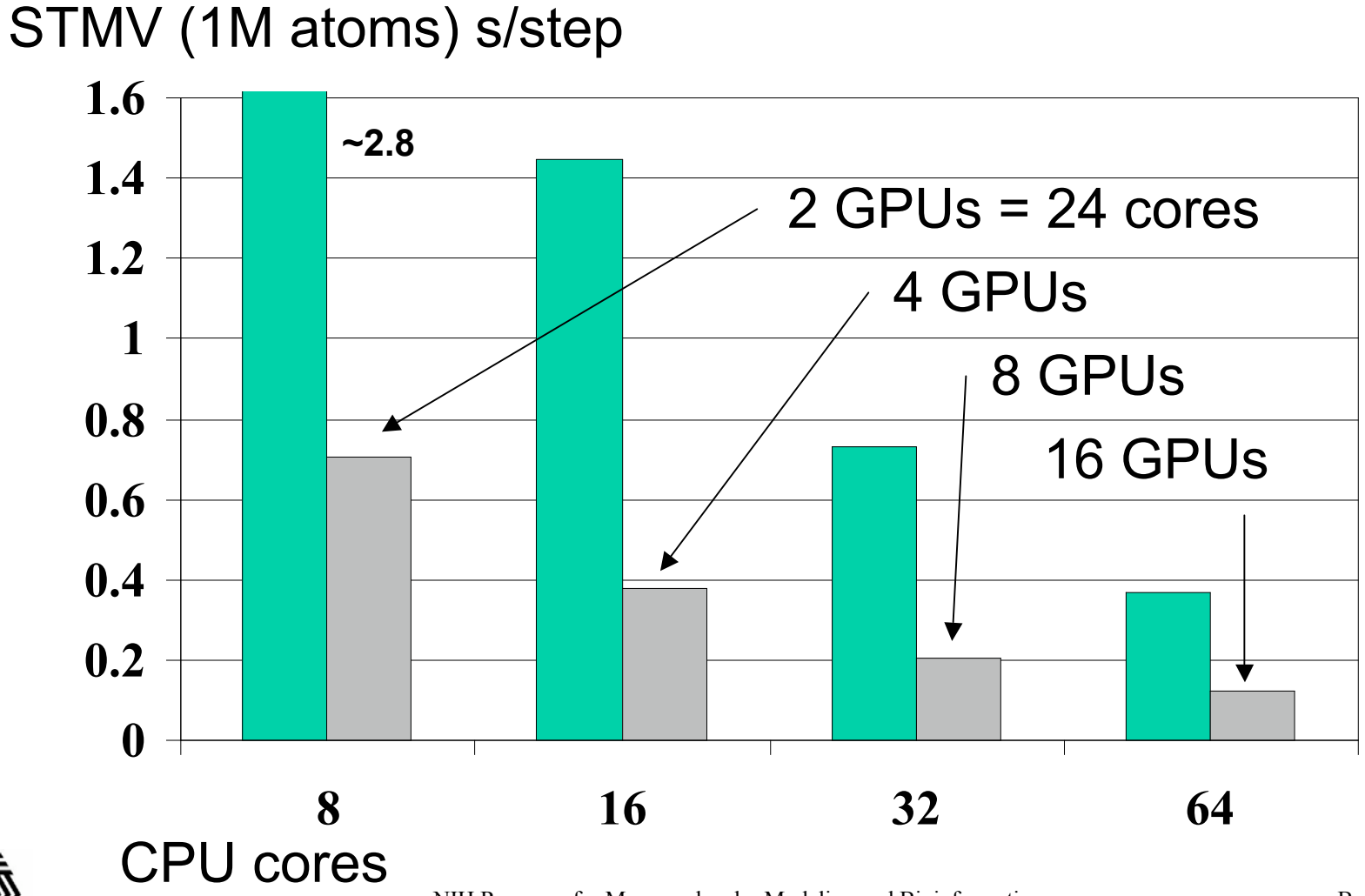


NCSA “8+2” Lincoln Cluster

- 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?

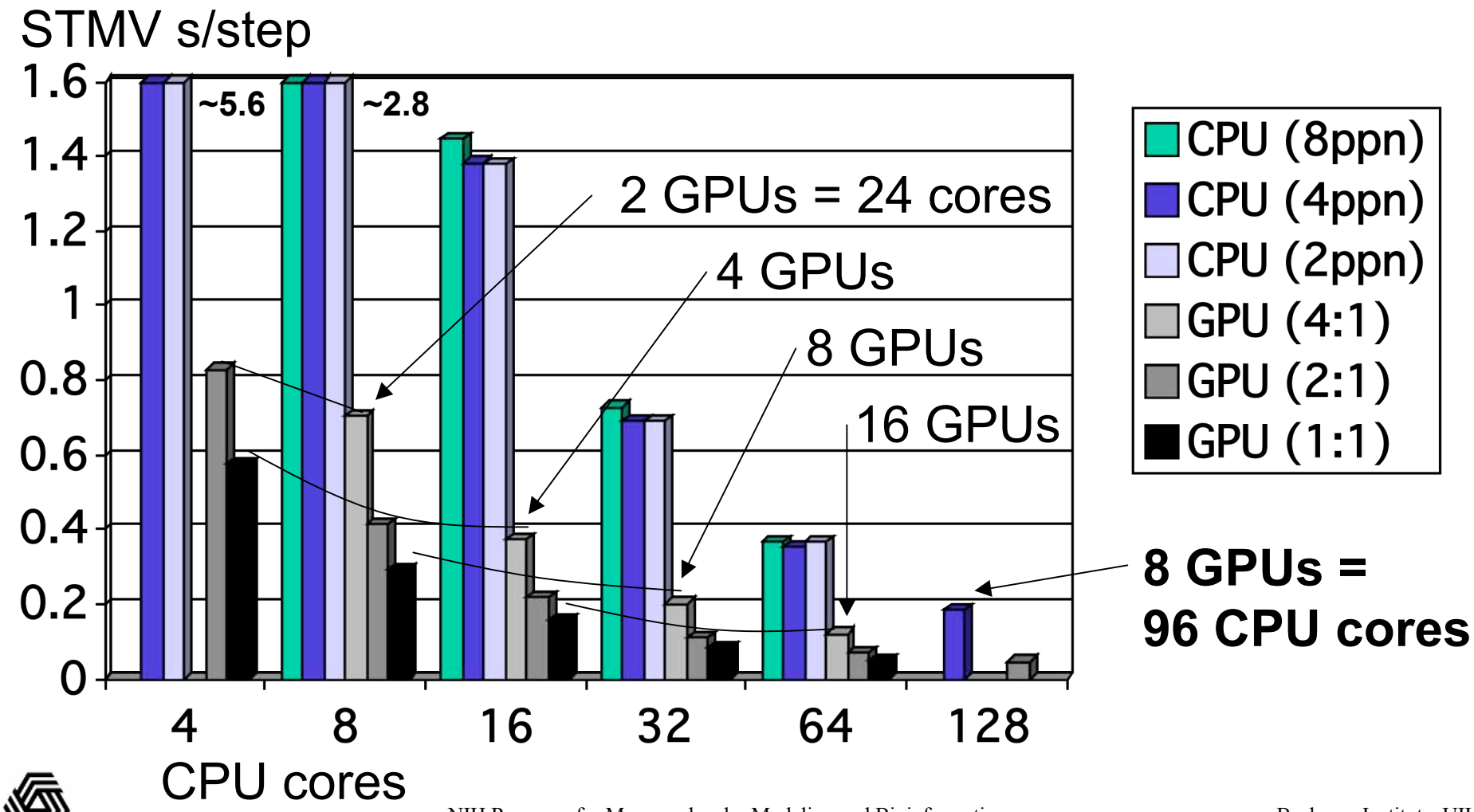
NCSA Lincoln Cluster Performance

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

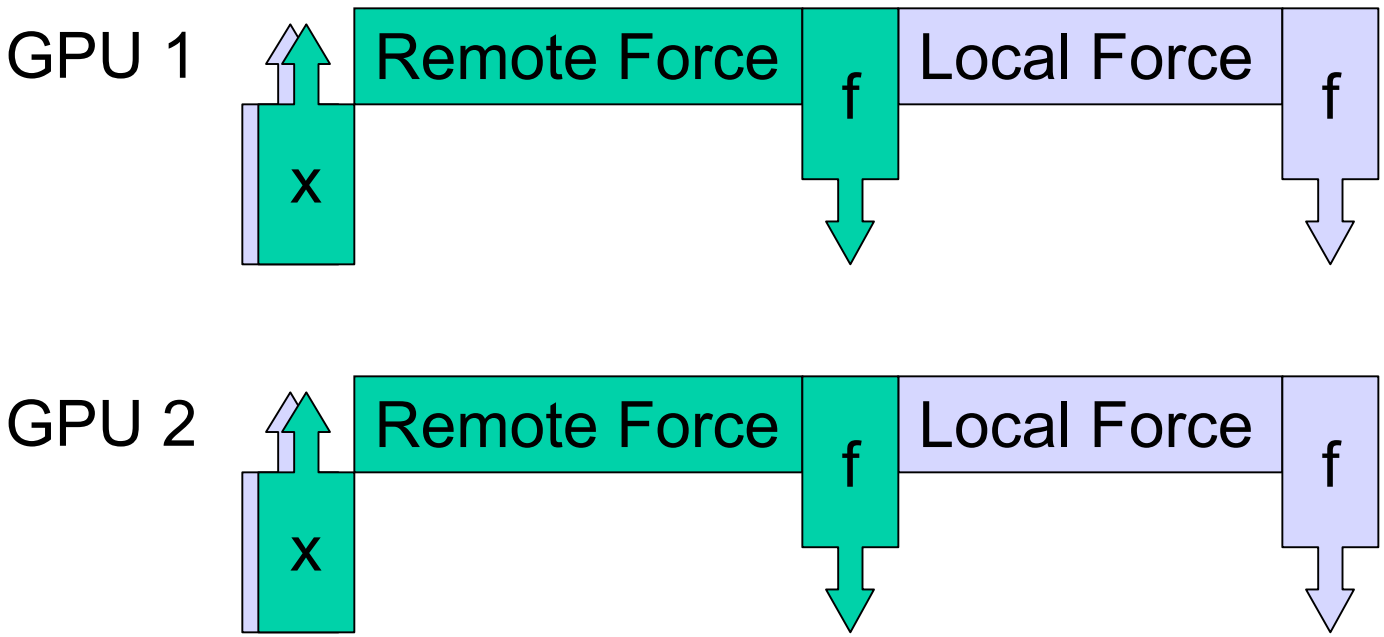


NCSA Lincoln Cluster Performance

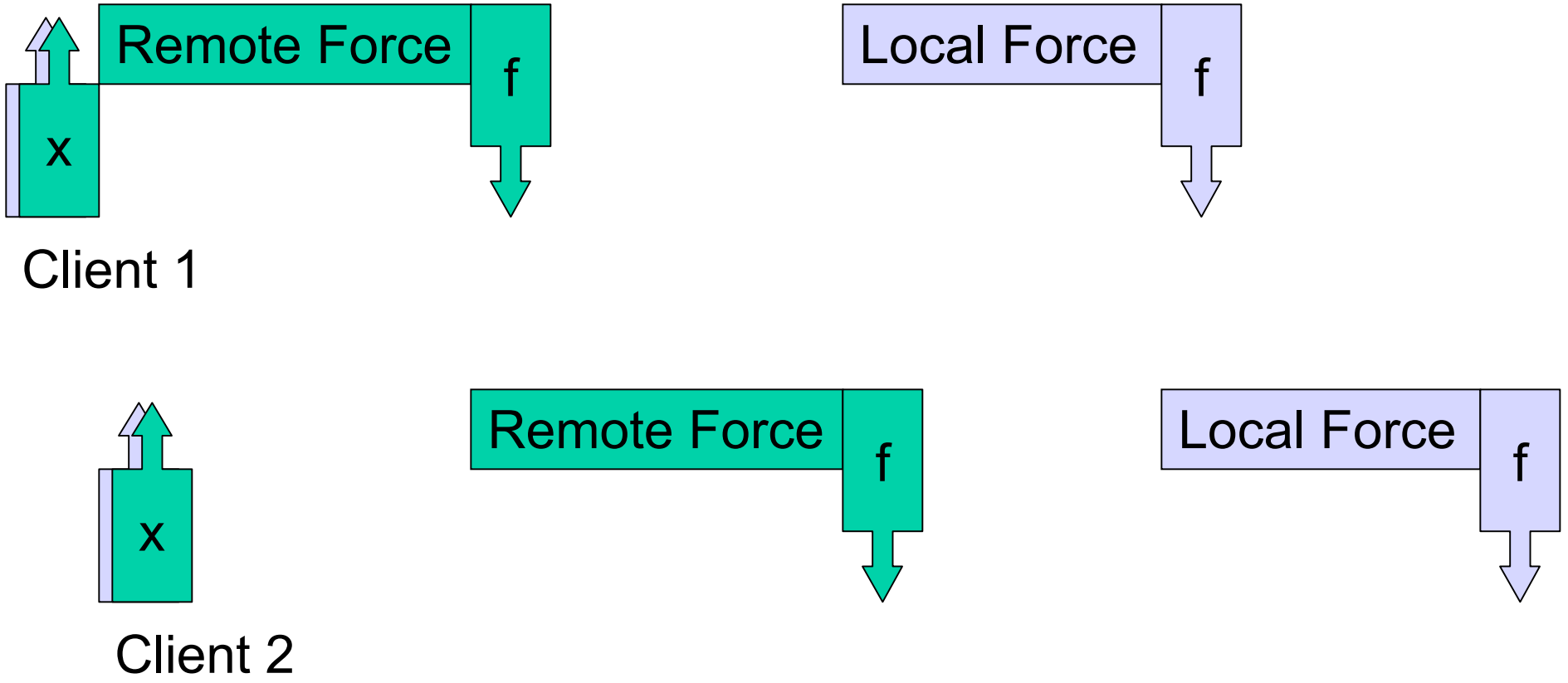
(8 cores and 2 GPUs per node)



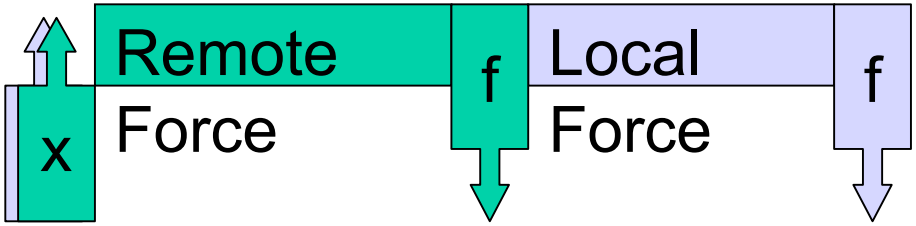
No GPU Sharing (Ideal World)



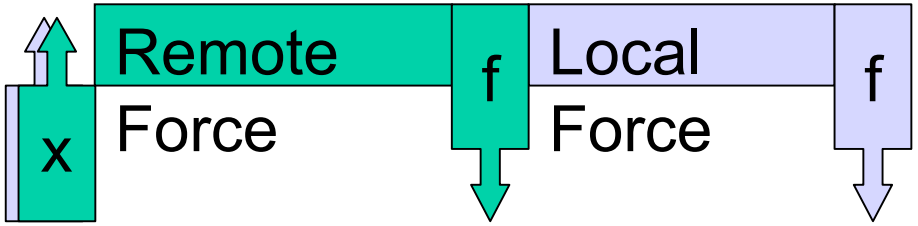
GPU Sharing (Desired)



GPU Sharing (Feared)

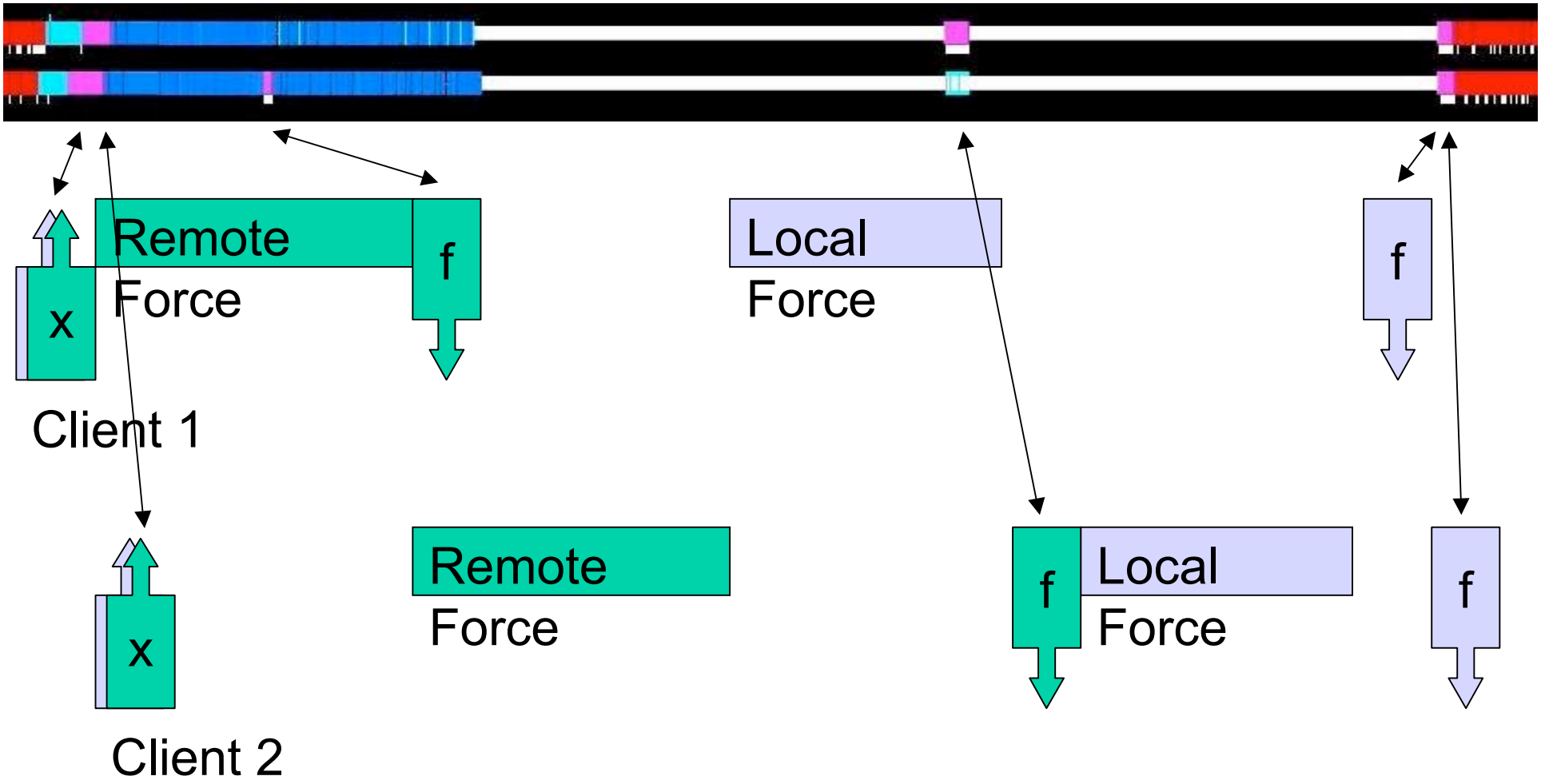


Client 1



Client 2

GPU Sharing (Observed)



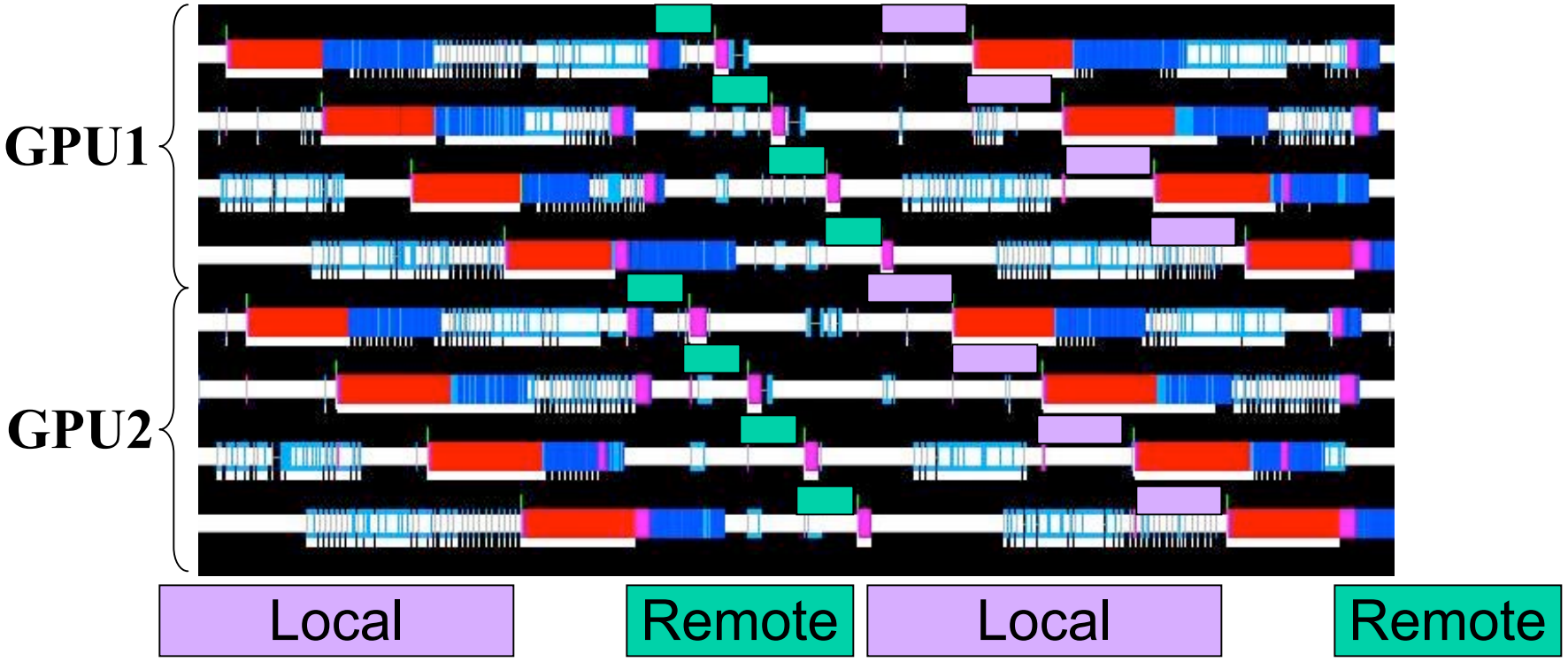
GPU Sharing (Explained)

- 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

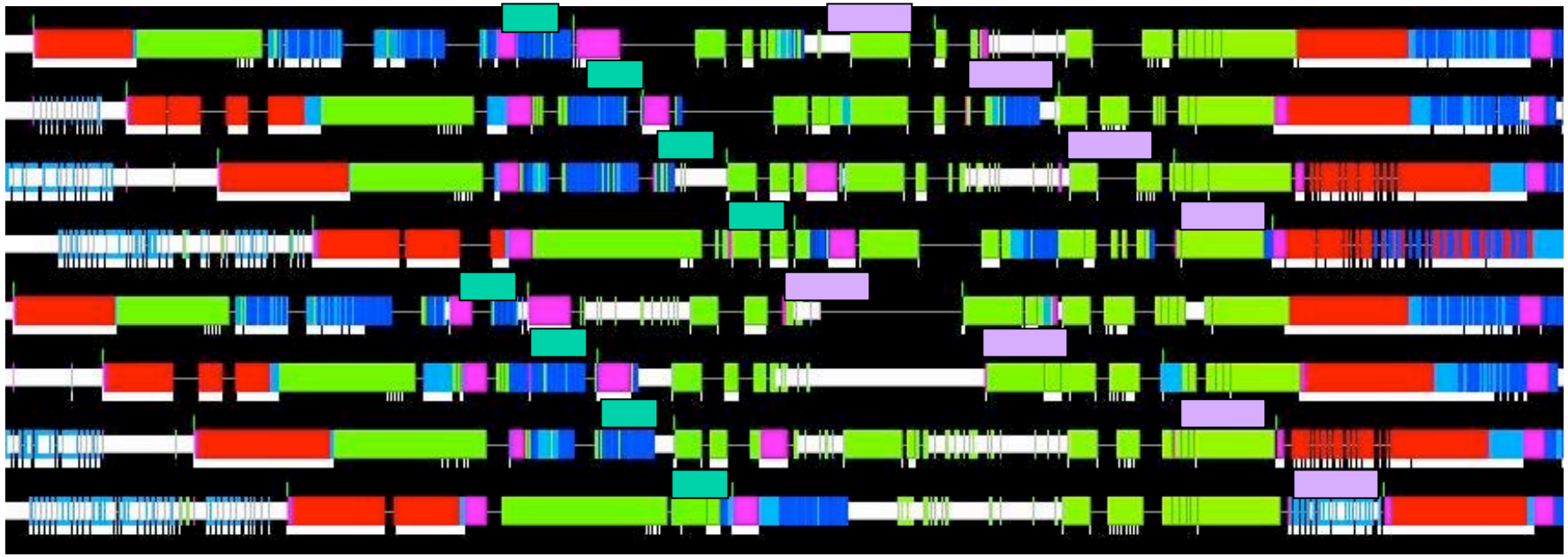
Inter-client Communication

- 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



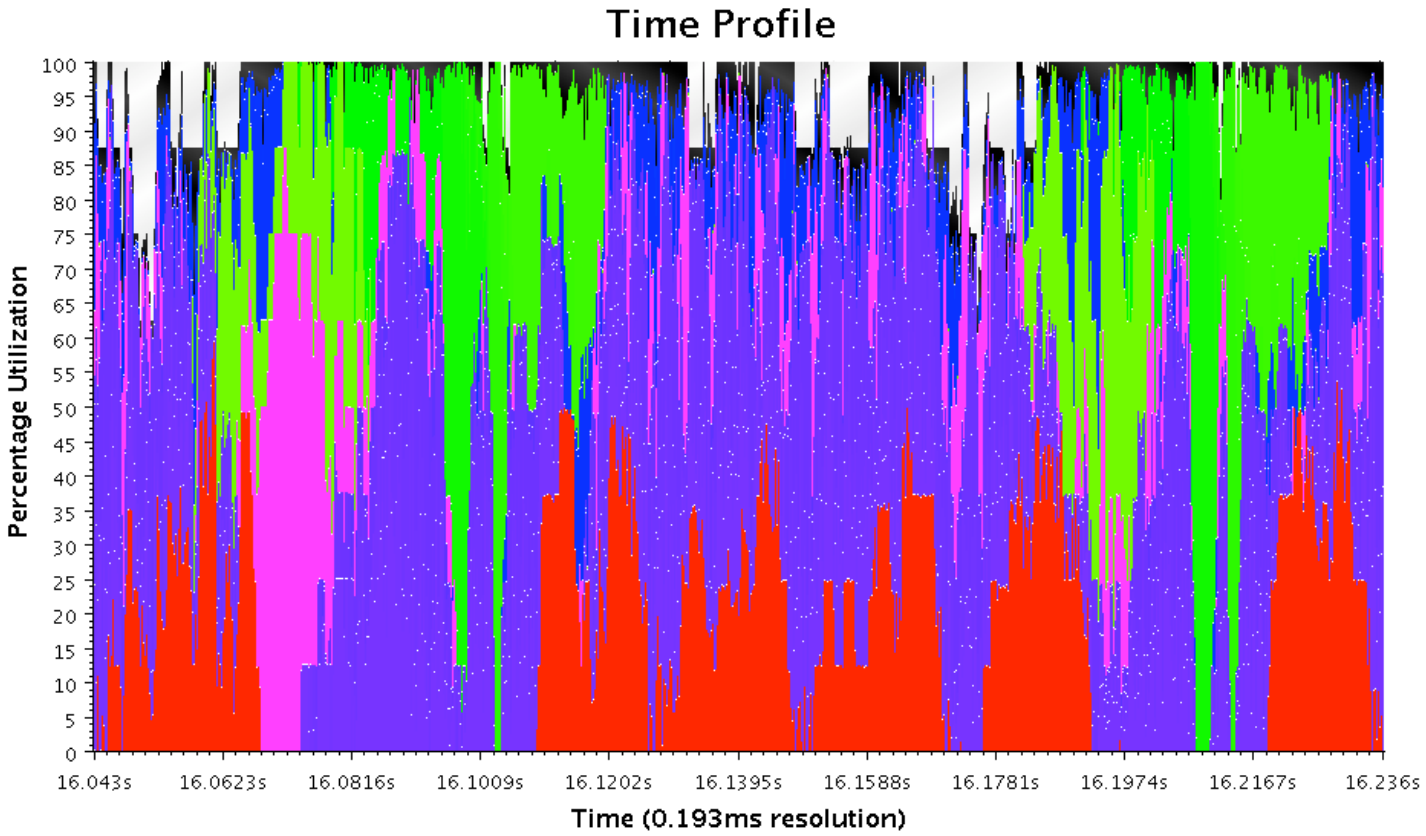
Weakness of Token-Passing

- 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

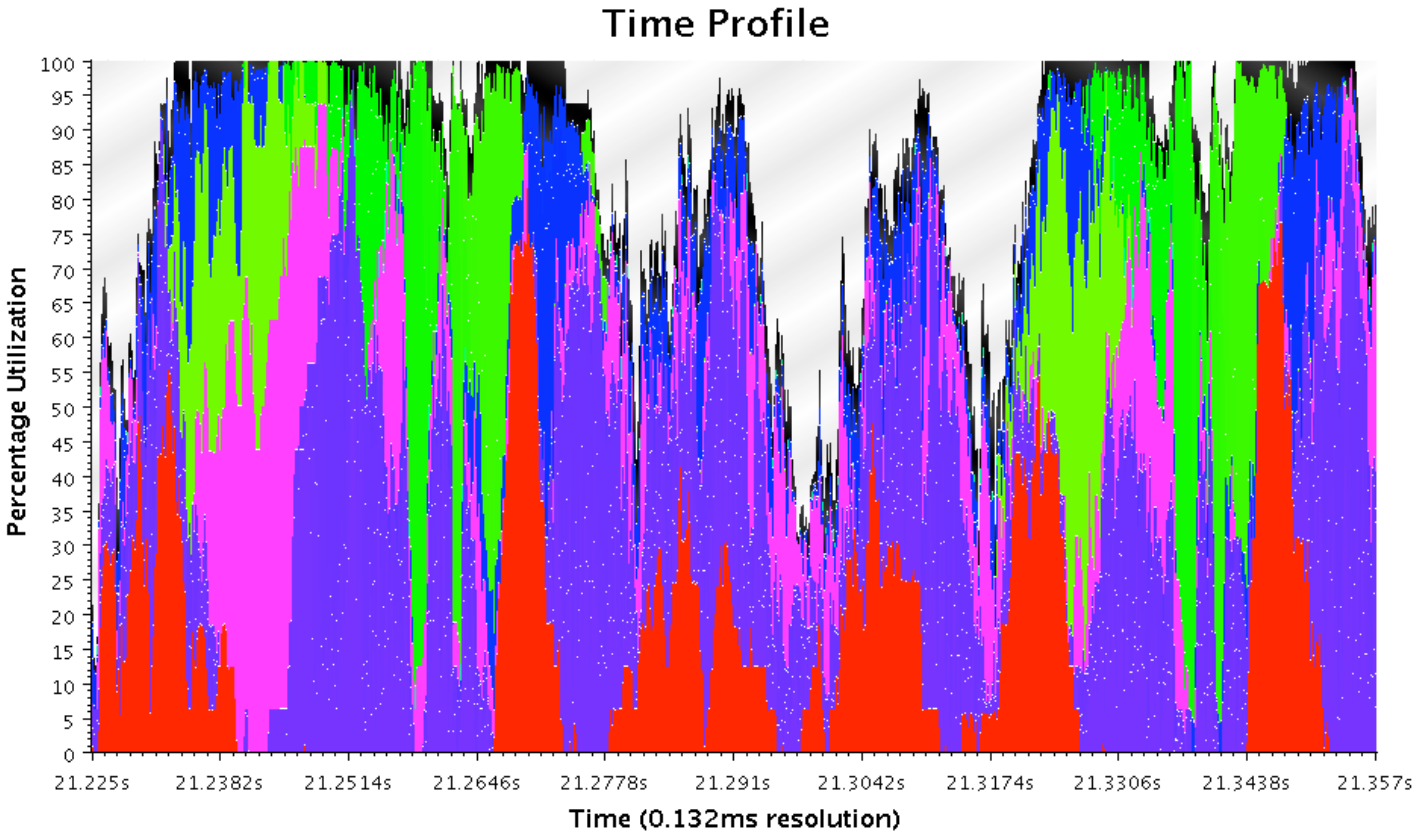
Current Compromise

- Fermi should overlap multiple streams
- If GPU is shared:
  - Submit remote work
  - Wait for remote work to complete
    - Gives other processes a chance to submit theirs
  - Submit local work
- If GPU is not shared:
  - Submit remote and local work immediately

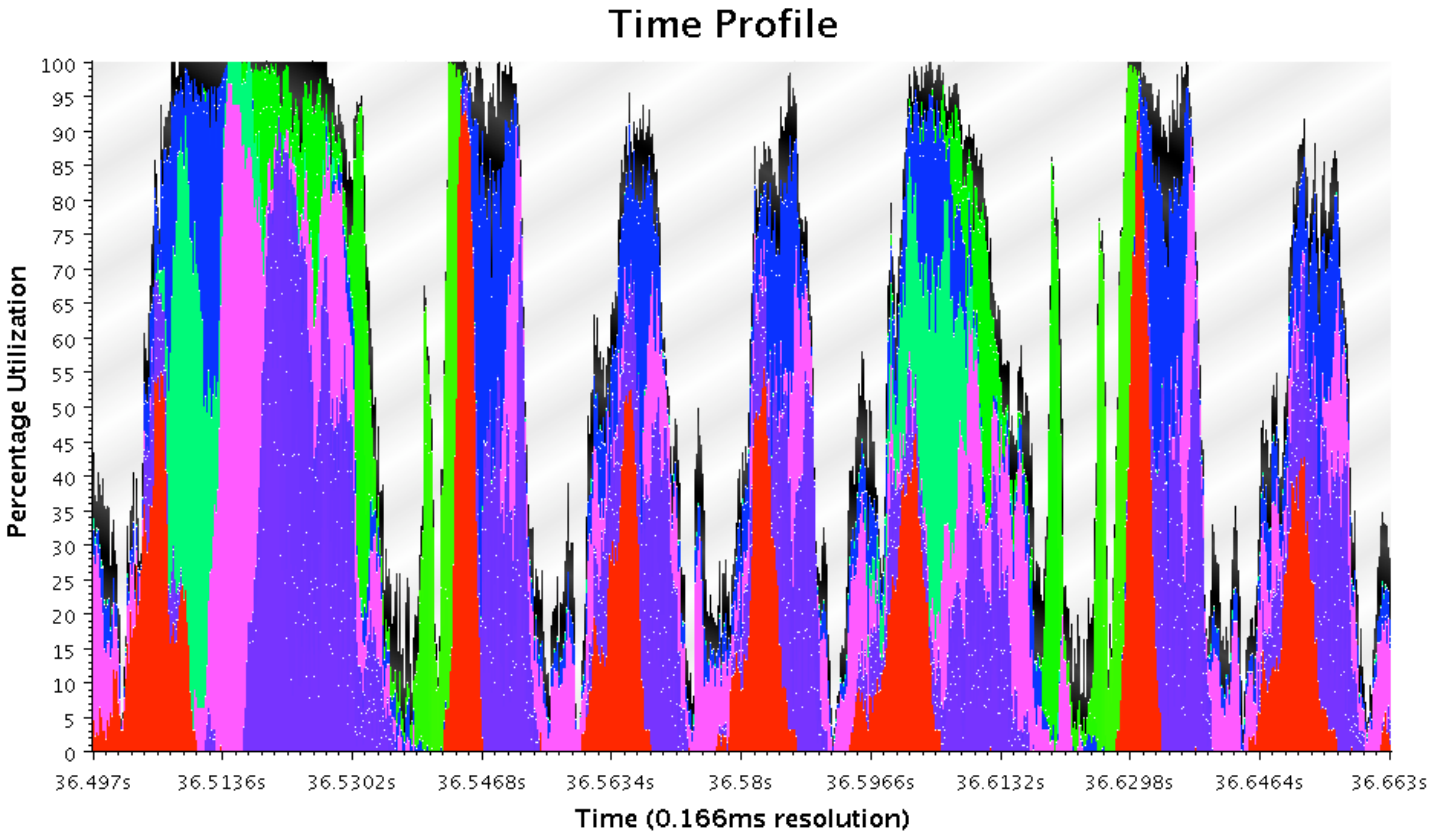
8 GPUs + 8 CPU Cores



8 GPUs + 16 CPU Cores



8 GPUs + 32 CPU Cores



Recent NAMD GPU Developments

- Production features in 2.7b3 release (7/6/2010):
  - 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 2.7b4 release (9/17/2010):
  - Sort blocks in order of decreasing work
  - Recursive bisection within patch on 32-atom boundaries
  - Warp-based pairlists based on sorted atoms

Sorting Blocks

- Sort patch pairs by increasing distance.
- Equivalent to sort by decreasing work.
- Slower blocks start first, fast blocks last.
- Reduces idle time, total runtime of grid.

Sorting Atoms

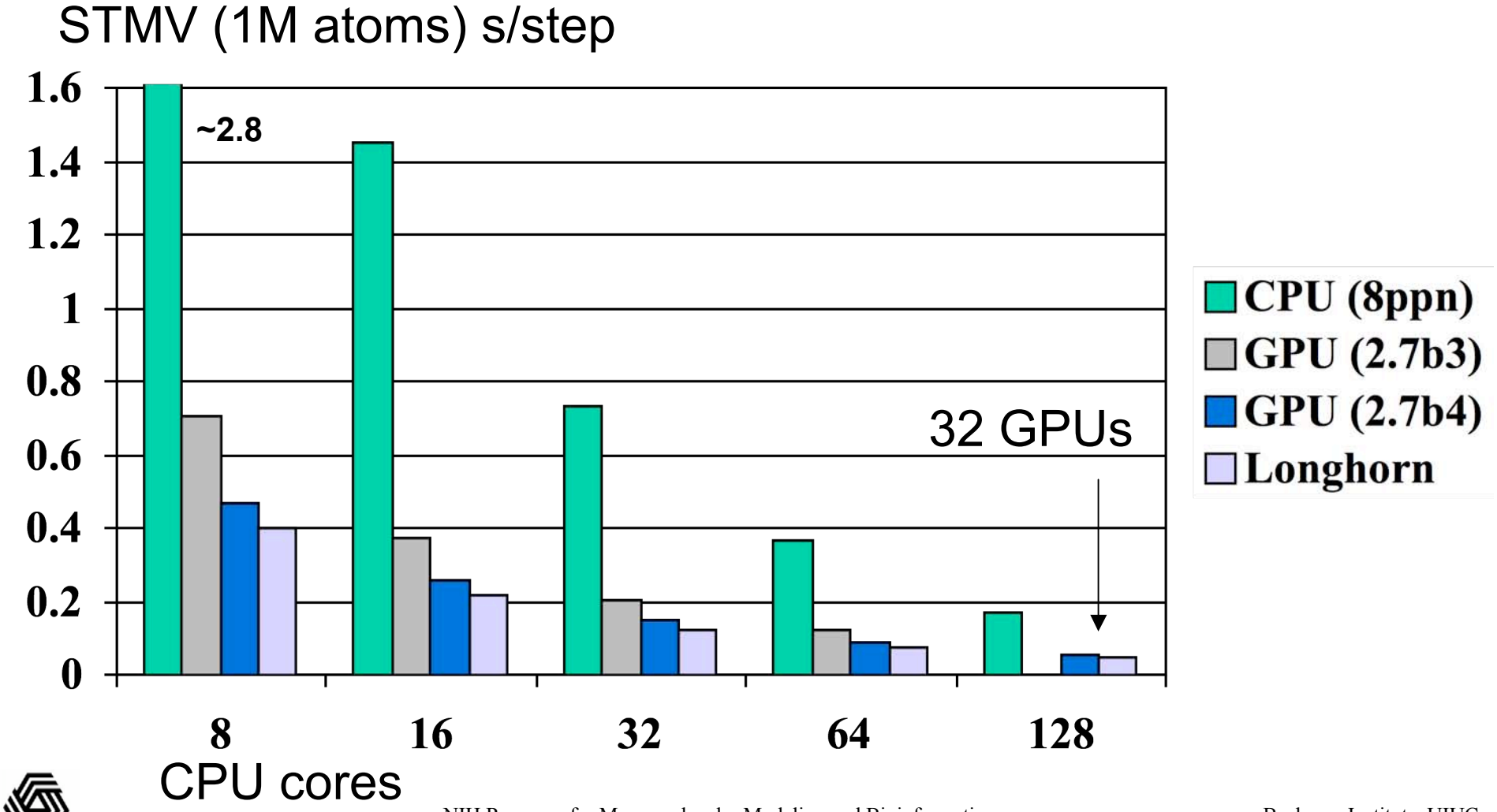
- Reduce warp divergence on cutoff tests
- Group nearby atoms in the same warp
- One option is space-filling curve
- Used recursive bisection instead
  - Split only on 32-atom boundaries
  - Find major axis, sort, split, repeat…

Warp-based Pairlists

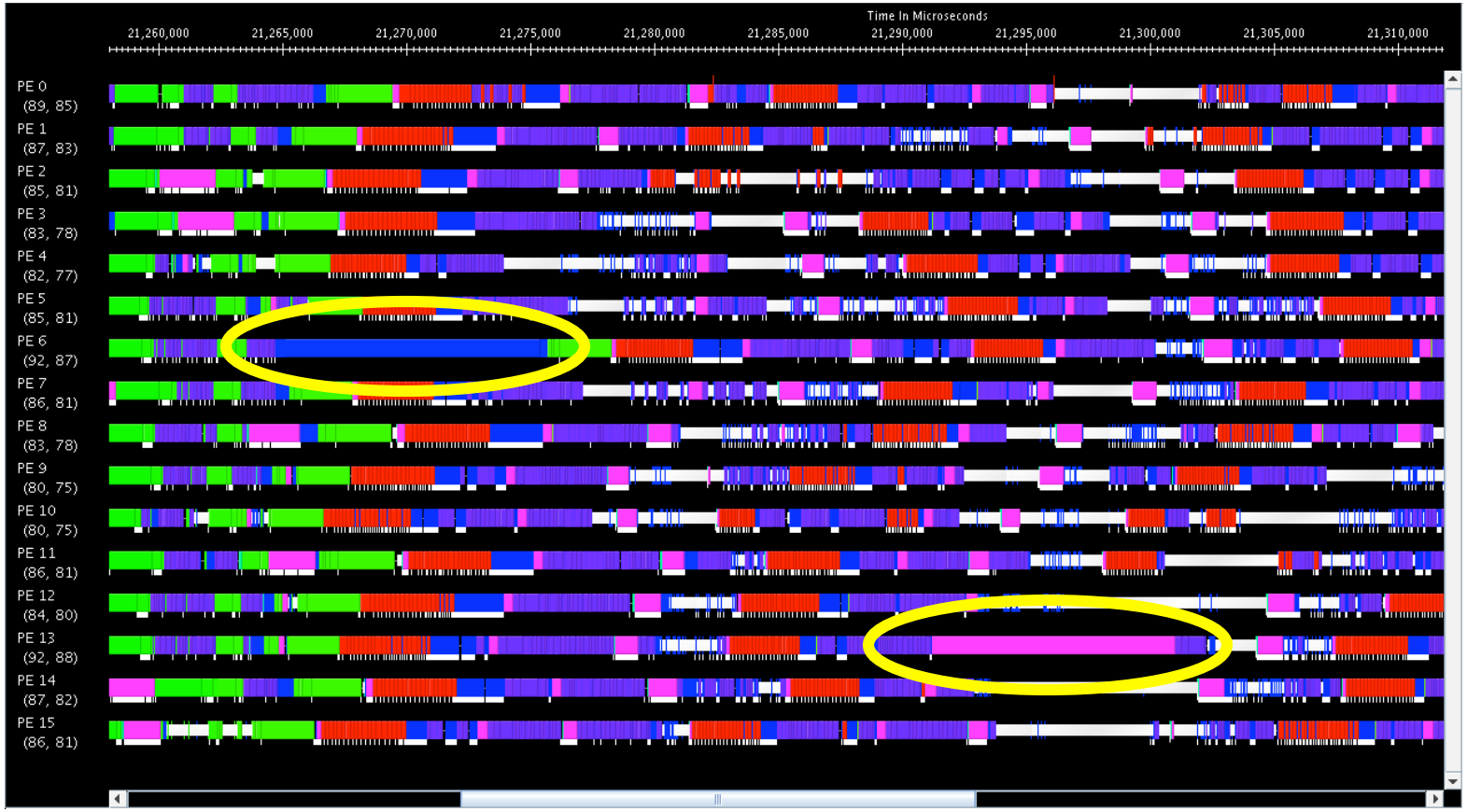
- List generation
  - Load 16 atoms into shared memory
  - Any atoms in this warp within pairlist distance?
  - Combine all (4) warps as bits in char and save.
- List use
  - Load set of 16 atoms if any bit is set in list
  - Only calculate if this warp's bit is set
  - Cuts kernel runtime by 50%

Lincoln and Longhorn Performance

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



System Noise Still Present



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
  - Improve scaling of PME reciprocal sum
- 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/