Improving NAMD Performance on Multi-GPU Platforms

David J. Hardy

Theoretical and Computational Biophysics Group
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
http://www.ks.uiuc.edu/~dhardy/

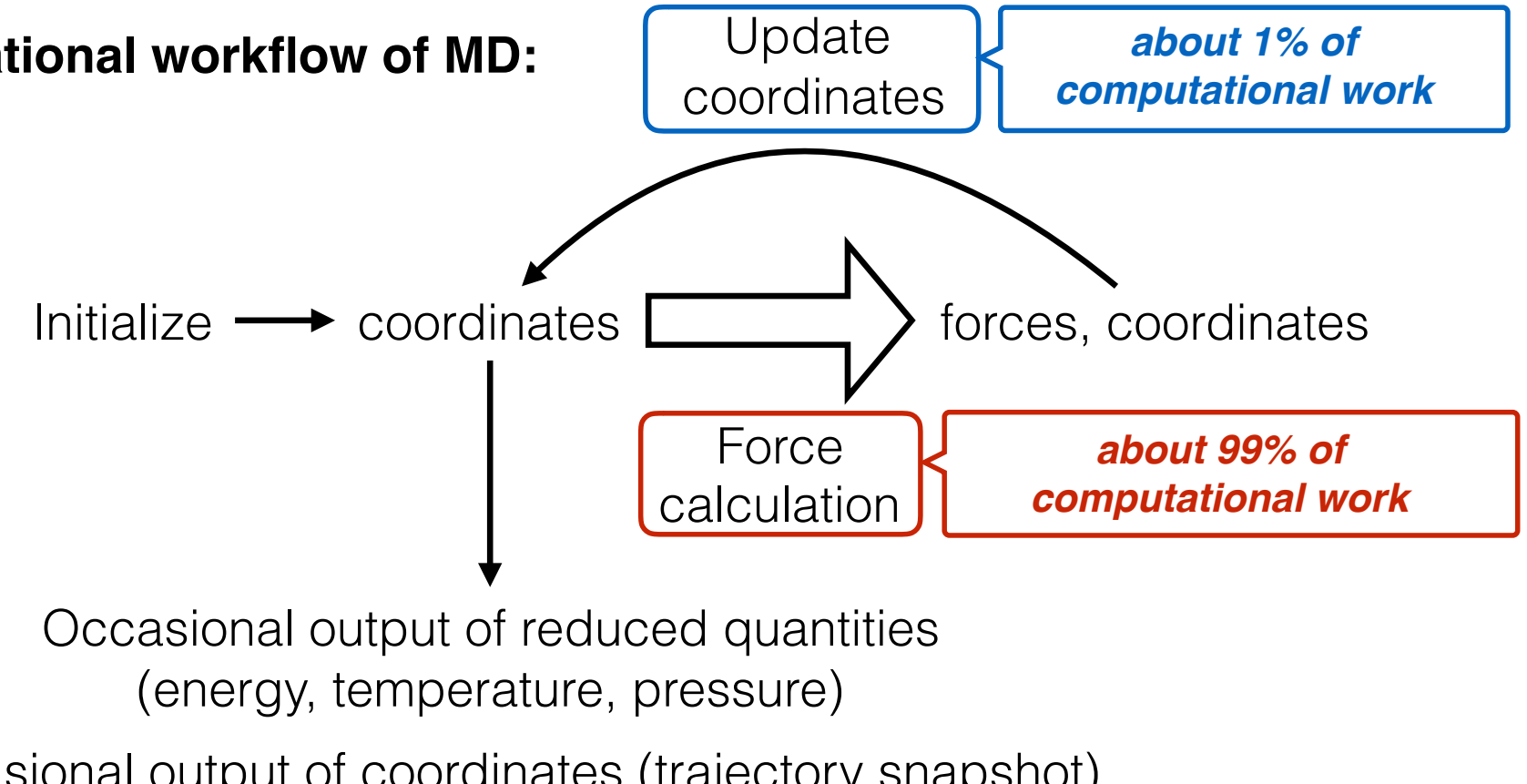
16th Annual Workshop on Charm++ and its Applications
April 11, 2018

Outline

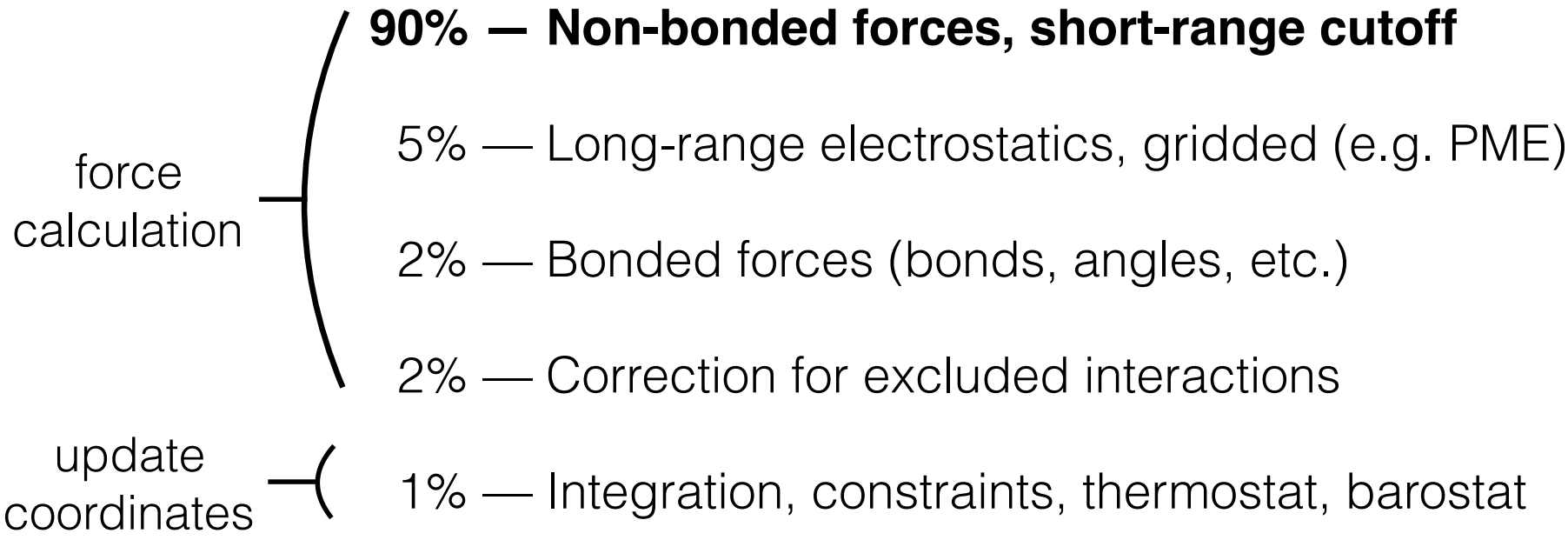
- NAMD's use of GPUs as coprocessors, a historical perspective
  - NAMD has been developed for more than 20 years
  - **First full-featured molecular dynamics code to adopt CUDA**
    - Stone, et al. *J Comput Chem*, 28:2618-2640, 2007
- The challenge posed by today's multi-GPU architectures
- How can Charm++ help address these challenges?

Parallelism in Molecular Dynamics Limited to Each Timestep

Computational workflow of MD:



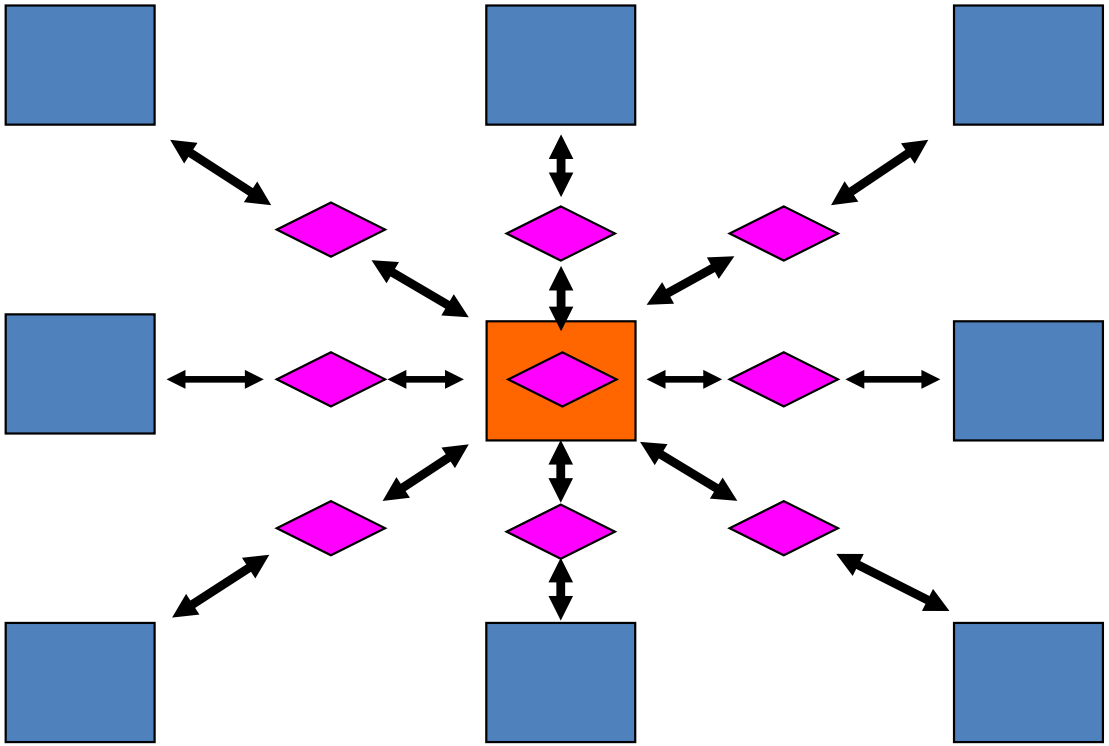
Work Dominated by Nonbonded Forces



Apply GPU acceleration first to the most expensive part

NAMD Hybrid Decomposition with Charm++

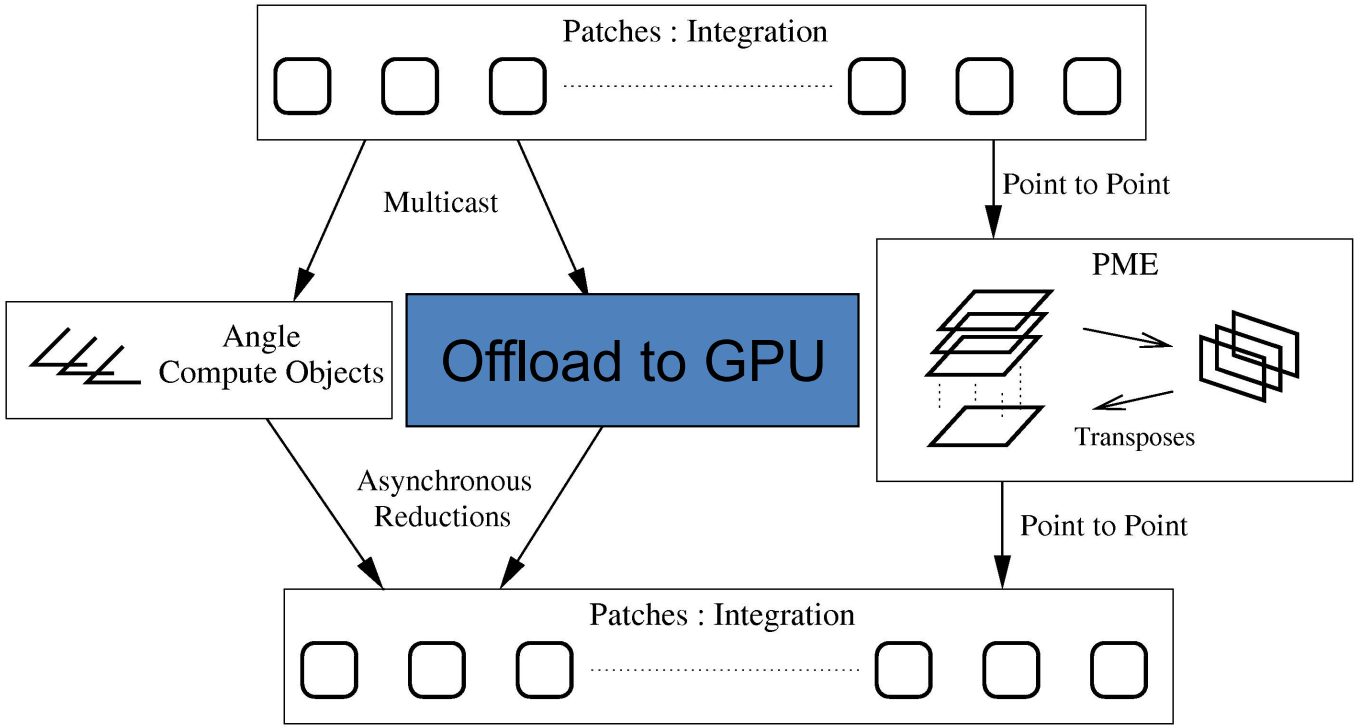
Kale et al., *J. Comp. Phys.* 151:283-312, 1999



- Spatially decompose data and communication
- Separate but related work decomposition
- **“Compute objects” create much greater amount of parallelization**, facilitating iterative, measurement-based load balancing system, all from **use of Charm++**

Overlap Calculations, Offload Nonbonded Forces

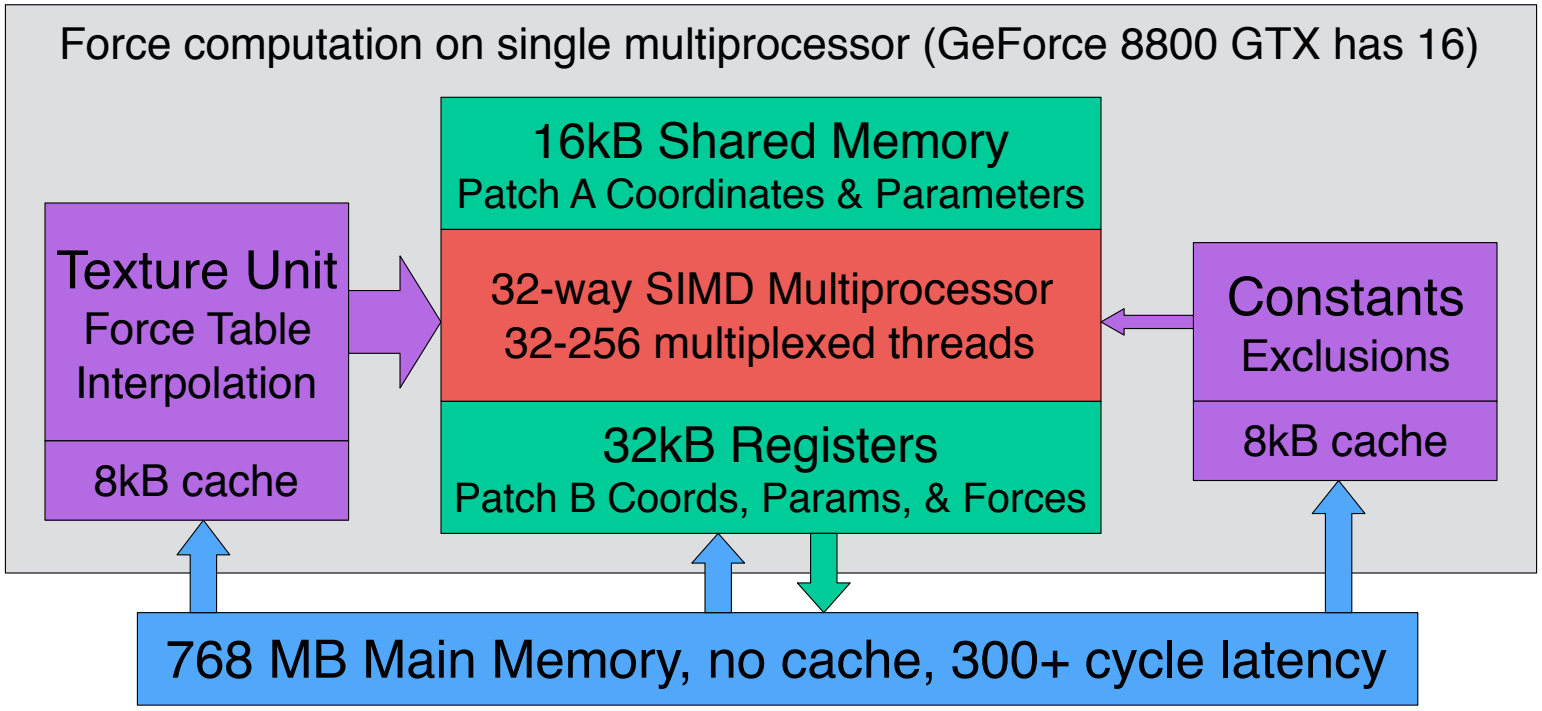
Phillips et al., *SC2002*



Objects are assigned to processors and queued as data arrives

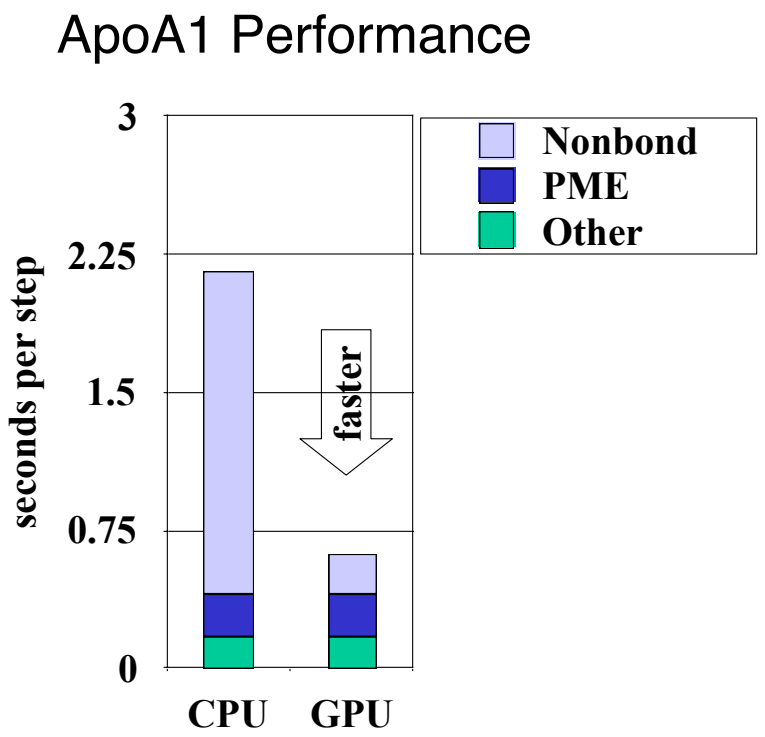
Early Nonbonded Forces Kernel Used All Memory Systems

- 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.



NAMD Performance Improved Using Early GPUs

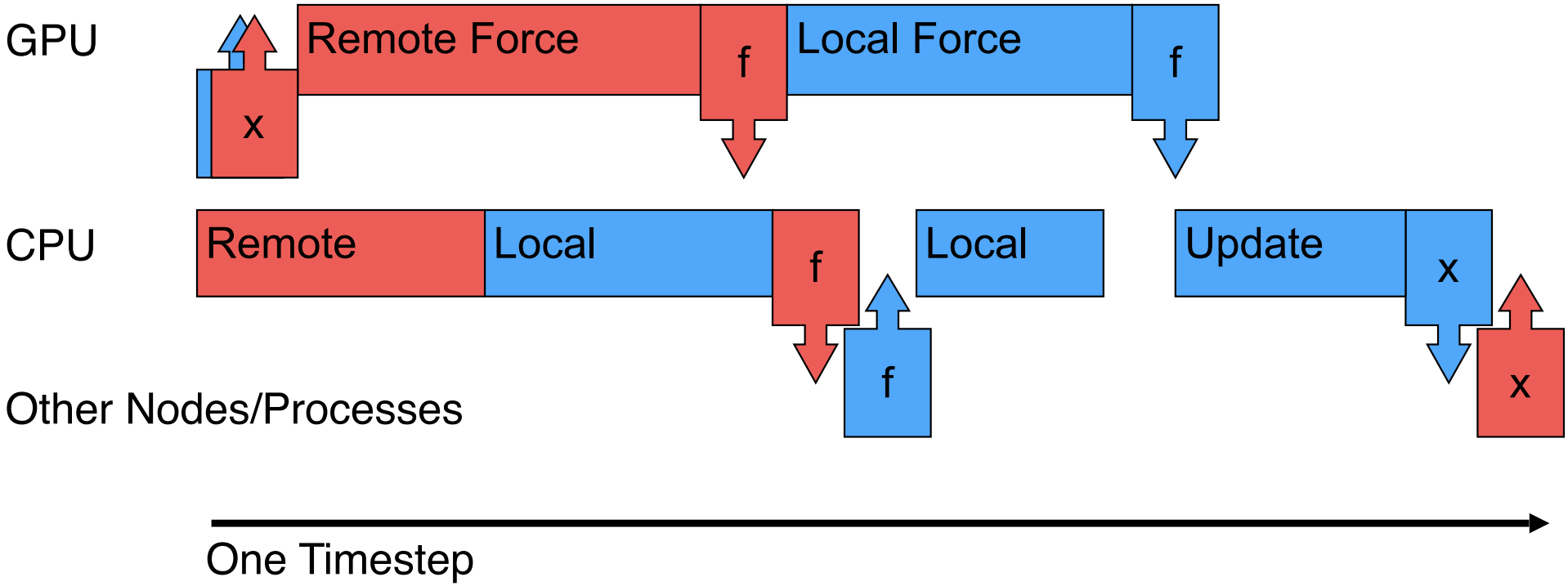
- Full NAMD, not test harness
- Useful performance boost
  - 8x speedup for nonbonded
  - 5x speedup overall w/o PME
  - 3.5x speedup overall w/ PME
  - GPU = quad-core CPU
- Plans for better performance
  - Overlap GPU and CPU work.
  - Tune or port remaining work.
    - PME, bonded, integration, etc.



2.67 GHz Core 2 Quad Extreme + GeForce 8800 GTX

Reduce Communication Latency by Separating Work Units

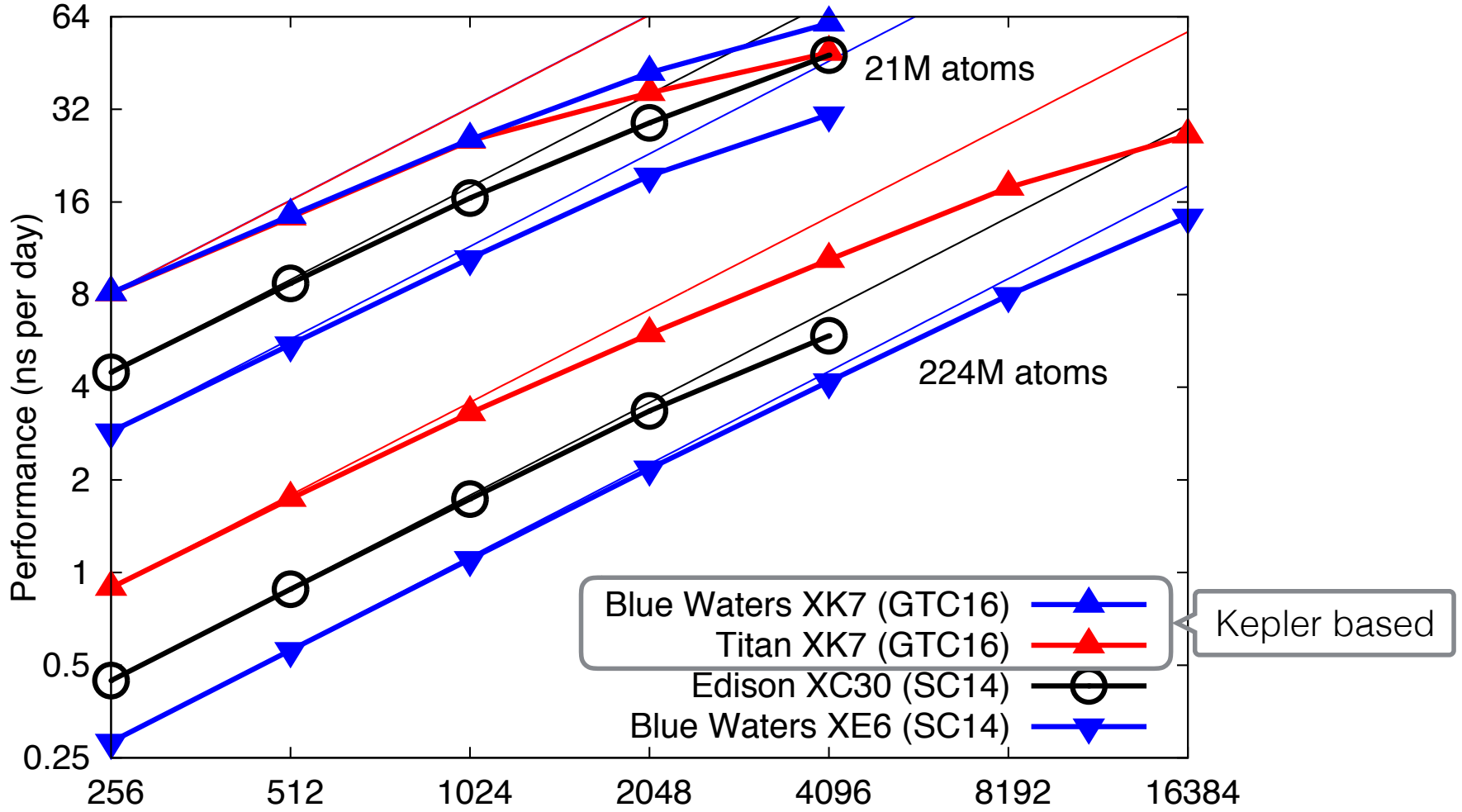
Phillips et al., SC2008



Early GPU Fits Into Parallel NAMD as Coprocessor

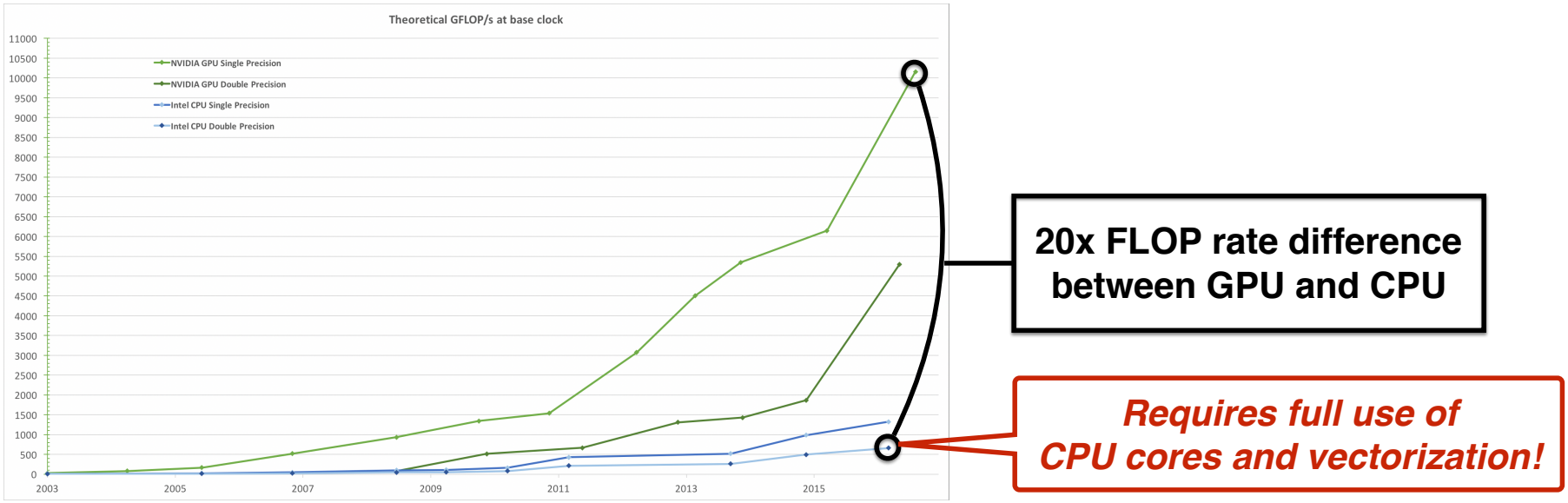
- Offload most expensive calculation: non-bonded forces
- Fits into existing parallelization
- **Extends existing code without modifying core data structures**
- Requires work aggregation and kernel scheduling considerations to optimize remote communication
- **GPU is treated as a coprocessor**

NAMD Scales Well on Kepler Based Computers



(2fs timestep)

Large Rate Difference Between Pascal and CPU



- Balance between GPU and CPU capability keeps shifting towards GPU
- NVIDIA plots show only through Pascal — Volta widens the performance gap!
- Difference made worse by multiple GPUs per CPU (e.g. AWS, DGX, Summit)
- Past efforts to balance work between GPU and CPU are **now CPU bound**

Reduce Latency, Offload All Force Computation

- Overlapped GPU communication and computation (2012)
- Offload atom-based work for PME (2013)
  - Use higher order interpolation with coarser grid
  - Reduce parallel FFT communication

Emphasis on improving communication latency

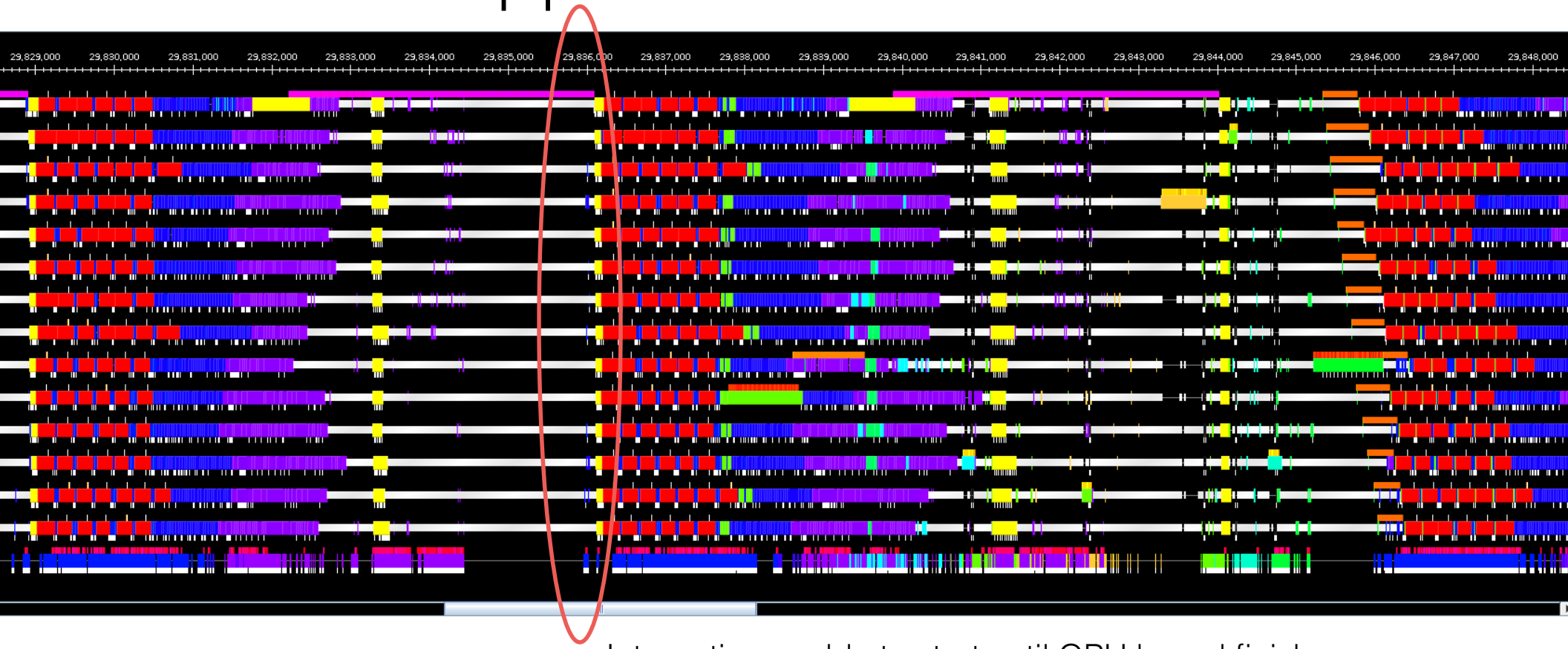
- Faster nonbonded force kernels (2016)
- Offload entire PME using cuFFT (for single node use) (2016)
- Offload remaining force terms (2017)
  - Includes: bonds, angles, dihedrals, impropers, crossterms, exclusions

Emphasis on using GPUs more effectively

Overlapped GPU Communication and Computation

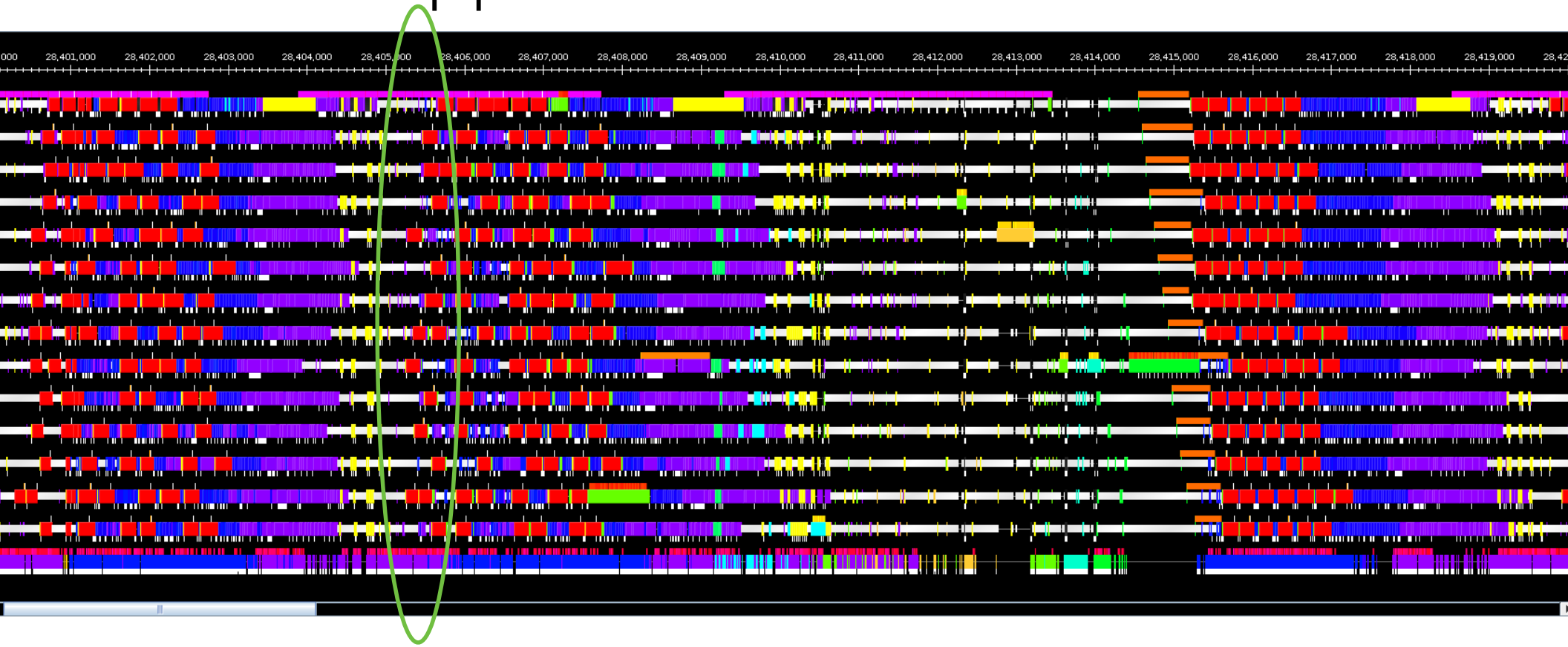
- **Allows incremental results from a single grid to be processed on CPU before grid finishes on GPU**
- Allows merging and prioritizing of remote and local work
- GPU side:
  - Write results to host-mapped memory (also without streaming)
  - __threadfence_system() and __syncthreads()
  - Atomic increment for next output queue location
  - Write result index to output queue
- CPU side:
  - Poll end of output queue (int array) in host memory

Non-overlapped Kernel Communication



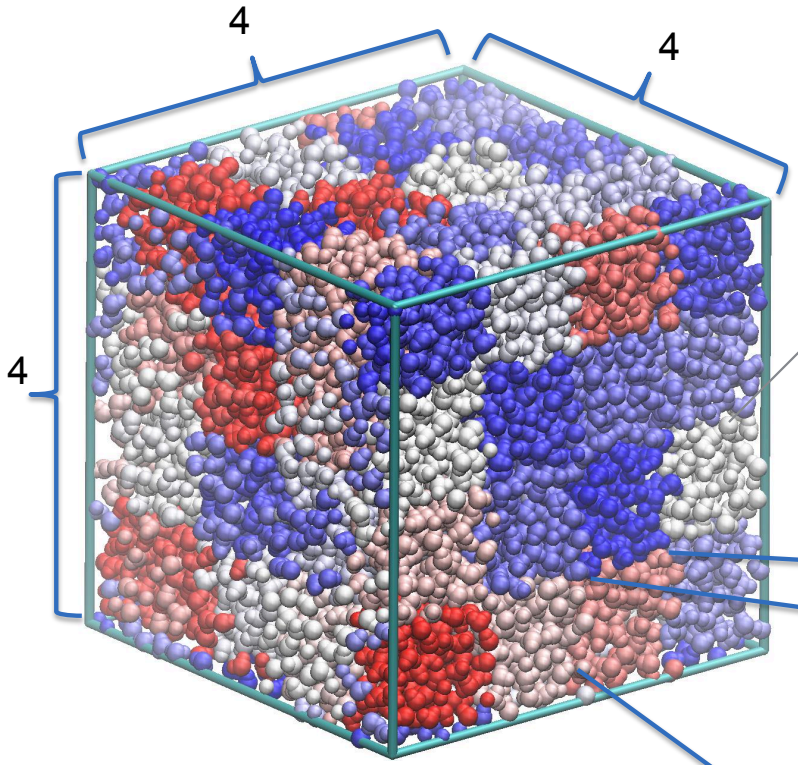
Integration unable to start until GPU kernel finishes

Overlapped Kernel Communication

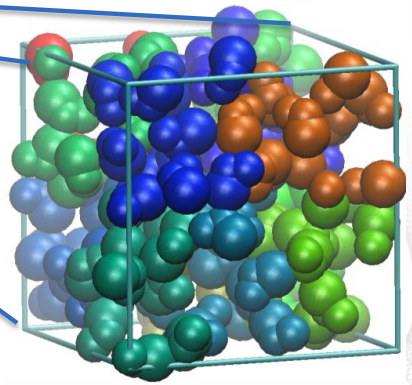


GPU kernel communicates results while running; patches begin integration as soon as data arrives

FASTER Non-bonded force computation in NAMD

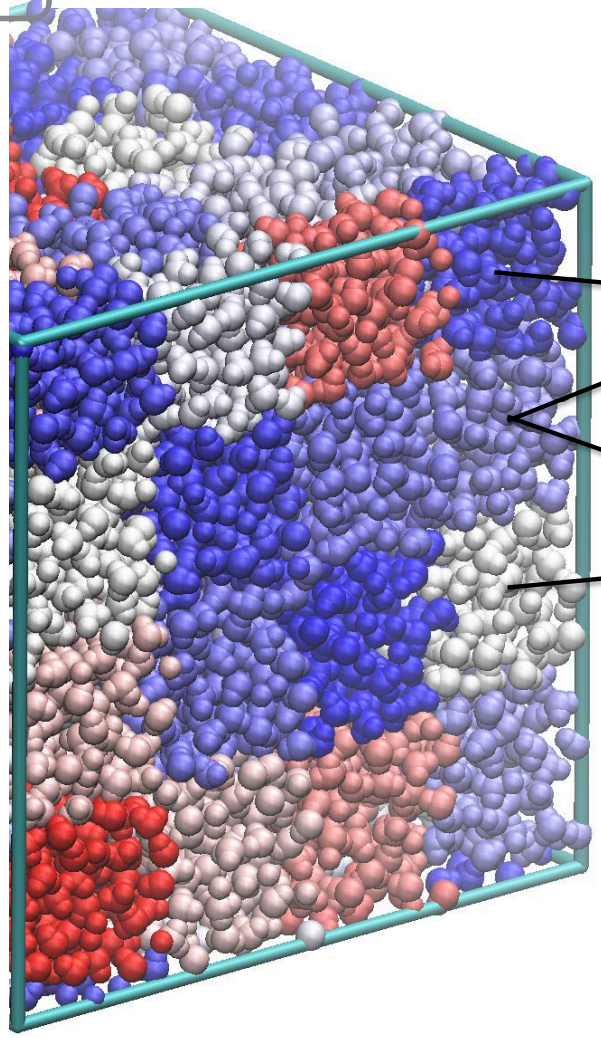


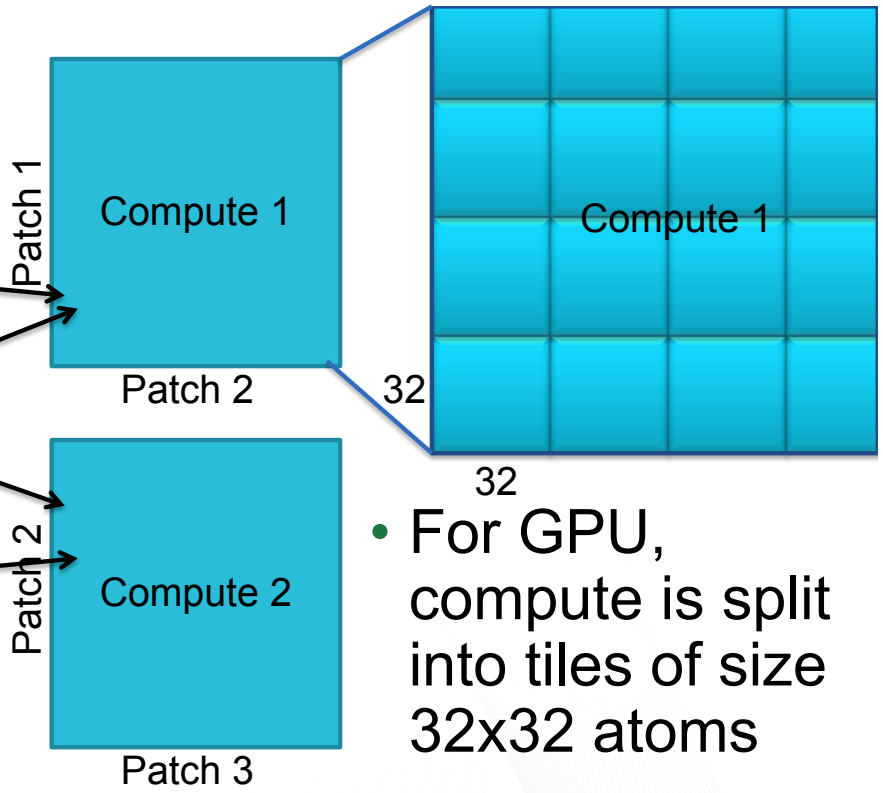
- Two levels of spatial sorting
  - Simulation box is divided into patches
  - Within the patch, atoms are sorted spatially into groups of 32 using orthogonal recursive bisection method



FASTER

Non-bonded force compute

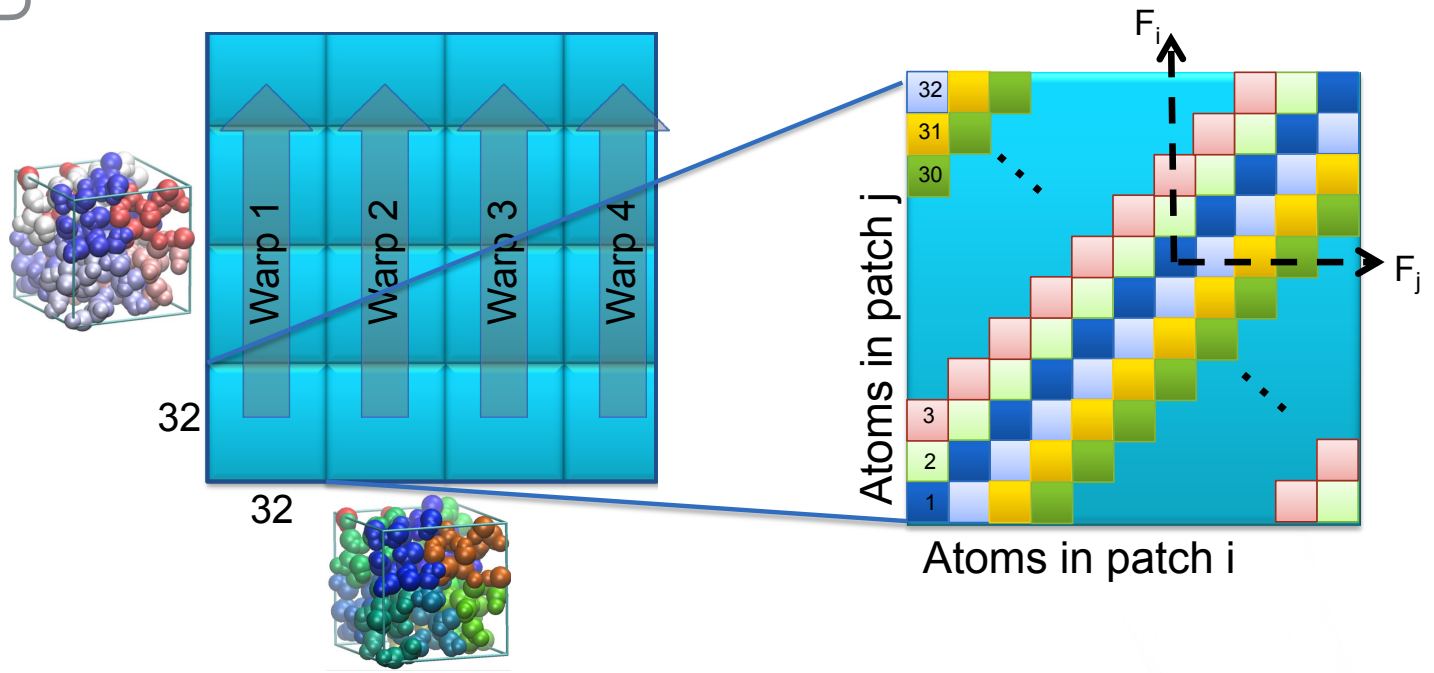




- For GPU, compute is split into tiles of size 32x32 atoms

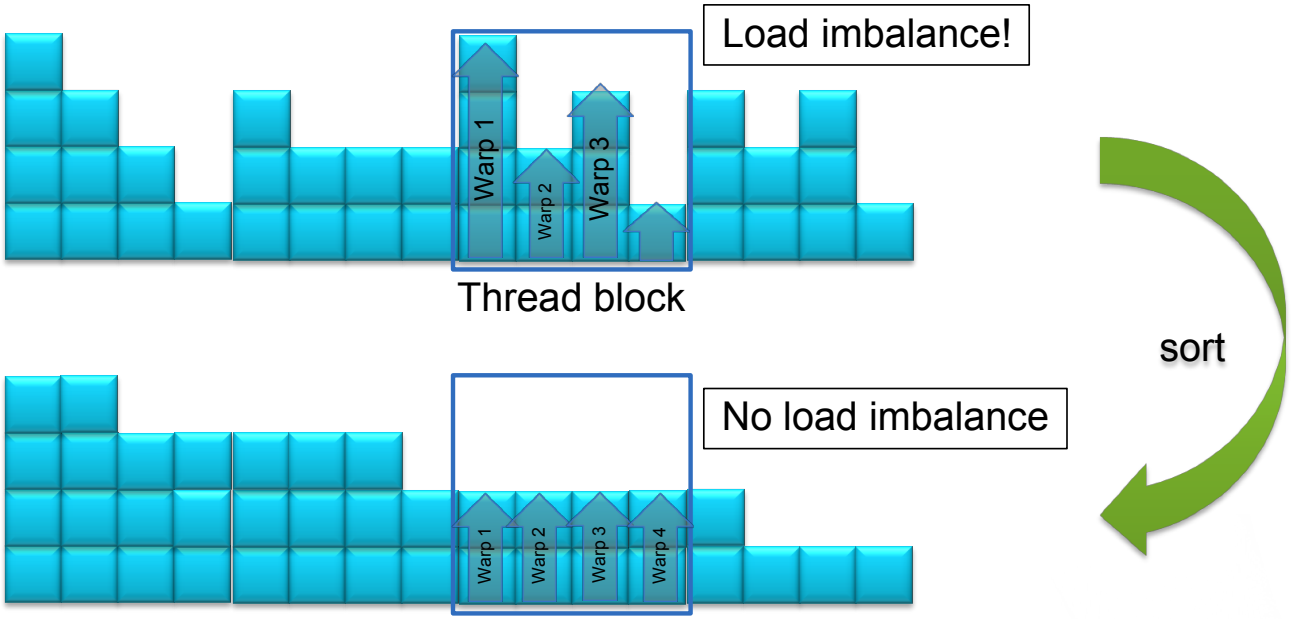
- Compute = all pairwise interactions between two patches

FASTER Non-bonded force computation



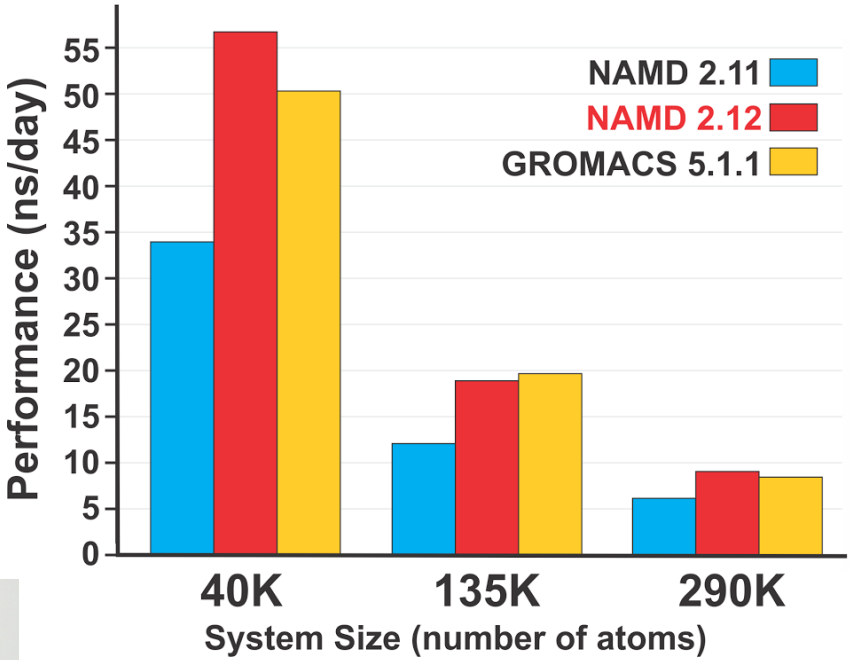
- One warp per tiles
- Loop through 32x32 tile diagonally
  - Avoids race condition when storing forces F_i and F_j
- Bitmask used for exclusion lookup

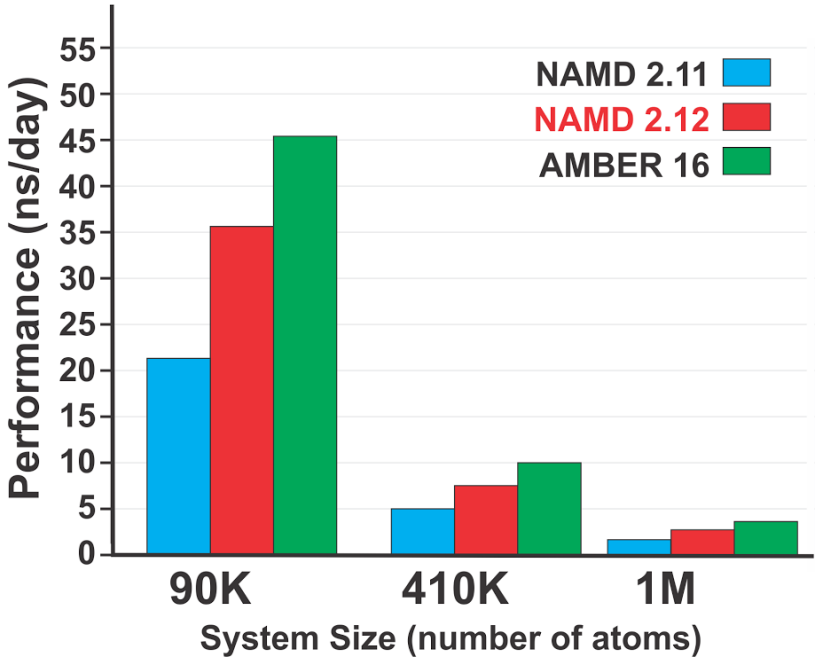
Neighbor list sorting



- Tile lists executed on the same thread block should have approximately the same work load
- Simple solution is to sort according to tile list length
- Also minimizes tail effects at the end of kernel execution

Single-Node GPU Performance Competitive on Maxwell



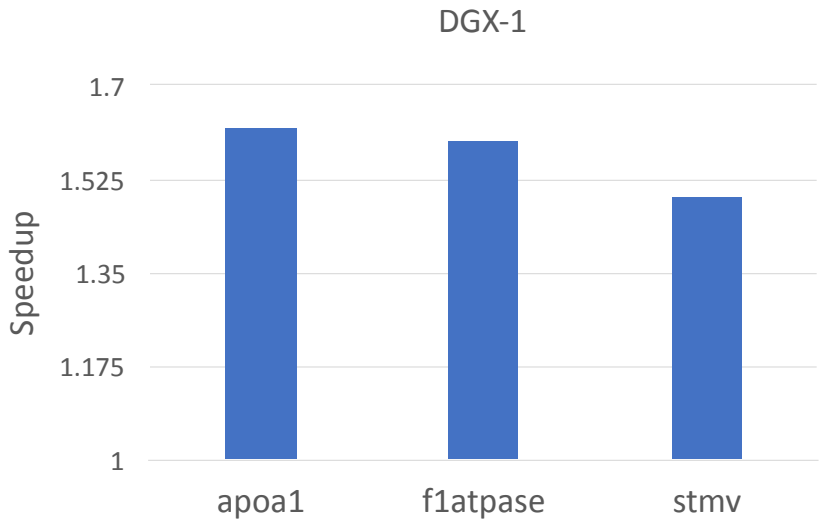


New kernels by **Antti-Pekka Hynninen**, **NVIDIA**

Stone, Hynninen, et al., *International Workshop on OpenPOWER for HPC (IWOPH'16)*, 2016

More Improvement from Offloading Bonded Forces

- GPU offloading for bonds, angles, dihedrals, impropers, exclusions, and crossterms
- Computation in single precision
- Forces are accumulated in 24.40 fixed point
- Virials are accumulated in 34.30 fixed point
- Code path exists for double precision accumulation on Pascal and newer GPUs
- **Reduces CPU workload and hence improves performance on GPU-heavy systems**



New kernels by **Antti-Pekka Hynninen, NVIDIA**

Supercomputers Increasing GPU to CPU Ratio

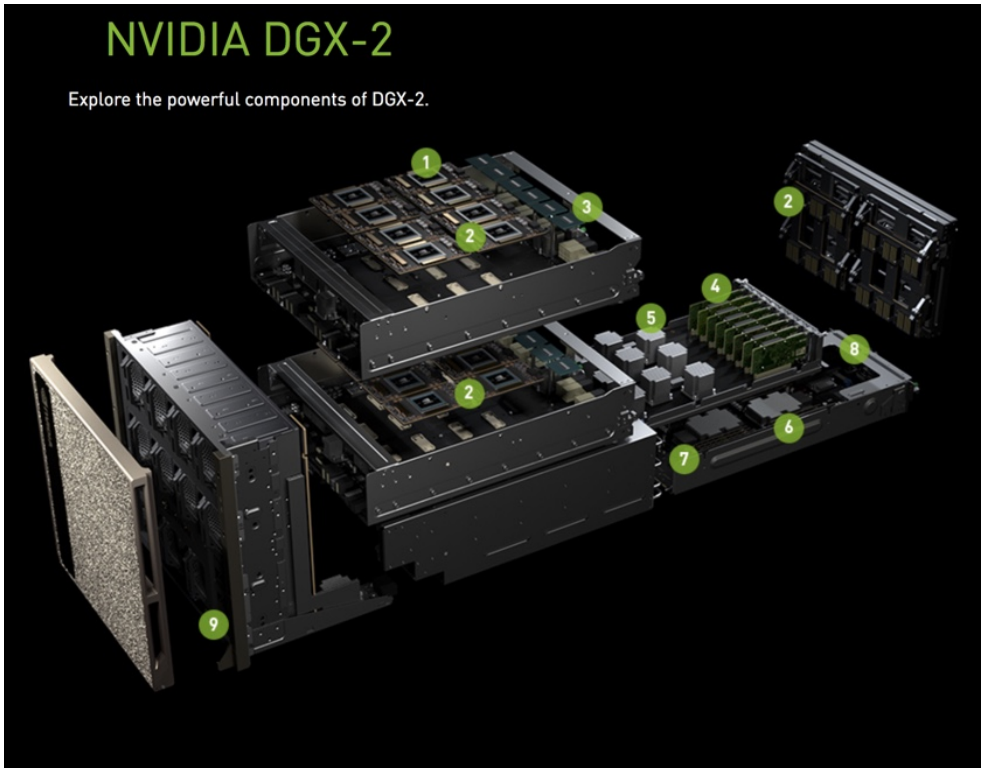
Blue Waters, Titan with Cray XK7 nodes
1 K20 / 16-core AMD Opteron

Summit nodes
6 Volta / 42 cores IBM Power 9

⇨ Only 7 cores supporting each Volta!

Revolutionary GPU-based Hardware

- 16 Volta GPUs
- 16 x 32 GB HBM2
- Fast switch makes memory uniformly accessible
- 2 Intel Xeon Platinum CPUs (2 x 28 cores)
- 1.5 TB main memory

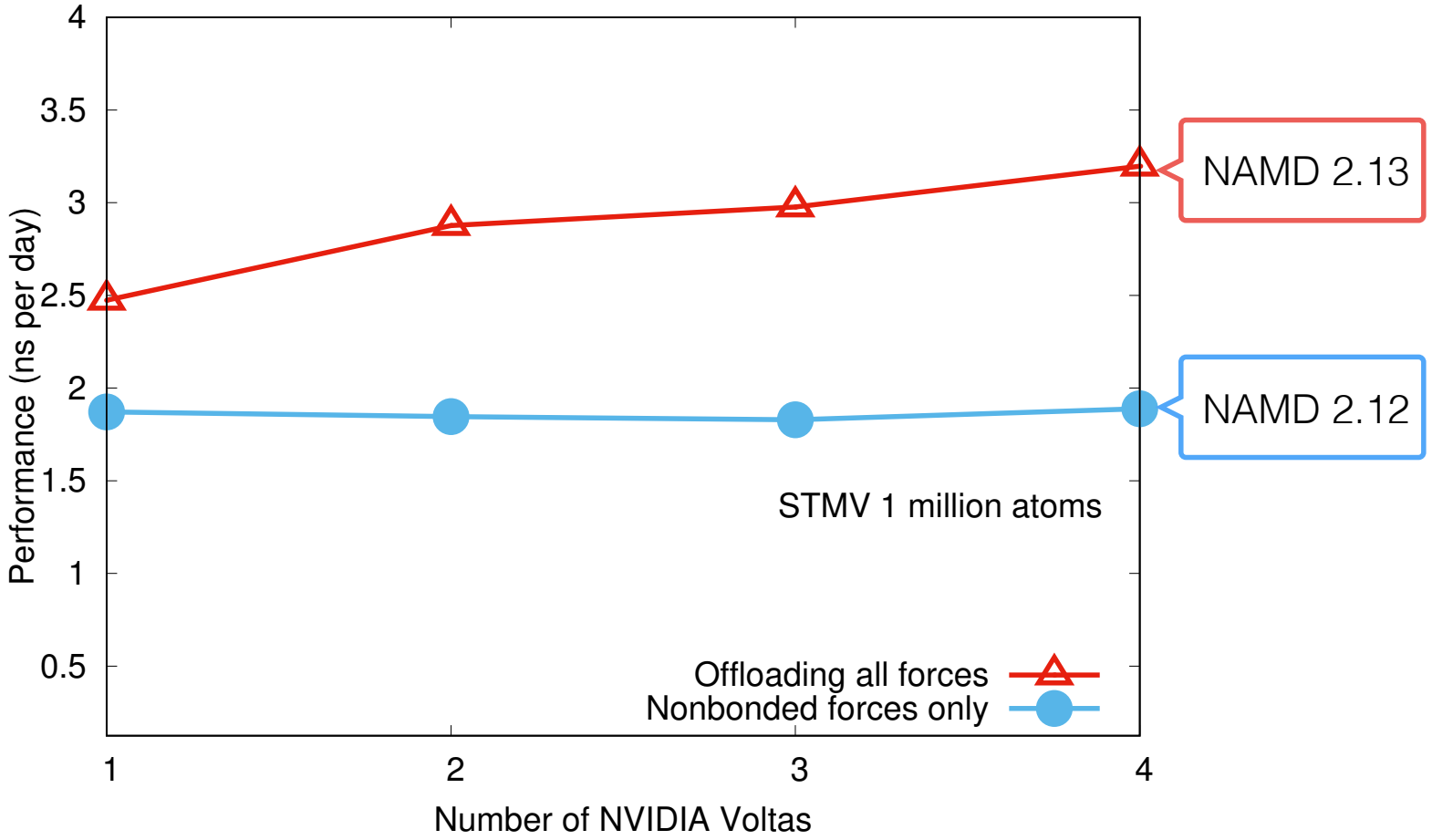


Equivalent compute power to about 160 nodes of Blue Waters

DGX-2: **3.5 CPU cores / GPU** vs. Blue Waters: **16 CPU cores / GPU**

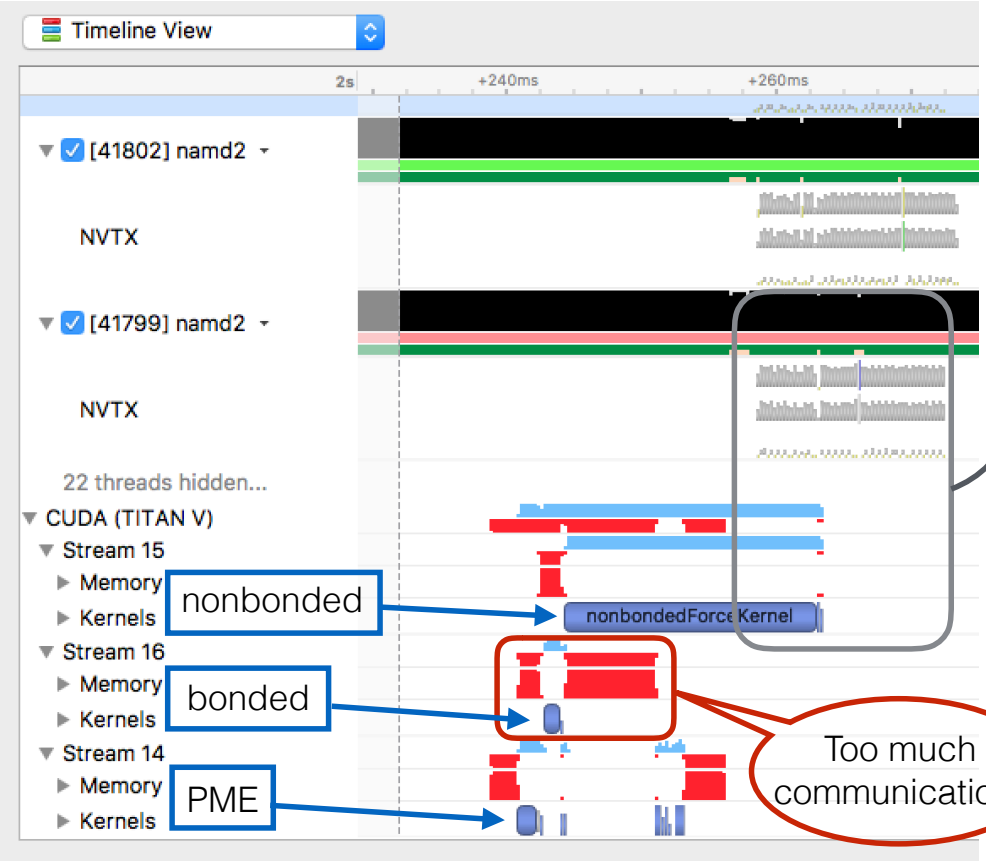
Limited Scaling Even After Offloading All Forces

Results on NVIDIA DGX-1 (Intel Haswell using 28-cores with Volta V100 GPUs)



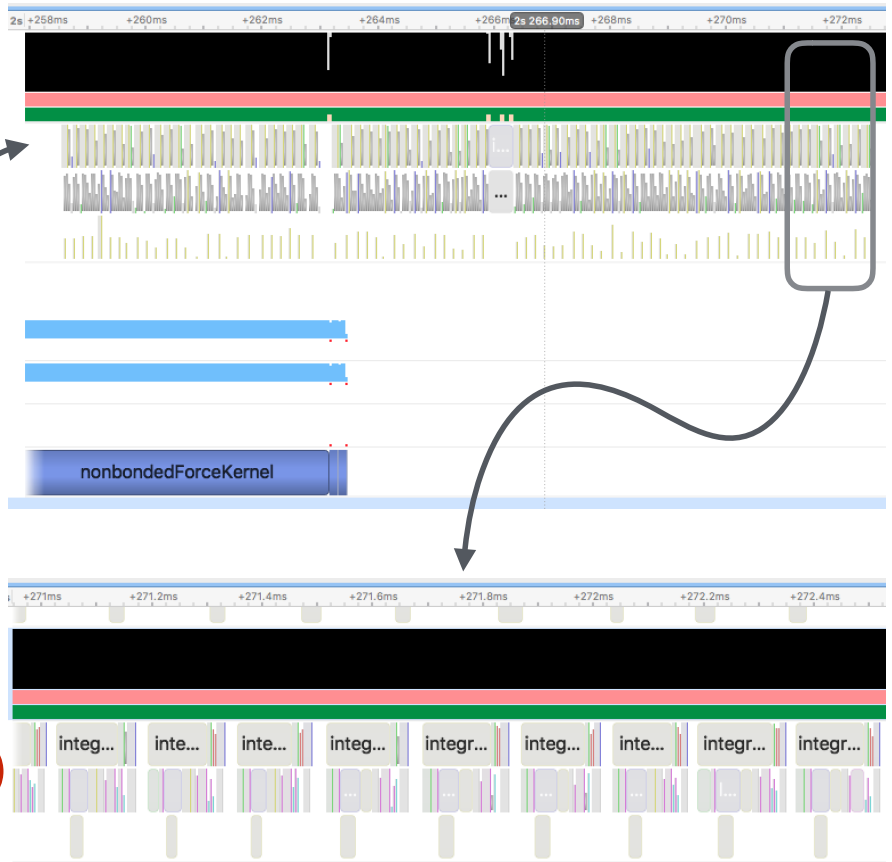
CPU Integrator Calculation (1%) Causing Bottleneck

Nsight Systems profiling of NAMD running STMV (1M atoms) on 1 Volta & 28 CPU cores



GPU is not being kept busy

Too much CPU work:
2200 patches across 28 cores



Patches running sequentially within each core

CPU integrator work is mostly data parallel, but…

- Uses double precision for positions, velocities, forces
  - **Data layout is array of structures (AOS)**, not well-suited to vectorization
- Each NAMD “patch” runs integrator in **separate user-level thread** to make source code more accessible
  - Benefit from vectorization is reduced, loop over 200–600 atoms in each patch
- Too many **exceptional cases** handled within same code path
  - E.g. fixed atoms, pseudo-atom particles (Drude and lone pair)
  - Test conditionals for simulation options and rare events (e.g. trajectory output) **every timestep**

CPU integrator work is mostly data parallel, but…

- Additional communication is required
  - Send reductions for kinetic energies and virial
  - Receive broadcast periodic cell rescaling when simulating constant pressure
- A few core **algorithms have sequential parts**, with reduced parallelism
  - Not suitable for CPU vectorization
  - **Rigid bond constraints** — successive relaxation over each hydrogen group
  - **Random number generator** — Gaussian numbers using Box–Muller transform with rejection
  - Reductions calculated over “hydrogen groups” — irregular data access patterns

Strategies for Overcoming Bottleneck

- Data structures for CPU vectorization
  - **Convert atom data storage from AOS (array of structures) form into vector friendly SOA (structure of arrays) form**
- Algorithms for CPU vectorization
  - **Replace non-vectorizing RNG code with vectorized version**
  - **Replace rigid bond constraints sequential algorithm with one capable of fine-grained parallelism** (maybe LINCS or Matrix-SHAKE)
- **Offload integrator to GPU**
  - Main challenge is aggregating patch data
  - Use vectorized algorithms, adapt curand for Gaussian random numbers

Goal: Developing GPU-based NAMD

- CPU primarily manages GPU kernel launching
  - CPU prepares and aggregates data structures for GPU, **handles Charm++ communication**
- Reduces overhead of host-to-device memory transfers
  - **Data lives on the GPU**, clusters of patches
  - Communicate edge patches for force calculation and atom migration
- Design new data structures capable of GPU or CPU vectorization
  - Major refactor of code to **reveal more data parallelism**
- Kernel-based design with consistent interfaces across GPU and CPU
  - Need **fallback kernels for CPU** (e.g. continue to support Blue Waters and Titan)

Some Conclusions

For HPC apps in general and NAMD in particular

- Balance between CPU and GPU computational capability continues to shift in favor of the GPU
  - **The 1% workload from 10 years ago is now 60% or more in wall clock time**
- Any past code that has attempted to load balance work between CPU and GPU is today likely to be CPU bound
- Best utilization of GPU might require keeping all data on the GPU
  - **Motivates turning a previously CPU-based code using GPU as an accelerator into a GPU-based code using CPU as a kernel management and communication coprocessor**
- Volta + CUDA 9.x + CUDA Toolkit Libraries provide enough general purpose support to allow moving an application entirely to the GPU

What will Charm++'s role be for multi-GPU NAMD?

- Single-node multi-GPU case
  - With everything running on CPU-managed GPUs, Charm++ has nothing to do!
  - Even now, **NAMD patches are non-migratable**
- Multi-node multi-GPU case
  - Charm++ handles internode communication
  - **Load balancing is still ineffective, until Charm++ understands GPU work**
- Charm++ could help us make use of improvements to device communication, like NVLink

Acknowledgments

- Special thanks to:
  - John Stone and others from Theoretical Biophysics Research Group, Beckman Institute, UIUC
  - Ke Li and others from NVIDIA CUDA Team
  - NVIDIA NSight Systems Team
  - James Phillips, NCSA, UIUC
  - In memoriam Antti-Pekka Hynninen, NVIDIA
  - Ronak Buch and Karthik Senthil from Parallel Programming Laboratory, Dept of Computer Science, UIUC
- Grants:
  - NIH P41-GM104601 Center for Macromolecular Modeling and Bioinformatics
  - Summit Center for Accelerated Application Readiness, OLCF, ORNL