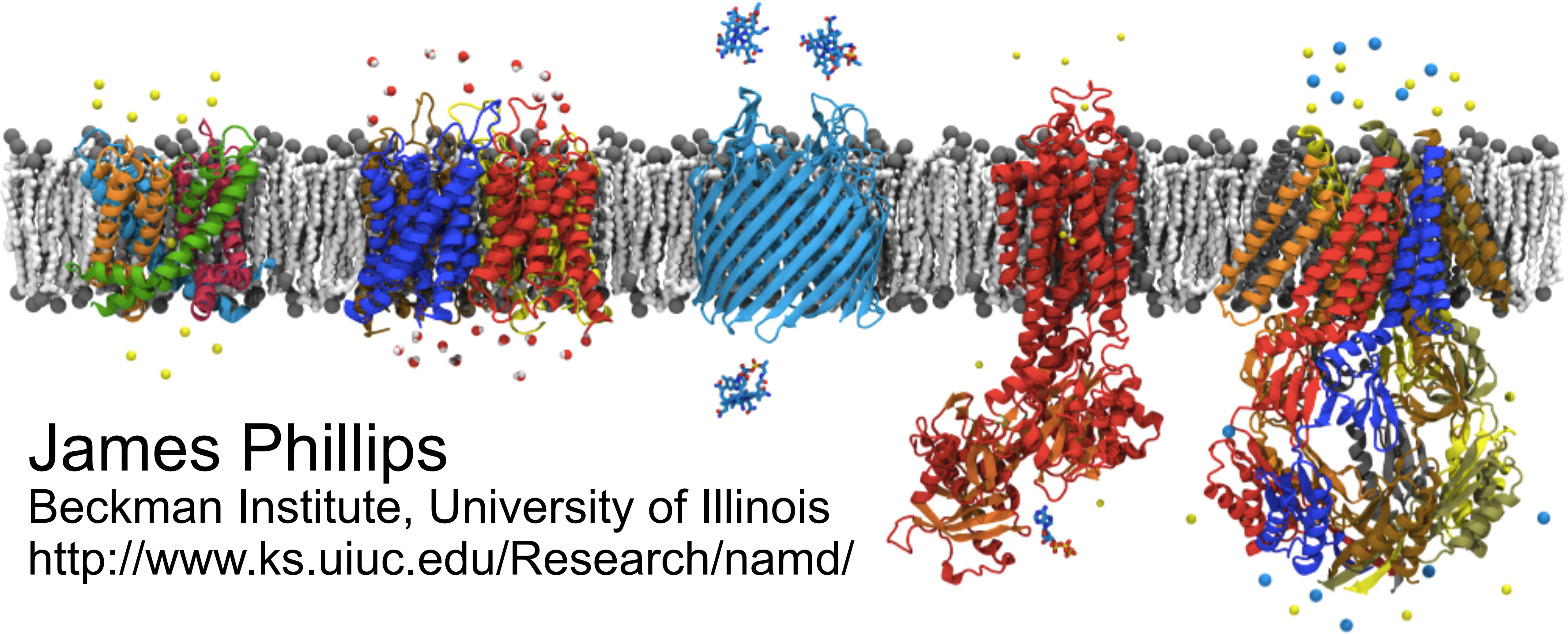
Towards *Process-Level* Charm++ Programming in NAMD



James Phillips
Beckman Institute, University of Illinois
http://www.ks.uiuc.edu/Research/namd/

NIH Biomedical Technology Research Center for Macromolecular Modeling and Bioinformatics

Developers of the widely used computational biology software VMD and NAMD

250,000 registered VMD users
72,000 registered NAMD users

Renewed 2012-2017 with 10.0 score (NIH)

research projects include: virus capsids, ribosome, photosynthesis, protein folding, membrane reshaping, animal magnetoreception

600 publications (since 1972)
over 54,000 citations

5 faculty members
8 developers
1 systems administrator
17 postdocs
46 graduate students
3 administrative staff

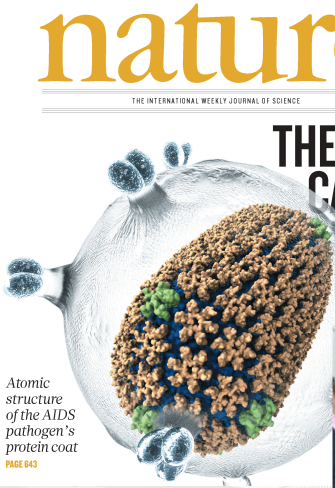
Achievements Built on People

Tajkorshid, Luthey-Schulten, Stone, Schulten, Phillips, Kale, Mallon

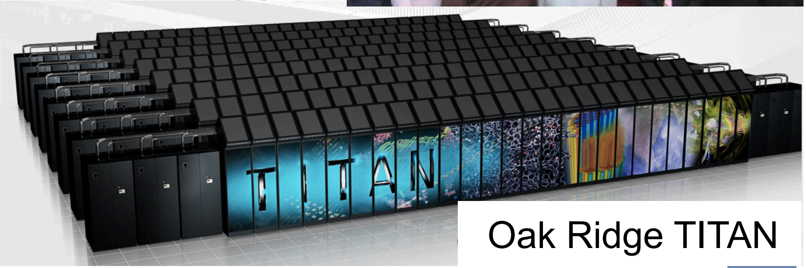
NAMD Serves NIH Users and Goals

Practical Supercomputing for Biomedical Research

- 72,000 users can't all be computer experts.
  - 18% are NIH-funded; many in other countries.
  - 21,000 have downloaded more than one version.
  - 5000 citations of NAMD reference papers.
- One program available on all platforms.
  - Desktops and laptops – setup and testing
  - Linux clusters – affordable local workhorses
  - Supercomputers – free allocations on XSEDE
  - Blue Waters – sustained petaflop/s performance
  - GPUs - next-generation supercomputing
- User knowledge is preserved across platforms.
  - No change in input or output files.
  - Run any simulation on **any number of cores.**
- Available free of charge to all.


nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
THE HIV-1 CAPSID
Atomic structure of the AIDS pathogen's protein coat
PAGE 643

Hands-On Workshops



Oak Ridge TITAN

NAMD Benefits from Charm++ Collaboration

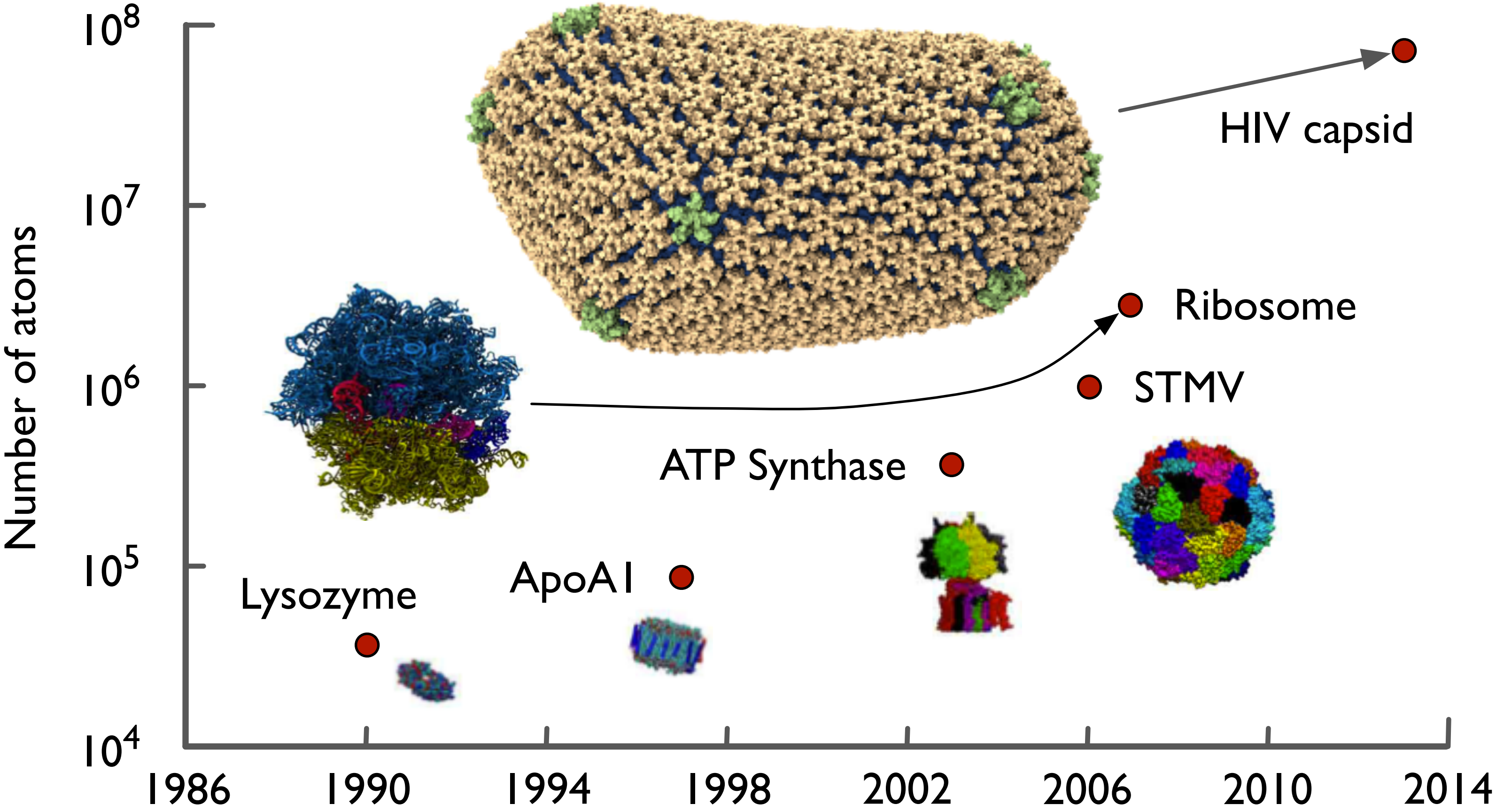
- Illinois Parallel Programming Lab
  - Prof. Laxmikant Kale
  - charm.cs.illinois.edu

KLAUS SCHULTEN
Center for the Physics of Living Cells
University of Illinois
2012 IEEE COMPUTER SOCIETY
SIDNEY FERNBACH
MEMORIAL AWARD
LAXMIKANT V. KALE
National Center for Supercomputing
Applications · University of Illinois

- Long standing collaboration
  - Since start of Center in 1992
  - Gordon Bell award at SC2002
  - Joint Fernbach award at SC12

- Synergistic research
  - NAMD requirements drive and validate CS work
  - Charm++ software provides unique capabilities
  - Enhances NAMD performance in many ways

Structural data drives simulations

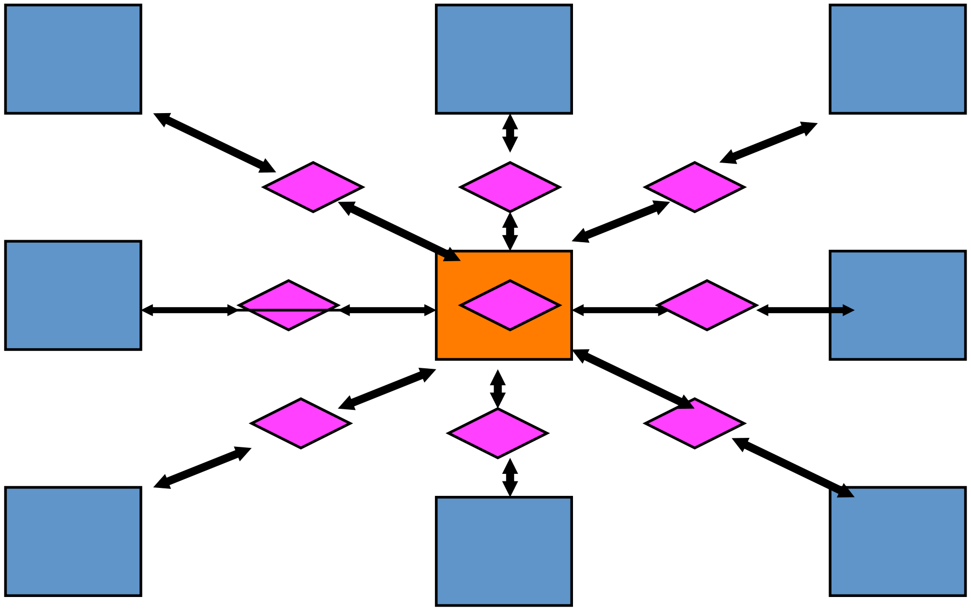


Charm++ Used by NAMD

- Parallel C++ with *data driven* objects.
- Asynchronous method invocation.
- Prioritized scheduling of messages/execution.
- Measurement-based load balancing.
- Portable messaging layer.

NAMD Hybrid Decomposition

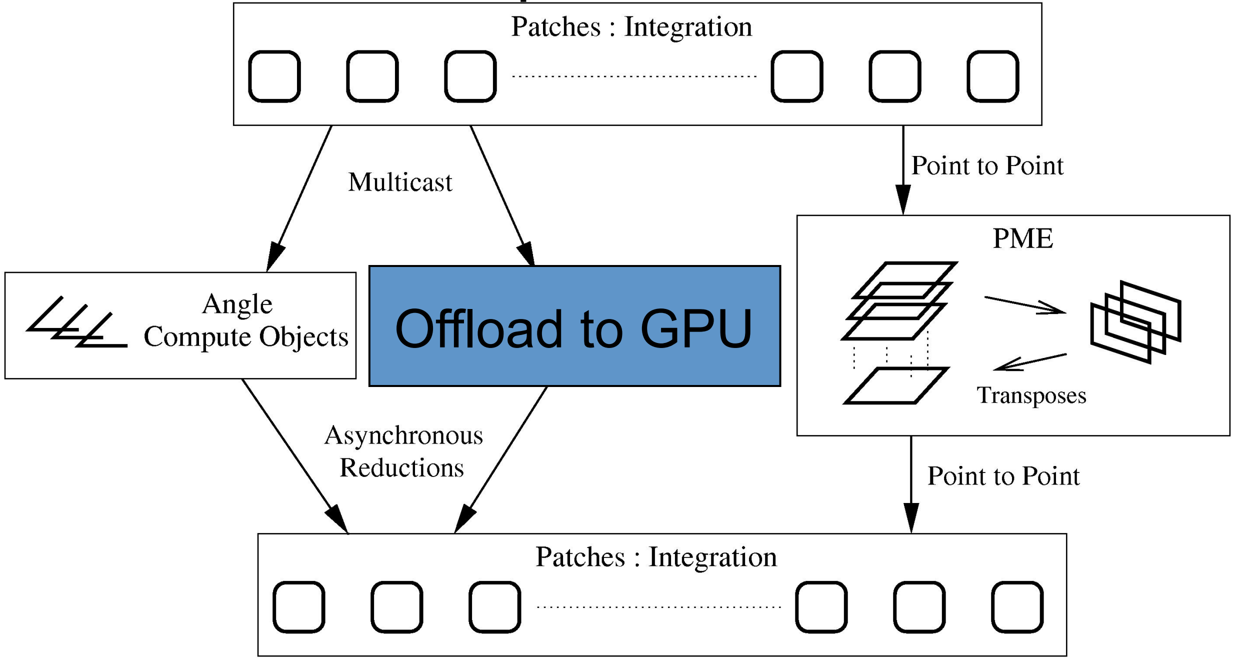
Kale *et al., J. Comp. Phys.* 151:283-312, 1999.



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

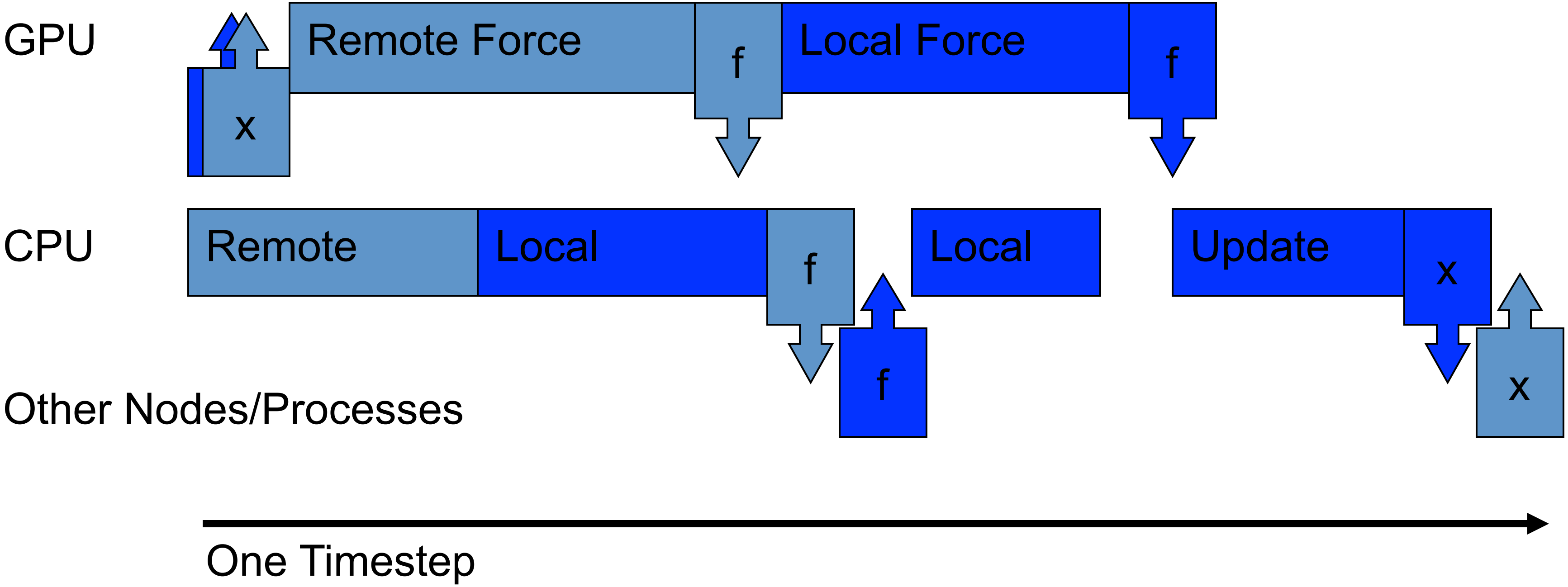
NAMD Overlapping Execution

Phillips *et al.*, *SC2002*.



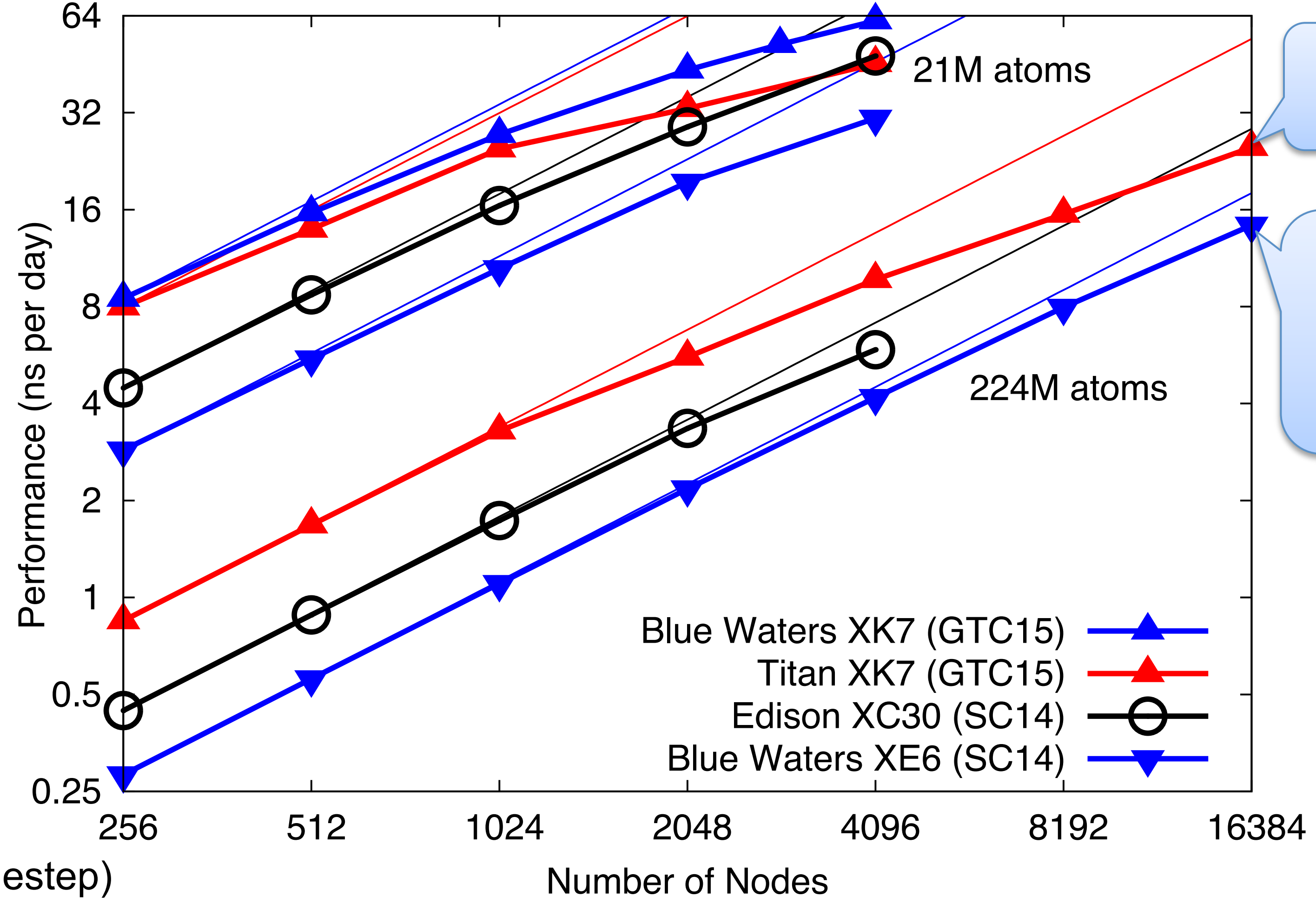
Objects are assigned to processors and queued as data arrives.

Overlapping GPU and CPU with Communication



Phillips *et al.*, SC2008

NAMD on Petascale Platforms Today



25 ns/day, 7 ms/step

14 ns/day, 79% parallel efficiency on 224M atoms

(2fs timestep)

Future NAMD Platforms

- NERSC Cori / Argonne Theta (2016)
  - Knight’s Landing (KNL) Xeon Phi
  - Single-socket nodes, Cray Aries network
- Oak Ridge Summit (2018)
  - IBM Power 9 CPUs + NVIDIA Volta GPUs
  - 3,400 fat nodes, dual-rail InfiniBand network
- Argonne Aurora (2018)
  - Knight’s Hill (KNH) Xeon Phi

Charm++ Programming Model

- Programmer:
  - – Reasons about (arrays of) chares
  - – Writes entry methods for chares
  - – Entry methods send messages
- Runtime:
  - – Manages (re)mapping of chares to PEs

What if PEs share a host?

- Communication can bypass network
- Opportunity for optimization!
  - Multicast and reduction trees (easy)
  - Communication-aware load balancer (hard)
- May share a GPU (inefficiently)
  - Likely need CUDA Multi-Process Service

What if PEs share a host?

- Charm++ detects “physical nodes”.
- NAMD optimizations:
  - Place patches based on physical nodes.
  - Place computes on same physical nodes.
  - Optimize trees for patch positions, forces.
  - Optimize global broadcast and reductions.

What if PEs share a host?

- Non-SMP NAMD runs are common.
  - Avoid bottlenecks in Linux malloc(), free(), etc.
  - Don’t waste cores on communication threads.
  - Best performance for small simulations.
- This will likely be changing:
  - SMP builds are now faster on Cray for all sizes.
  - Fixing communication thread lock contention.

What if PEs share a process?

- Also share a host (see preceding slides).
- Share one copy of static data.
- Communicate by passing pointers.
- Share one CUDA context.
  - Use CUDA streams to overlap on GPU.
  - Avoid using shared default stream.

What if PEs share a process?

- Each process is Charm++ “node”.
- No-pack messages to same-node PEs.
- Node-level locks and PE-private variables.
- Messages to “nodegroup” run on any PE.
- Communication thread handles network.
- CkLoop for OpenMP-style parallelism.

What if PEs share a socket?

- Shared memory controller and L3 cache.
  - Duplicate data reduces cache efficiency.
  - Work with same data at same time if possible.
    - OpenMP and CkLoop do this naturally.
    - Possible to run one PE/socket and use OpenMP or CkLoop to parallelize across cores on socket.

What is most relevant for NAMD?

- One process per node
  - Single-node (multicore)
  - Largest simulations, memory limited
  - At most one process per GPU/MIC (offload)
- One or two processes per socket
  - Cray XE/XC or 64-core Opteron cluster
- Manually set CPU affinity:
  - E.g., +pemap 0-6,8-14 +commap 7,15

Process-level NAMD Today

- Patch position/force trees
  - Use nodegroup to avoid delaying messages
- GPU/MIC offload
  - Aggregate work, serialize control
- PME electrostatics on petascale
  - Single pencil per node to reduce message count
- PME offload to GPU
  - Aggregate work, serialize control

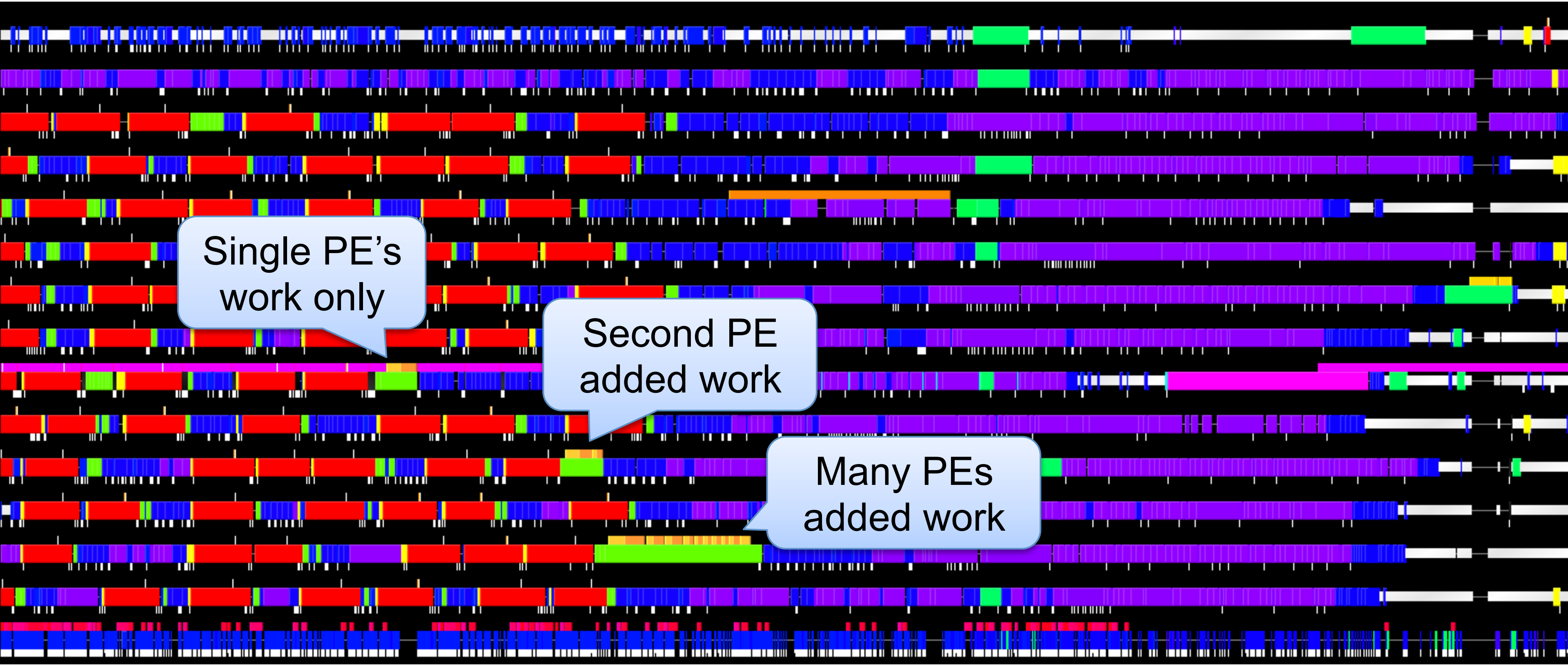
Process-level NAMD Idioms

- Paired group and nodegroup for trees
- CkLoop or nodegroup for bottlenecks
  - PME FFT and transpose messages
- Access chare on other PE via pointer
  - GPU data and result processing
- GPU control-serialization queue
  - Want one PME charge grid CUDA stream per PE
  - But there are locks hidden in CUDA calls

Serialization Queue Idiom

- When PE is ready to submit work to GPU:
  - Lock queue.
  - Queue marked as busy? Add work and unlock.
  - Not marked as busy?
    - Mark as busy, unlock queue, submit work, lock queue.
    - While work in queue: unlock, submit work, lock.
  - Mark queue as not busy and unlock queue.

Serialization Queue Trace



Suggested Charm++ Features

- nodearray chare entries run on any PE in node
  - Serialized per element unless [reentrant]
- [mobile] entries for groups and arrays
  - Execute on any PE in node similar to nodegroup
- Set-exclusive entry points
  - Serialize calls on same chare to entries in set
- Set-reentrant entry points
  - Serialize calls outside set on same chare

Charm++ Programming Model

- Programmer:
  - – Reasons about (arrays of) chares
  - – Writes entry methods for chares
  - – Entry methods send messages
- Runtime:
  - – Manages (re)mapping of chares to PEs

Charm++ Programming Model

- Programmer:
  - Reasons about (arrays of) chares
  - Writes entry methods for chares
  - Labels entry methods as [reentrant], etc.
  - Entry methods send messages
- Runtime:
  - Manages (re)mapping of chares to nodes/PEs

What if PEs share a core?

- Hardware threads up to SMT8 on Power8.
- Shared L1/L2 cache – same as previous.
- Shared execution units.
  - Busy-waiting can slow PEs on same core.
  - Load balancer measurements more variable.

Hybrid Charm++/OpenMP?

- Leverage vendor-optimized OpenMP 4.X
- One thread team per PE
  - Team master thread runs Charm++ scheduler
  - Use pthreads, atomics, etc. as now
  - Likely one team (PE) per core
- OpenMP directives distribute loops to threads
  - Also SIMD directives for vector instructions

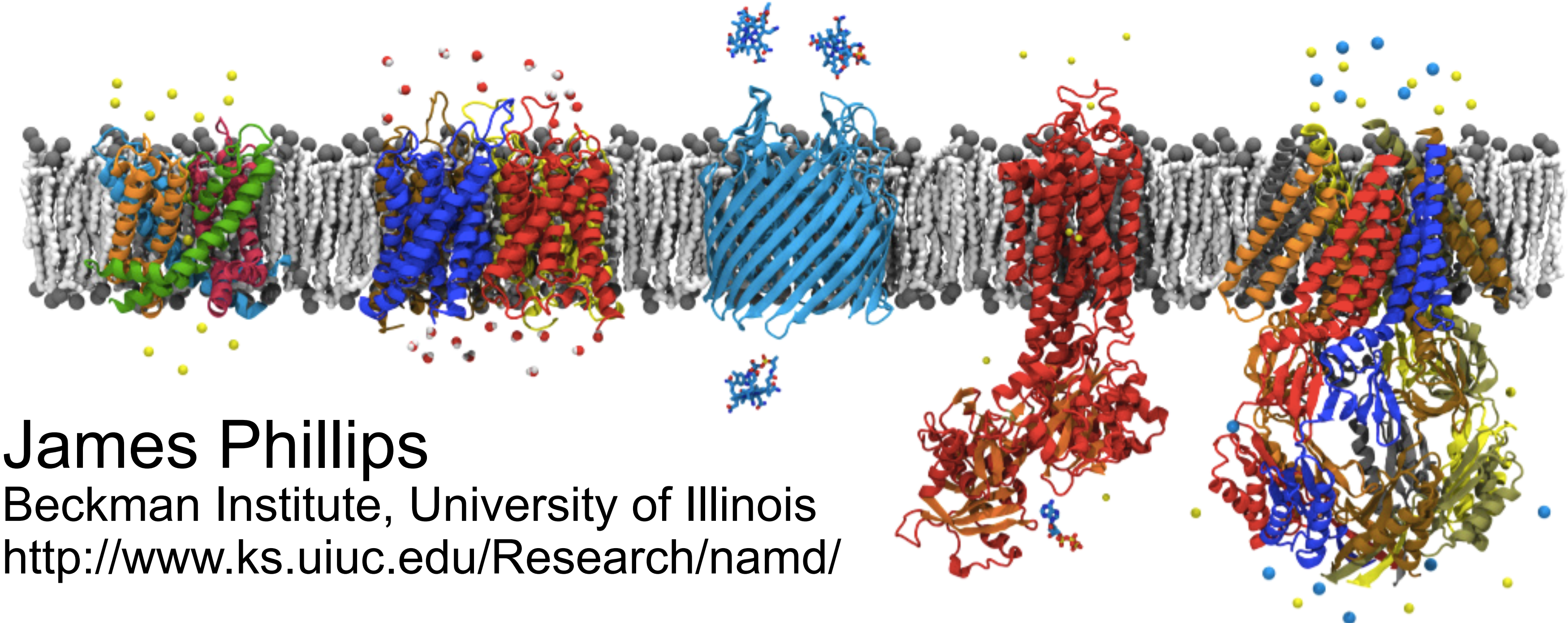
Charm++ Programming Model

- Programmer:
  - Reasons about (arrays of) chares
  - Writes entry methods for chares
  - Labels entry methods as [reentrant], etc.
  - Exposes loop-level parallelism via OpenMP
  - Entry methods send messages
- Runtime:
  - Manages (re)mapping of chares to nodes/PEs

Conclusions

- Process-level programming is needed.
- Current Charm++ support is inelegant.
- Small Charm++ changes expose:
  - Additional scheduling flexibility
  - Entry point concurrency internal to chares
  - Loop-level parallelism internal to chares

Thanks to: NIH, NSF, DOE, NCSA,
NVIDIA (**Sarah Tariq**, Patric Zhao, Sky Wu, Justin Luitjens, Nikolai Sakharnykh),
Cray (Sarah Anderson, Ryan Olson), NCSA (Robert Brunner),
PPL (Eric Bohm, Yanhua Sun, Gengbin Zheng, Nikhil Jain)
and 20 years of NAMD and Charm++ developers and users.



James Phillips
Beckman Institute, University of Illinois
http://www.ks.uiuc.edu/Research/namd/