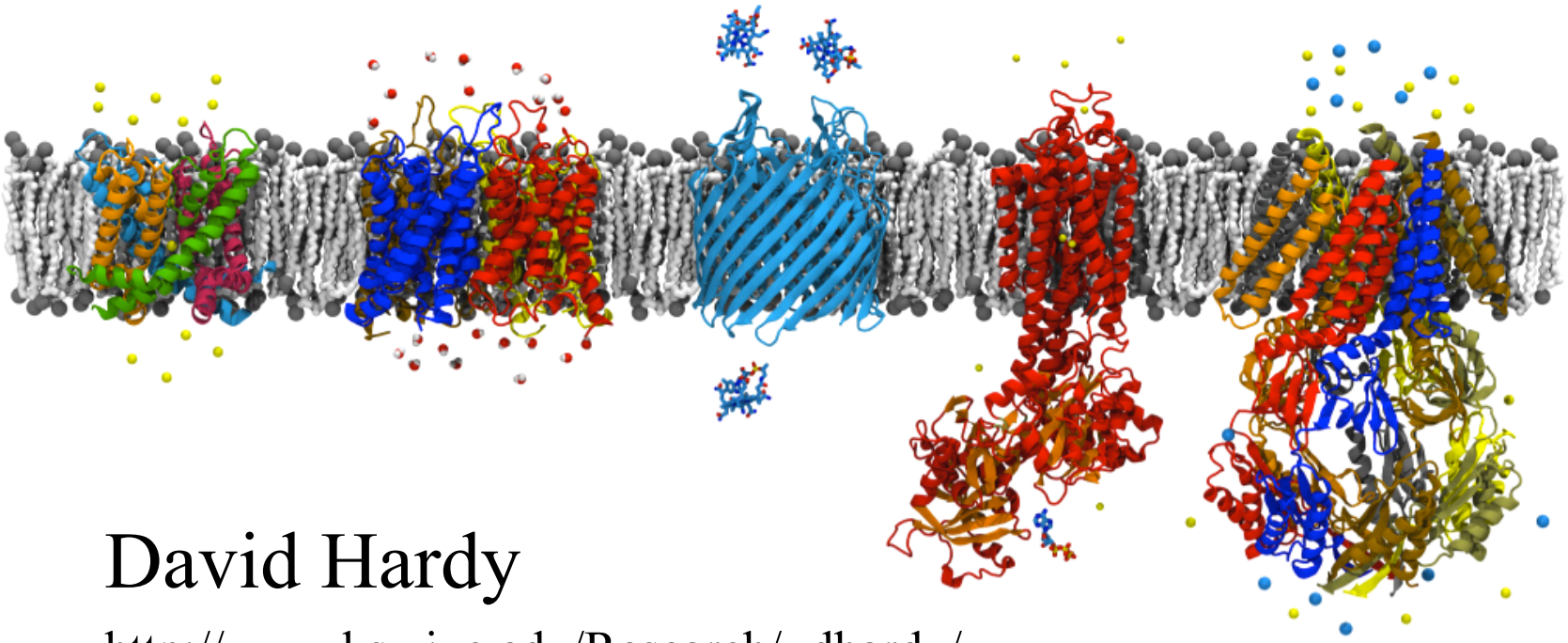
Short-range (Non-bonded) Interactions in NAMD



David Hardy

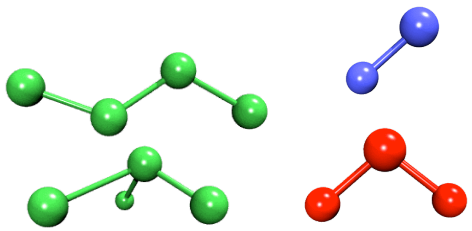
http://www.ks.uiuc.edu/Research/~dhardy/

NAIS: State-of-the-Art Algorithms for Molecular Dynamics

(Presenting the work of James Phillips.)

Molecular Dynamics

$$m_i \frac{d^2\vec{r}_i}{dt^2} = \vec{F}_i = -\vec{\nabla} U(\vec{R})$$ ← Integrate for 1 billion time steps



$$U(\vec{R}) = \underbrace{\sum_{bonds} k_i^{bond}(r_i - r_0)^2}_{U_{bond}} + \underbrace{\sum_{angles} k_i^{angle}(\theta_i - \theta_0)^2}_{U_{angle}} +$$

$$\underbrace{\sum_{dihedrals} k_i^{dihe}[1 + \cos(n_i\phi_i + \delta_i)]}_{U_{dihedral}} +$$

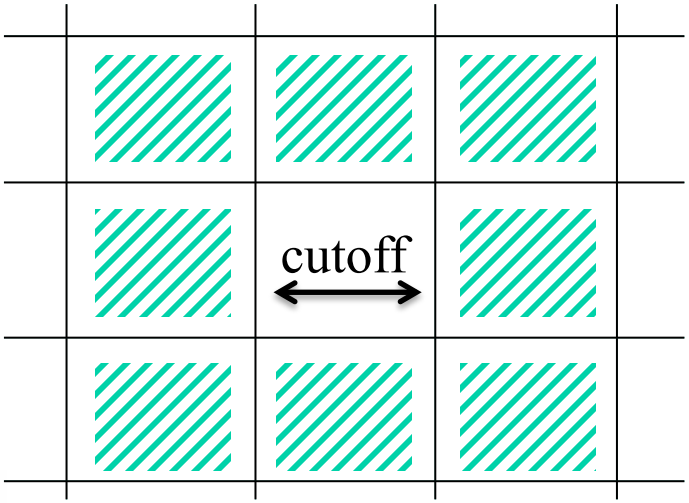
Non-bonded interactions require most computation →

$$\underbrace{\sum_i \sum_{j \neq i} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] + \sum_i \sum_{j \neq i} \frac{q_i q_j}{\epsilon r_{ij}}}_{U_{nonbond}}$$

van der Waals electrostatics

Short-range Non-bonded Interactions

- Sum interactions within cutoff distance a:
  - Perform spatial hashing of atoms into grid cells
  - For every grid cell, for each atom:
    - Loop over atoms in each neighboring cell
    - If $r_{ij}^2 < a^2$, sum potential energy, virial, and atomic forces
  - Use Newton's 3rd Law: $f_{ij} = -f_{ji}$



If cutoff distance is no bigger than cell, then loop over nearest neighbors

NAMD: *grid cells* are "patches"

NAMD: *spatial hashing* is "migration"

Excluded Pairs

- Self interactions are excluded
- Typically exclude pairs of atoms that are covalently bonded to each other or to a common atom
- Possible approaches:
  - Ignore and correct later
    - But this can cause large numerical errors
  - Detect during evaluation and skip

Algorithmic Enhancements (1)

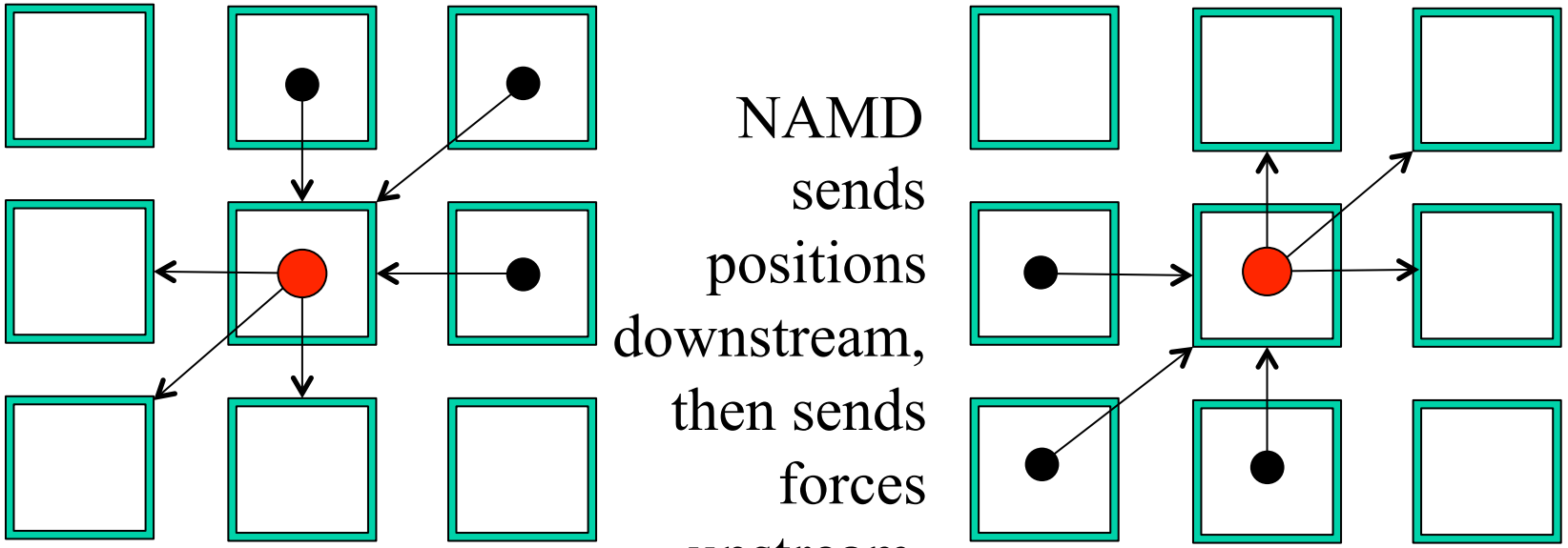
- Maintain pair lists
  - For each atom i, keep list of atoms j within cutoff
  - Extend cutoff distance ($a+\delta$), no update needed until an atom moves distance $\delta/2$
- Maintain "hydrogen groups"
  - Reduce amount of pairwise testing between atoms
  - Let ε be upper bound on hydrogen bond length
  - Test distance between "parent" atoms
    - If $r_{ij}^2 < (a - 2\varepsilon)^2$, then all atoms interact
    - If $r_{ij}^2 > (a + 2\varepsilon)^2$, then no atoms interact
    - Otherwise have to test all pairs

Algorithmic Enhancements (2)

- Combine pair lists and hydrogen groups
  - Use hydrogen groups to shortcut pair list generation
  - Check exclusions only when generating pair lists
  - During force computation, just need to test cutoff
- Interpolation tables for interactions
  - Avoid erfc and exp functions needed for PME
  - Avoid rsqrt (on x86)
  - Avoid additional branching and calculation for van der Waals switching function

Short-range Parallelization

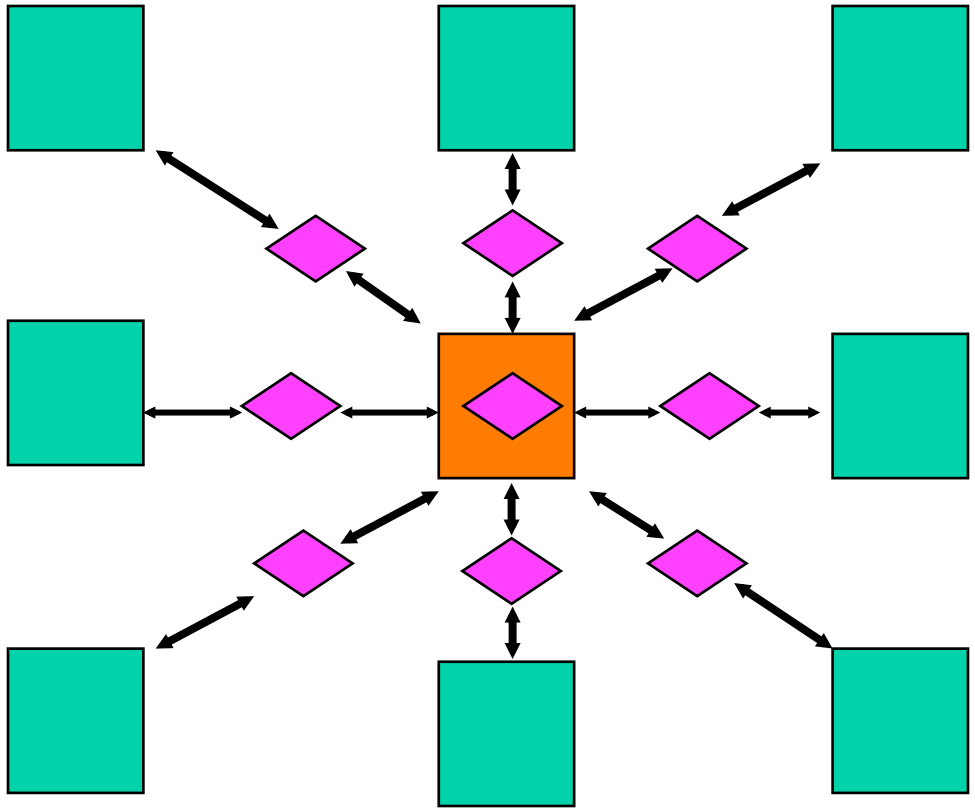
- Spatial decomposition
- Assign grid cells to PEs
- Maps naturally to 3D mesh topology
  - Communication with nearest neighbors



NAMD sends positions downstream, then sends forces upstream.

NAMD Hybrid Decomposition

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



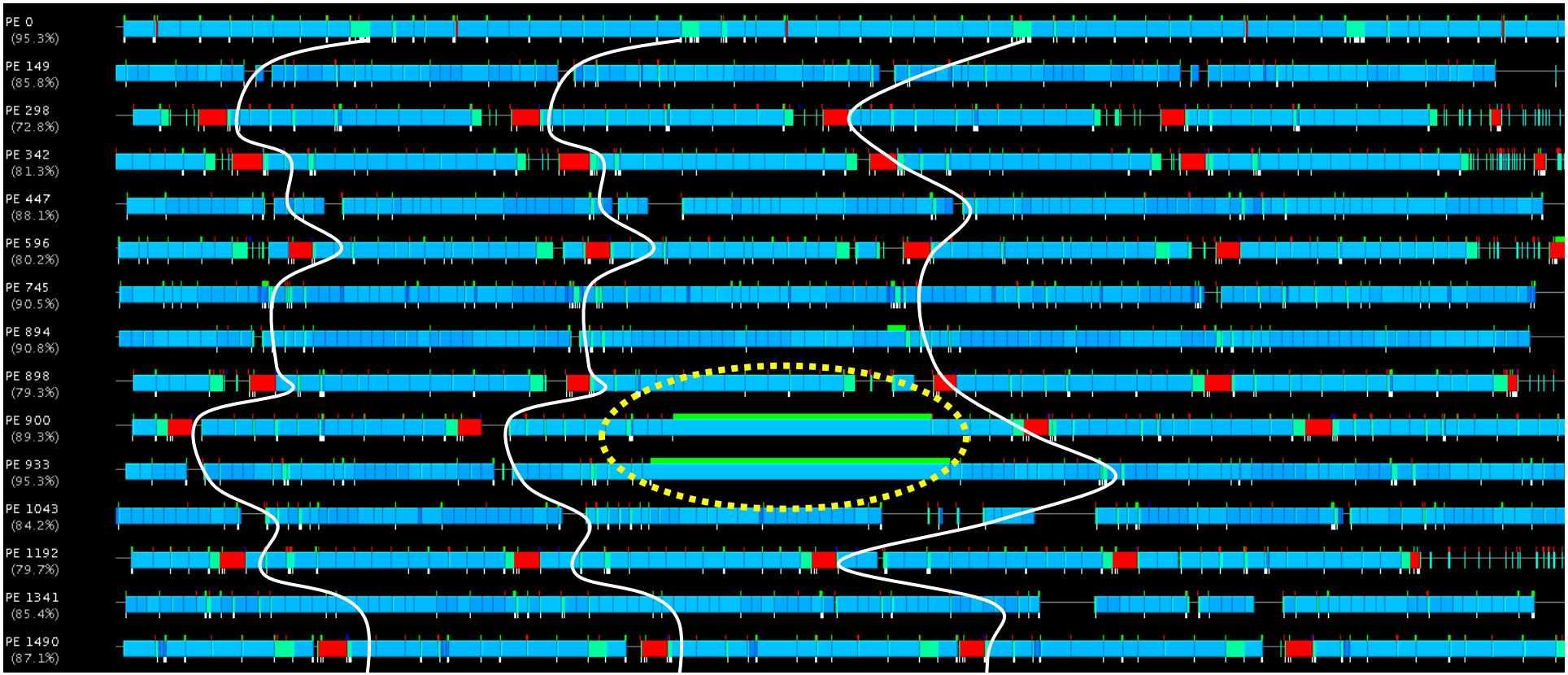
- Spatially decompose data and communication.
- Separate but related work decomposition.
- "Compute objects" facilitate iterative, measurement-based load balancing system.

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/

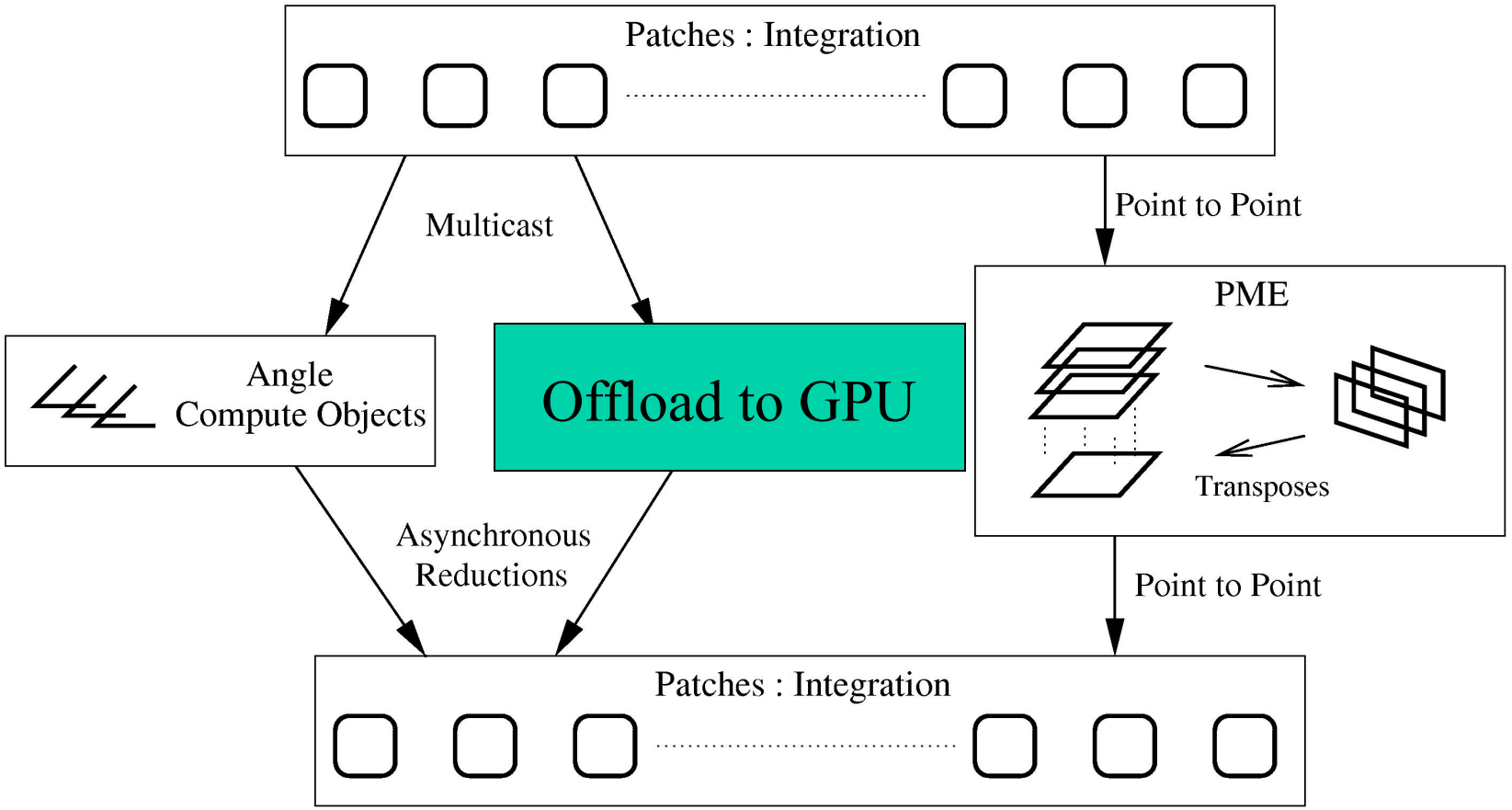
System Noise Example

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



NAMD Overlapping Execution

Phillips *et al., SC2002*.



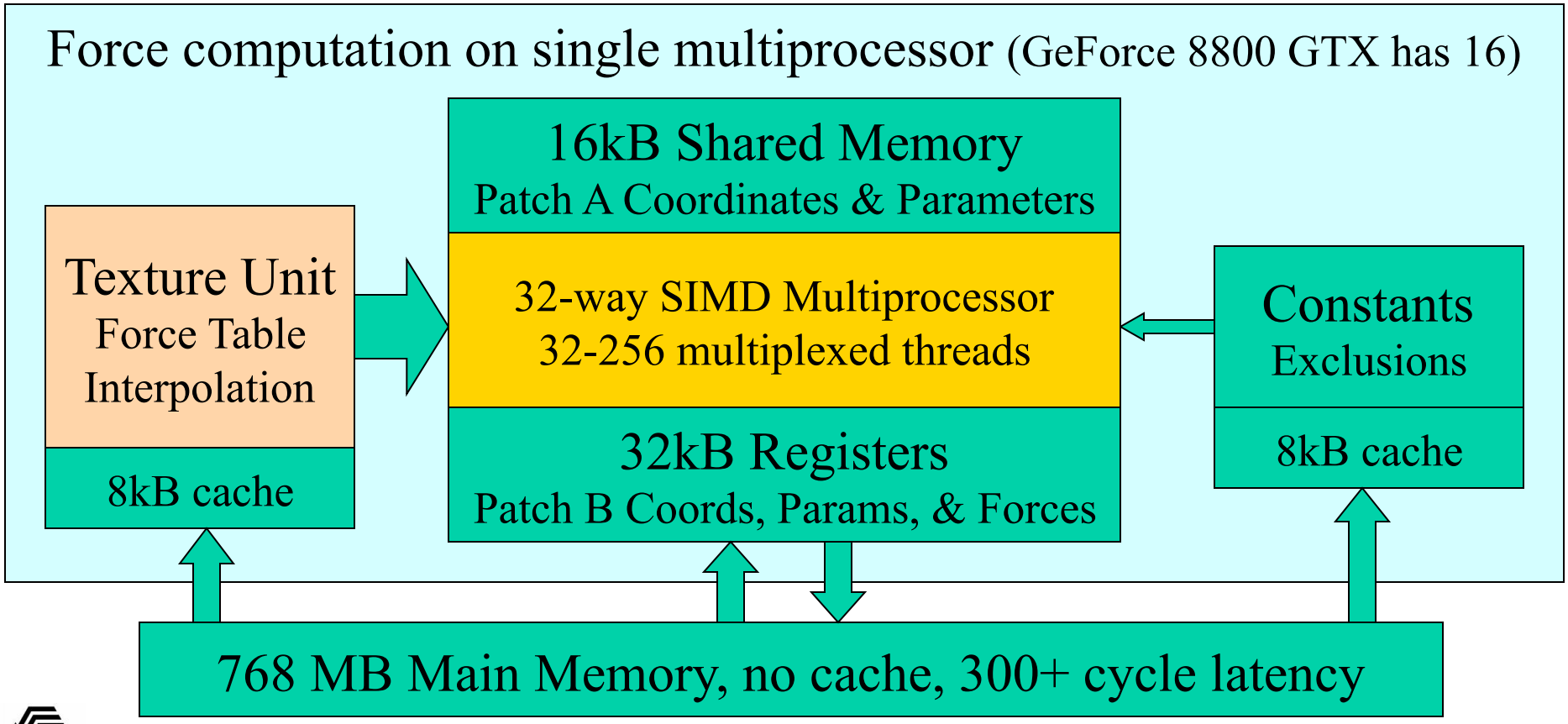
Objects are assigned to processors and queued as data arrives.

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.

Short-range 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.



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

Short-range 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
  }
}
```

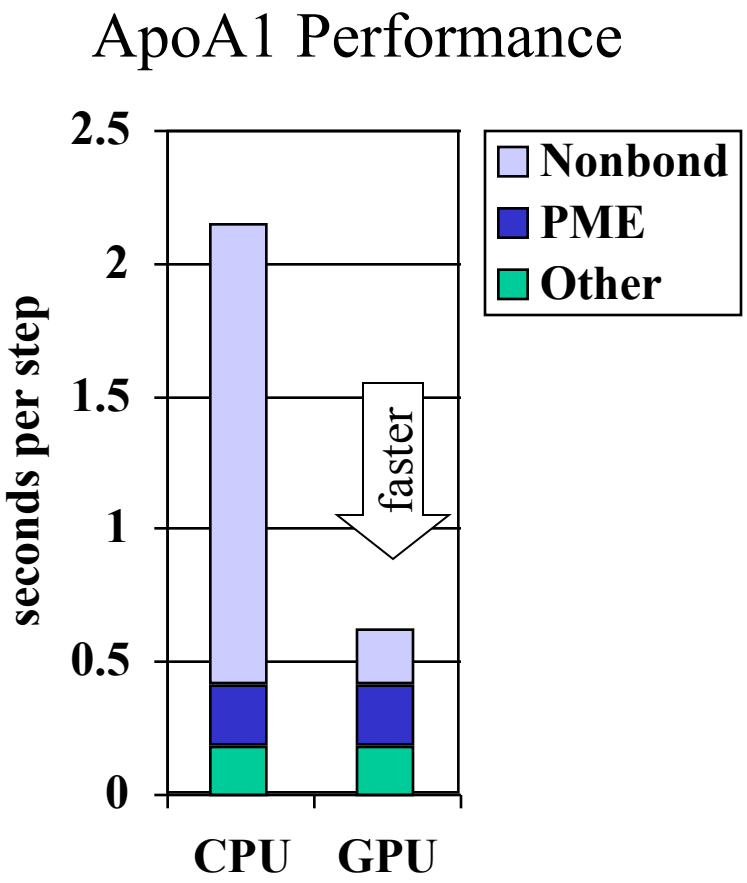
Stone *et al.*, *J. Comp. Chem.* 28:2618-2640, 2007.

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)

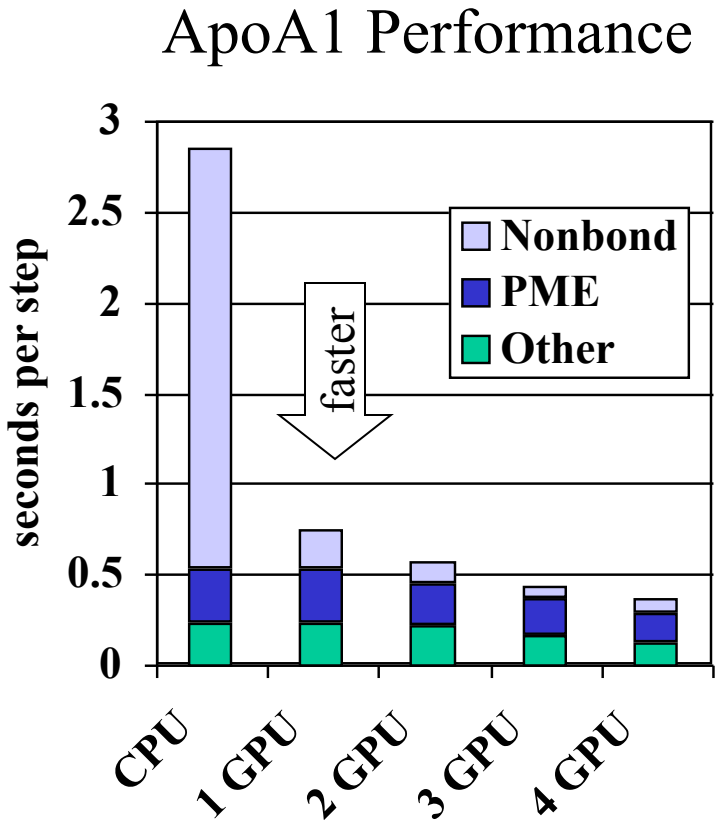
- Full NAMD, not test harness
- Useful performance boost
  - 8x speedup for nonbonded
  - 5x speedup overall w/o PME
  - 3.5x speedup overall w/ PME
  - GPU = quad-core CPU
- Plans for better performance
  - Overlap GPU and CPU work.
  - Tune or port remaining work.
    - PME, bonded, integration, etc.



2.67 GHz Core 2 Quad Extreme + GeForce 8800 GTX

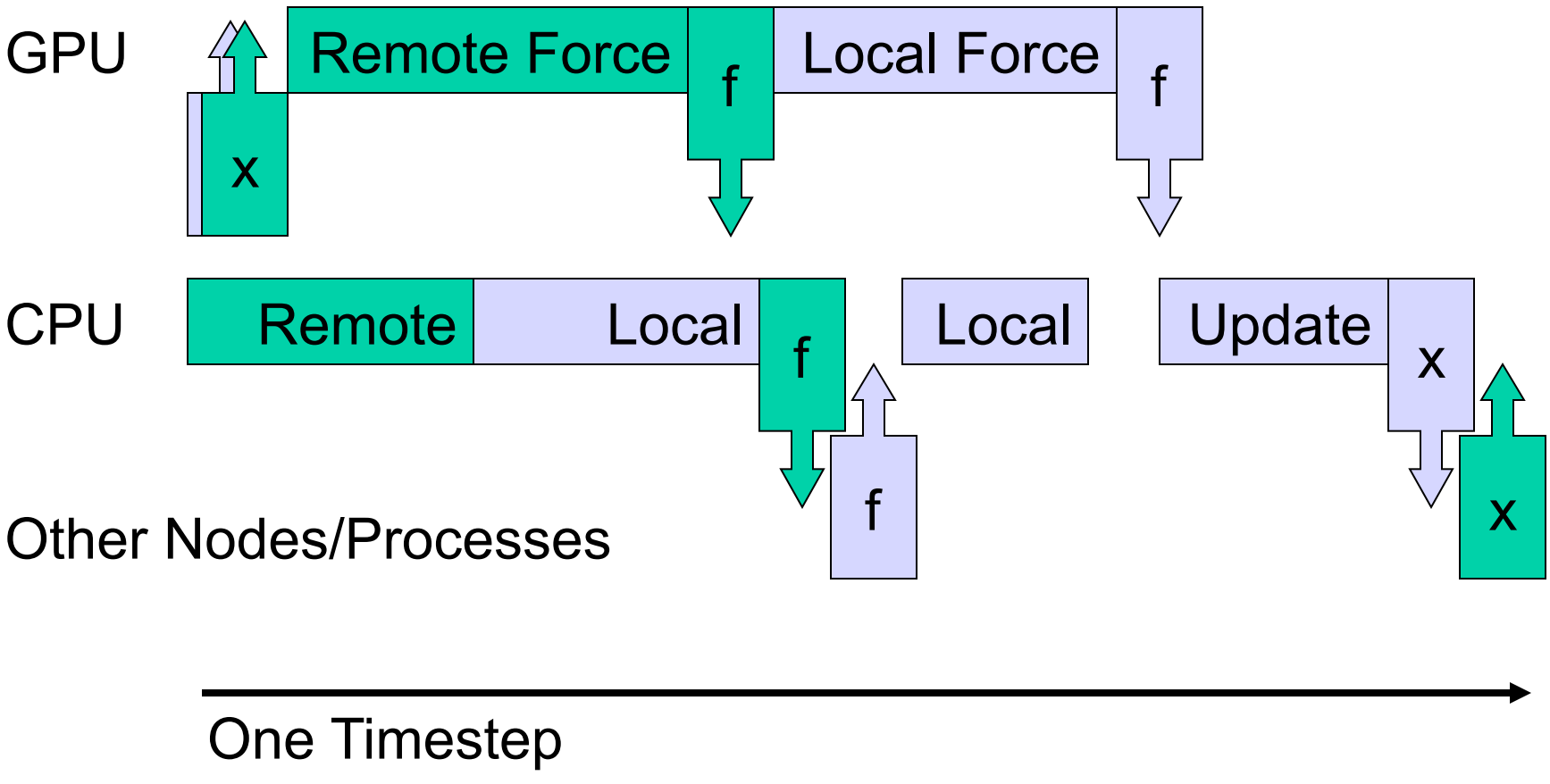
2007 GPU Cluster Performance

- 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.



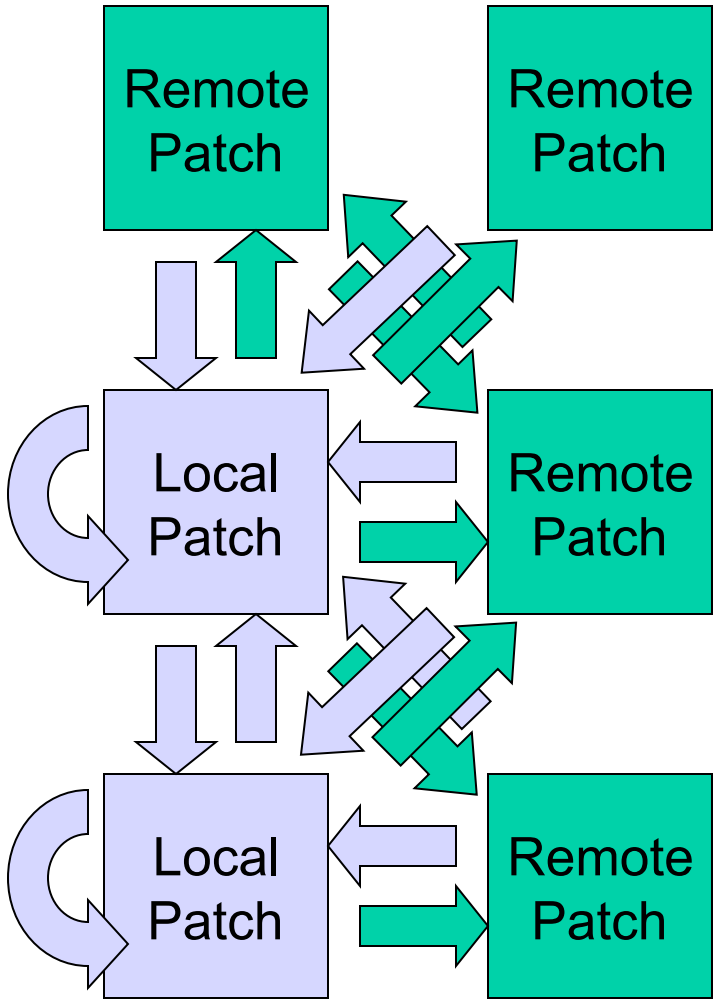
2.2 GHz Opteron + GeForce 8800 GTX

Overlapping GPU and CPU with Communication



"Remote Forces"

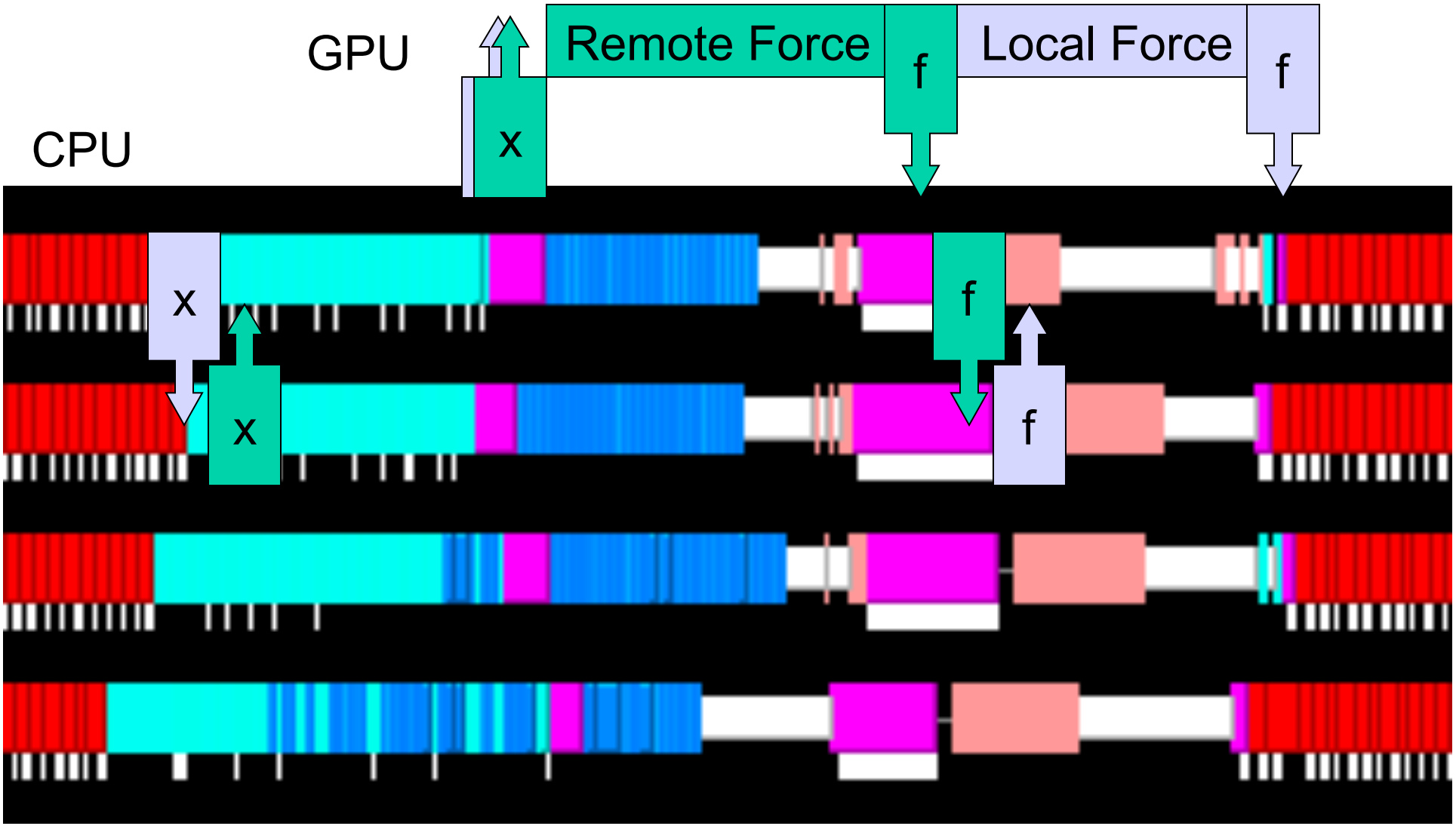
- Forces on atoms in a local patch are "local"
- Forces on atoms in a remote patch are "remote"
- Calculate remote forces first to overlap force communication with local force calculation
- Not enough local work to overlap it with position communication



Work done by one processor

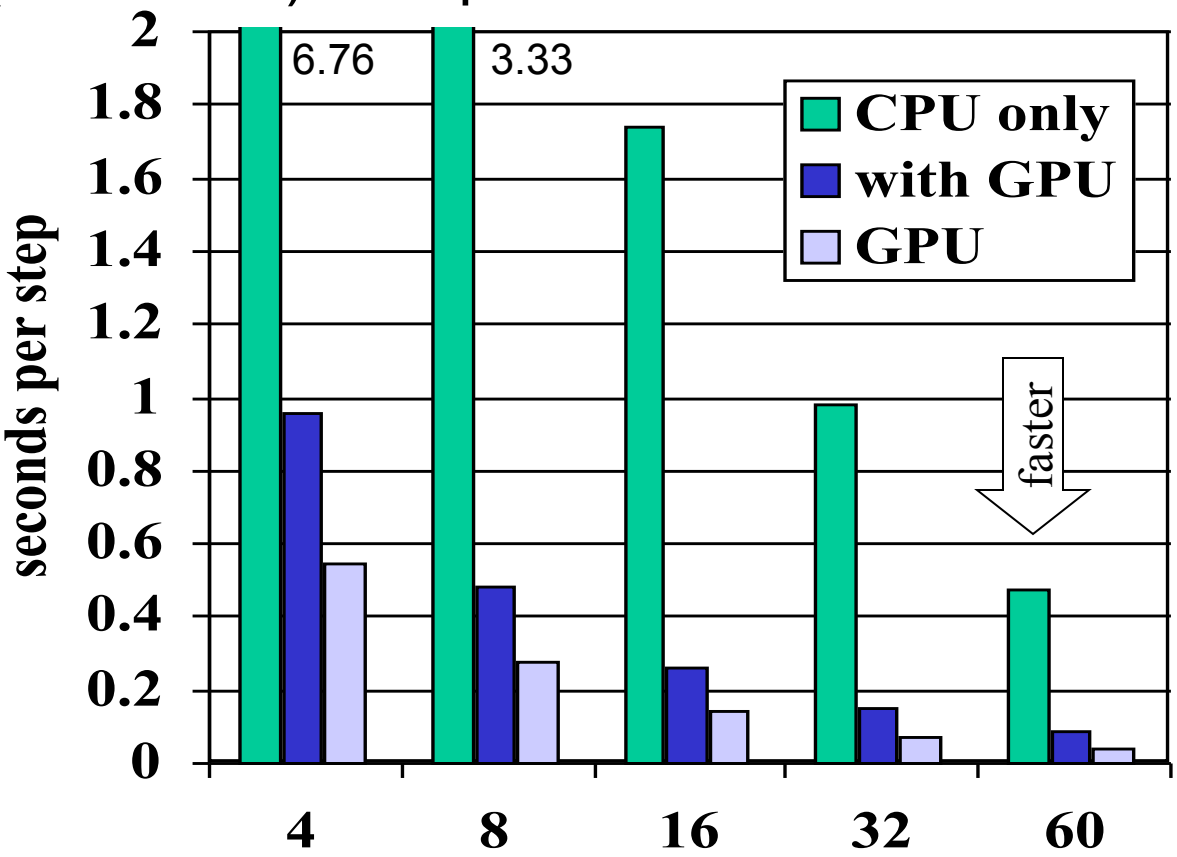
Actual Timelines from NAMD

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



NCSA "4+4" QP Cluster

STMV (1M atoms) s/step



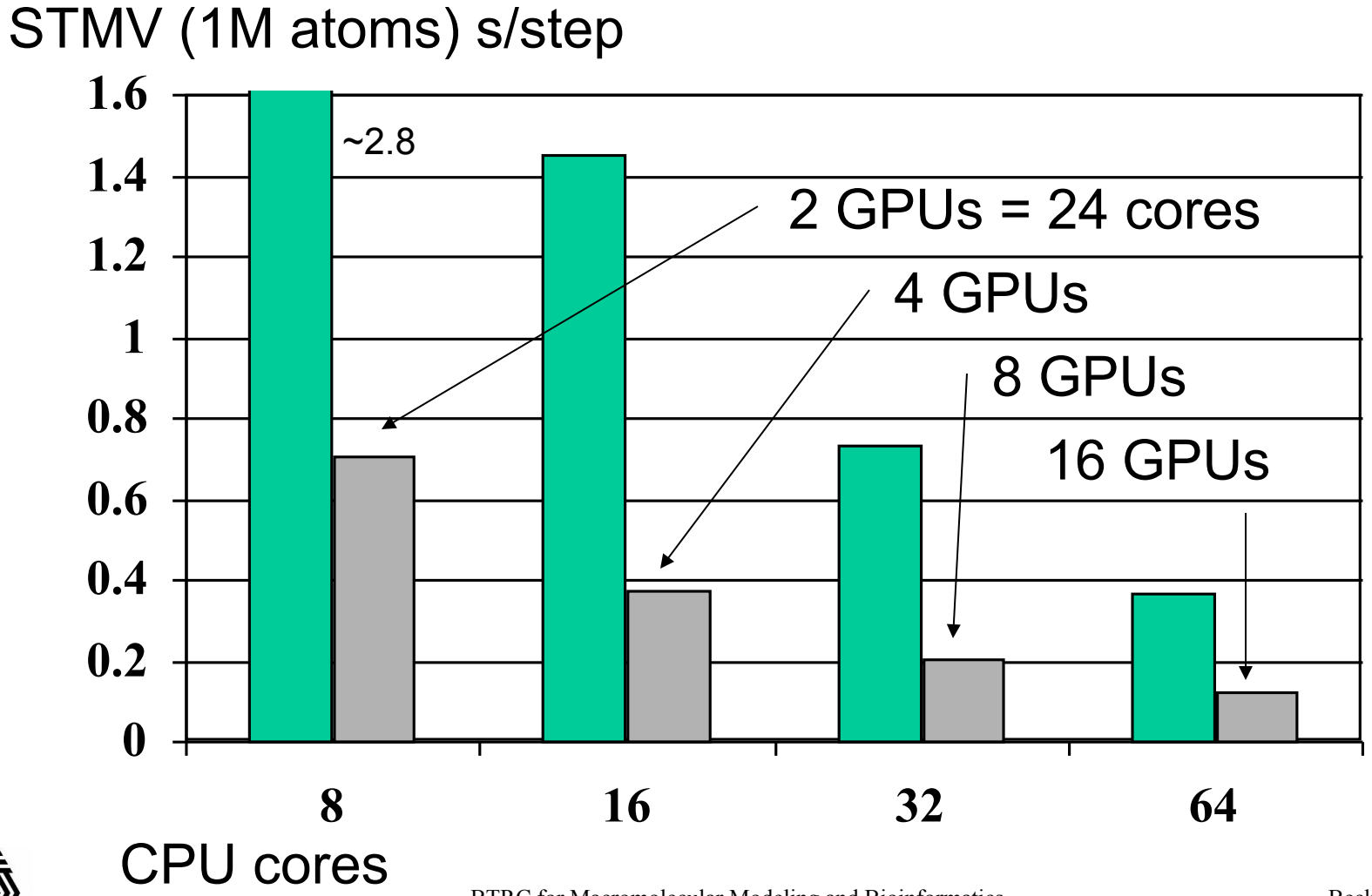
2.4 GHz Opteron + Quadro FX 5600

NCSA "8+2" Lincoln Cluster

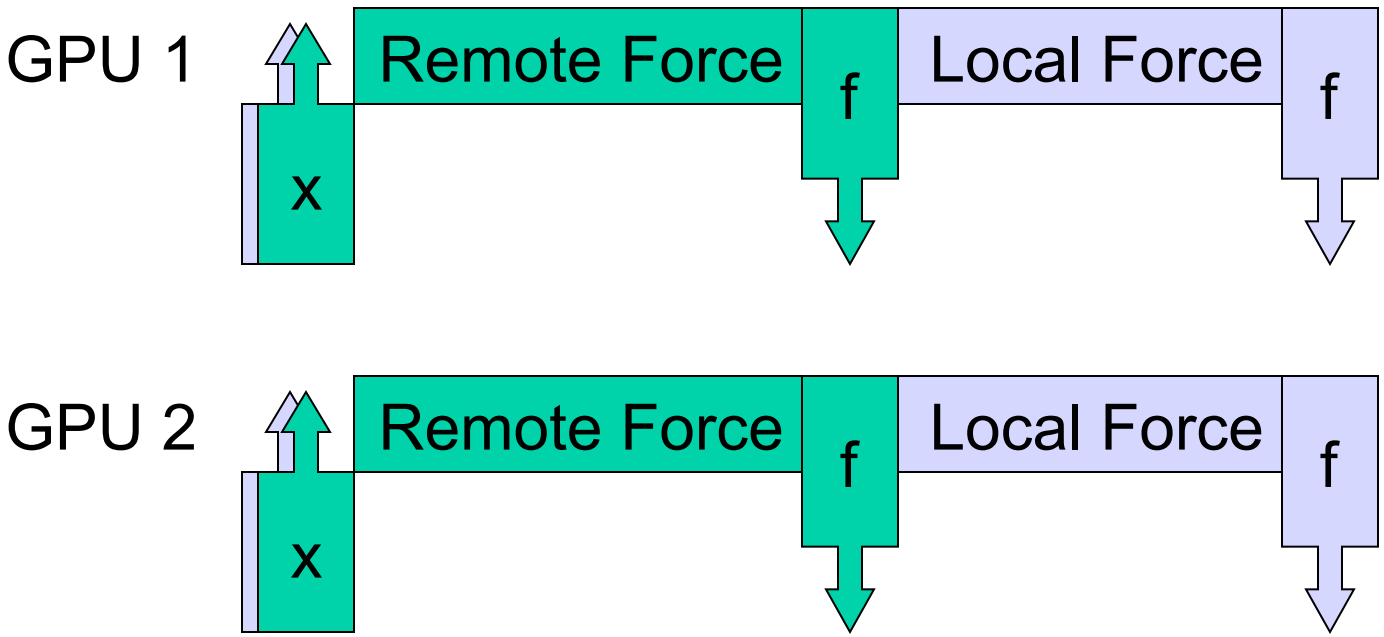
- CPU: 2 Intel E5410 Quad-Core 2.33 GHz
- GPU: 2 NVIDIA C1060
  - Actually S1070 shared by two nodes
- How to share a GPU among 4 CPU cores?
  - Send all GPU work to one process?
  - Coordinate via messages to avoid conflict?
  - Or just hope for the best?

NCSA Lincoln Cluster Performance

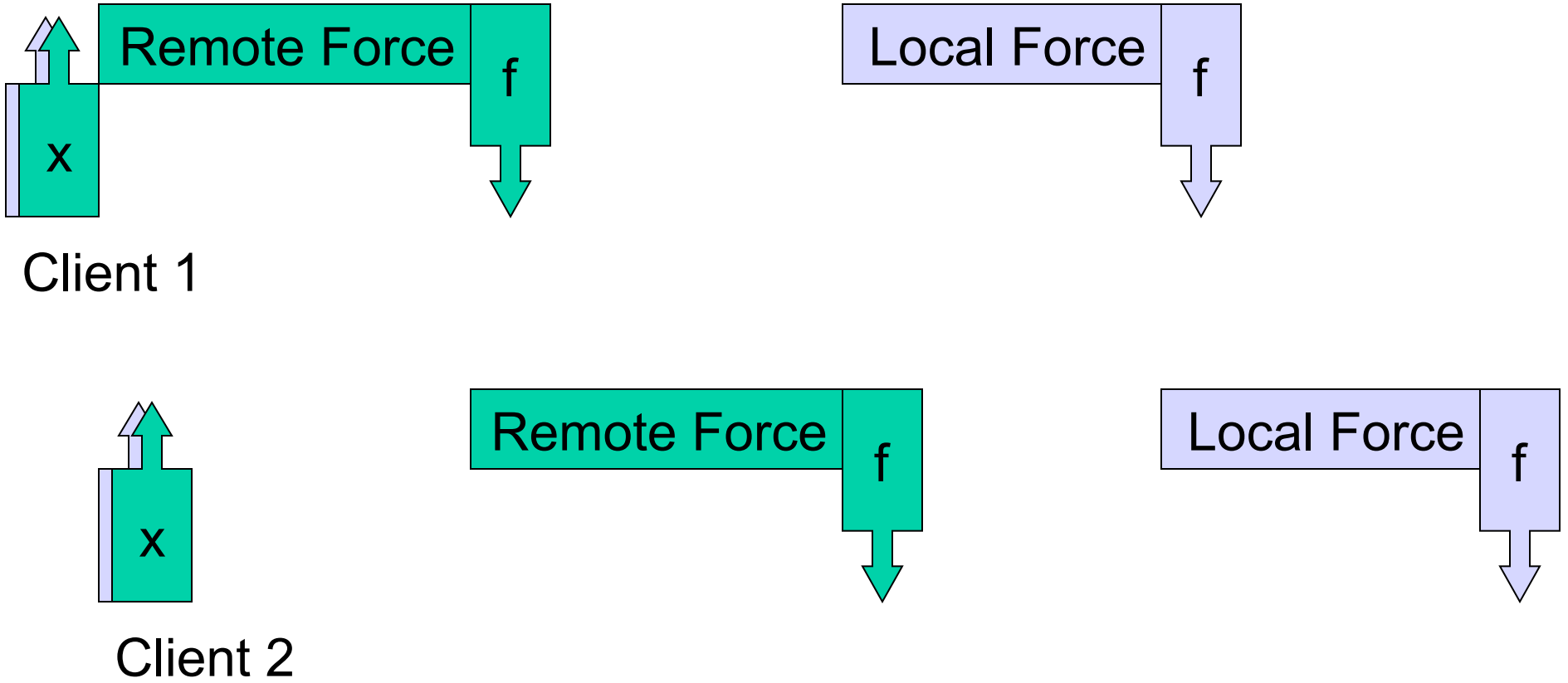
(8 Intel cores and 2 NVIDIA Telsa GPUs per node)



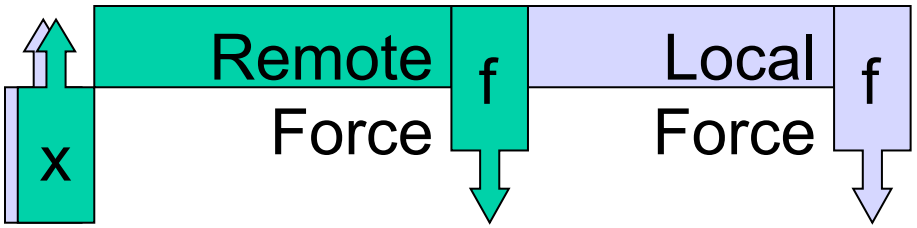
No GPU Sharing (Ideal World)



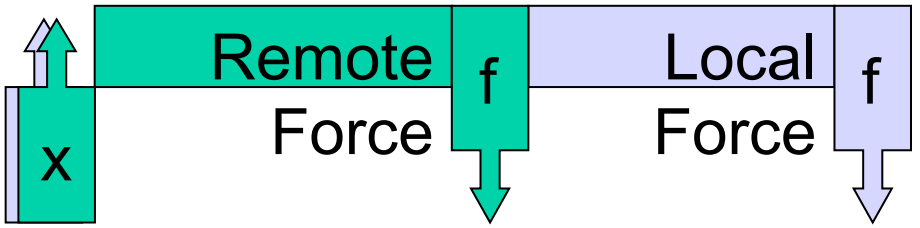
GPU Sharing (Desired)



GPU Sharing (Feared)

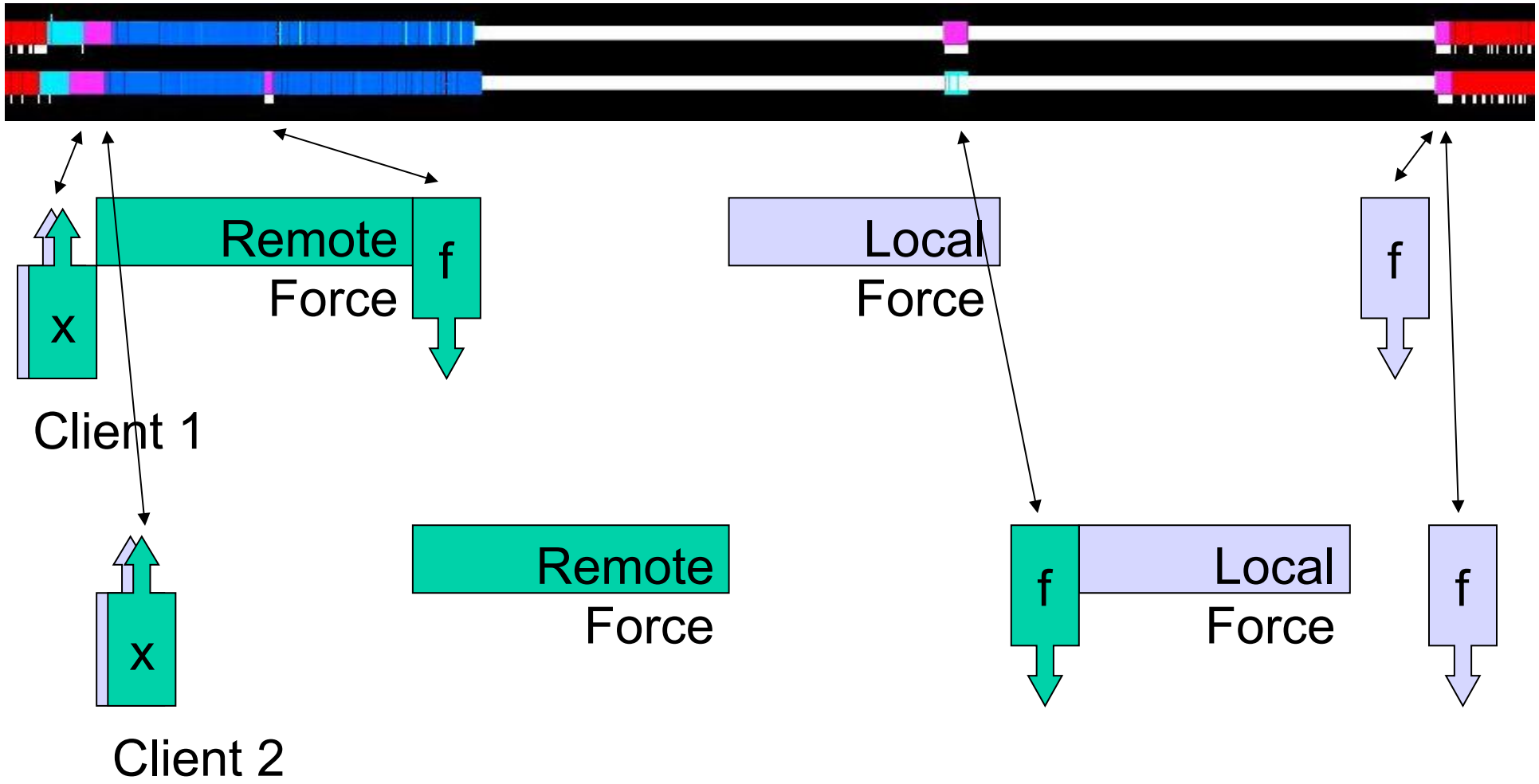


Client 1



Client 2

GPU Sharing (Observed)



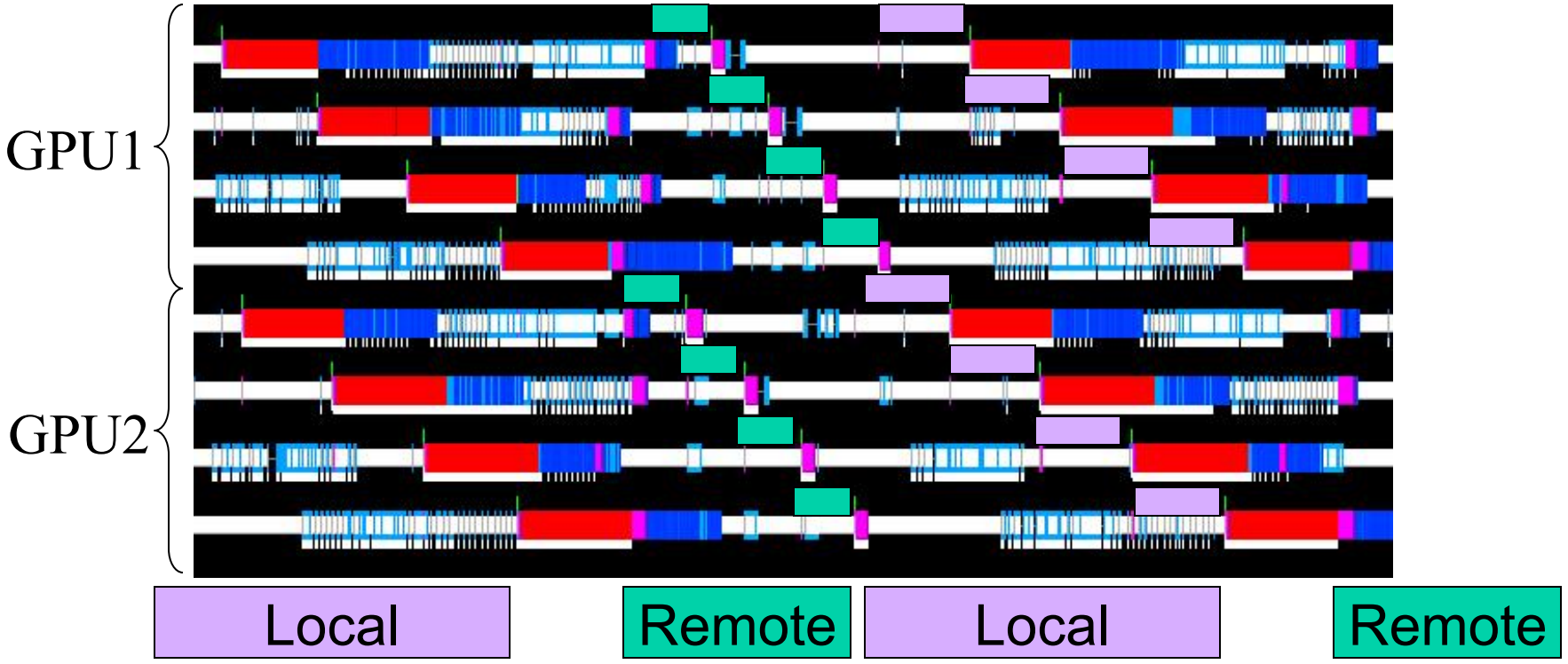
GPU Sharing (Explained)

- 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

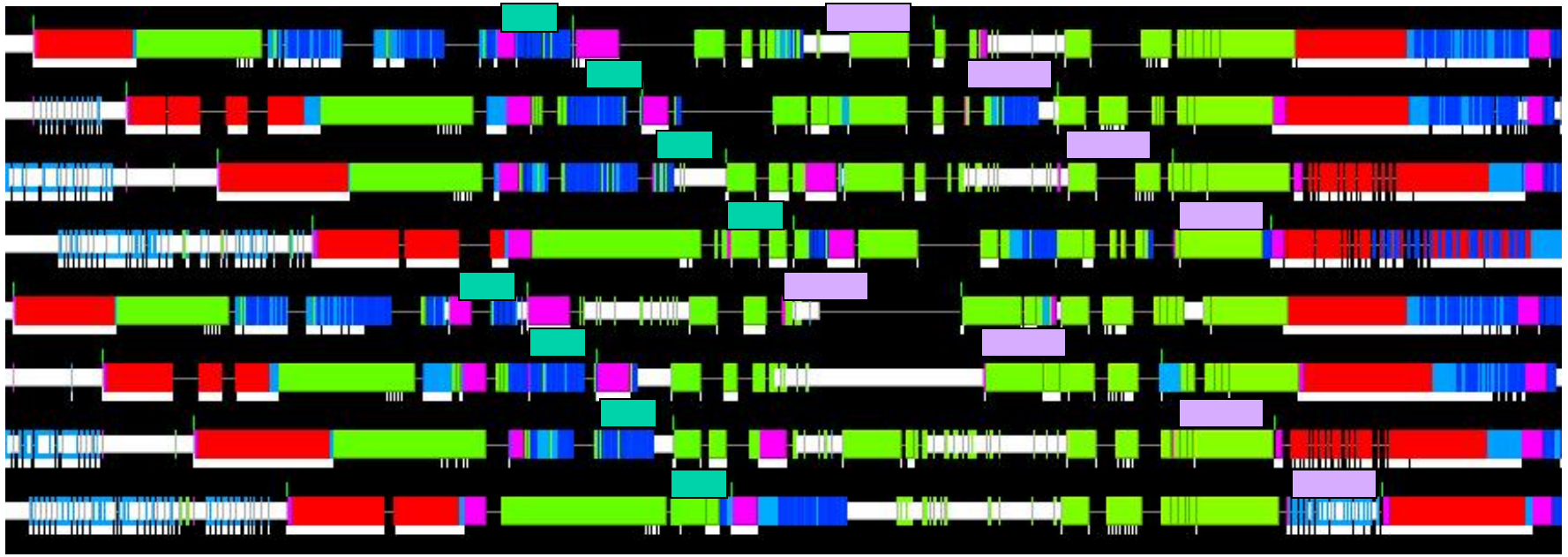
Inter-client Communication

- 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



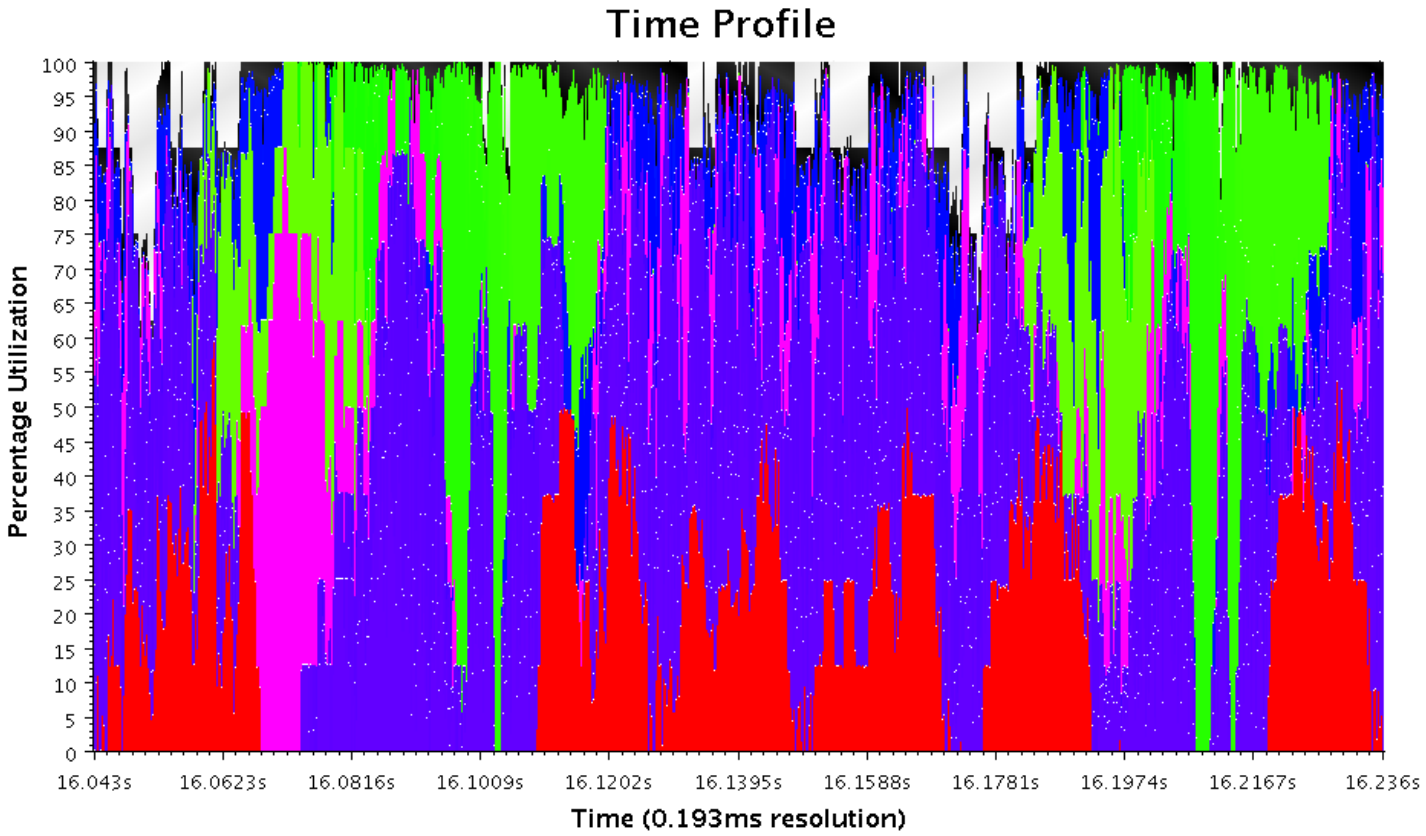
Weakness of Token-Passing

- 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

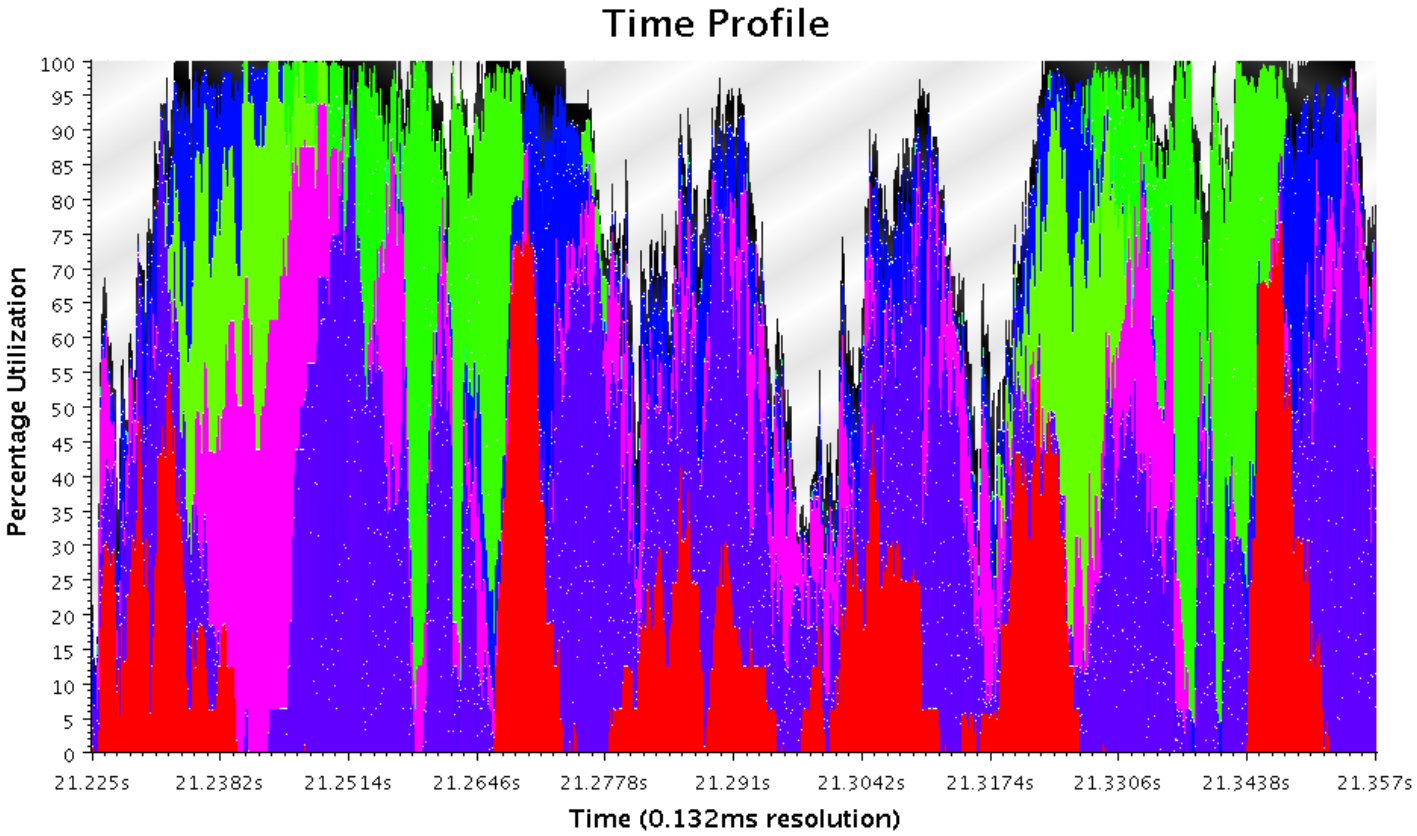
Current Compromise

- Use Fermi to 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

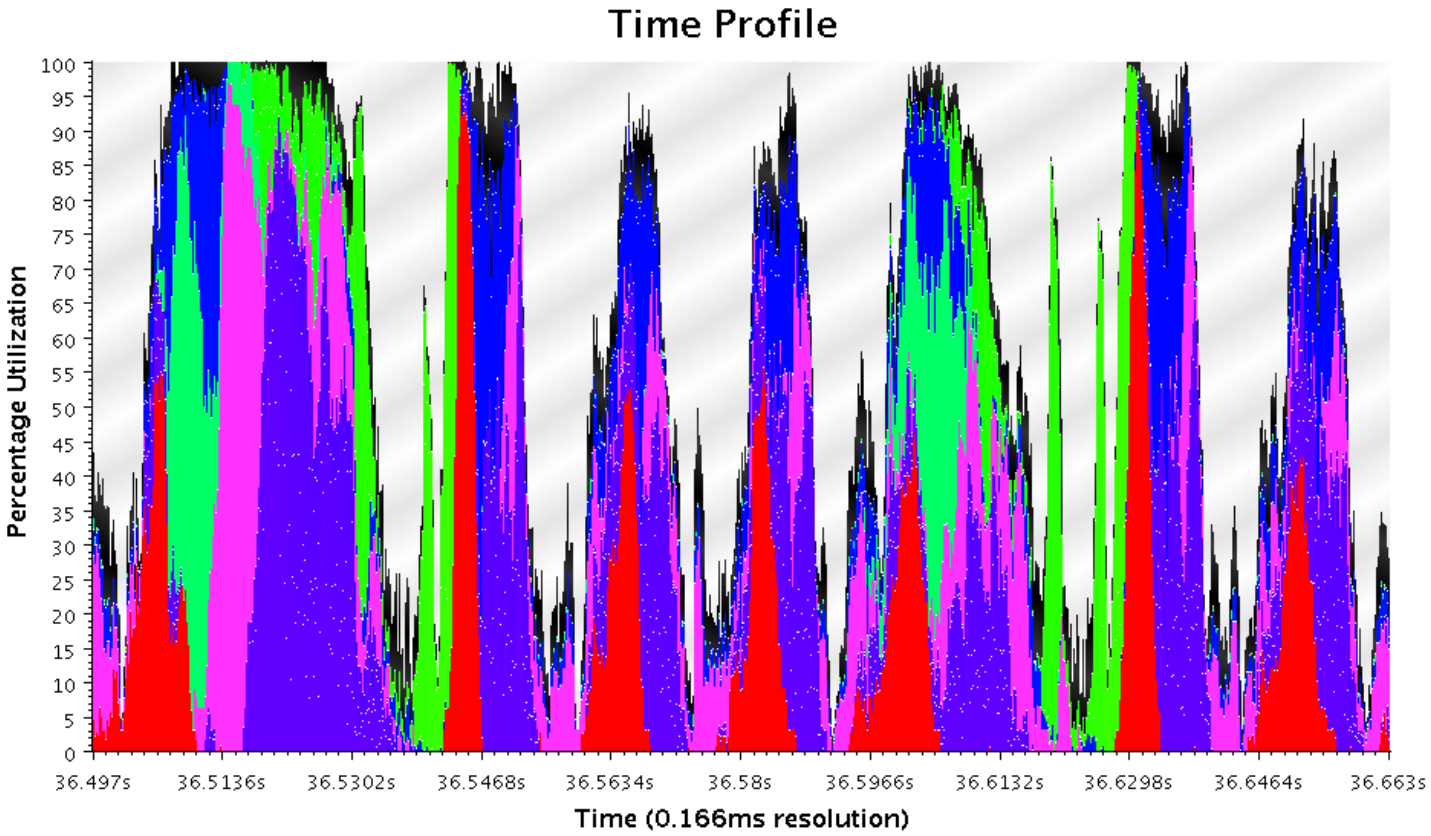
8 GPUs + 8 CPU Cores



8 GPUs + 16 CPU Cores



8 GPUs + 32 CPU Cores



Further 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 pair lists 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

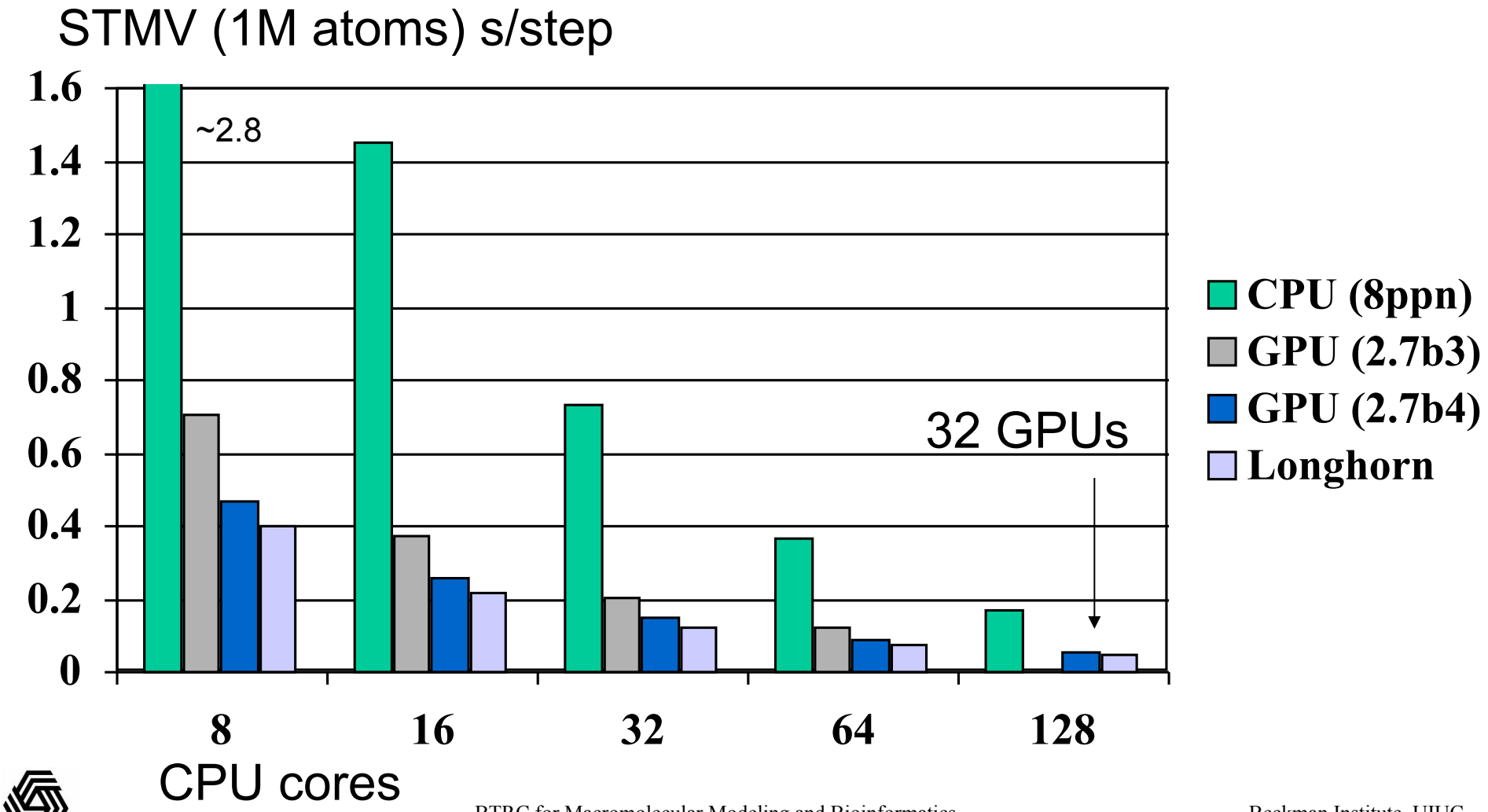
- Reduce warp divergence on cutoff tests
- Group nearby atoms in the same warp
- One option is space-filling curve
- Used recursive bisection instead
  - Split only on 32-atom boundaries
  - Find major axis, sort, split, repeat…

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