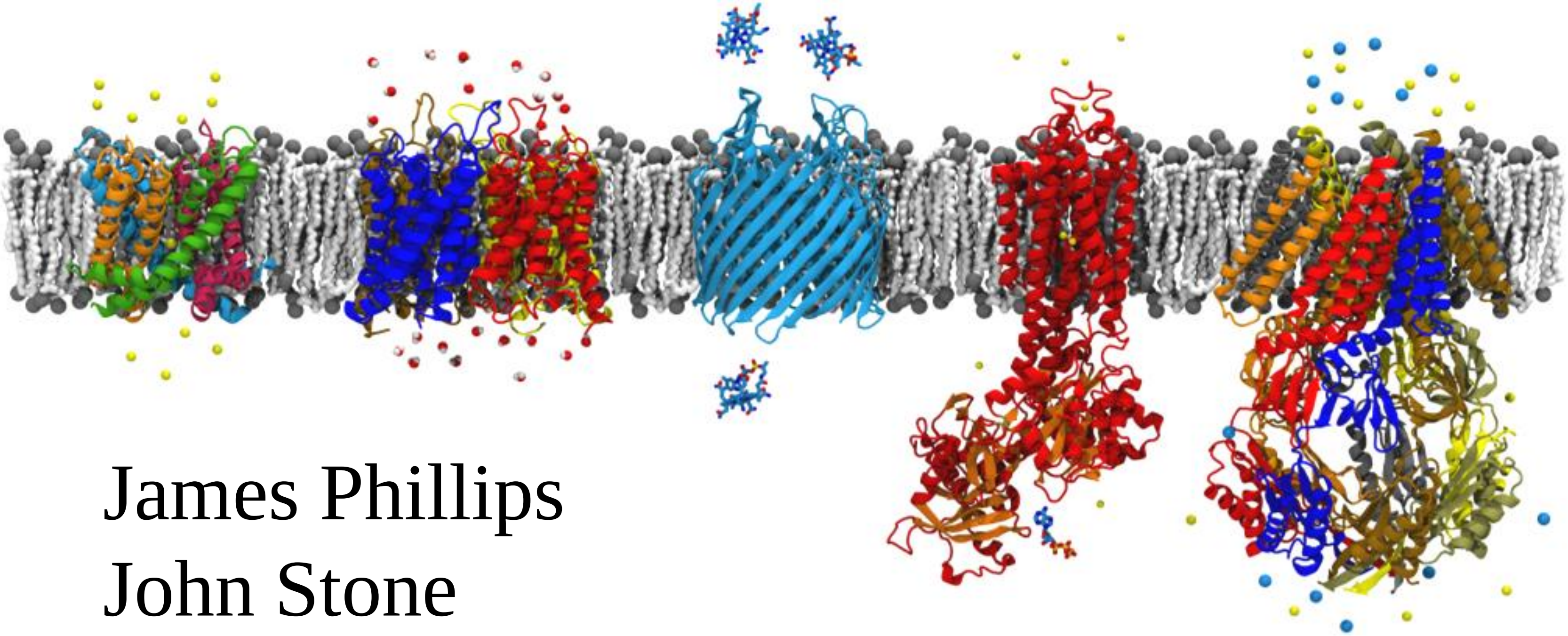
Experience with NAMD on GPU-Accelerated Clusters



James Phillips
John Stone
Klaus Schulten

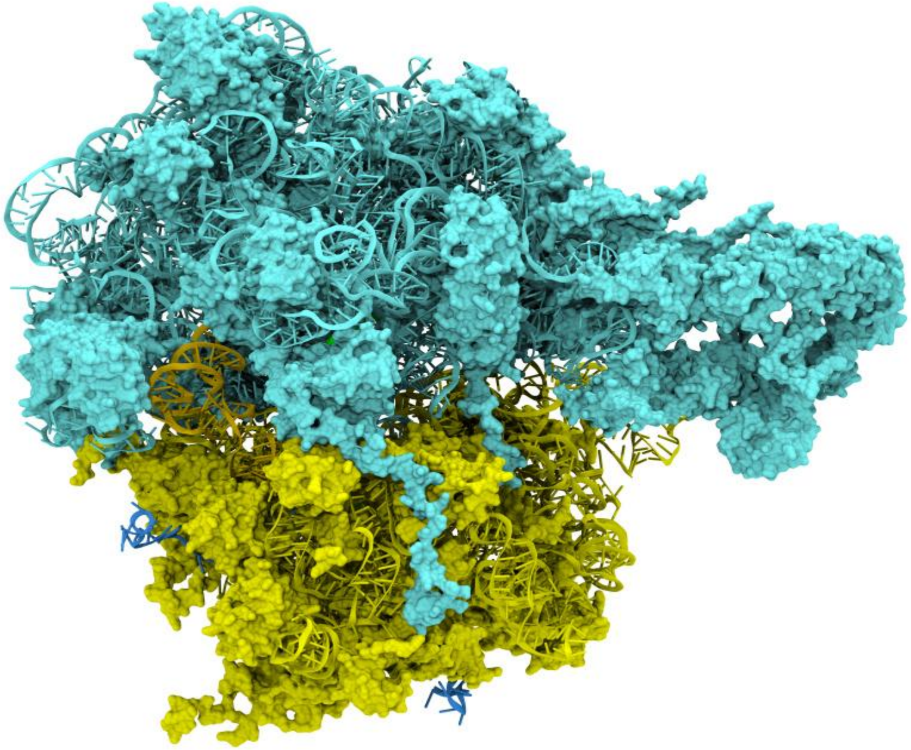
http://www.ks.uiuc.edu/Research/gpu/

Outline

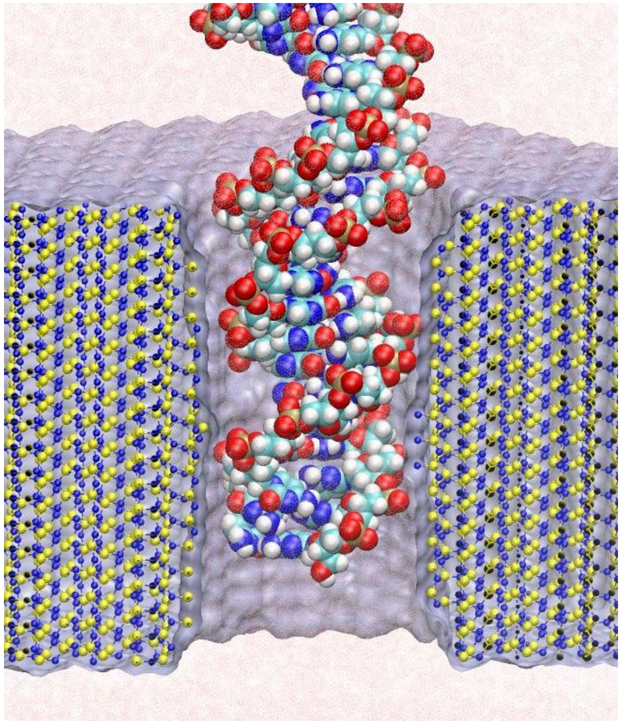
- Motivational images of NAMD simulations
- Why all the fuss about GPUs?
- What is message-driven programming?
- Adapting NAMD to GPU-accelerated clusters
- Older cluster performance results
- NCSA Lincoln cluster performance results
- Does CUDA like to share?

Computational Microscopy

Ribosome: synthesizes proteins from genetic information, target for antibiotics



Silicon nanopore: bionanodevice for sequencing DNA efficiently



NAMD: Practical Supercomputing

- 30,000 users can't all be computer experts.
  - 18% are NIH-funded; many in other countries.
  - 5600 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.

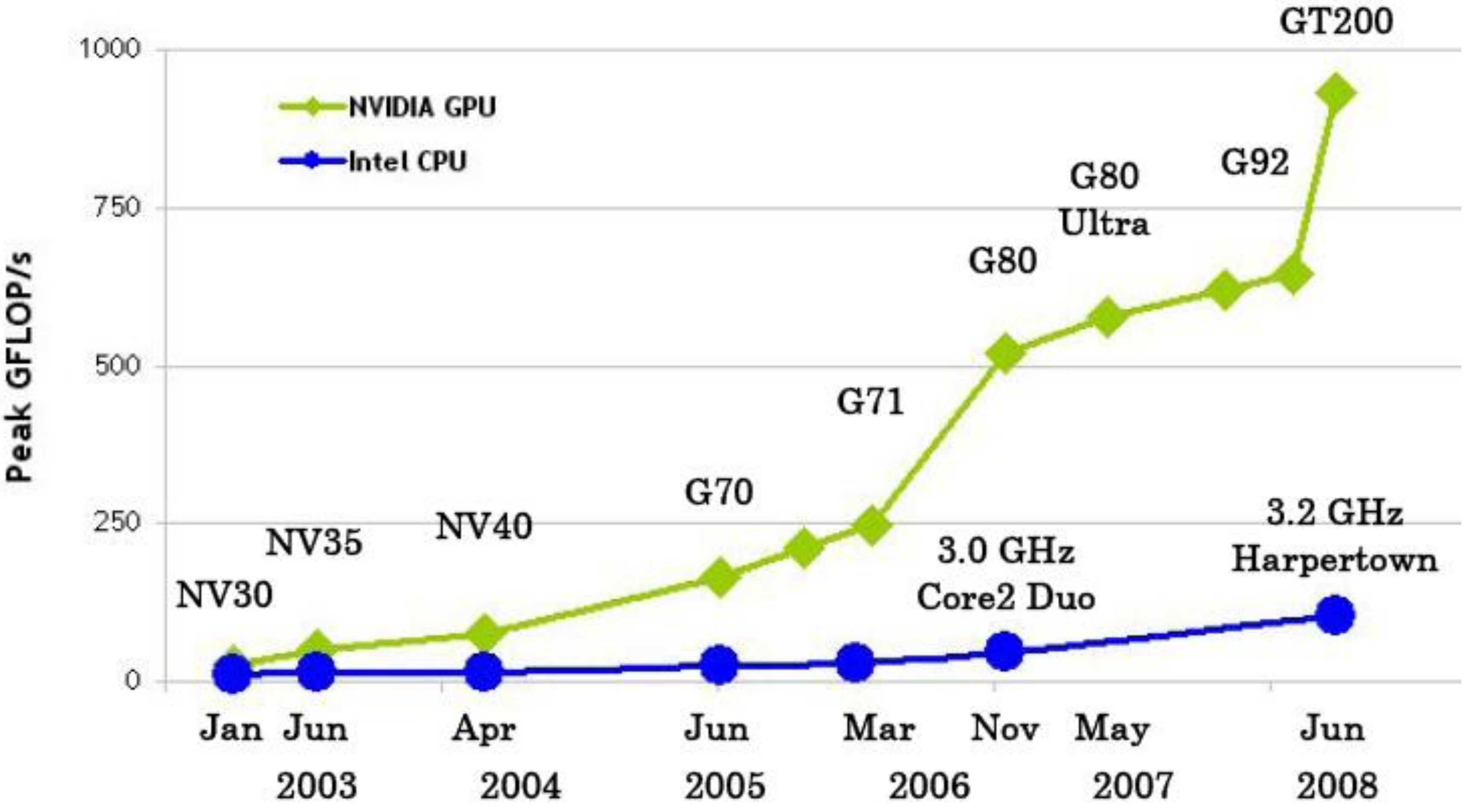
- Linux clusters – affordable workhorses
  - x86, x86-64, and Itanium processors
  - Gigabit ethernet, Myrinet, InfiniBand, Quadrics, Altix, etc

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

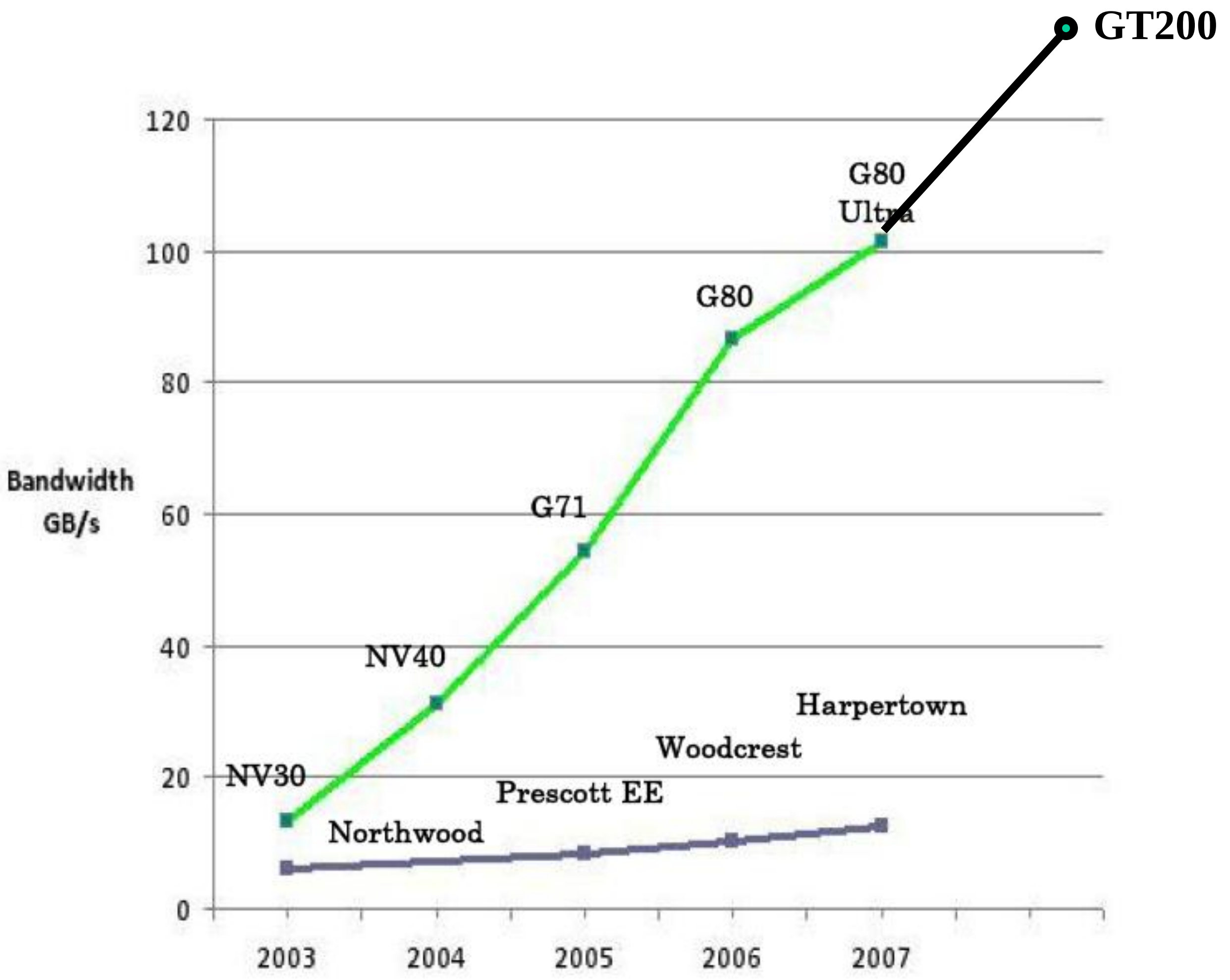
Our Goal: Practical Acceleration

- 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

Peak Single-precision Arithmetic Performance Trend



Peak Memory Bandwidth Trend

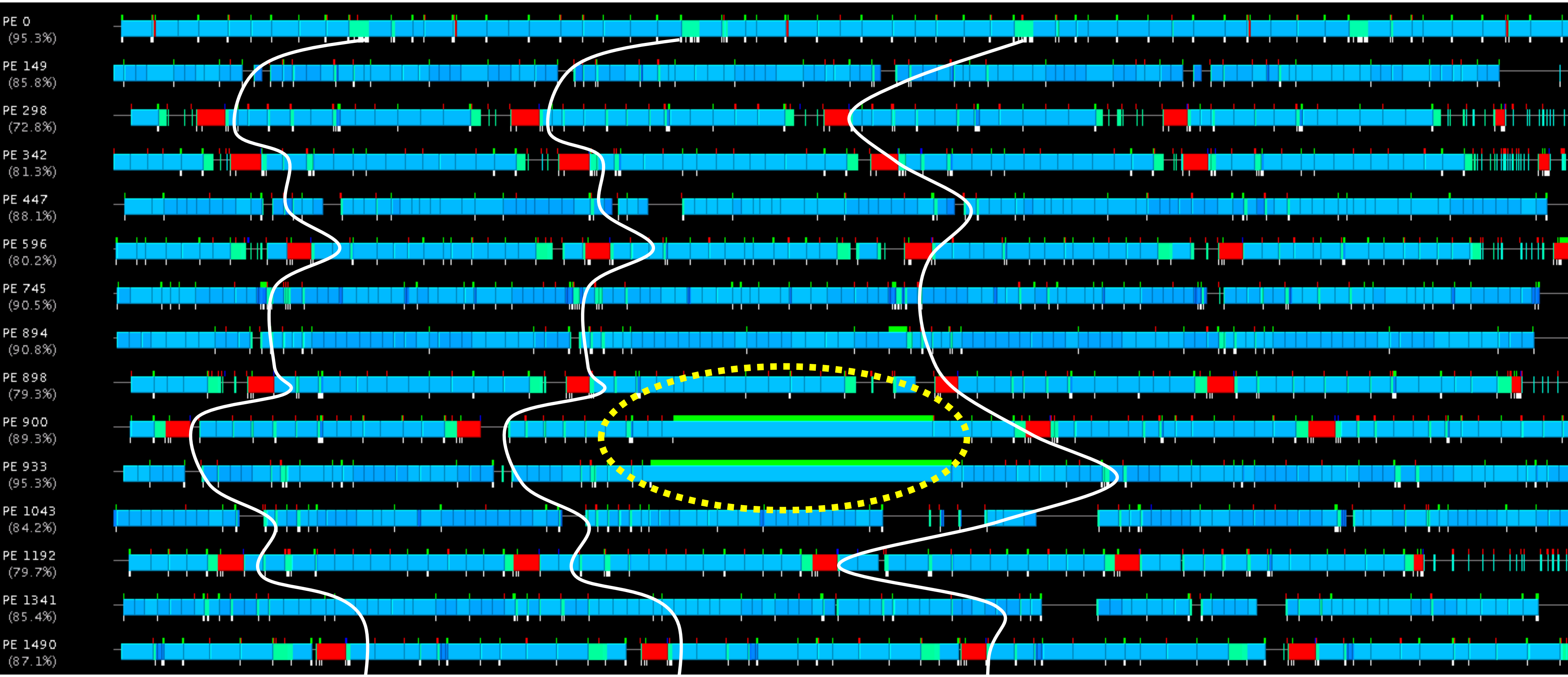


Message-Driven Programming

- 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”

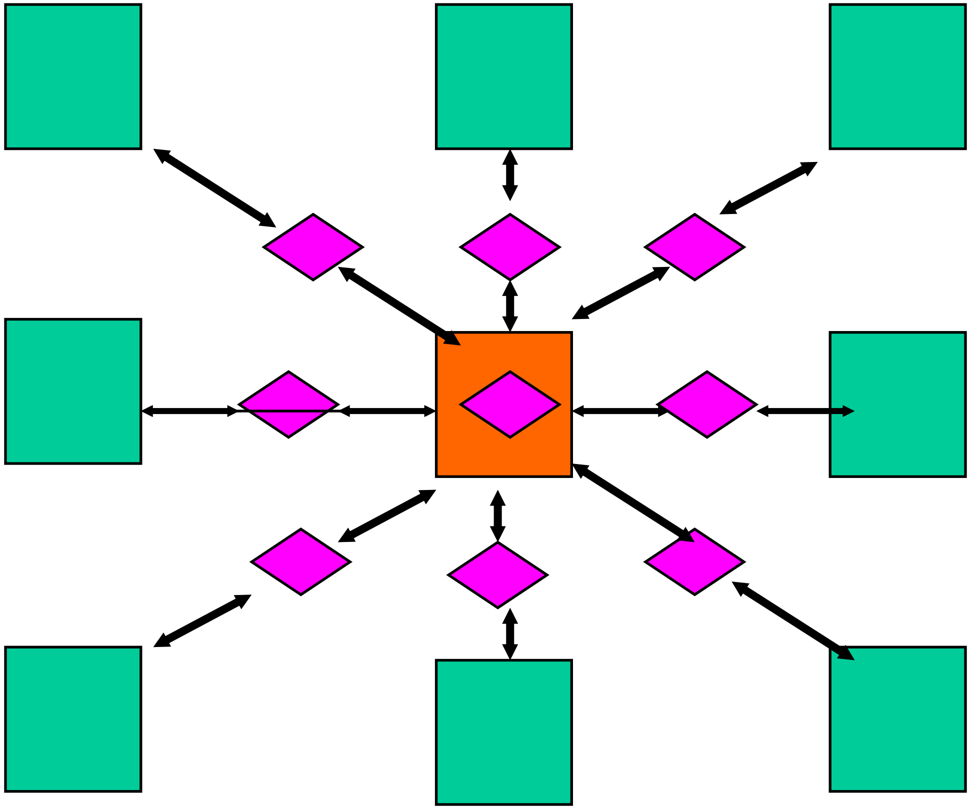


Message-Driven CUDA?

- 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.

NAMD Hybrid Decomposition

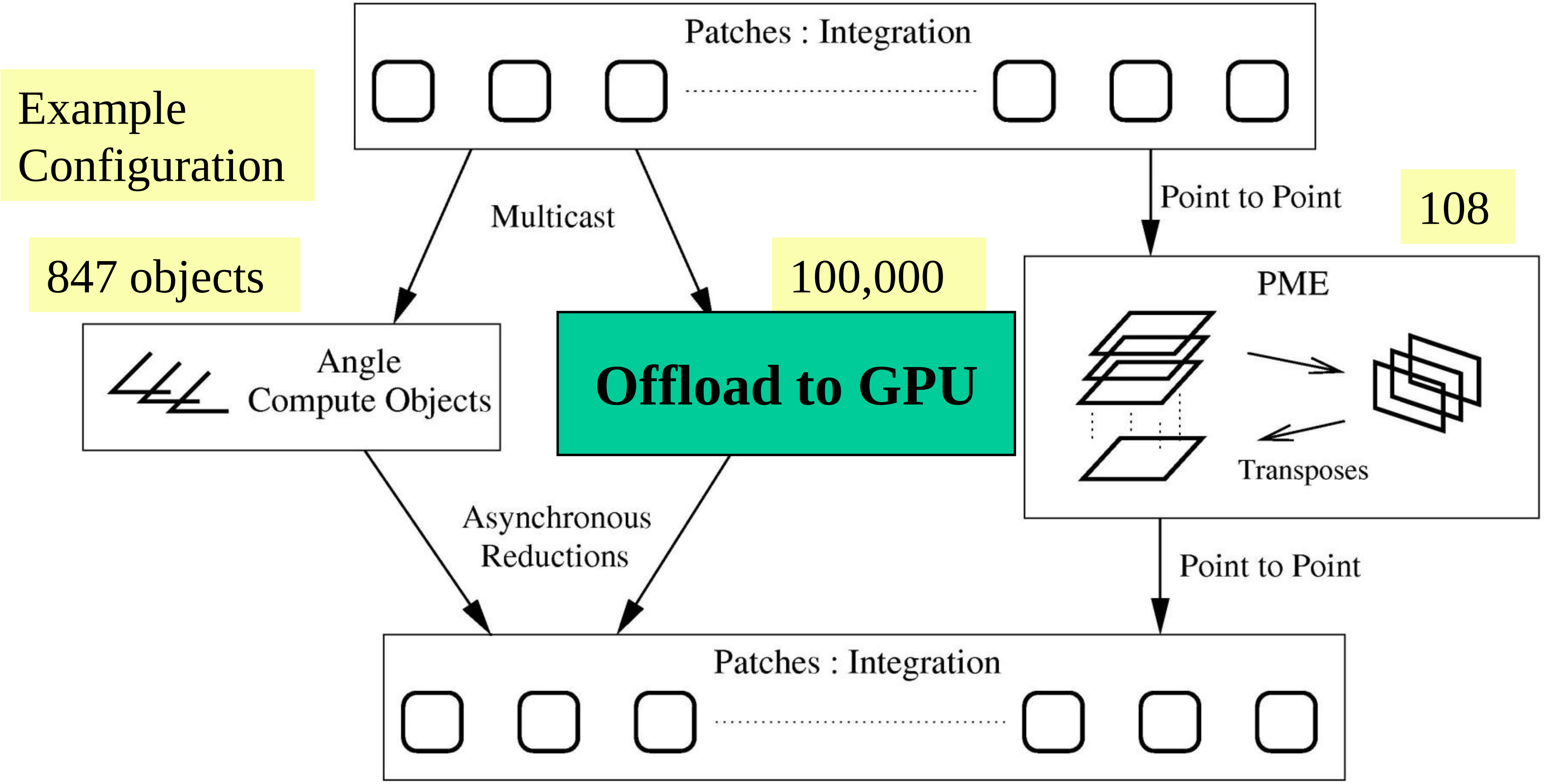
Kale *et al.*, *J. Comp. Phys.* **151**:283-312, 1999.



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

NAMD Overlapping Execution

Phillips *et al.*, *SC2002*.



Example Configuration

847 objects

100,000

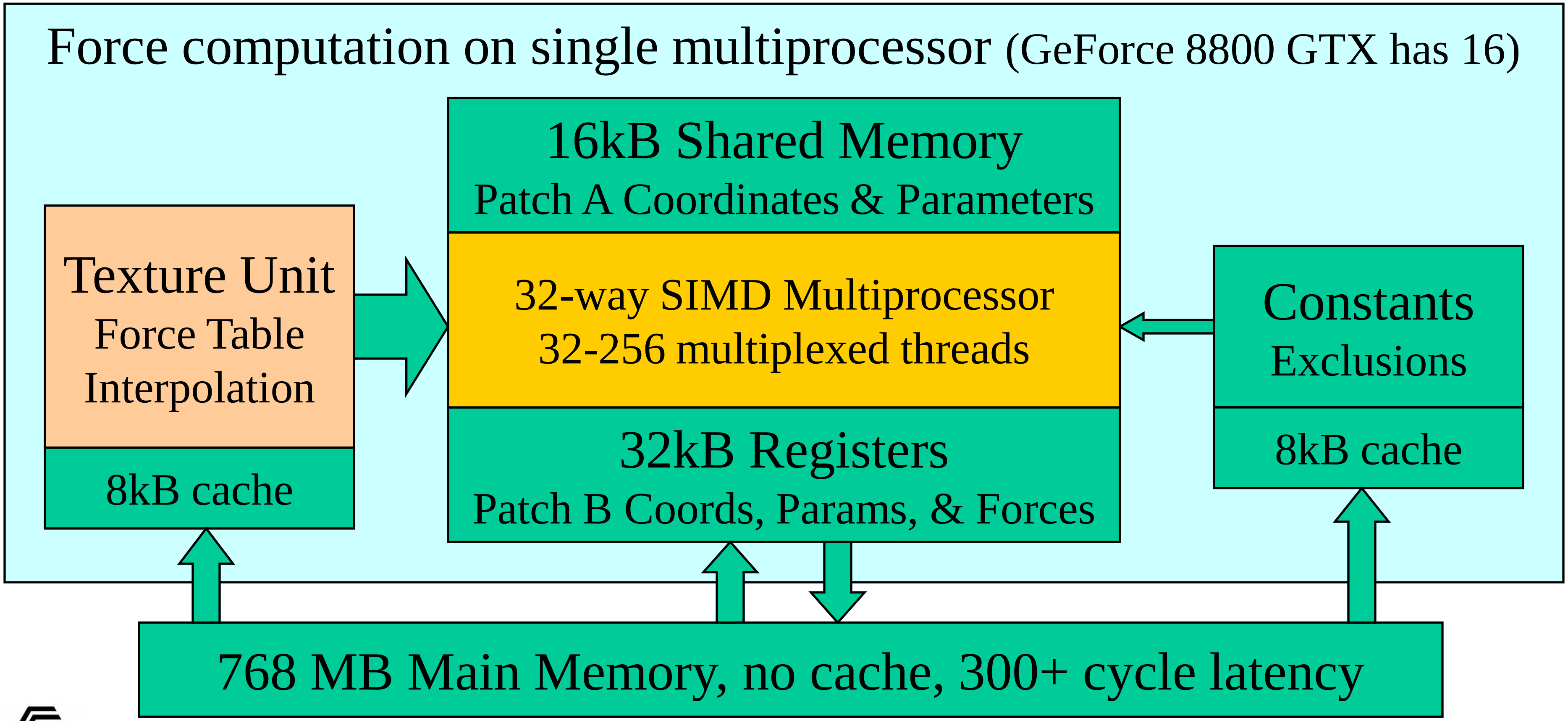
108

Offload to GPU

Objects are assigned to processors and queued as data arrives.

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

National Center for Research Resources

Beckman Institute, UIUC

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 ) {
```

Force Interpolation

```
    float4 ft = texfetch(force_table, 1.f/sqrt(r2));
```

Exclusions

```
    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);
    }
```

Parameters

```
    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
```

Accumulation

```
    iforce.x += dx * f;   iforce.y += dy * f;    iforce.z += dz * f;
    iforce.w += 1.f;  // interaction count or energy
  }
}
```

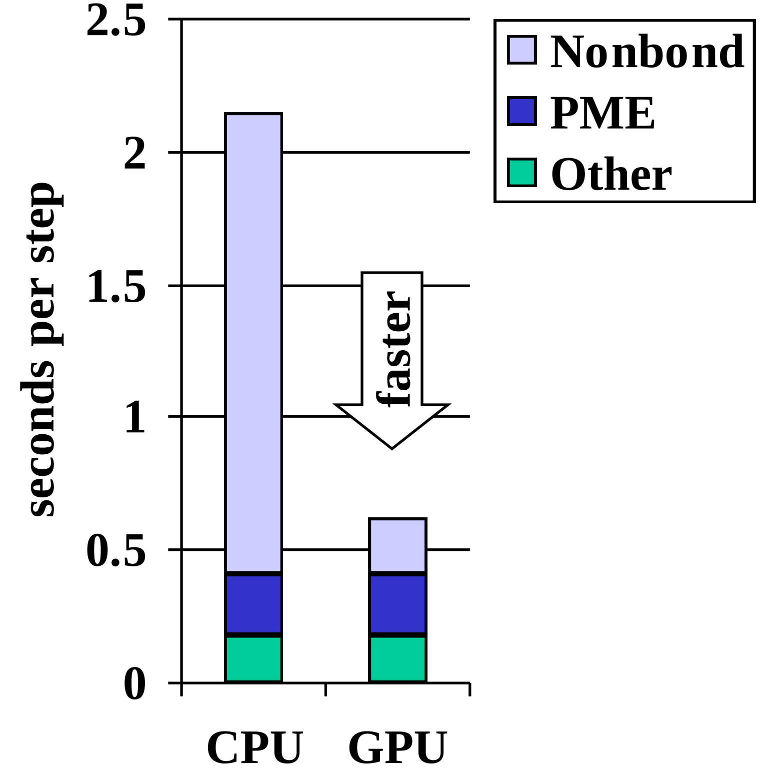
National Center for Research Resources

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

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.

ApoA1 Performance

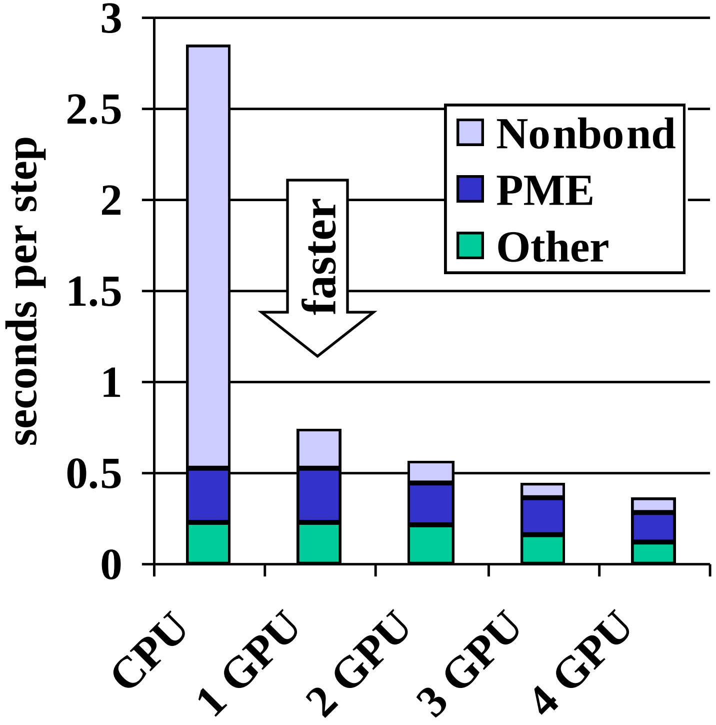


2.67 GHz Core 2 Quad Extreme + GeForce 8800 GTX

2007 GPU Cluster Performance

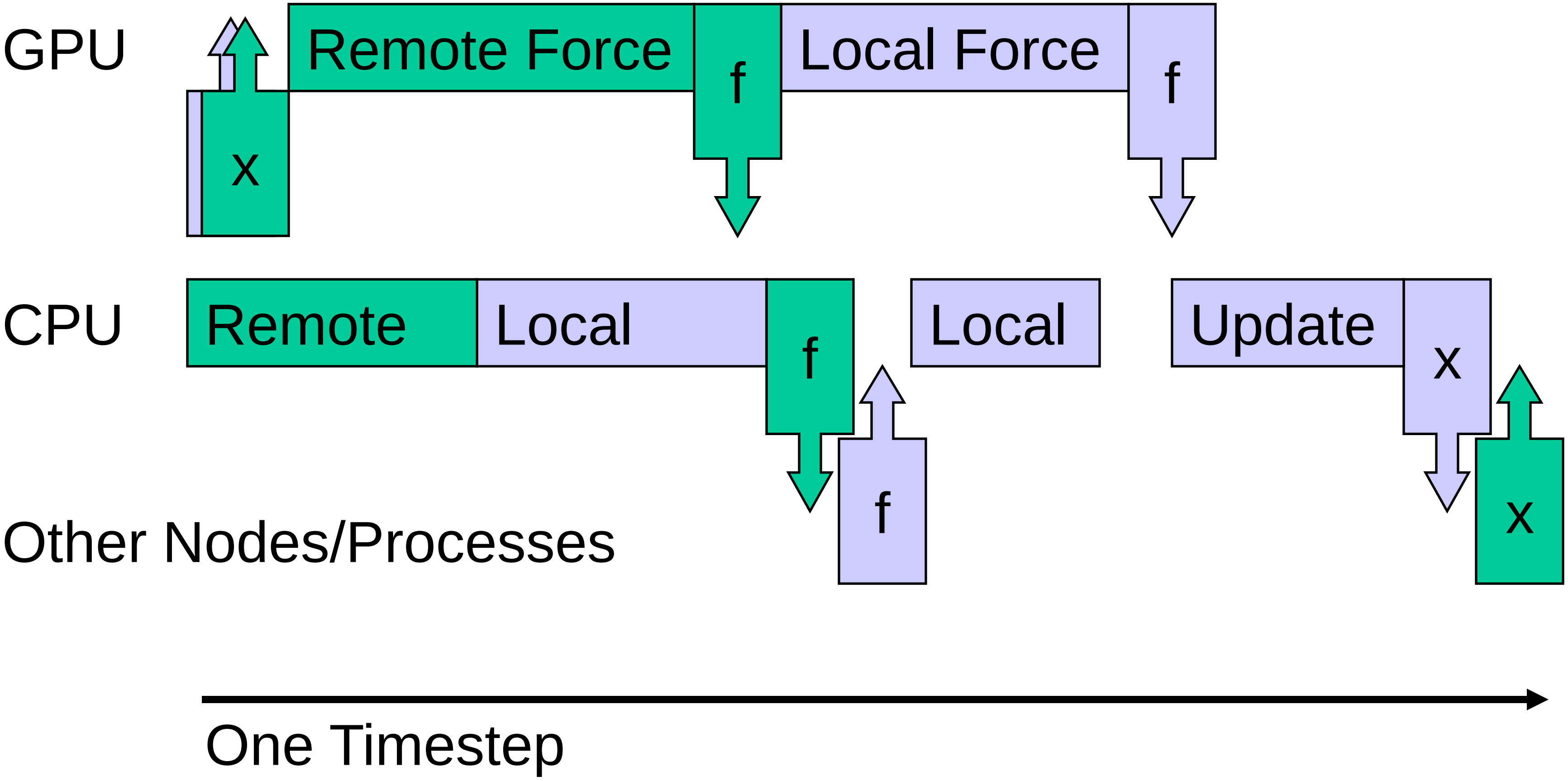
- Poor scaling unsurprising
  - 2x speedup on 4 GPUs
  - Gigabit ethernet
  - Load balancer disabled
- Plans for better scaling
  - InfiniBand network
  - Tune parallel overhead
  - Load balancer changes
    - Balance GPU load.
    - Minimize communication.

ApoA1 Performance



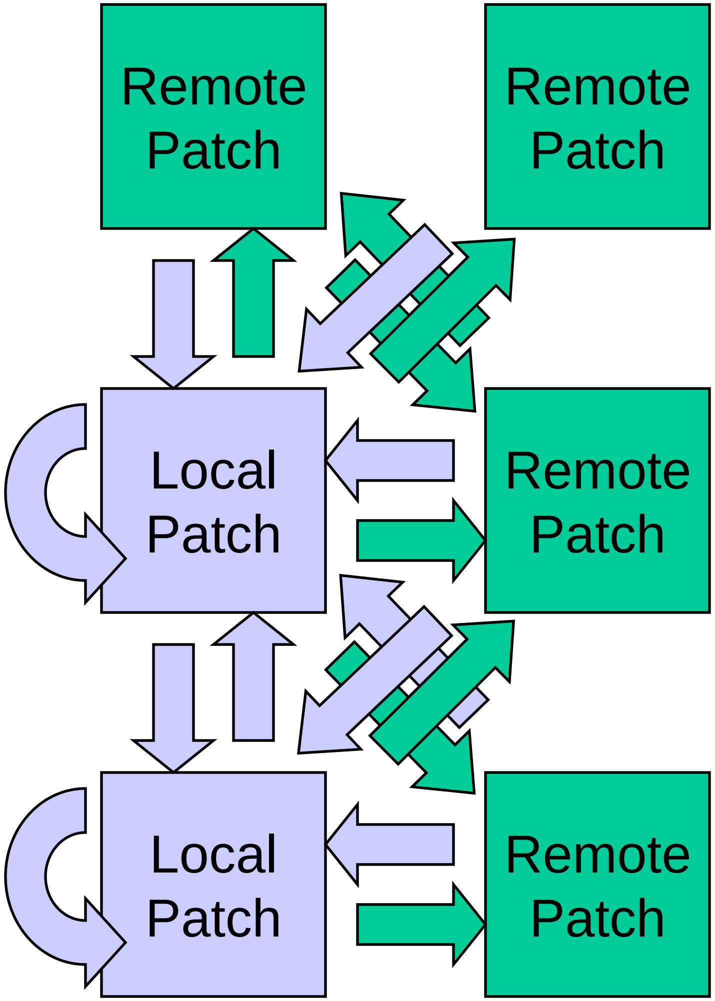
2.2 GHz Opteron + GeForce 8800 GTX

Overlapping GPU and CPU with Communication



“Remote Forces”

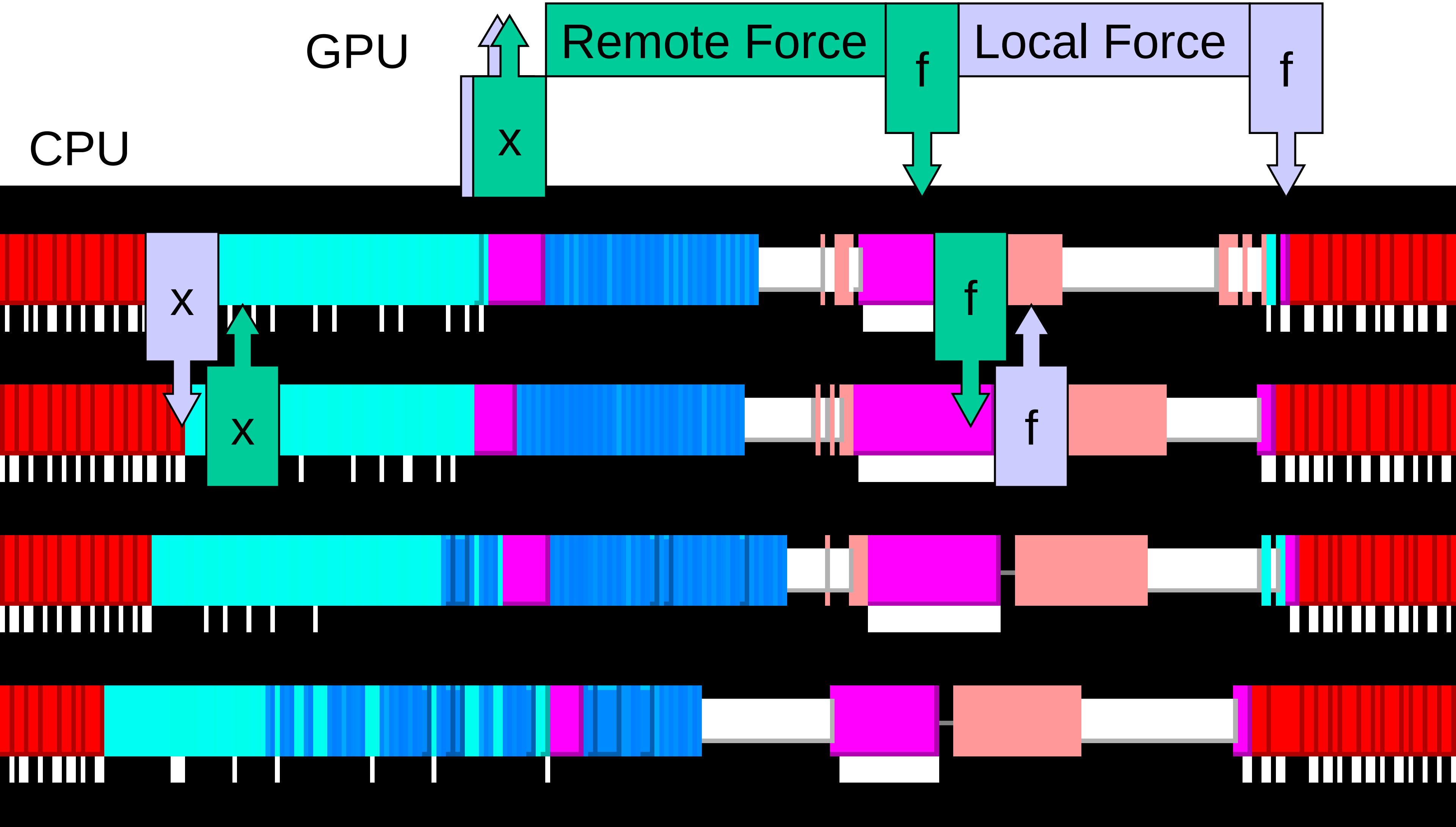
- Forces on atoms in a local patch are “local”
- Forces on atoms in a remote patch are “remote”
- Calculate remote forces first to overlap force communication with local force calculation
- Not enough work to overlap with position communication



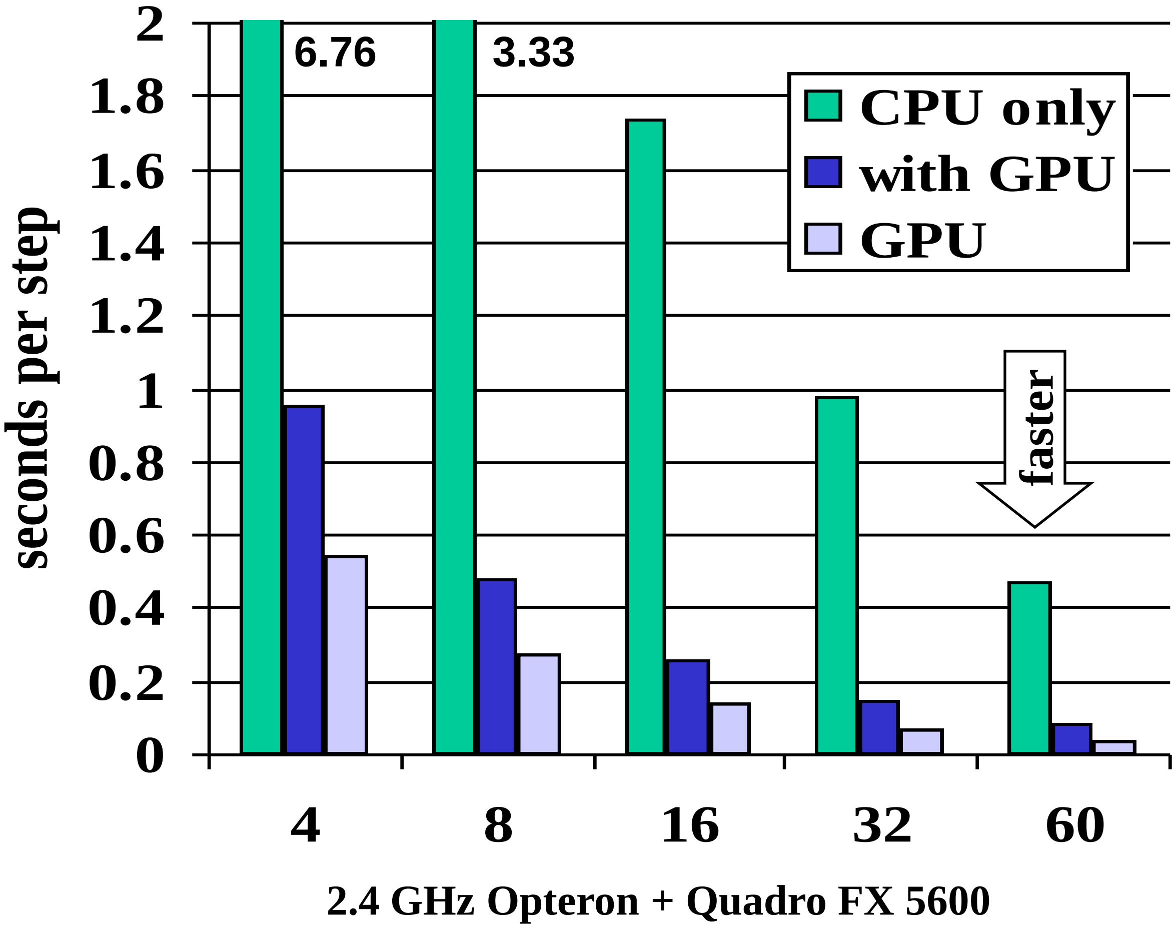
Work done by **one** processor

Actual Timelines from NAMD

Generated using Charm++ tool “Projections”



NCSA "4+4" QP Cluster



GPU Cluster Observations

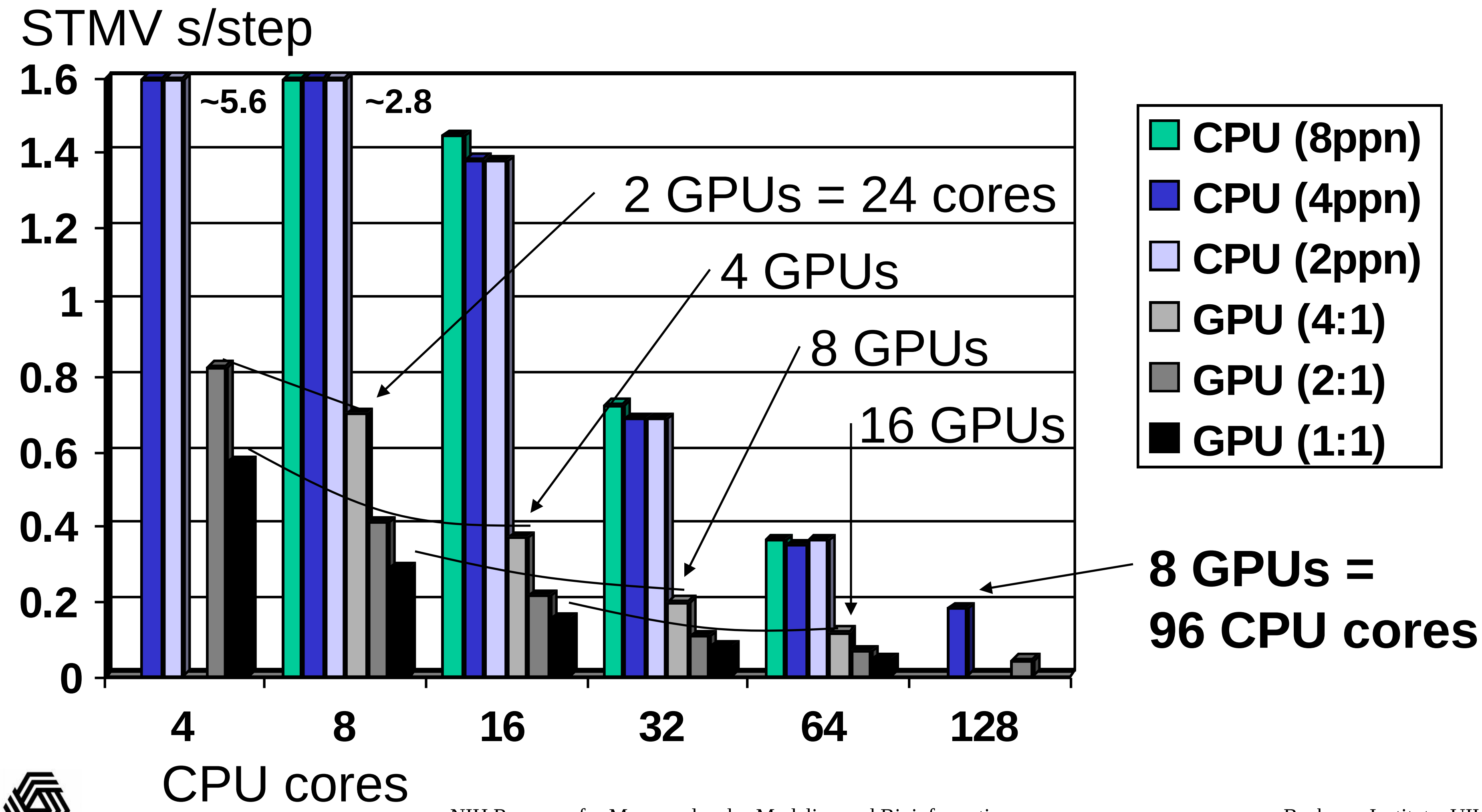
- Tools needed to control GPU allocation
  - Simplest solution is rank % devicesPerNode
  - Doesn’t work with multiple independent jobs
- CUDA and MPI can’t share pinned memory
  - Either user copies data or disable MPI RDMA
  - Need interoperable user-mode DMA standard
- Speaking of extra copies…
  - Why not DMA GPU to GPU?
  - Even better, why not RDMA over InfiniBand?

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, very early results)



~5.6

~2.8

2 GPUs = 24 cores

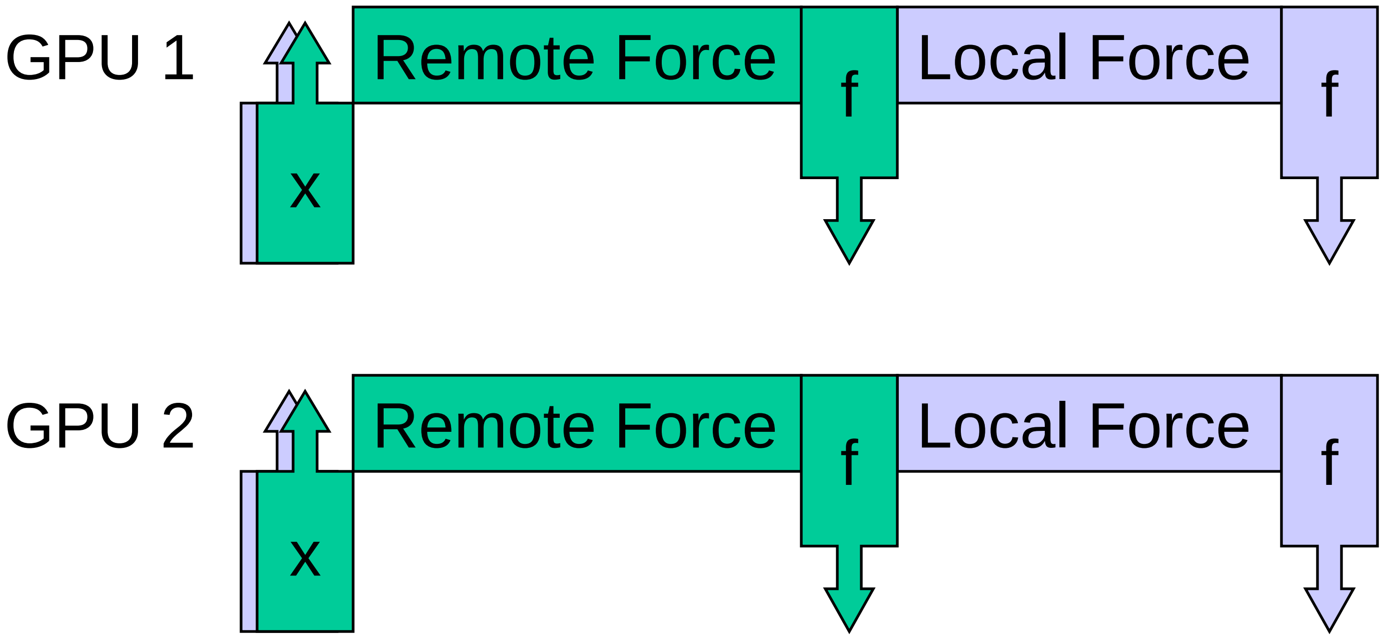
4 GPUs

8 GPUs

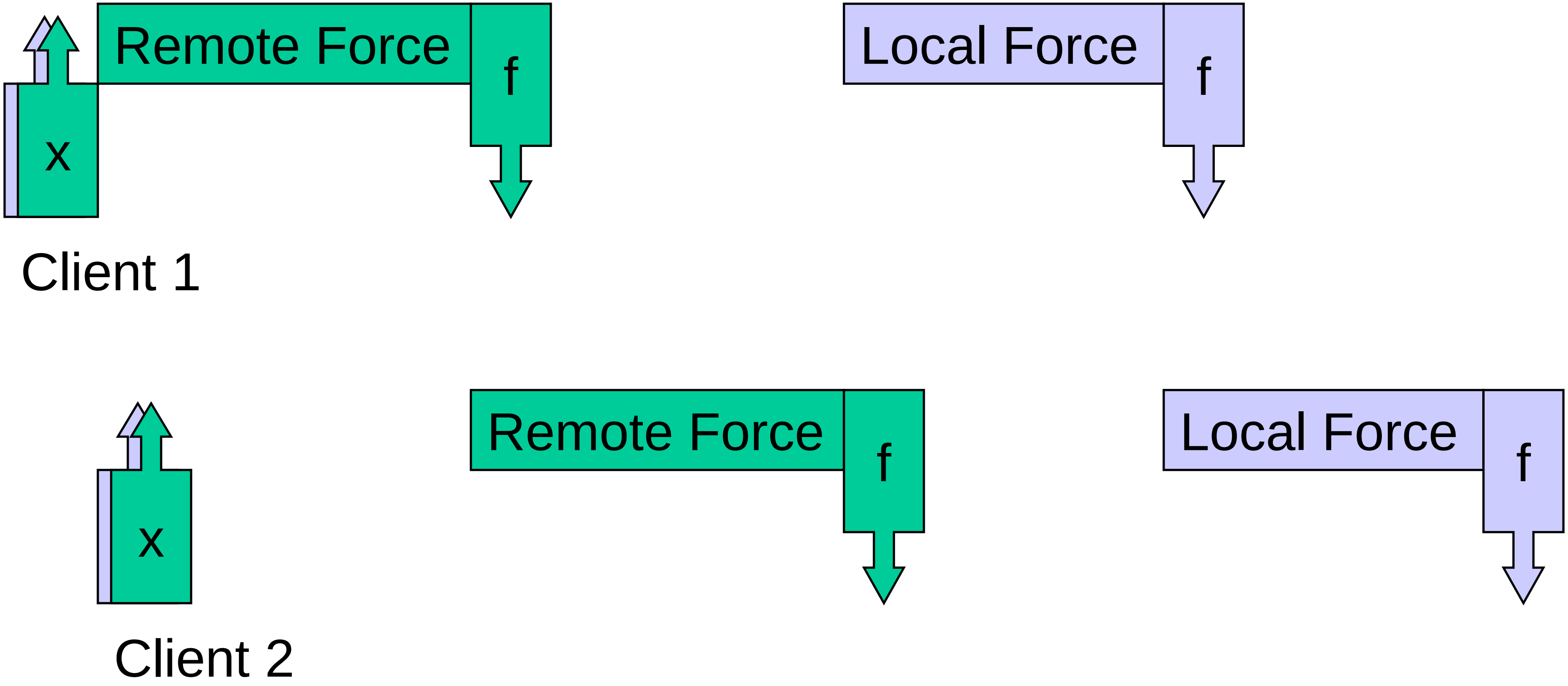
16 GPUs

8 GPUs = 96 CPU cores

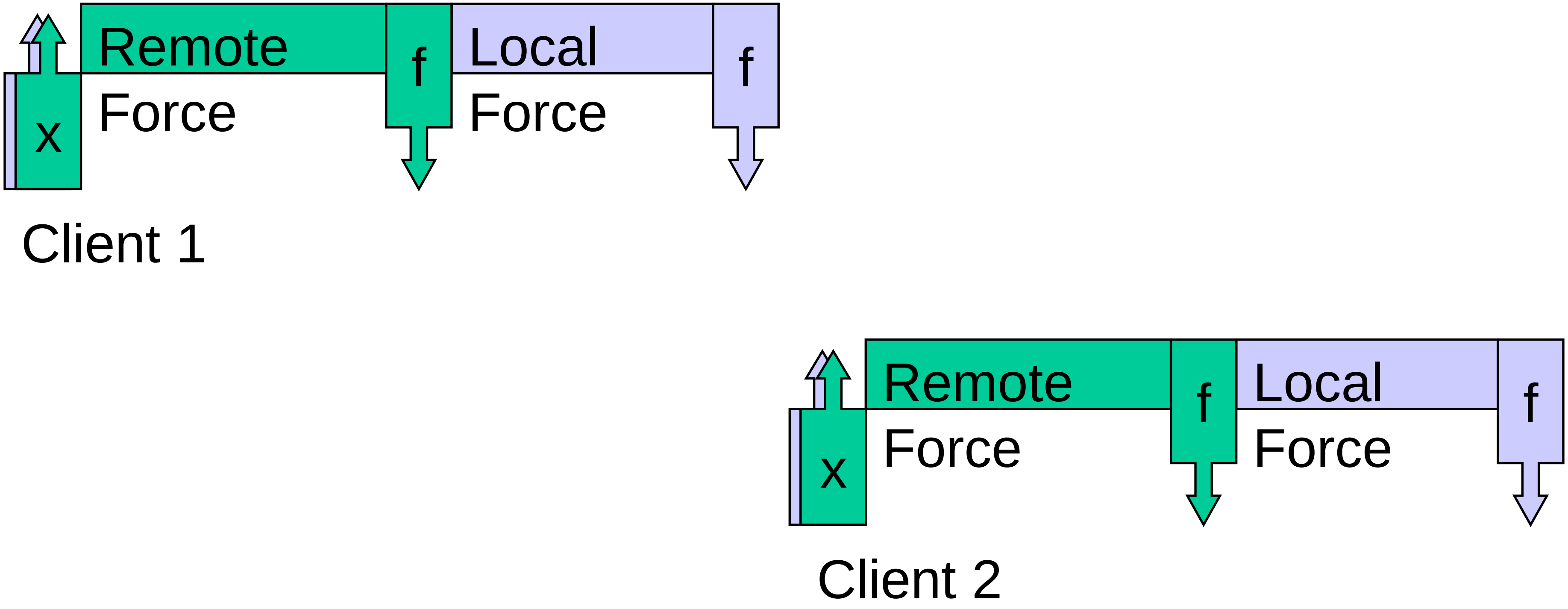
No GPU Sharing (Ideal World)



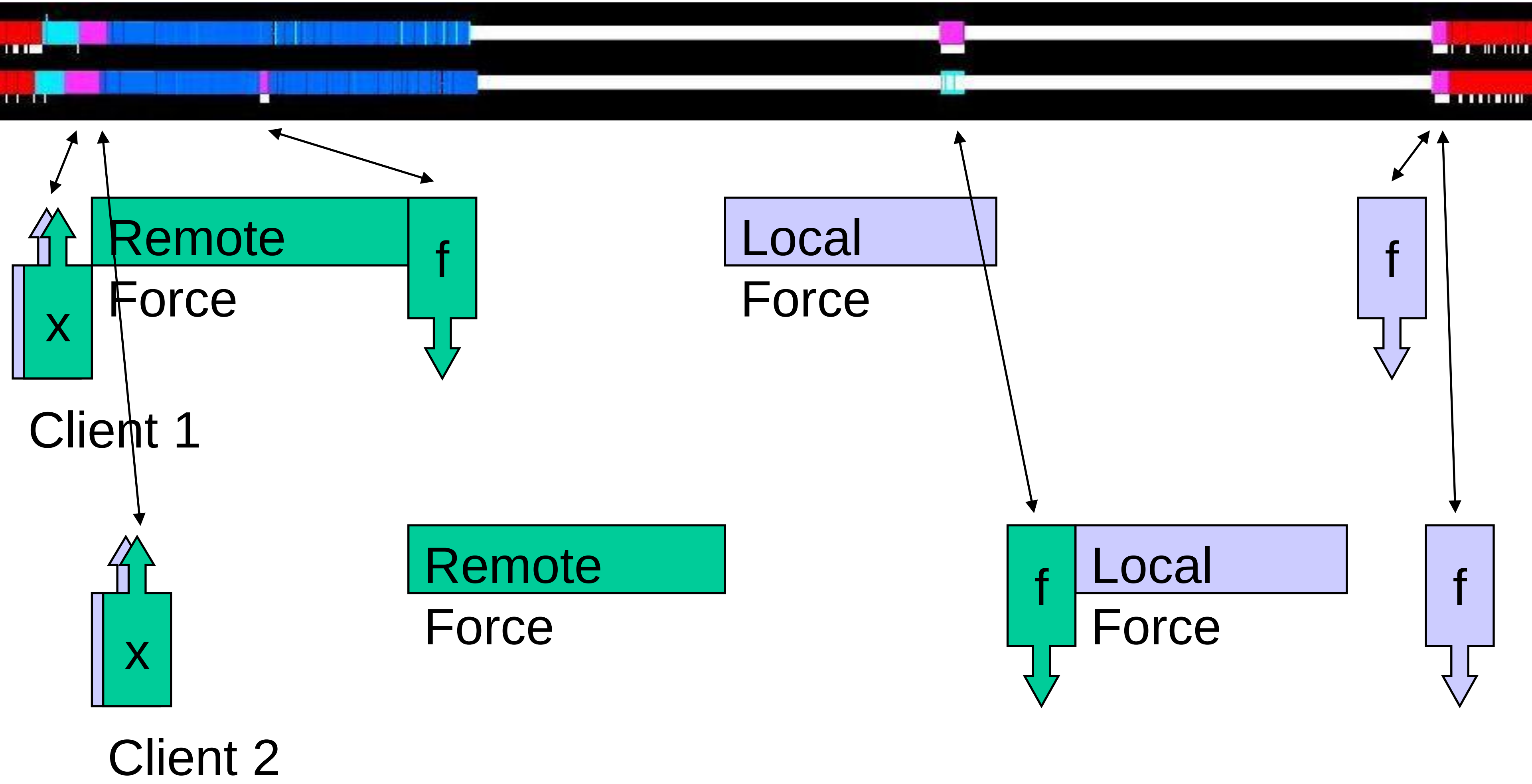
GPU Sharing (Desired)



GPU Sharing (Feared)



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, …

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)
- Faster processors will need a faster network!

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, NVIDIA
- NIH support: P41-RR05969

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