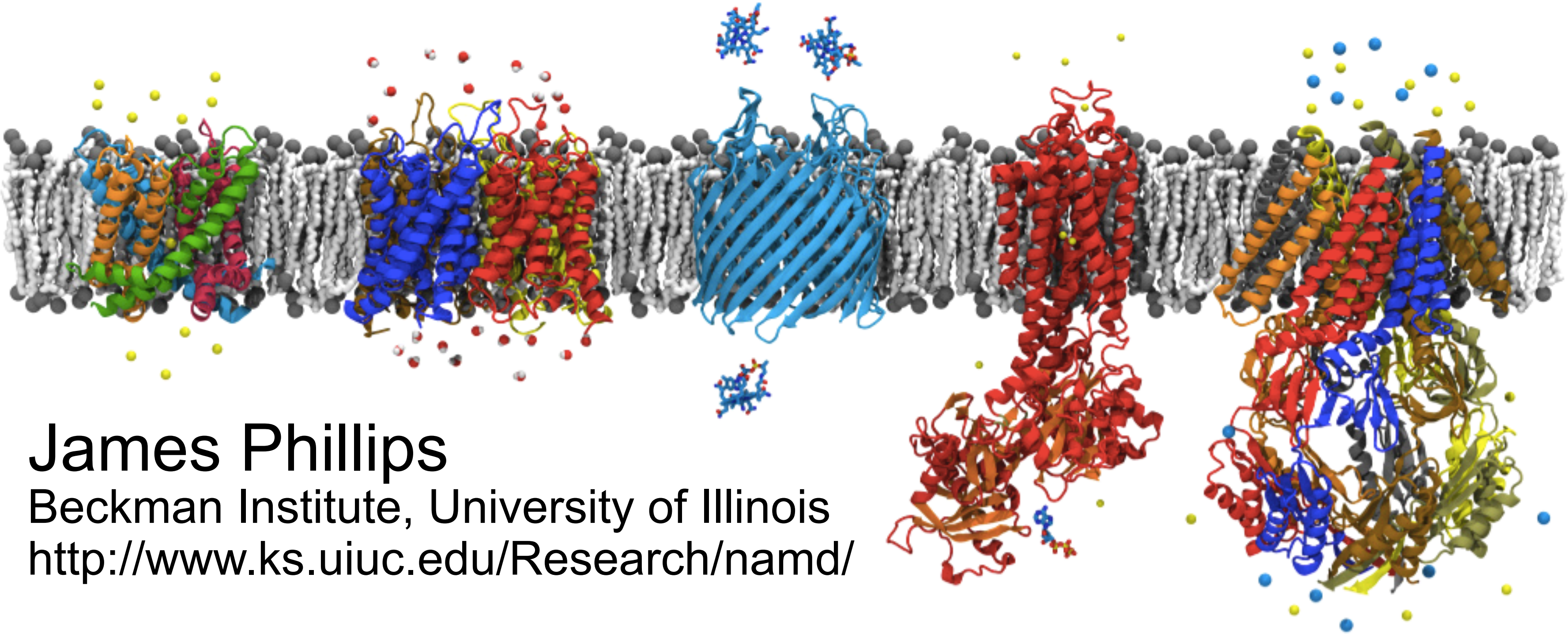
Scriptable Asynchronous Multi-Copy Algorithms in NAMD via Charm++ Partitions



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
80,000 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

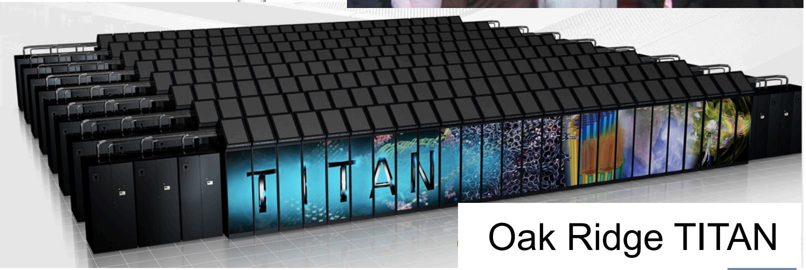
Tajkorshid, Luthey-Schulten, Stone, Schulten, Phillips, Kale, Mallon

NAMD Mission Statement:

Practical Supercomputing for Biomedical Research

- 80,000 users can't all be computer experts.
  - 18% are NIH-funded; many in other countries.
  - 24,000 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 – from desktop to supercomputer
- 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.

Hands-On Workshops



Oak Ridge TITAN

NAMD Benefits from Charm++ Collaboration

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

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

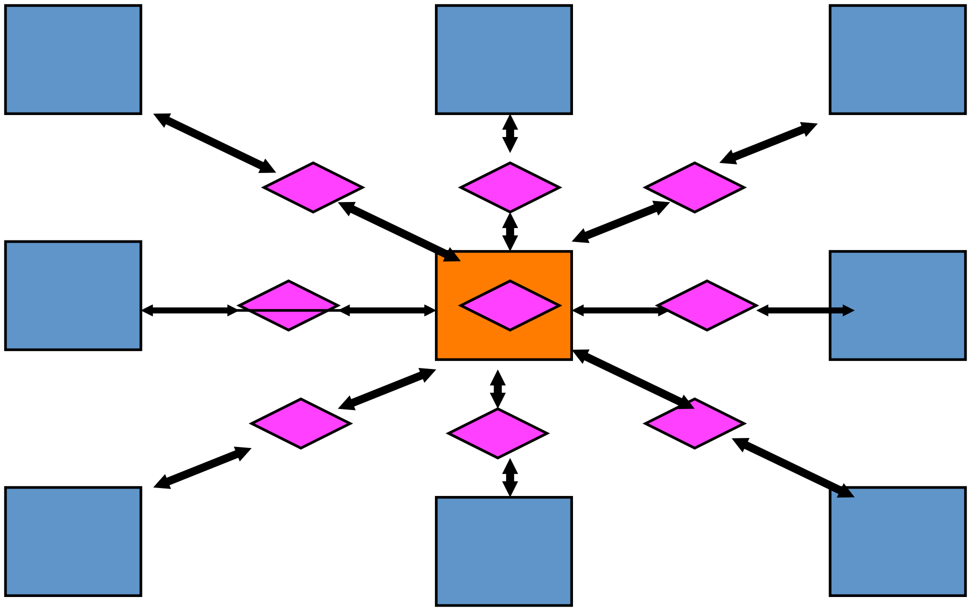
- Synergistic research
  - NAMD requirements drive and validate CS work
  - Charm++ software provides unique capabilities
  - Enhances NAMD performance in many ways

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

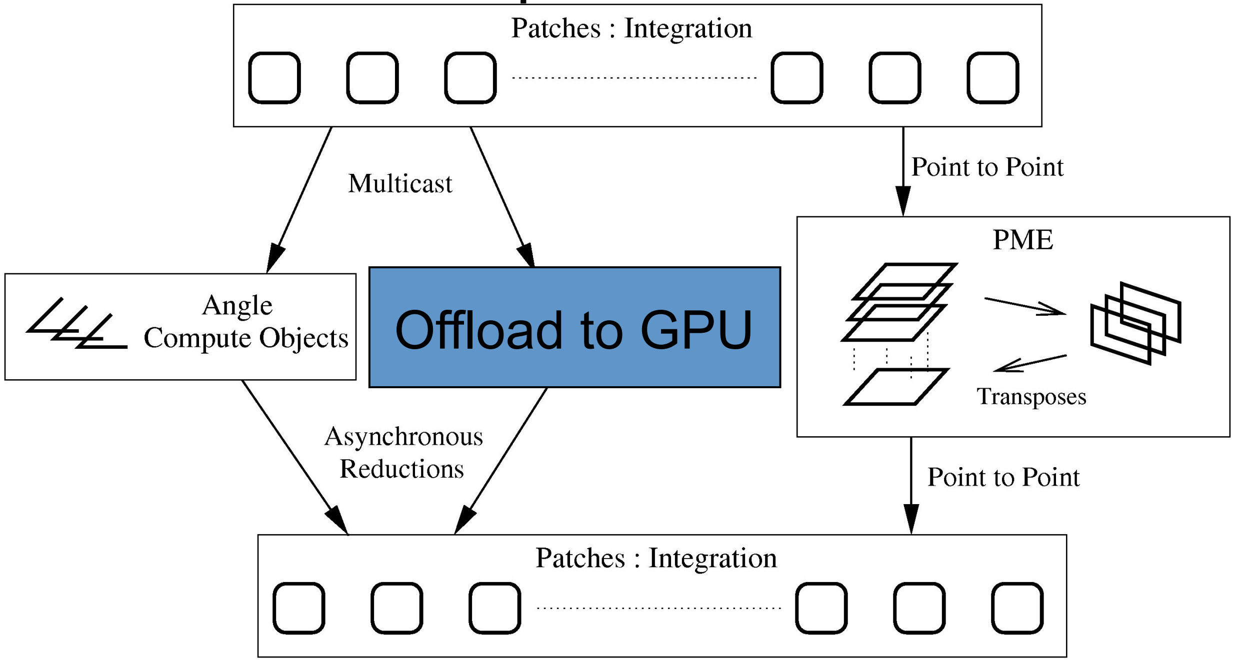
Kale *et al.*, *J. Comp. Phys.* 151:283-312, 1999.



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

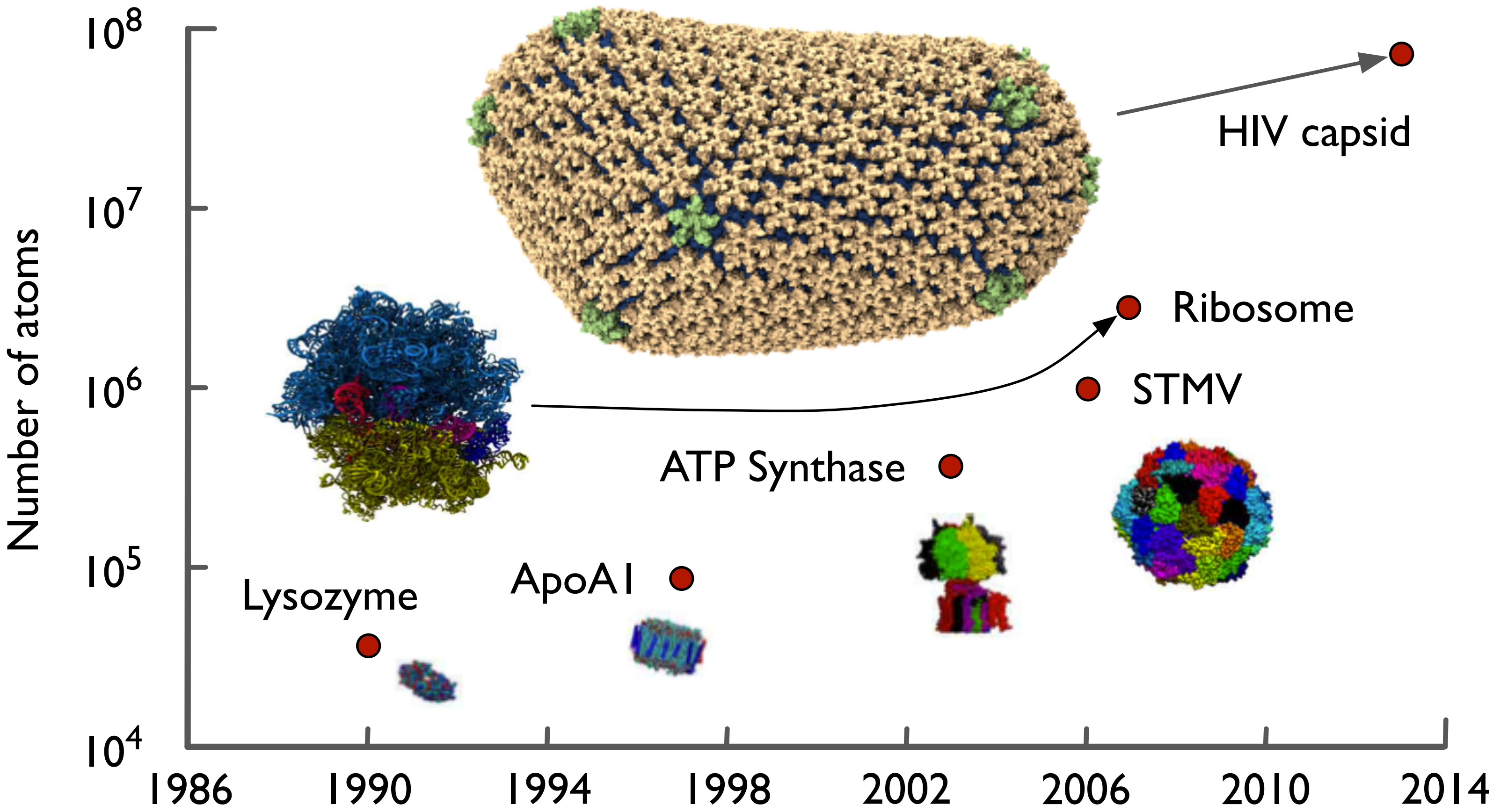
NAMD Overlapping Execution

Phillips *et al., SC2002.*

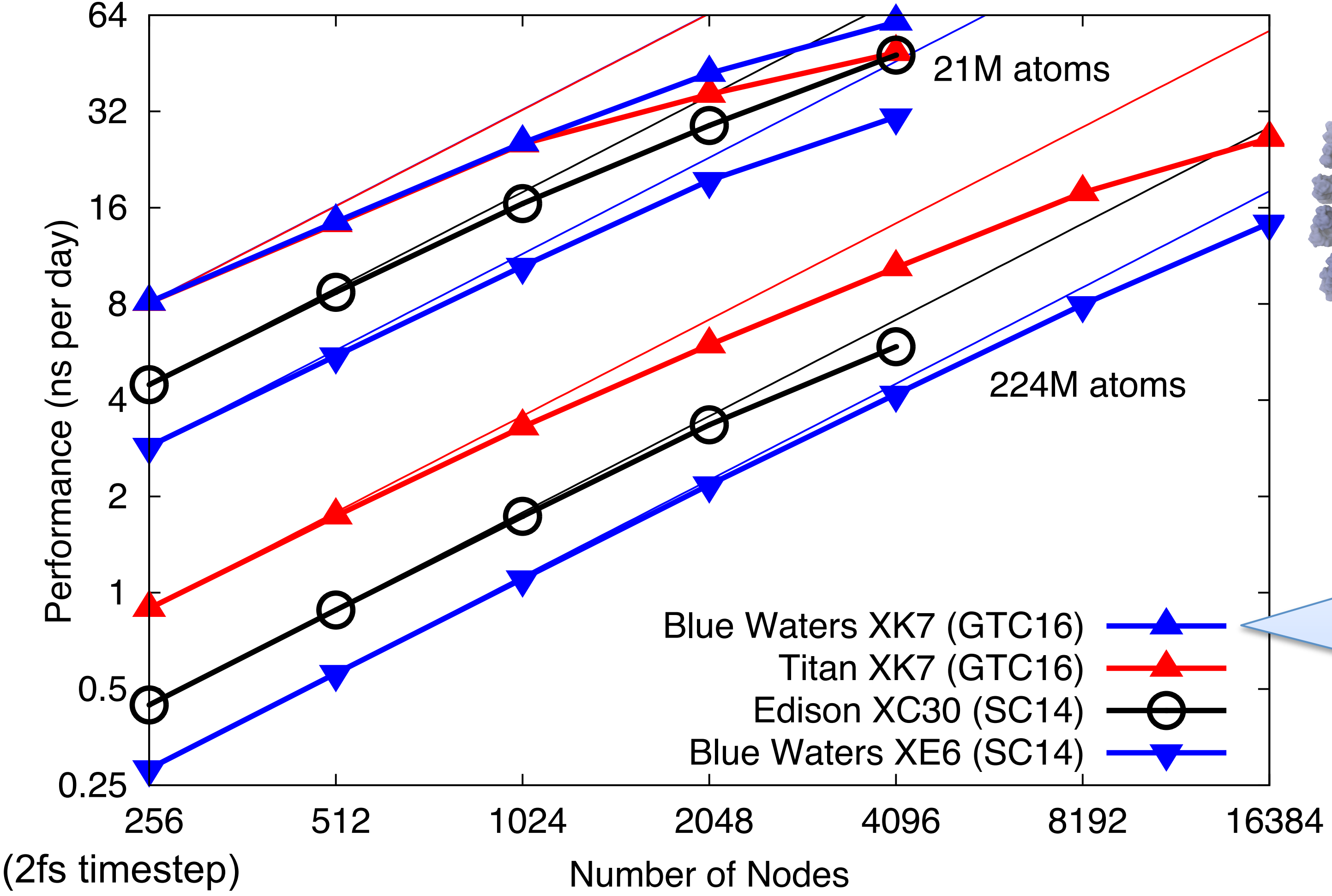


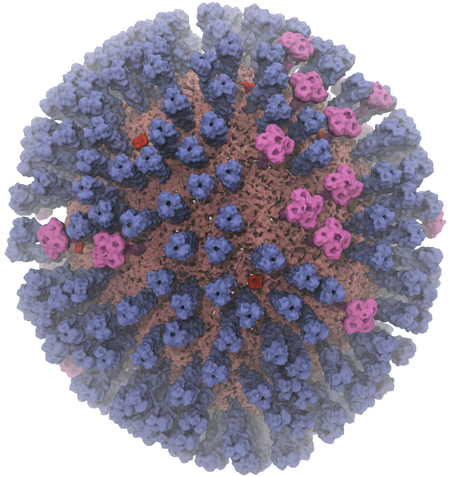
Objects are assigned to processors and queued as data arrives.

Structural data drives larger simulations



NAMD Runs Large Petascale Simulations Well

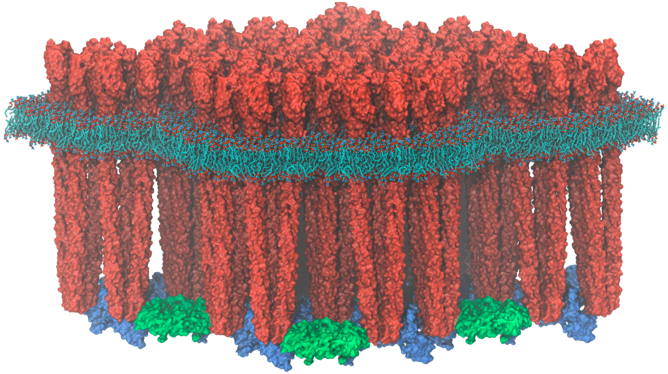




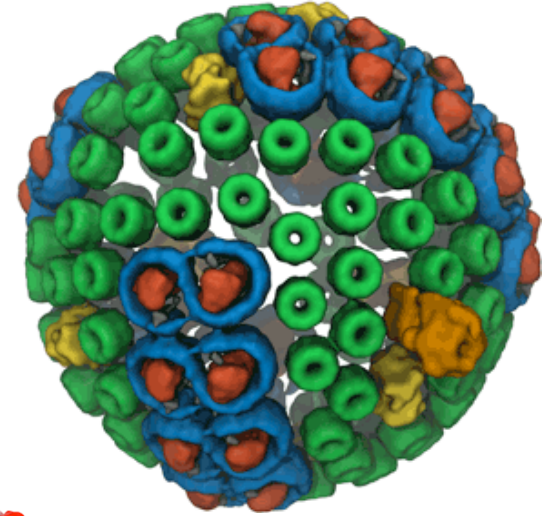
Influenza, 210M atoms
Amaro Lab, UCSD

Topology-aware scheduler

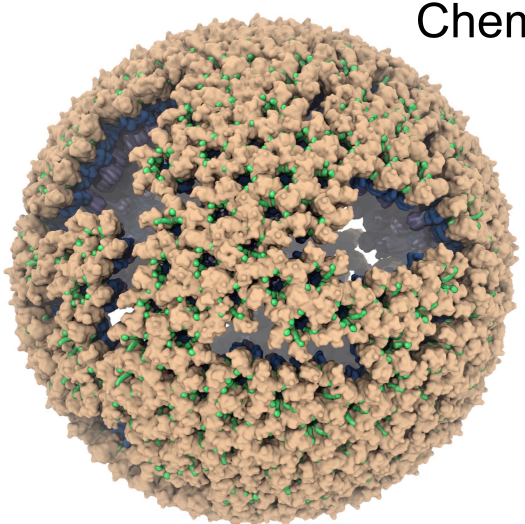
A Sampling of Petascale Projects Using NAMD



Chemosensory Array

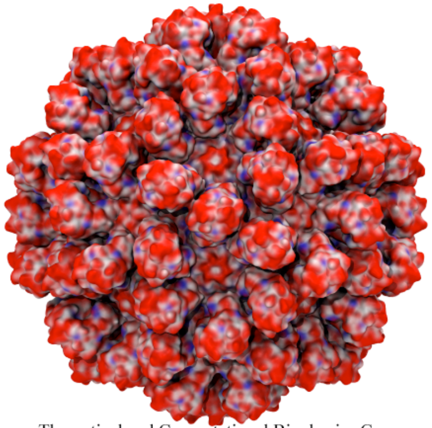


Chromatophore



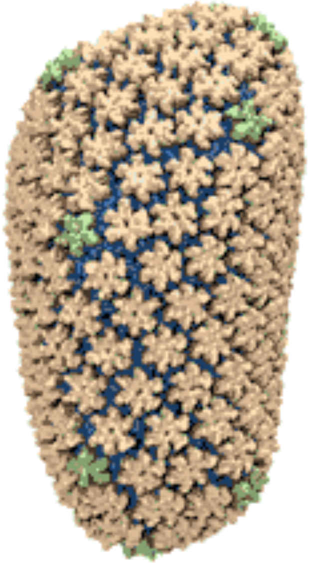
Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champaign

Rous Sarcoma Virus



Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champaign

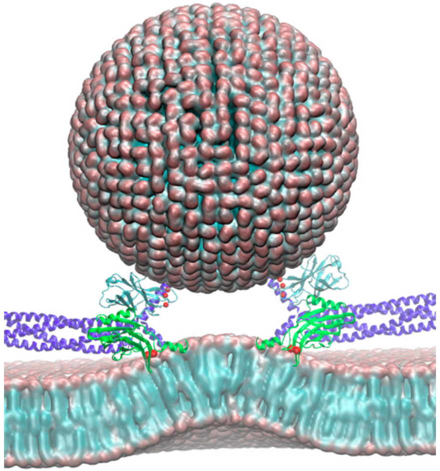
Rabbit Hemorrhagic Disease



HIV

Future NAMD Platforms

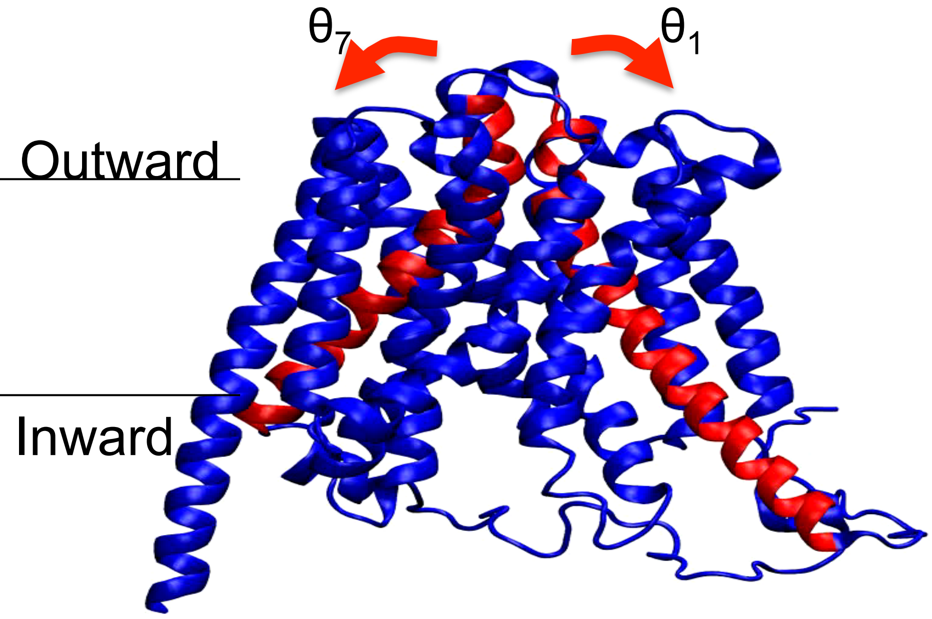
- NERSC Cori / Argonne Theta (2016)
  - Knight's Landing (KNL) Xeon Phi
  - Single-socket nodes, Cray Aries network
  - Theta Early Science Project:
    "Free Energy Landscapes of
    Membrane Transport Proteins"
- Oak Ridge Summit (2018)
  - IBM Power 9 CPUs + NVIDIA Volta GPUs
  - 3,400 fat nodes, dual-rail InfiniBand network
  - CAAR Project "Molecular Machinery of the Brain"
- Argonne Aurora (2018)
  - Knight's Hill (KNH) Xeon Phi



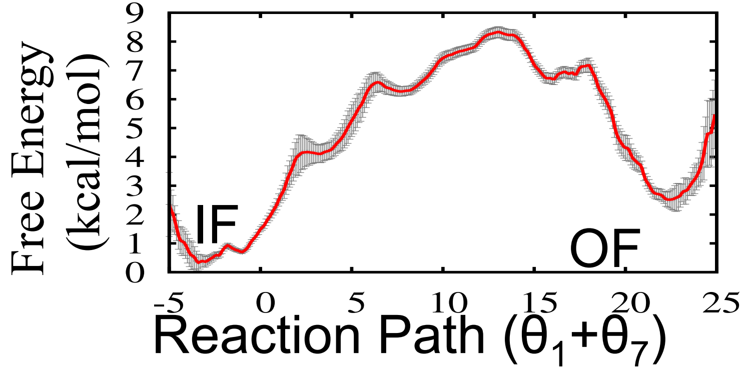
Synaptic vesicle and presynaptic membrane

Replica Exchange Enables Advanced Sampling

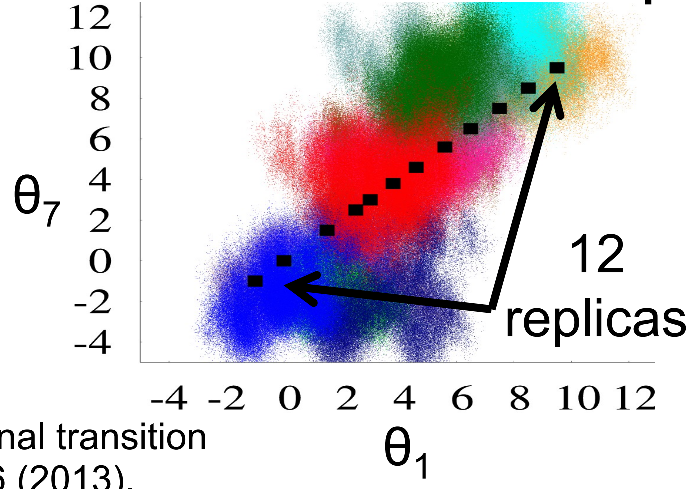
Bias-Exchange Umbrella Sampling
on quaternion-based order parameters



GlpT transporter in membrane

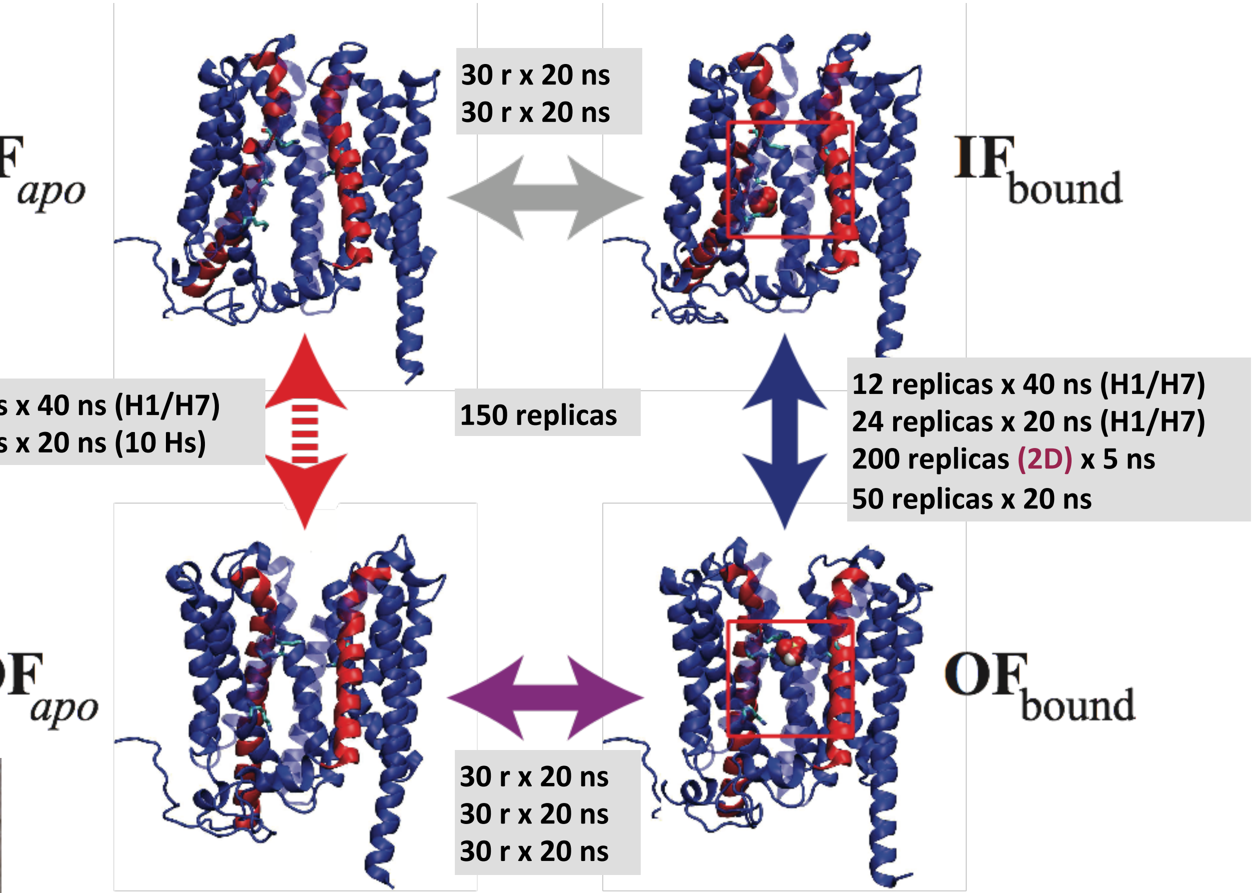


Efficient Reaction Path Sampling



M. Moradi and E. Tajkhorshid, Mechanistic picture for conformational transition of a membrane transporter at atomic resolution. PNAS, 110:18916 (2013).

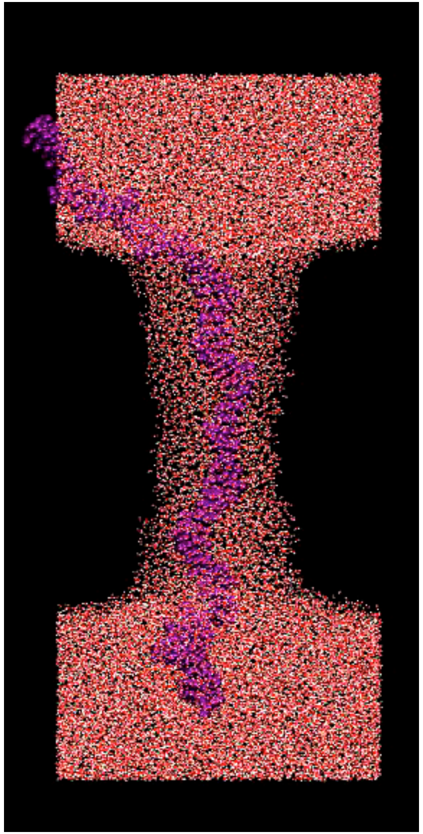
Multi-Copy Algorithm Sophistication Increases



Mahmoud Moradi, bias-exchange umbrella sampling simulations of GlpT

Tcl Scripting Enables Portable Customization

- Top-level protocols:
  - Replica exchange (originally via sockets)
  - Minimize, heat, equilibrate
  - Simulated annealing
- Long-range forces on selected atoms
  - Torques and other steering forces
  - Adaptive bias free energy perturbation
  - Coupling to external coarse-grain model
- Special boundary forces
  - Applies potentially to every atom
  - Several optimizations for efficiency
  - Shrinking phantom pore for DNA



Why NAMD and VMD Use Tcl

- History: Programs are ~20 years old.
- Maturity: Package management, portable.
- Stability: Interfaces haven't changed.
- Flexibility: Encapsulates mini-languages.
- Approachability: Looks like a simple scripting language, doesn't scare non-programmers.

- Next NAMD adds optional Python interpreter.

Tcl and Charm++ in NAMD

- Tcl runs on PE 0 only
  - Tcl parses config file until end or “run”
  - Send startup messages, run scheduler
    - Scheduler processes messages, starts run
    - At end of run, exit scheduler on quiescence
  - Tcl continues parsing config file...

Tcl Overview

- Variables: $var $array($key) $array($i.field)
- Strings: abc 123 “$sub” {$nosub} [eval this]
- Commands: command $byvalue byname
  - To create commands: proc {args} {script}
  - upvar and uplevel access calling namespace
  - Control structures are just commands
- Simple core enables great flexibility.

NAMD 2.9 Multi-Copy Tcl Interface

- Blocking communication (MPI semantics):
  - replicaSend *data dest*
  - replicaRecv *source*
  - replicaSendrecv *data dest source*
- Utility functions:
  - myReplica
  - numReplicas
  - replicaBarrier

Replica Exchange in Tcl

```
while {$i_run < $num_runs} {
  run $steps_per_run
  save_array
  incr i_step $steps_per_run
  set TEMP $saved_array(TEMP)
  set POTENTIAL $saved_array(POTENTIAL)
  puts $history_file \
    "$i_step $replica(index) $NEWTEMP $TEMP $POTENTIAL"
  if { $i_run % 2 == 0 } { set swap a; set other b
  } else { set swap b; set other a }
  set doswap 0
  if { $replica(index) < $replica(index.$swap) } {
    set temp $replica(temperature)
    set temp2 $replica(temperature.$swap)
    set BOLTZMAN 0.001987191
    set dbeta [expr ((1.0/$temp) - (1.0/$temp2)) / $BOLTZMAN]
    set pot $POTENTIAL
    set pot2 [replicaRecv $replica(loc.$swap)]
    set delta [expr $dbeta * ($pot2 - $pot)]
    set doswap [expr $delta < 0. || exp(-1.*$delta) > rand()]
    replicaSend $doswap $replica(loc.$swap)
    if { $doswap } { set rid $replica(index);
        set rid2 $replica(index.$swap) }
  }
  if { $replica(index) > $replica(index.$swap) } {
    replicaSend $POTENTIAL $replica(loc.$swap)
    set doswap [replicaRecv $replica(loc.$swap)]
  }
  set newloc $r
  if { $doswap } {
    set newloc $replica(loc.$swap)
    set replica(loc.$swap) $r
  }
  set replica(loc.$other) [replicaSendrecv \
    $newloc $replica(loc.$other) $replica(loc.$other)]
  set oldidx $replica(index)
  if { $doswap } {
    set OLDTEMP $replica(temperature)
    array set replica [replicaSendrecv [array get replica] $newloc $newloc]
    set NEWTEMP $replica(temperature)
    rescalevels [expr sqrt(1.0*$NEWTEMP/$OLDTEMP)]
    langevinTemp $NEWTEMP
  }
  incr i_run
}
```

MPI Implementation (NAMD 2.9)

- Charm++ initialization:
  - MPI_Comm_split(MPI_COMM_WORLD,
    myReplicaID, my_rank_in_replica, &MPI_COMM_LOCAL);
  - MPI_Comm_split(MPI_COMM_WORLD,
    my_rank_in_replica, myReplicaID, &MPI_COMM_CROSS);

- Direct mapping to MPI communication:
  - void CmiReplicaSend(void *buf, int count, int dest) {
    MPI_Send(buf, count, MPI_BYTE, dest, 1, MPI_COMM_CROSS);
    }

- Limited by performance of Charm++ MPI communication layer

LRTS Implementation (NAMD 2.10)

- Converse asynchronous messaging:
  - void CmiInterSyncSendFn(int destPE, int partition int size, char *msg)
  - void CmiInterSyncNodeSendFn(int destNode, int partition, int size, char *msg)

- MPI-style blocking communication in Charm++:

```
entry void send(Pointer srcPointer, int srcSize, int dstPart, int dst) {
      serial {
        packSend(dst,dstPart,(char*)srcPointer.data,srcSize,recv_data_idx);
      }
      when recv_ack() serial {
        CpvAccess(breakScheduler) = 1;
      }
};
```

Credit: Nikhil Jain, NCSA funding

Atom-Exchange Tcl Interface

- replicaAtomSend *dest*
- replicaAtomRecv *source*
- replicaAtomSendrecv *dest source*

- Needed if settings can't be modified.
- Direct patch-to-patch communication
  - Requires "replicaUniformPatchGrids yes"

NAMD Replica Exchange Limitations

- One-to-one replicas to Charm++ partitions:
  - Available hardware must match science.
  - Batch job size must match science.
  - Replica count fixed at job startup.
  - No hiding of inter-replica communication latency.
  - No hiding of replica performance divergence.
- Can a different programming model help?

Swift (swift-lang.org) programming model: all progress driven by concurrent dataflow

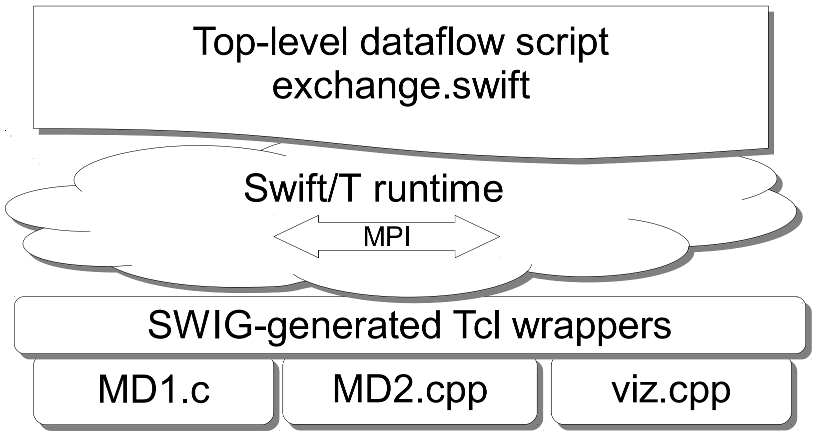
```
(int r) myproc (int i, int j)
{
    int f = F(i);
    int g = G(j);
    r = f + g;
}
```

- `F()` and `G()` implemented in native code or external programs
- `F()` and `G()` run in concurrently in different processes
- `r` is computed when they are both done

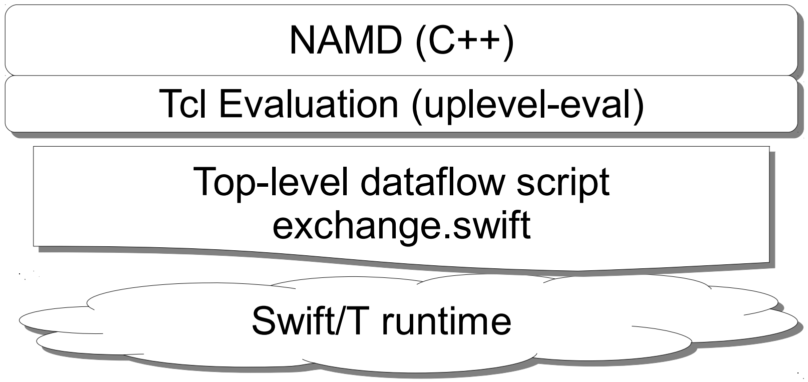
- This parallelism is *automatic*
- Works recursively throughout the program's call graph

NAMD and Swift/T

Typical Swift/T Structure



NAMD Structure



Phillips et al., Petascale Tcl with NAMD, VMD, and Swift/T.
In SC'14 workshop on High Performance Technical Computing in Dynamic Languages, SC '14.

Replica Exchange in Swift/T

```
foreach i in [0:num_replicas-1] {
  TEMPERATURE[i] = min_temp * exp(log(max_temp/min_temp)*(itof(i)/itof(num_replicas-1)) );
  states[1][i], POTENTIAL[1][i] = run_t(ifile, i, 1, steps_per_run, TEMPERATURE[i], 300);
}
foreach f in [2:num_runs] {
  if ( f%%2 == 1 ) { sources[f][0] = 0; }
  if ( (num_replicas+f)%%2 == 1 ) { sources[f][num_replicas-1] = num_replicas-1; }
  foreach i in [f%%2+1:num_replicas-1:2] {
    BOLTZMAN = 0.001987191;
    dbeta = ((1.0/TEMPERATURE[i-1]) - (1.0/TEMPERATURE[i])) / BOLTZMAN;
    float delta = dbeta * (POTENTIAL[f-1][i] - POTENTIAL[f-1][i-1]);
    boolean doswap = (delta < 0.0) || (exp(-delta) > random());
    printf("frame %d reps %d %d swap %s\n", f, i-1, i, doswap);
    if ( doswap ) { sources[f][i] = i-1; sources[f][i-1] = i; } else { sources[f][i] = i; sources[f][i-1] = i-1; }
  }
  foreach i in [0:num_replicas-1] {
    int isrc = sources[f][i];
    states[f][i], POTENTIAL[f][i] = run_t(states[f-1][isrc], i, f, steps_per_run, TEMPERATURE[i], TEMPERATURE[isrc]);
  }
}
```

NAMD Swift/T Limitations

- Unfamiliar language and semantics
- Requires dedicated server process
- Based on MPI – limits SMP performance
- Not designed for large data objects
  - Prototype writes and reads files (slow)
- Designed for a different class of problems
  - Maybe we can build something simpler

In-Memory Checkpointing

- checkpointStore *key* ?*replica* or global?
- checkpointLoad *key* ?*replica* or global?
- checkpointSwap *key* ?*replica* or global?
- checkpointFree *key* ?*replica* or global?

- *One-sided* – remote simulation continues running
  - Patch-level storage and communcation
- Useful for single-copy protocols as well
  - Extension of existing “checkpoint” and “revert”

Workflow-Enabling Commands

- replicaEval *replica script*
  - Evaluate *script* in remote partition's interpreter
  - *One-sided* – remote simulation continues running
  - Returns result
- replicaYield ?*seconds*?
  - Could be used to break up polling loops
  - In practice use (blocking) replicaRecv to wait instead
- replicaDcdFile *index*|off ?*filename*?
  - Redirect trajectory output

Simple Work Queue in Tcl

```
if { ! [myReplica] } {
…
 proc enqueue_work_0 work {
   if [workers_idle] {
     replicaSend $work [pop_worker]
   } else {
     push_work $work
   }
 }
  proc dependent_set_0 {var val} {
   upvar #0 $var v
   if { [info exists v] } {
     error "dependency variable $var set twice: old value $v, new value $val"
   }
   set v $val
  }
  proc dependent_work_0 {known future} {
   if { $future == {} } {
     enqueue_work_0 $known
     return
   }
   …
   uplevel #0 [list trace add variable $dname write [
                                               list dependent_trace_0 $known $future]]
  }
…
```

```
proc enqueue_work work {
  replicaEval 0 [list enqueue_work_0 $work]
}

proc dependent_work {known future} {
  replicaEval 0 [list dependent_work_0 $known $future]
}

proc dependent_set {var val} {
  replicaEval 0 [list dependent_set_0 $var $val]
}

proc schedule_work {} {
  while { 1 } {
    set w [replicaEval 0 "dequeue_work_0 [myReplica]"]
    if { $w == {} } {
      set w [replicaRecv 0]
    }
    eval $w
  }
}

replicaBarrier
schedule_work
```

Multiplexed Replica Exchange

```
proc run_replica { dict rescale } {
  global num_runs steps_per_run …
  dict with dict {
    firsttimestep $i_step
    if { $checkpointname != {} } {
      checkpointLoad $checkpointname $checkpointloc
      checkpointFree $checkpointname $checkpointloc
    } else { # start or restart … }
    if { $rescale != 1.0 } { rescalevels $rescale }
    if { $i_run >= $num_runs } { # exit … }
    langevinTemp $temperature
    replicaDcdFile $dcdindex
    run $steps_per_run
    incr i_run;  incr i_step $steps_per_run
    set checkpointname $i_run.$index;  set checkpointloc [myReplica]
    checkpointStore $checkpointname $checkpointloc
    replica_puts $history_file "$i_step $index $temperature $TEMP $POT"
    if { $i_run % 2 == 1 } {set iswap $index_a } else { set iswap $index_b }
  }
  if { $index < $iswap } {
    dependent_work [list swap_replica $dict] [list dict.$i_run.$iswap]
  } elseif { $index > $iswap } {
    dependent_set dict.$i_run.$index $dict
  } else {
    enqueue_work [list run_replica $dict 1.0]
  }
}
```

```
proc swap_replica {self swap} {
  set temp [dict get $self temperature]; set rid [dict get $self index]
  set temp2 [dict get $swap temperature]; set rid2 [dict get $swap index]
  set BOLTZMAN 0.001987191
  set dbeta [expr ((1.0/$temp) - (1.0/$temp2)) / $BOLTZMAN]
  set pot [dict get $self POT]; set pot2 [dict get $swap POT]
  set delta [expr $dbeta * ($pot2 - $pot)]
  set doswap [expr $delta < 0. || exp(-1. * $delta) > rand()]
  if { $doswap } {
    set i_run [dict get $self i_run]
    puts stderr ”ACCEPT $rid ($temp) $rid2 ($temp2) RUN $i_run"
    set rescale [expr sqrt(1.0*$temp/$temp2)]
    set rescale2 [expr sqrt(1.0*$temp2/$temp)]
    dict_swap self swap checkpointname checkpointloc history_file dcdindex
  } else {
    set rescale 1.0; set rescale2 1.0
  }
  dict with self {
    incr exchanges_attempted
    if { $doswap } { incr exchanges_accepted }
  }
  enqueue_work [list run_replica $swap $rescale2]
  run_replica $self $rescale
}
```

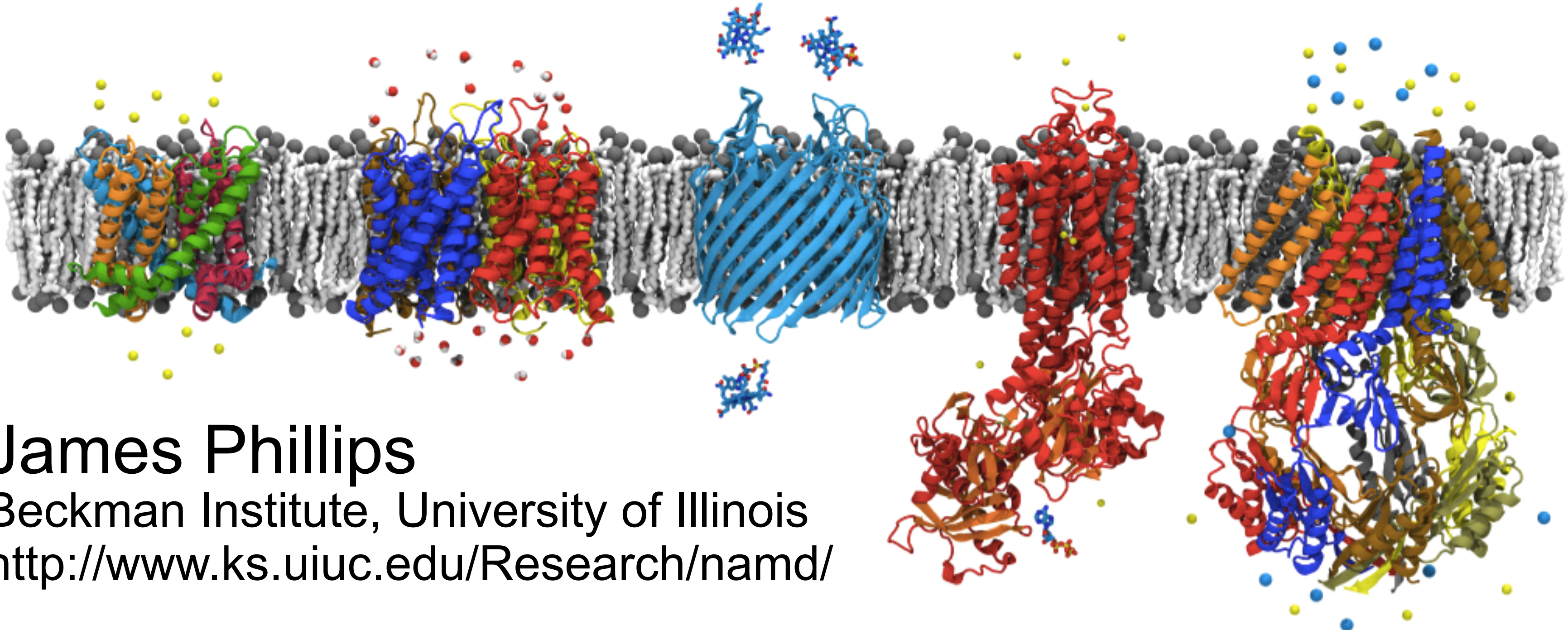
Conclusion: Multi-Copy Capabilities Evolve

- New: Workflow algorithms
  - Multiple in-memory checkpoints with inter-replica access
  - Global work queue, dependency-driven execution
  - First use: **Milestoning** (Lane Votapka, Amaro Lab, UCSD)

- Improved: Replica exchange simulations
  - Enhanced Tcl-based scripting of collective variables
  - **Multiplexing replicas** on smaller number of parallel partitions

- Available in NAMD 2.11
  - Released December 22, 2015

Thanks to: NIH, NSF, DOE, NCSA, ALCF, OLCF,
Nikhil Jain, Wei Jiang, Lei Huang, Mikolai Fajer, Yilin Meng, James Gumbart,
Yun Luo, Benoit Roux, Timothy G. Armstrong, Justin M. Wozniak, Michael Wilde,
and 20 years of NAMD and Charm++ developers and users.



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