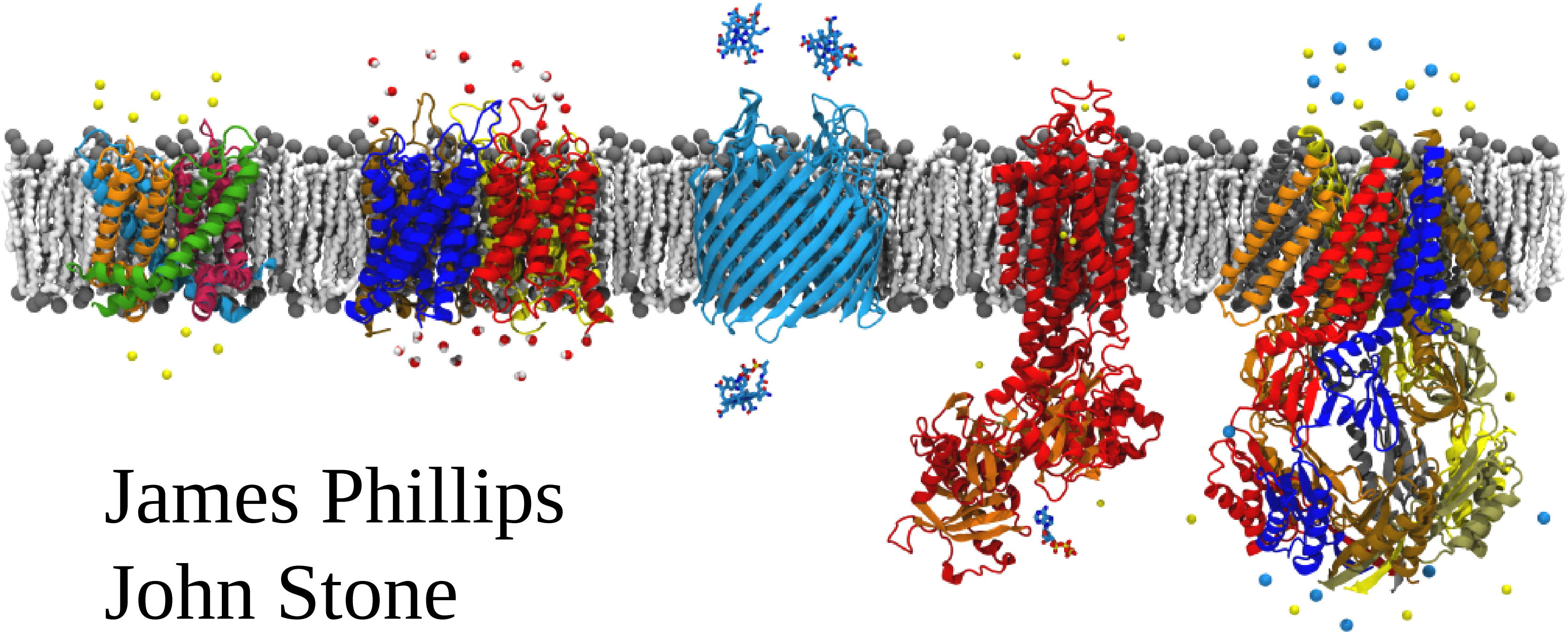
# Adapting a Message-Driven Parallel Application to GPU-Accelerated Clusters



James Phillips

John Stone

Klaus Schulten

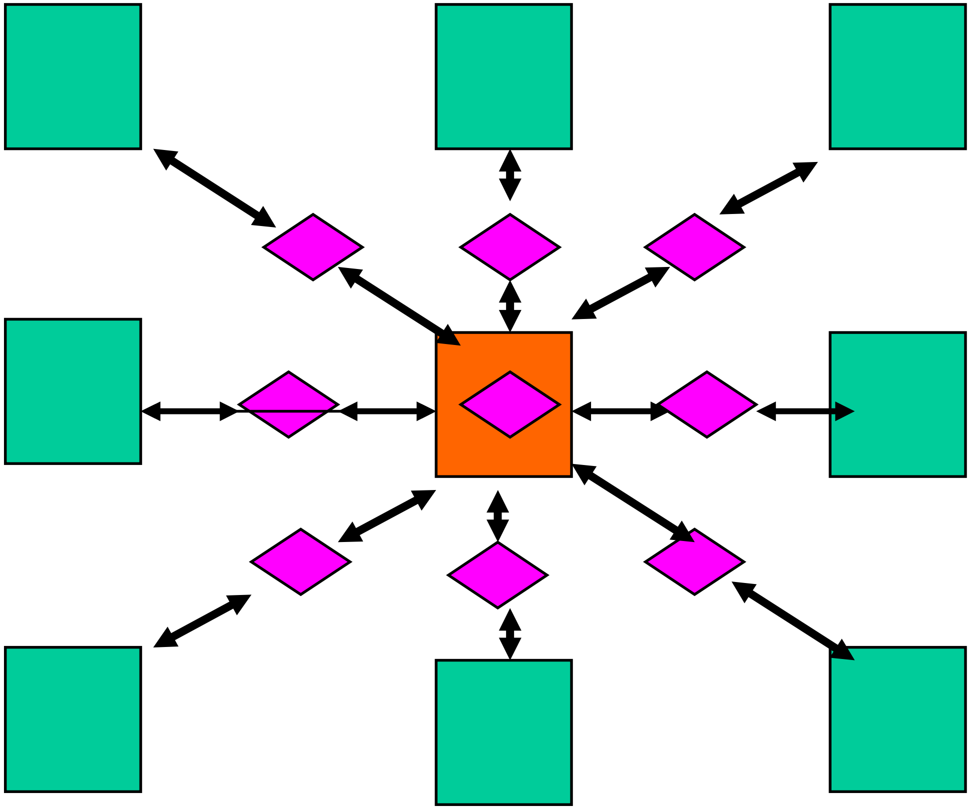
http://www.ks.uiuc.edu/Research/gpu/

# Outline

- NAMD and message-driven programming
- Adapting NAMD to GPU-accelerated clusters
- Old NCSA QP cluster performance results
- New NCSA Lincoln cluster performance results
- Does CUDA like to share?

# NAMD Hybrid Decomposition

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



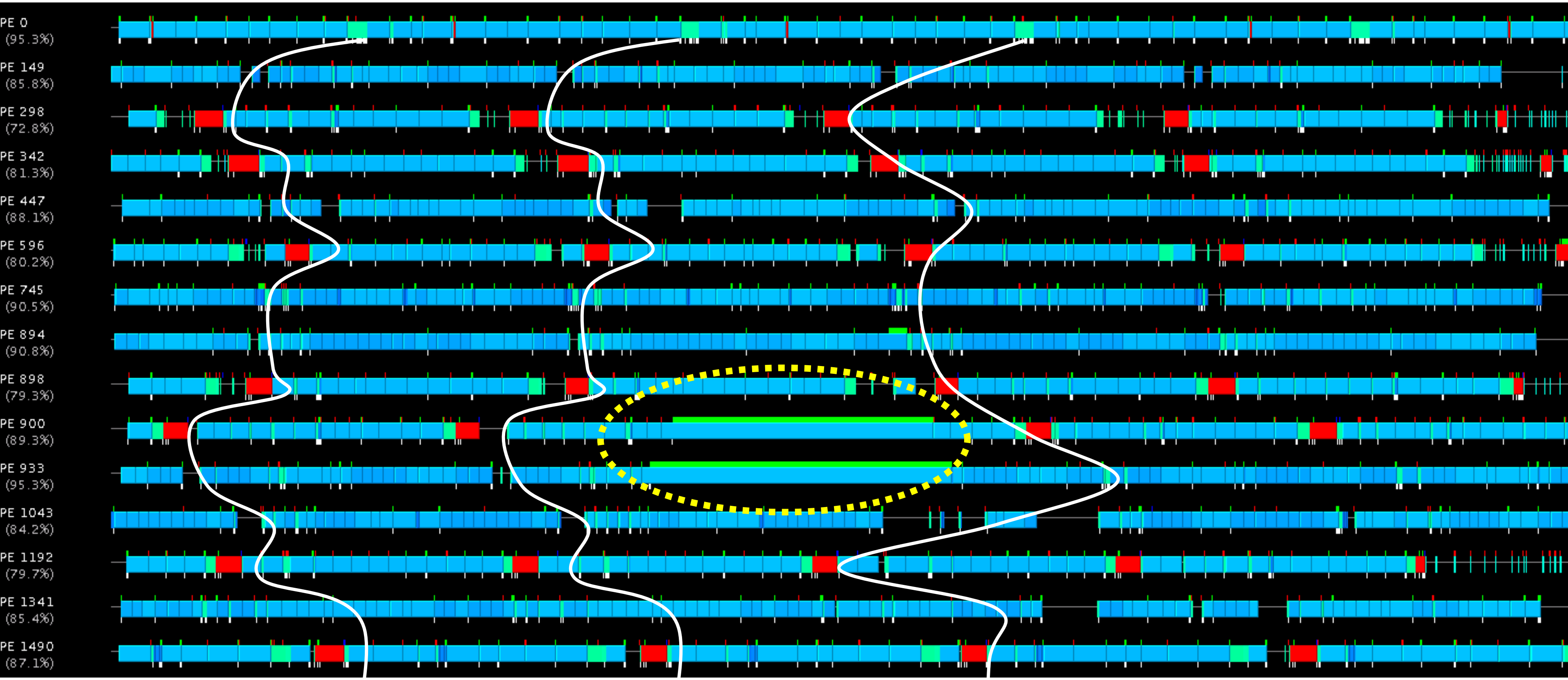
- Spatially decompose data and communication.
- Separate but related work decomposition.
- “Compute objects” facilitate iterative, measurement-based load balancing system.

# 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/

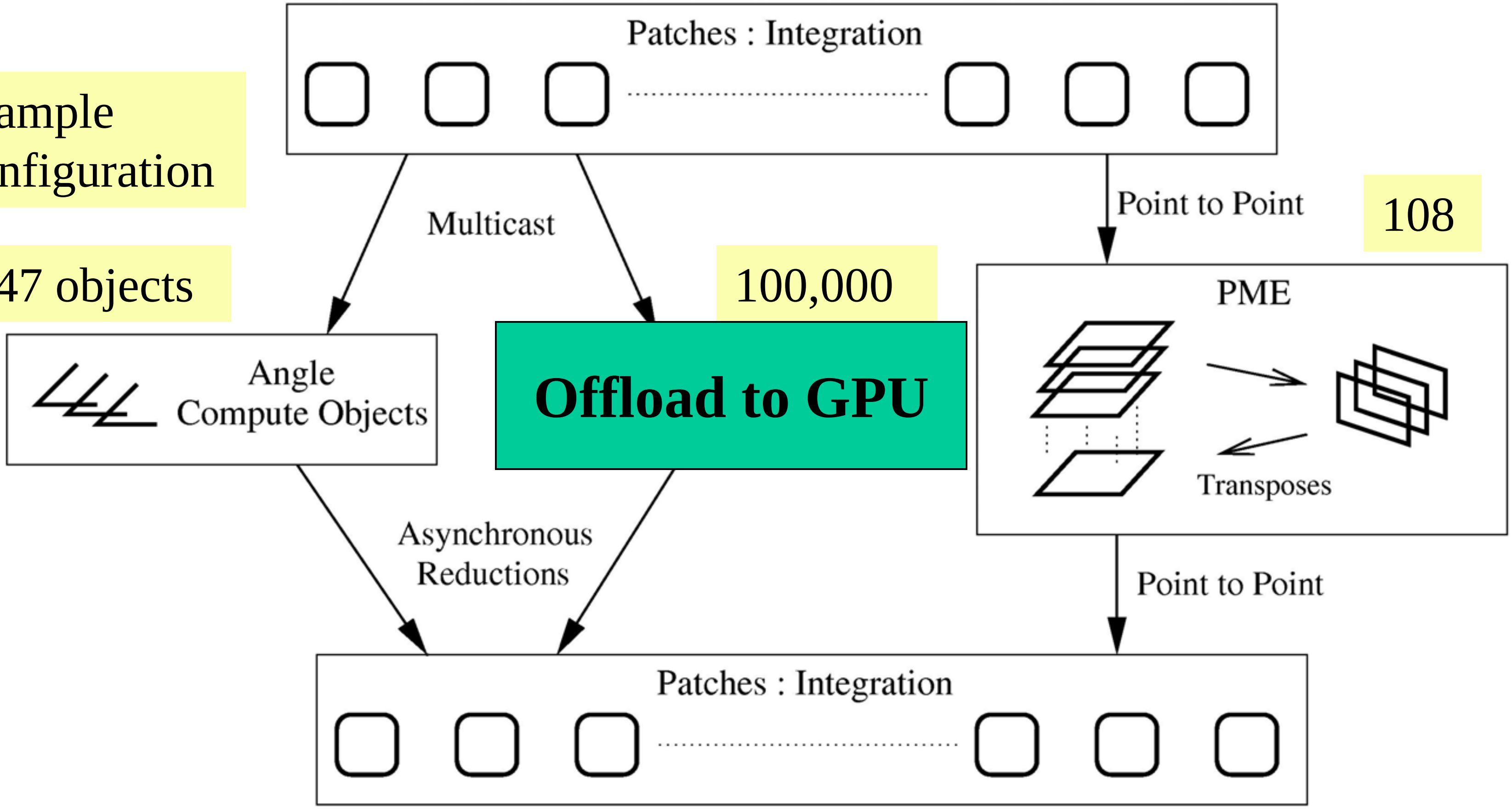
# System Noise Example

## Timeline from Charm++ tool "Projections"



# NAMD Overlapping Execution

Phillips *et al.*, *SC2002*.



Example Configuration

847 objects

100,000

108

**Offload to GPU**

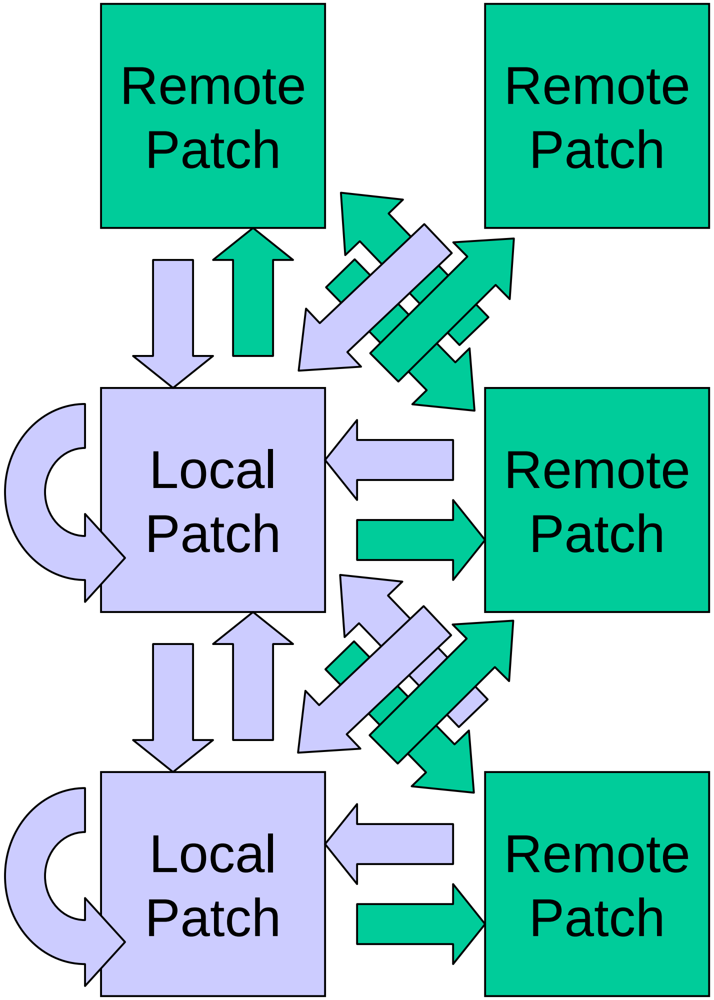
Objects are assigned to processors and queued as data arrives.

# 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.

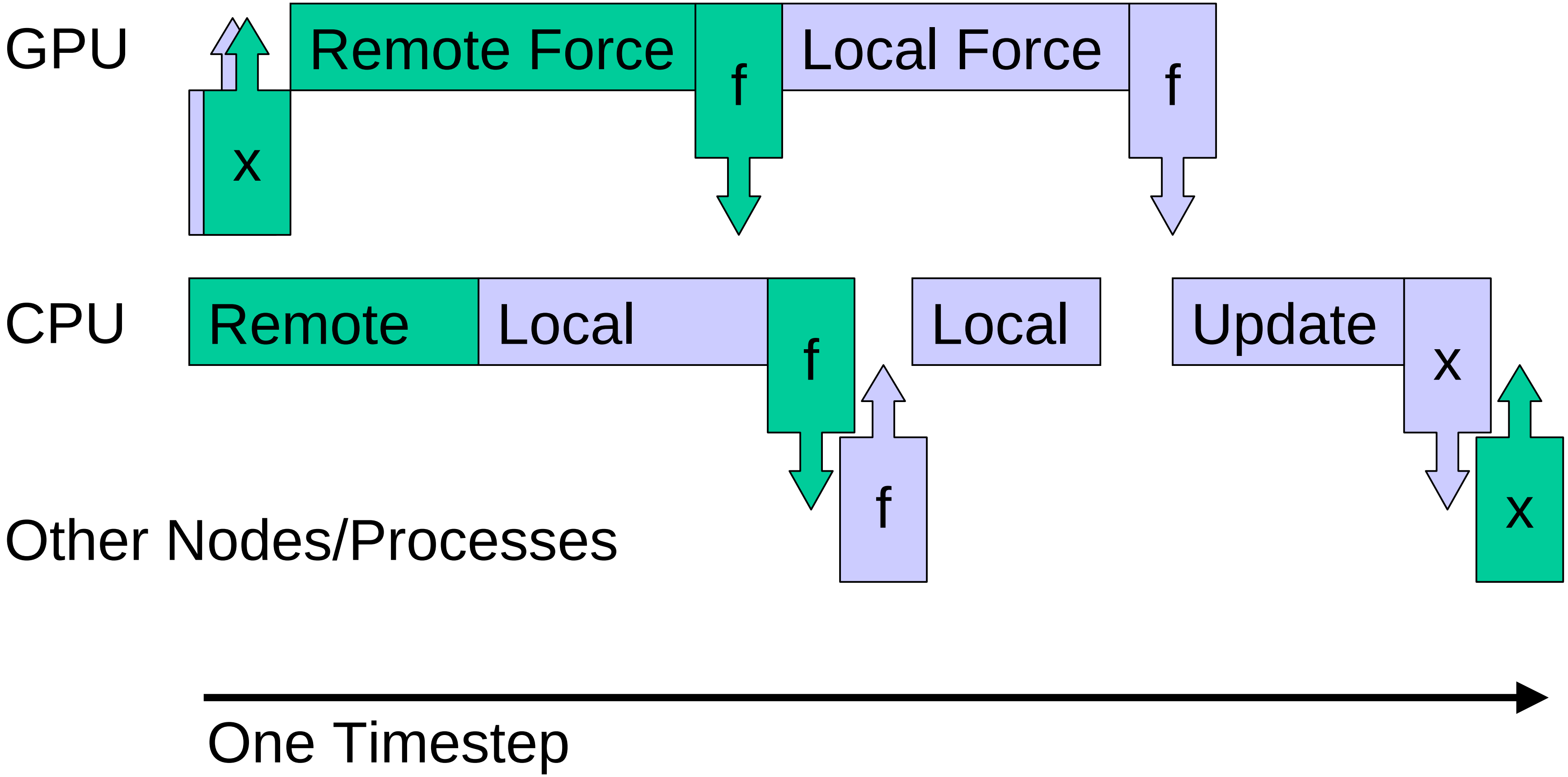
# “Remote Forces”

- 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



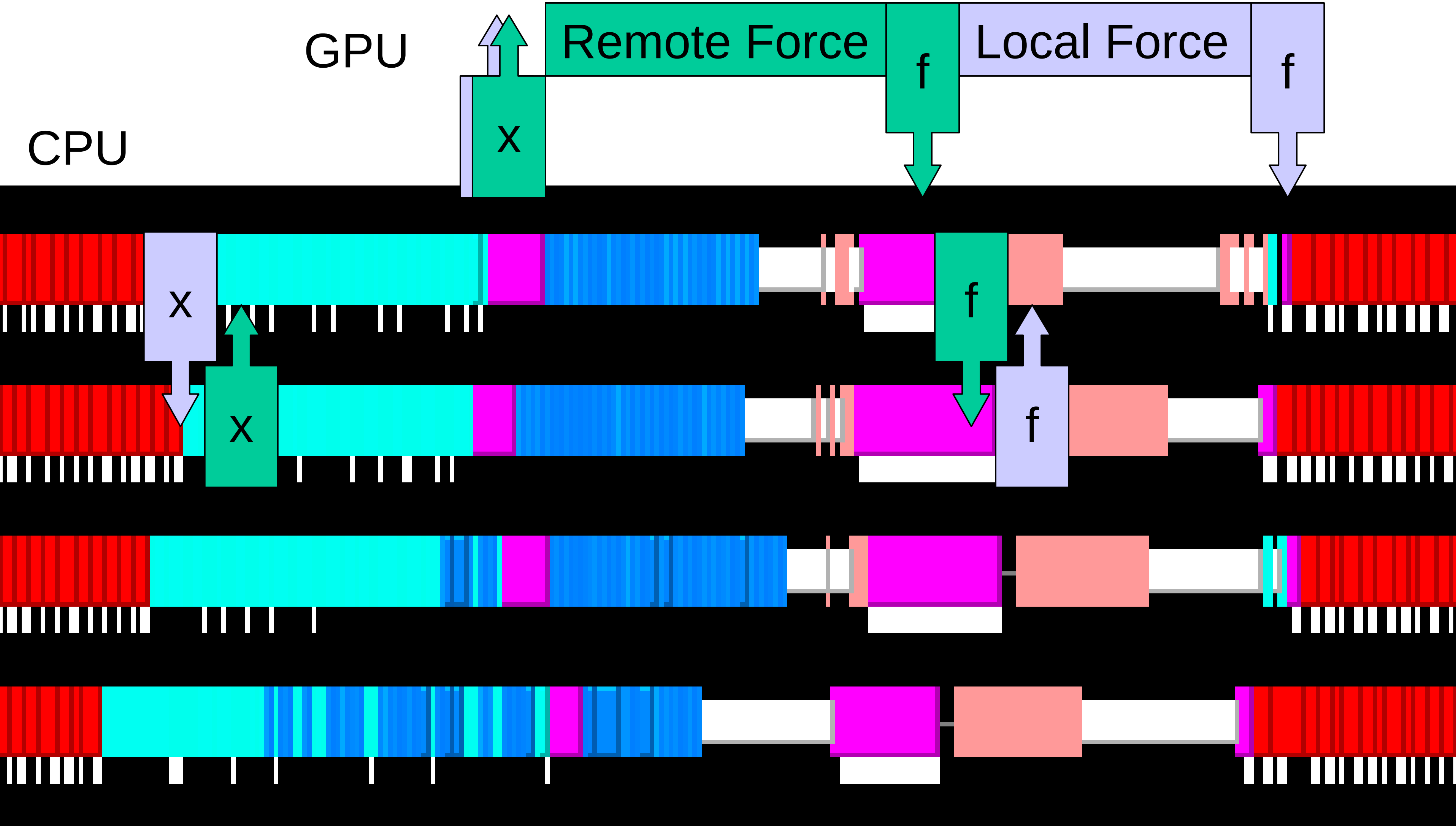
Work done by **one** processor

# Overlapping GPU and CPU with Communication

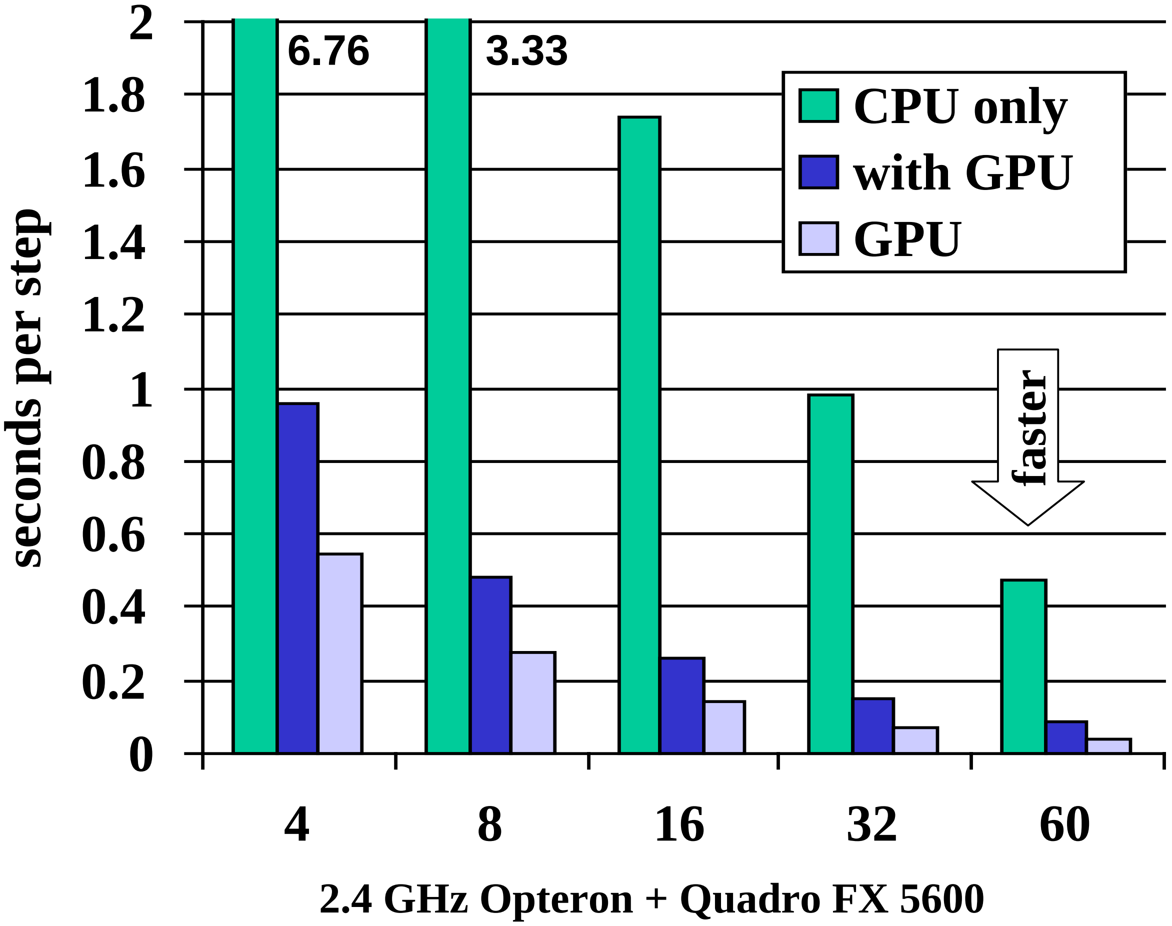


# Actual Timelines from NAMD

Generated using Charm++ tool “Projections”



# NCSA “4+4” QP Cluster



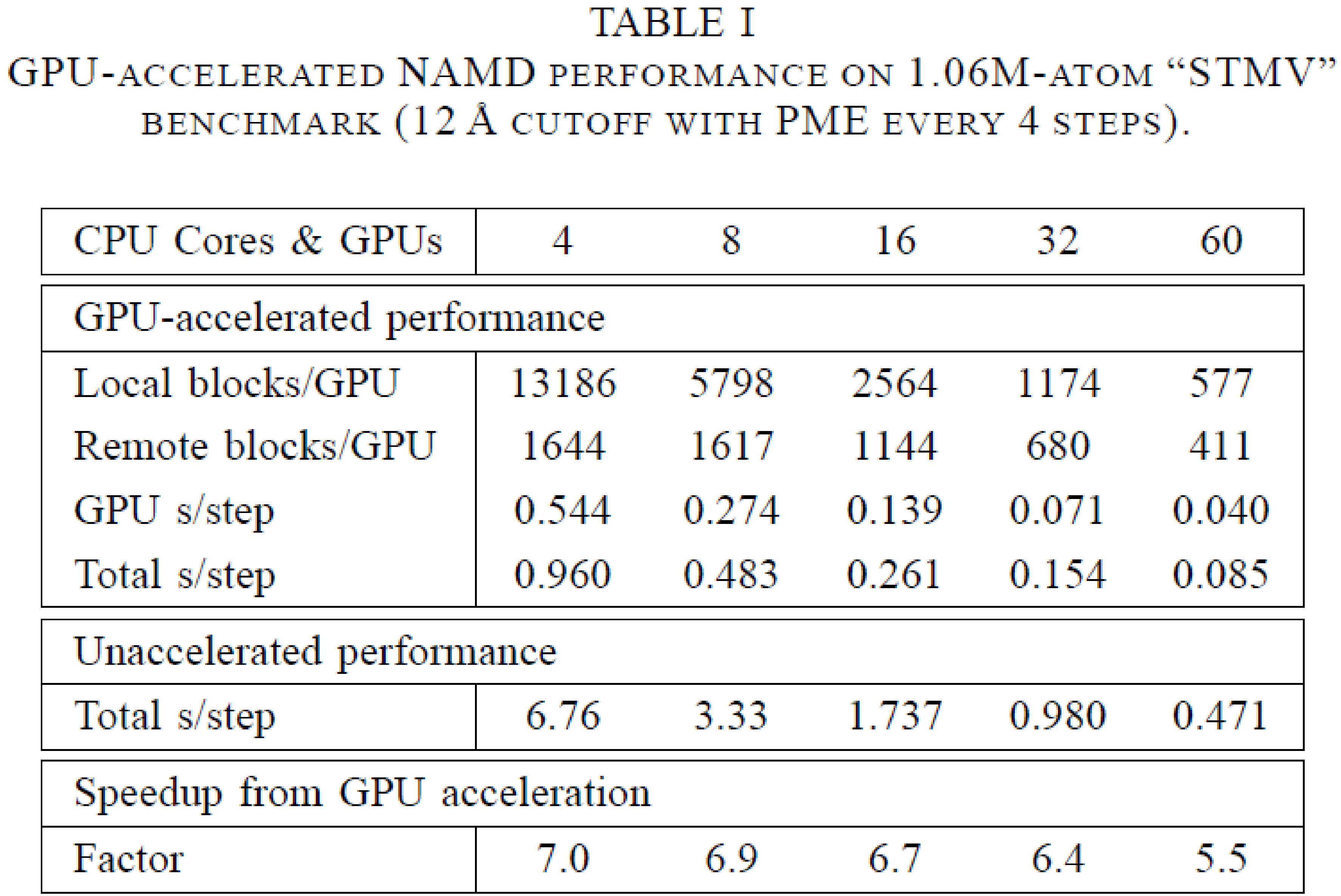
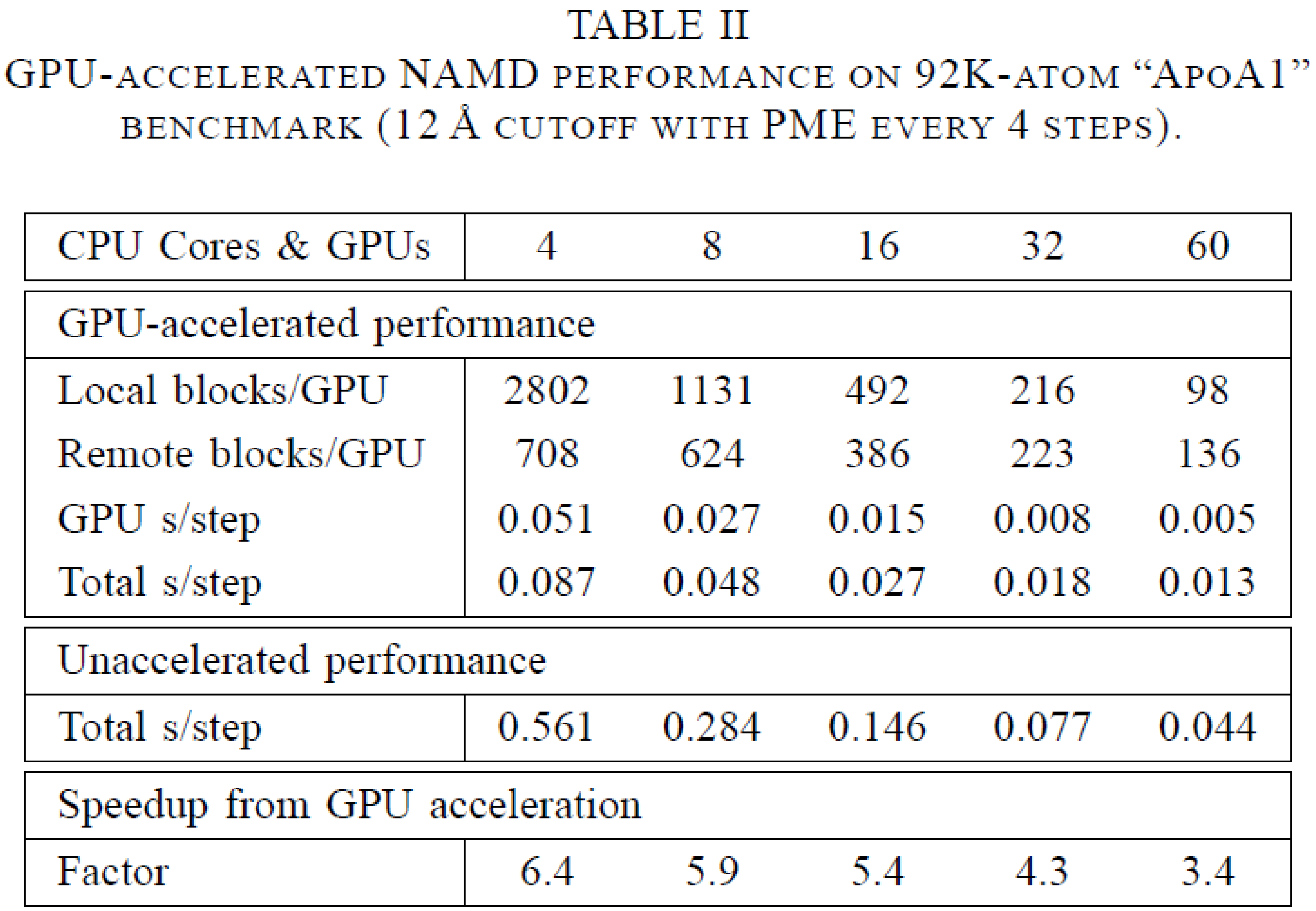


TABLE I

GPU-ACCELERATED NAMD PERFORMANCE ON 1.06M-ATOM "STMV" BENCHMARK (12 Å CUTOFF WITH PME EVERY 4 STEPS).

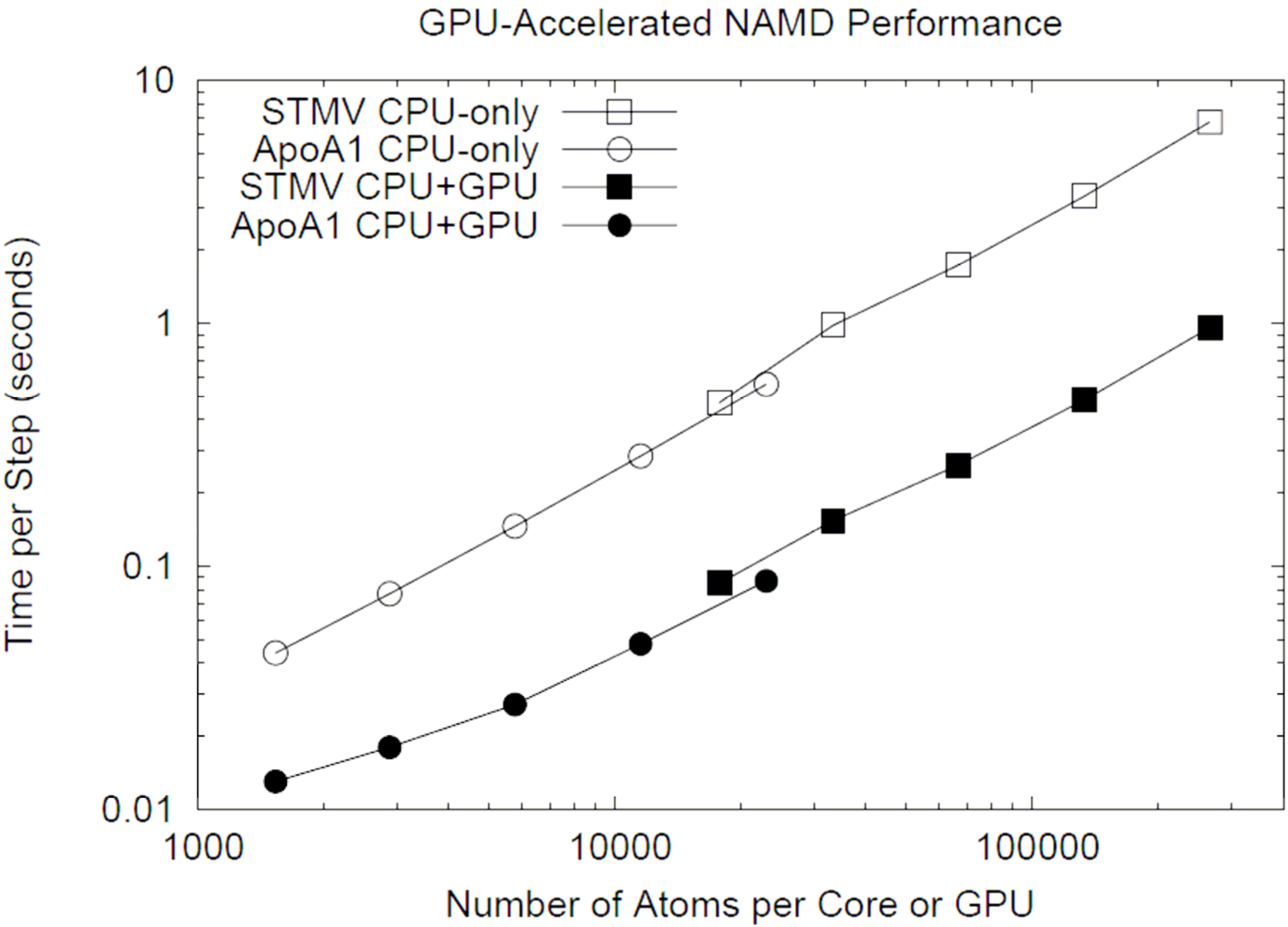
| CPU Cores & GPUs | 4 | 8 | 16 | 32 | 60 |
|---|---|---|---|---|---|
| **GPU-accelerated performance** | | | | | |
| Local blocks/GPU | 13186 | 5798 | 2564 | 1174 | 577 |
| Remote blocks/GPU | 1644 | 1617 | 1144 | 680 | 411 |
| GPU s/step | 0.544 | 0.274 | 0.139 | 0.071 | 0.040 |
| Total s/step | 0.960 | 0.483 | 0.261 | 0.154 | 0.085 |
| **Unaccelerated performance** | | | | | |
| Total s/step | 6.76 | 3.33 | 1.737 | 0.980 | 0.471 |
| **Speedup from GPU acceleration** | | | | | |
| Factor | 7.0 | 6.9 | 6.7 | 6.4 | 5.5 |



# TABLE II

## GPU-accelerated NAMD performance on 92K-atom "ApoA1" benchmark (12 Å cutoff with PME every 4 steps).

| CPU Cores & GPUs | 4 | 8 | 16 | 32 | 60 |
|---|---|---|---|---|---|
| **GPU-accelerated performance** | | | | | |
| Local blocks/GPU | 2802 | 1131 | 492 | 216 | 98 |
| Remote blocks/GPU | 708 | 624 | 386 | 223 | 136 |
| GPU s/step | 0.051 | 0.027 | 0.015 | 0.008 | 0.005 |
| Total s/step | 0.087 | 0.048 | 0.027 | 0.018 | 0.013 |
| **Unaccelerated performance** | | | | | |
| Total s/step | 0.561 | 0.284 | 0.146 | 0.077 | 0.044 |
| **Speedup from GPU acceleration** | | | | | |
| Factor | 6.4 | 5.9 | 5.4 | 4.3 | 3.4 |


GPU-Accelerated NAMD Performance
STMV CPU-only □
ApoA1 CPU-only ○
STMV CPU+GPU ■
ApoA1 CPU+GPU ●
Time per Step (seconds)
10
1
0.1
0.01
1000
10000
100000
Number of Atoms per Core or GPU

# GPU Cluster Observations

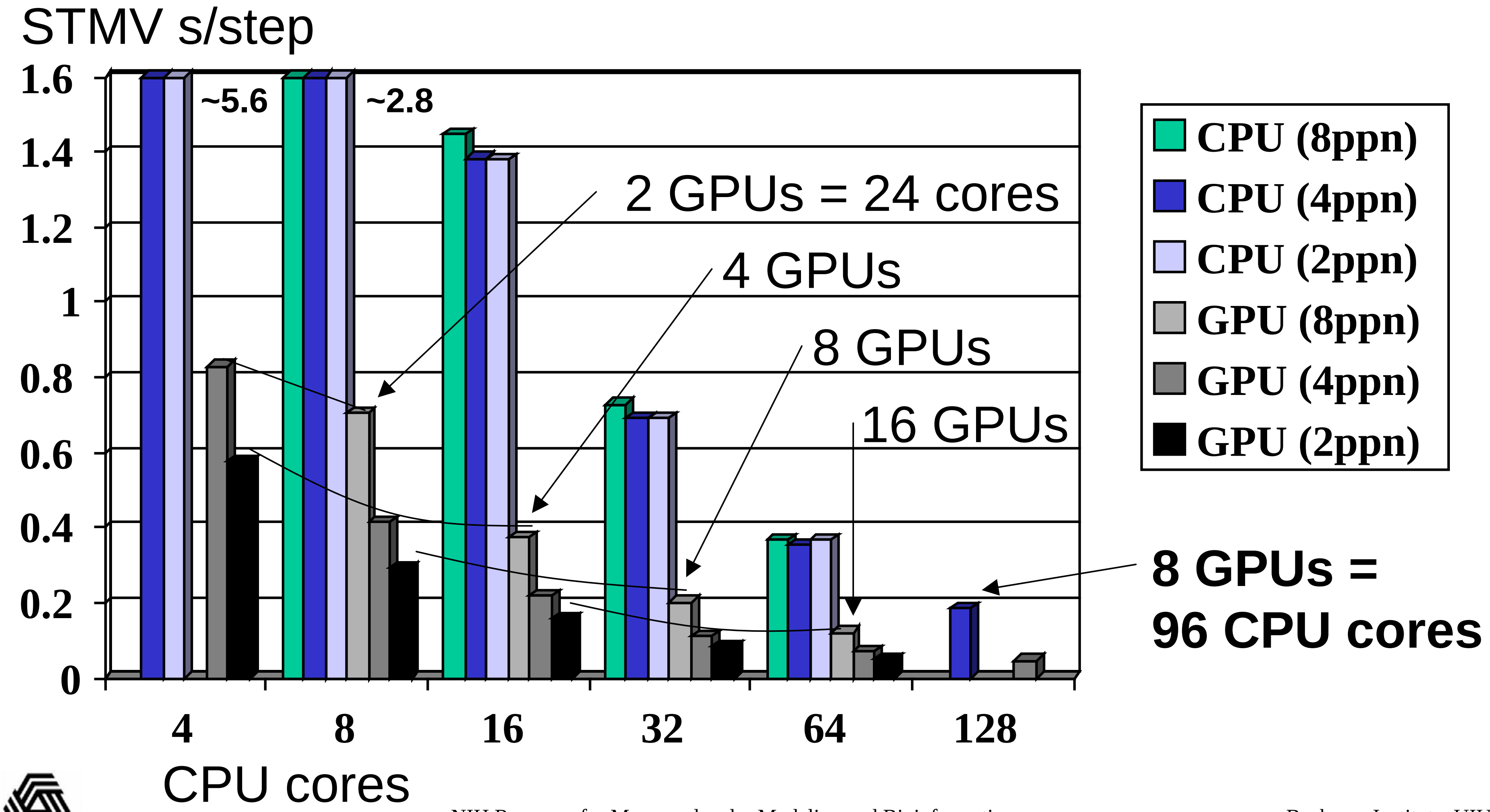
- Tools needed to control GPU allocation
  - Simplest solution is rank % devicesPerNode
  - Doesn’t work with multiple independent jobs
- CUDA and MPI can’t share pinned memory
  - Either user copies data or disable MPI RDMA
  - Need interoperable user-mode DMA standard
- Speaking of extra copies…
 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



2 GPUs = 24 cores

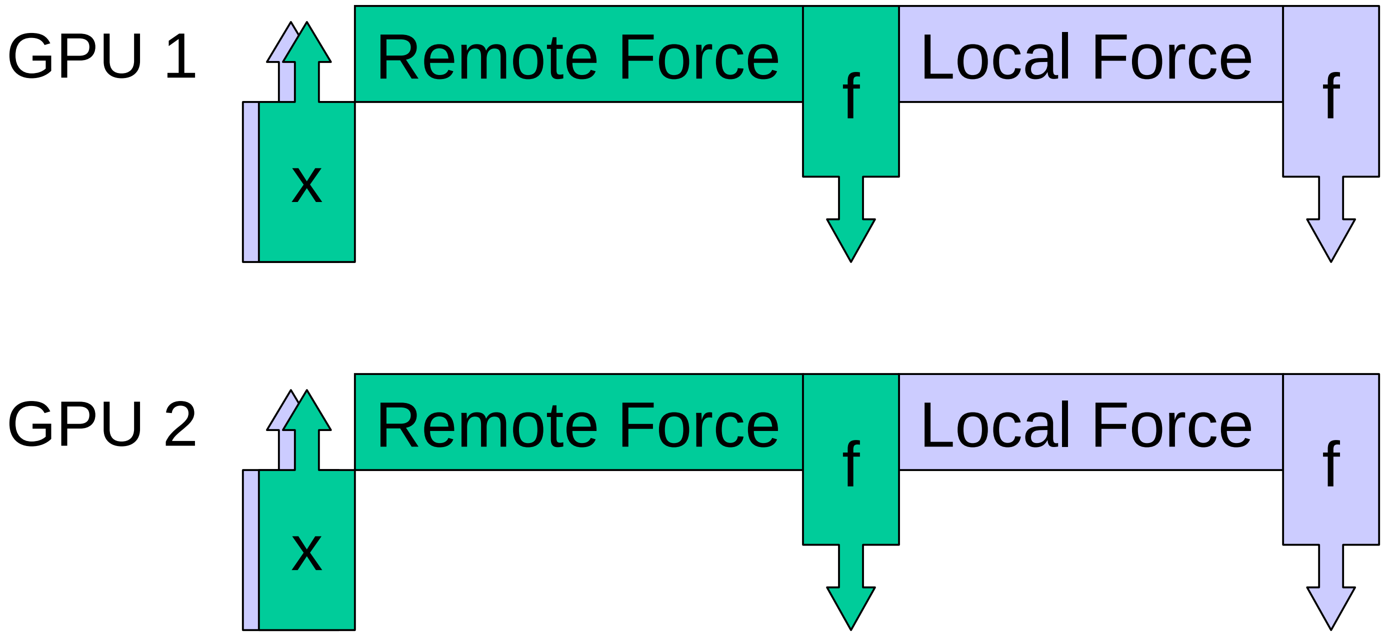
4 GPUs

8 GPUs

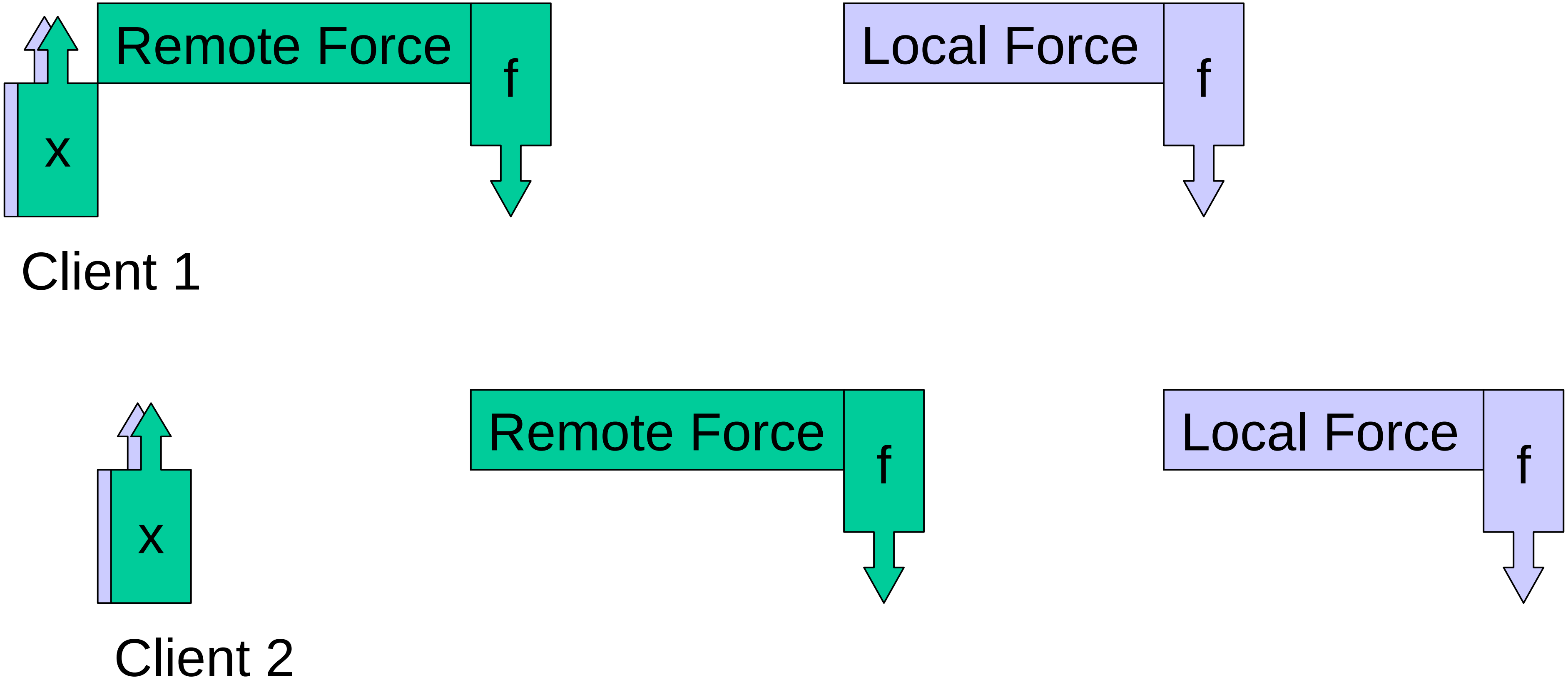
16 GPUs

**8 GPUs =**
**96 CPU cores**

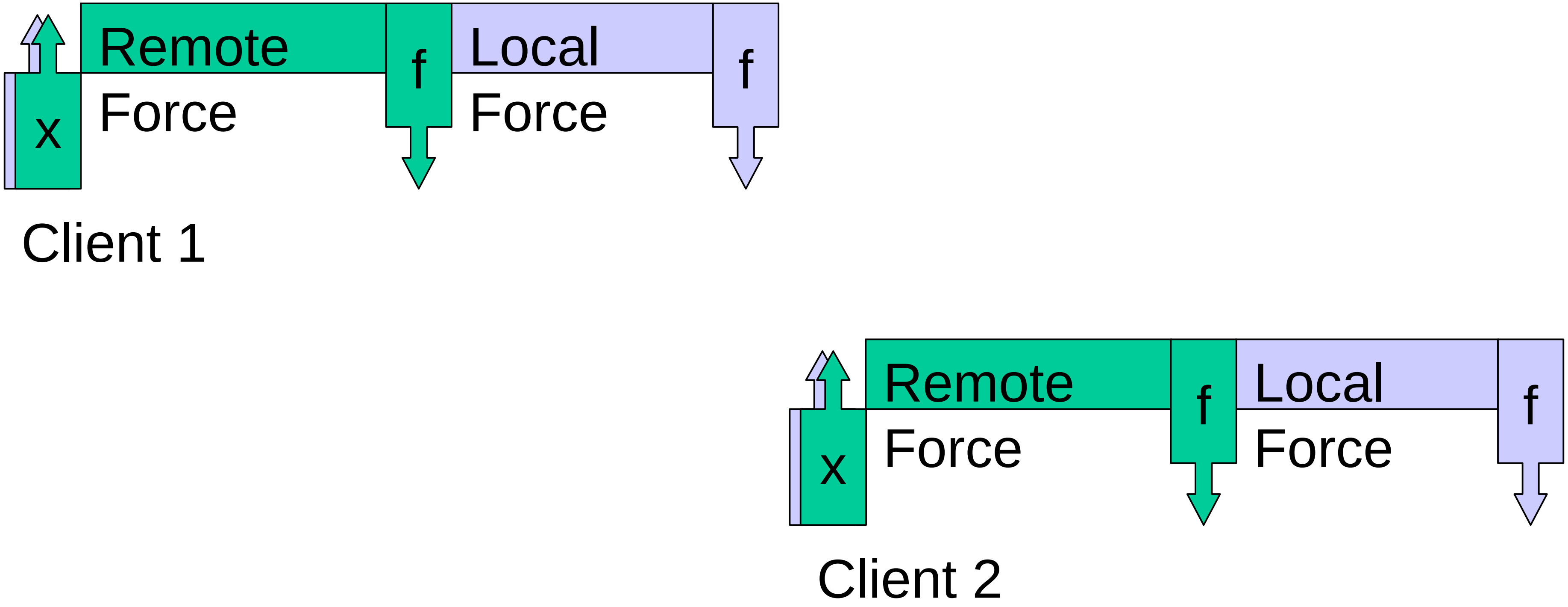
# No GPU Sharing (Ideal World)



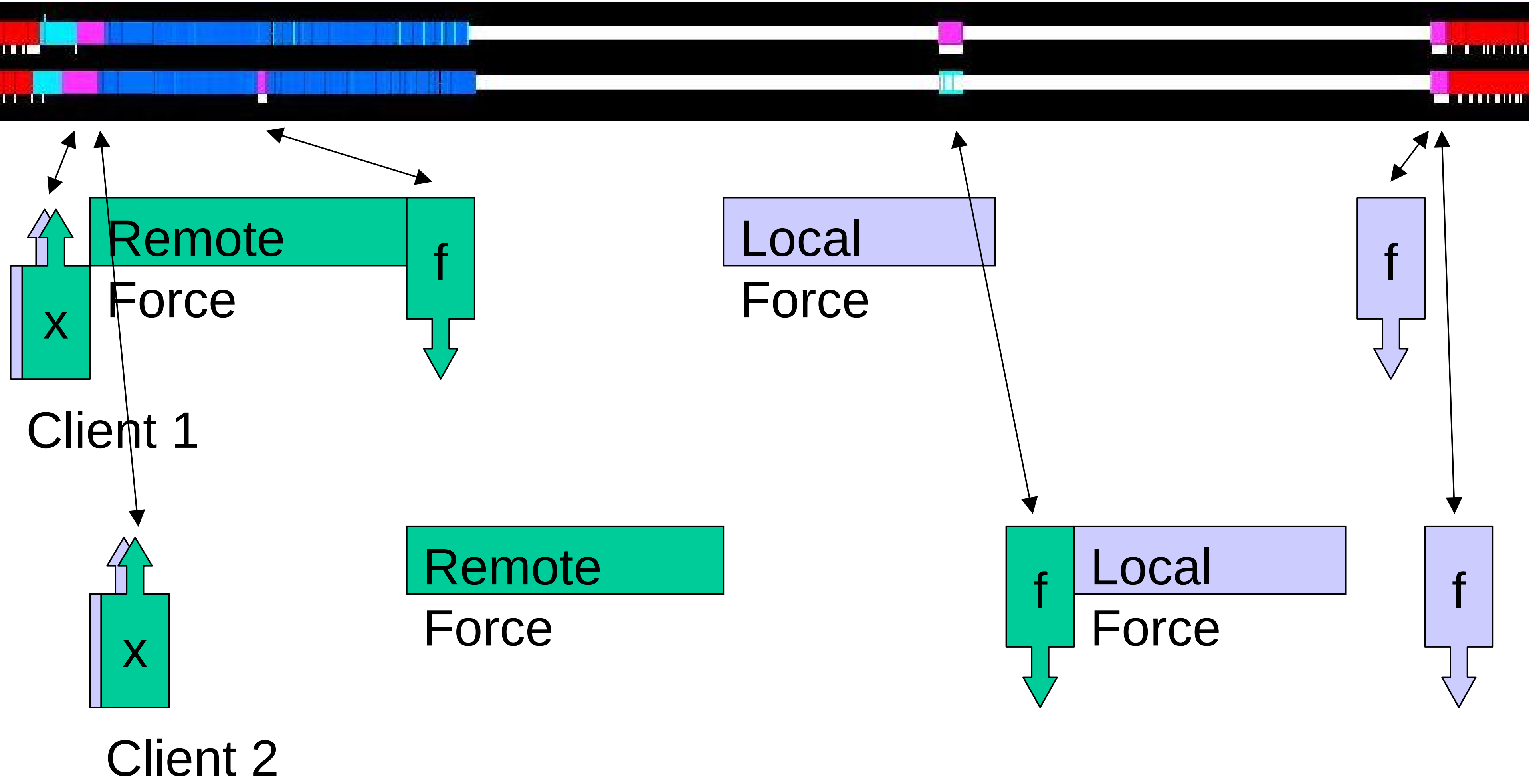
# GPU Sharing (Desired)



# 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:
  - Use locks (atomics) to merge kernels (not G80)
  - Explicit inter-client coordination

# 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/**