

Charm++ Applications

Laxmikant (Sanjay) Kale

<http://charm.cs.illinois.edu>

Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana Champaign

Applied Modeling & Simulation (AMS) Seminar Series
NASA Ames Research Center, July 14, 2014



Exascale Challenges

- Main challenge: variability
 - Static/dynamic
 - Heterogeneity: processor types, process variation, ..
 - Power/Temperature/Energy
 - Component failure
- Exacerbated by strong scaling needs from apps
 - Why?
- To deal with these, we must seek
 - Not full automation
 - Not full burden on app-developers
 - But: a good division of labor between the system and app developers

My Mantra

OM

I call it a mantra because I will repeat it a lot in this talk.
And its going to be my message to App Developers on
how to get ready for Adaptive Runtimes

a

My Mantra

OM

My Mantra

Oh...Maybe the order
doesn't matter

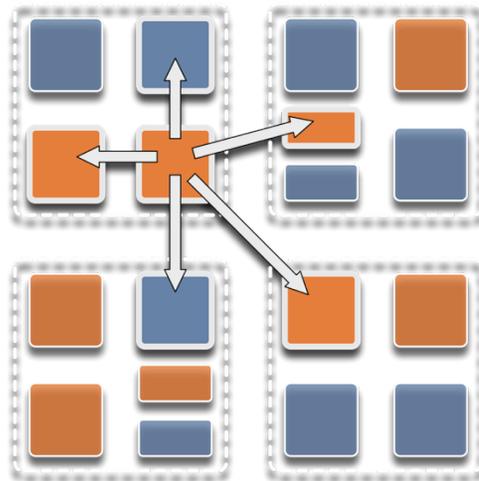
OMa

My Mantra

*ver decomposition
OaM
synchrony
igratability*

Overdecomposition

- Decompose the work units & data units into many more pieces than execution units
 - Cores/Nodes/..
- Not so hard: we do decomposition anyway



Migratability

- Allow these work and data units to be migratable at runtime
 - i.e. the programmer or runtime, can move them
- Consequences for the app-developer
 - Communication must now be addressed to logical units with global names, not to physical processors
 - But this is a good thing
- Consequences for RTS
 - Must keep track of where each unit is
 - Naming and location management

Asynchrony:

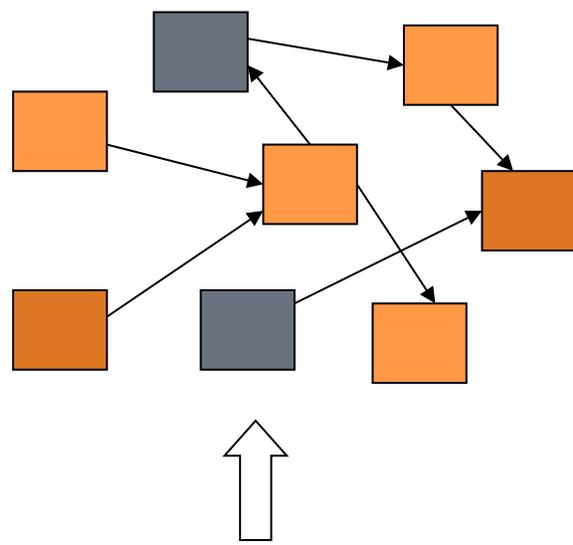
- Now: **Message-Driven Execution**
 - You have multiple units on each processor
 - They address each other via logical names
- Need for scheduling:
 - What sequence should the work units execute in?
 - One answer: let the programmer sequence them
 - Seen in current codes, e.g. some AMR frameworks
 - Message-driven execution:
 - Let the work-unit that happens to have data (“message”) available for it execute next
 - Let the RTS select among ready work units
 - Programmer should not specify what executes next, but can influence it via priorities

Realization of this model in Charm++

- Overdecomposed entities: chares
 - Chares are C++ objects
 - With methods designated as “entry” methods
 - Which can be invoked asynchronously by remote chares
 - Chares are organized into indexed collections
 - Each collection may have its own indexing scheme
 - 1D, ..7D,
 - Sparse
 - Bitvector or string as an index
 - Chares communicate via asynchronous method invocations
 - `A[i].foo(...)`; A is the name of a collection, i is the index of the particular chare.

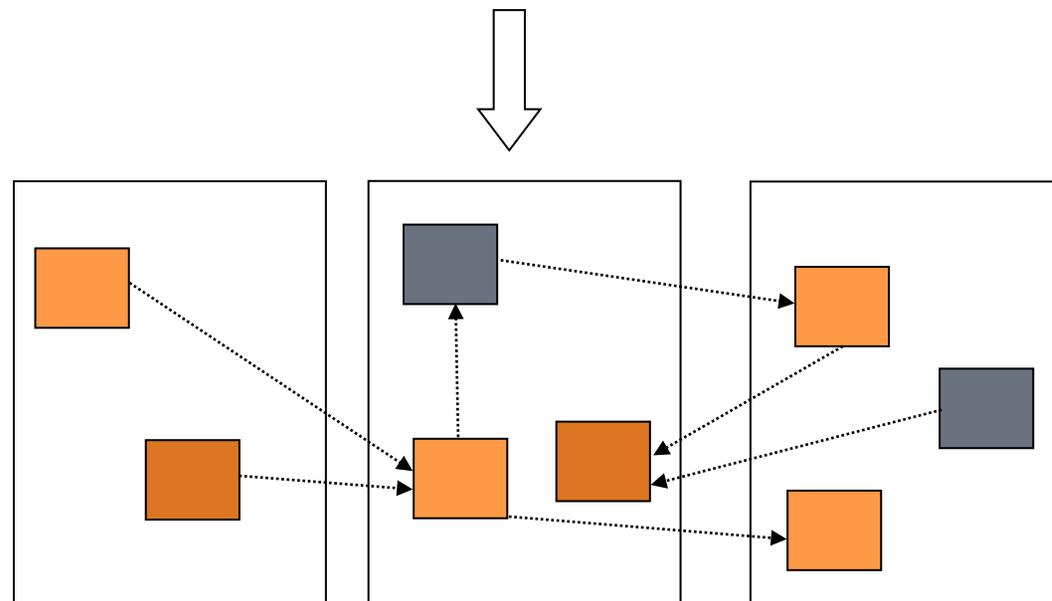
Charm++: Object-based overdecomposition

- Multiple “indexed collections” of C++ objects
- Indices can be multi-dimensional and/or sparse
- Programmer expresses communication between objects
 - with no reference to processors : `A[i].foo(...)`

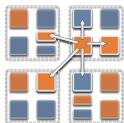


User View

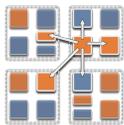
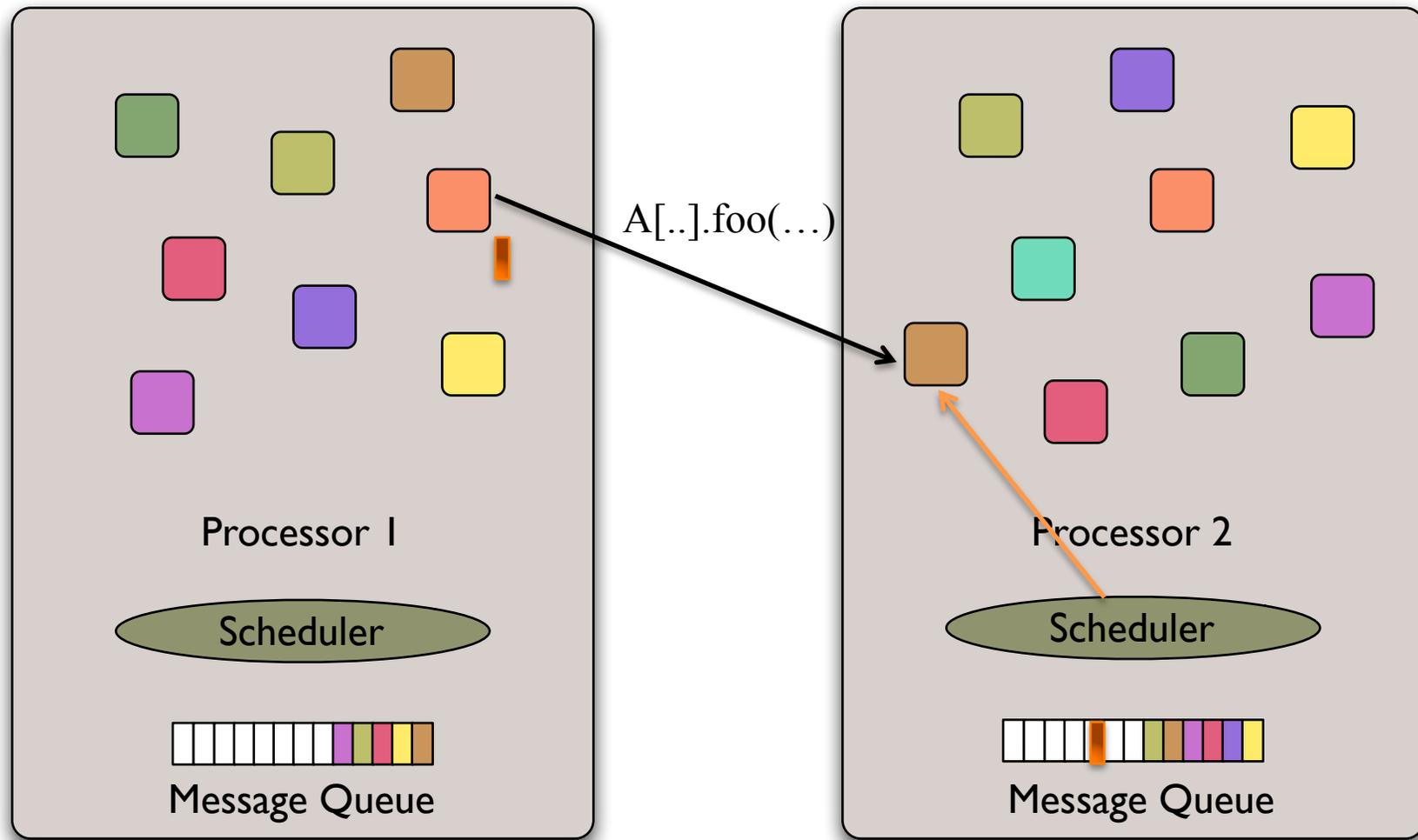
System implementation



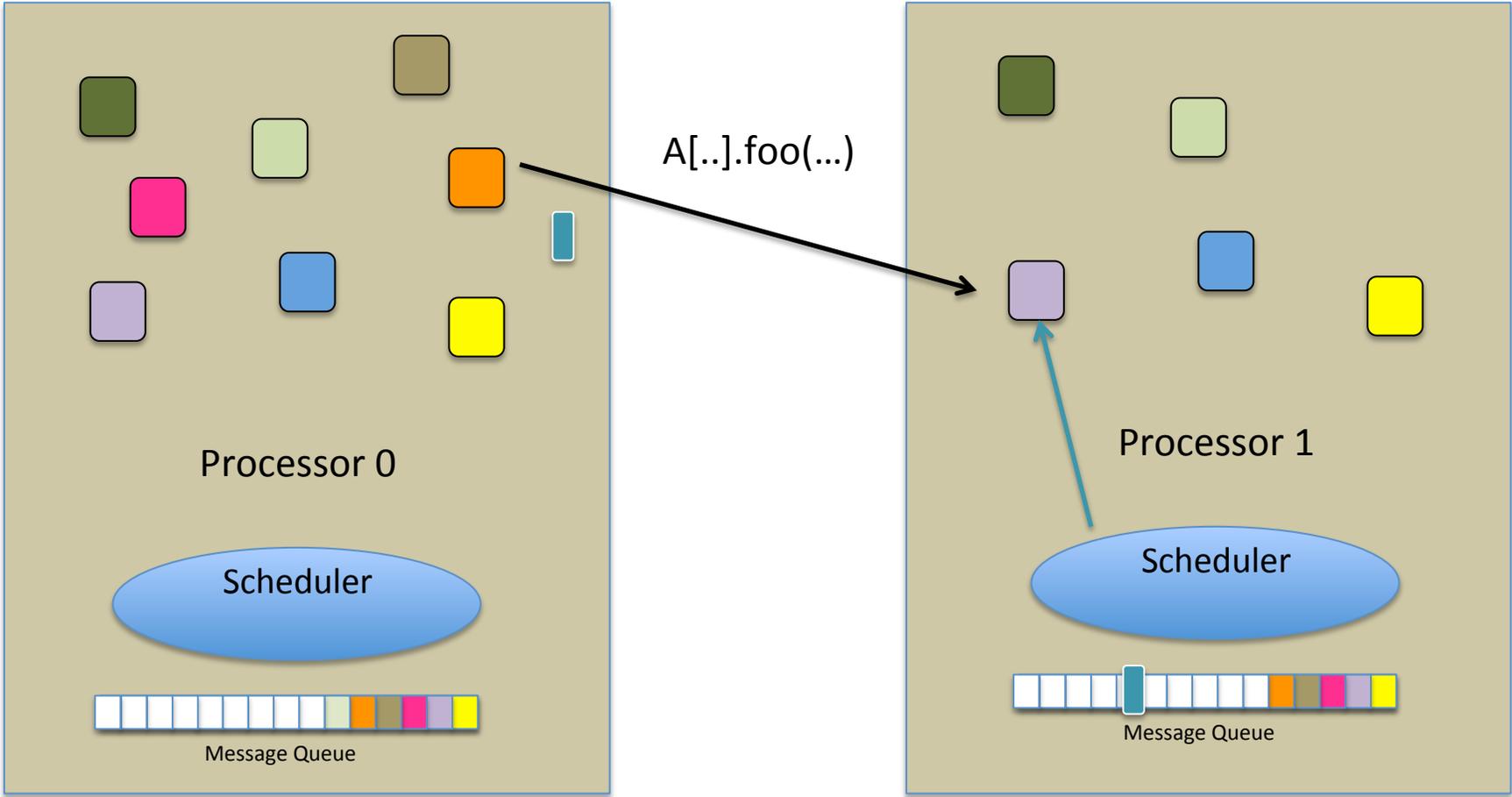
SICM2

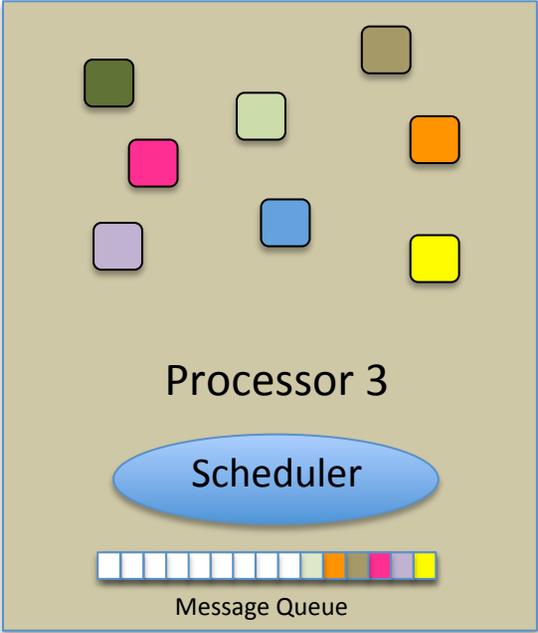
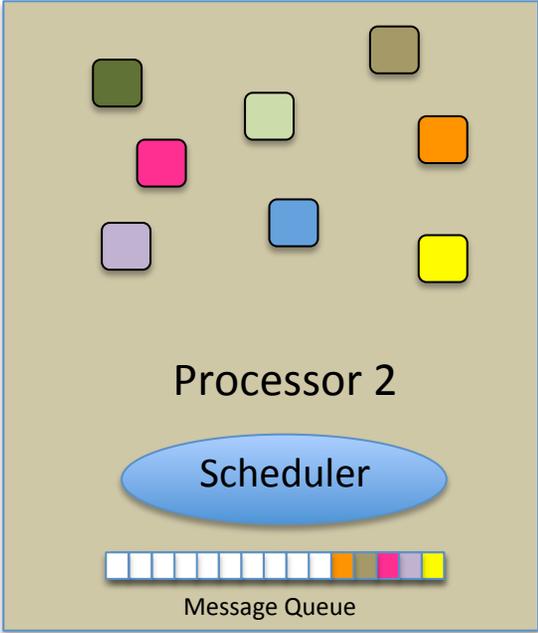
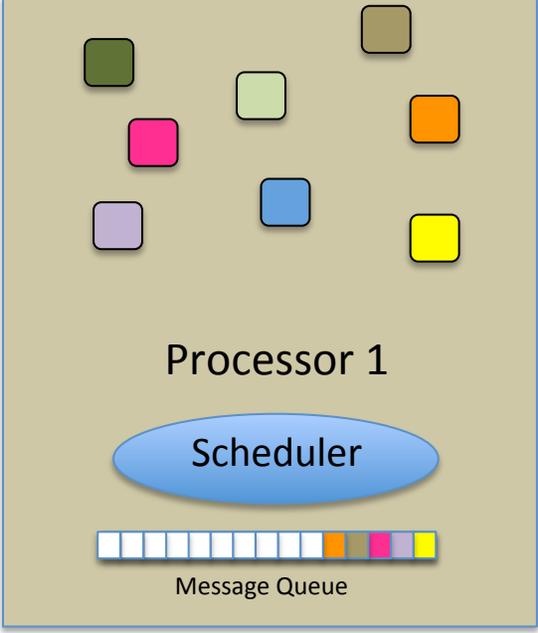
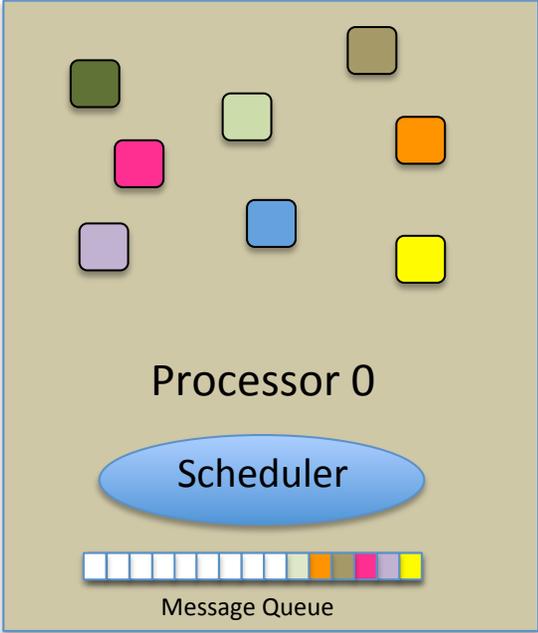


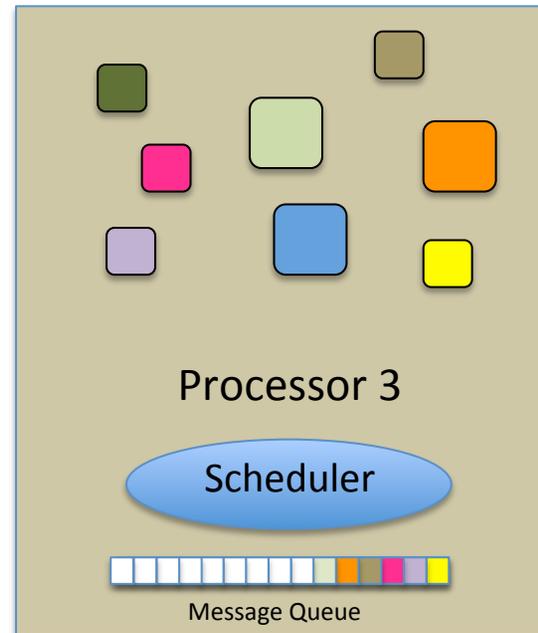
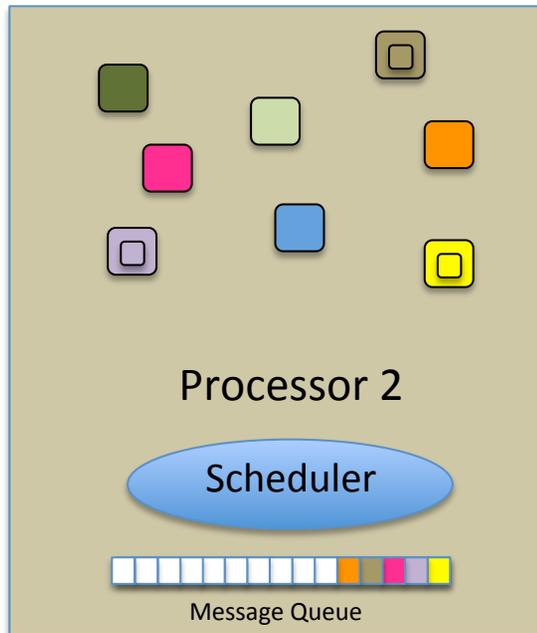
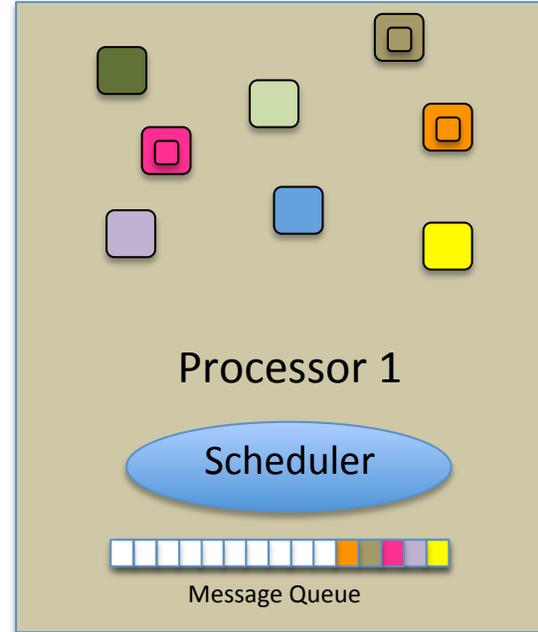
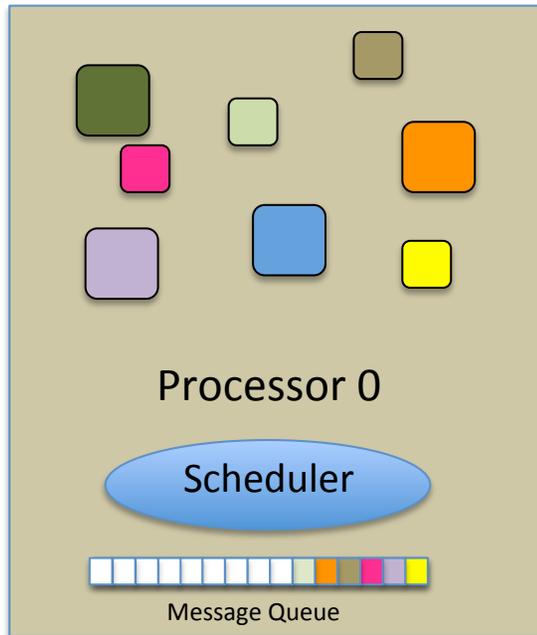
Message-driven Execution

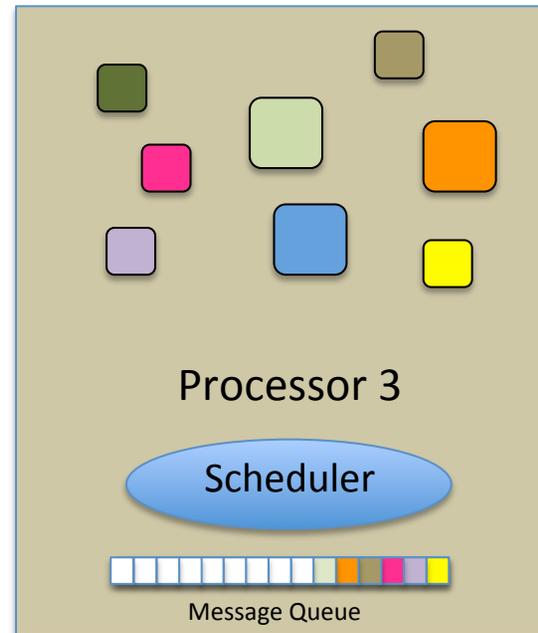
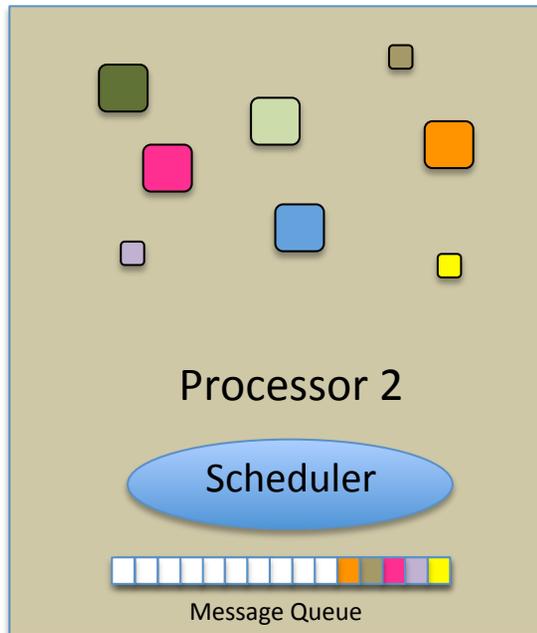
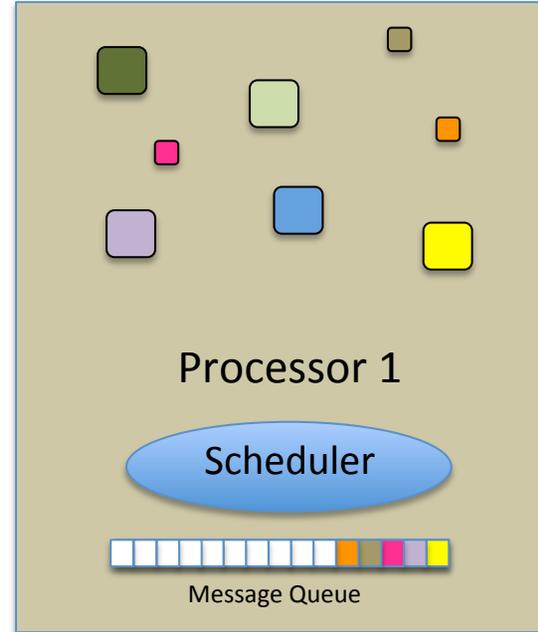
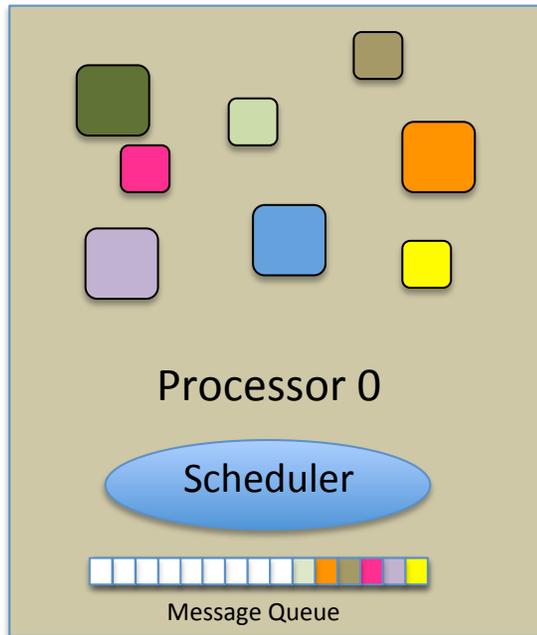


Message-driven Execution

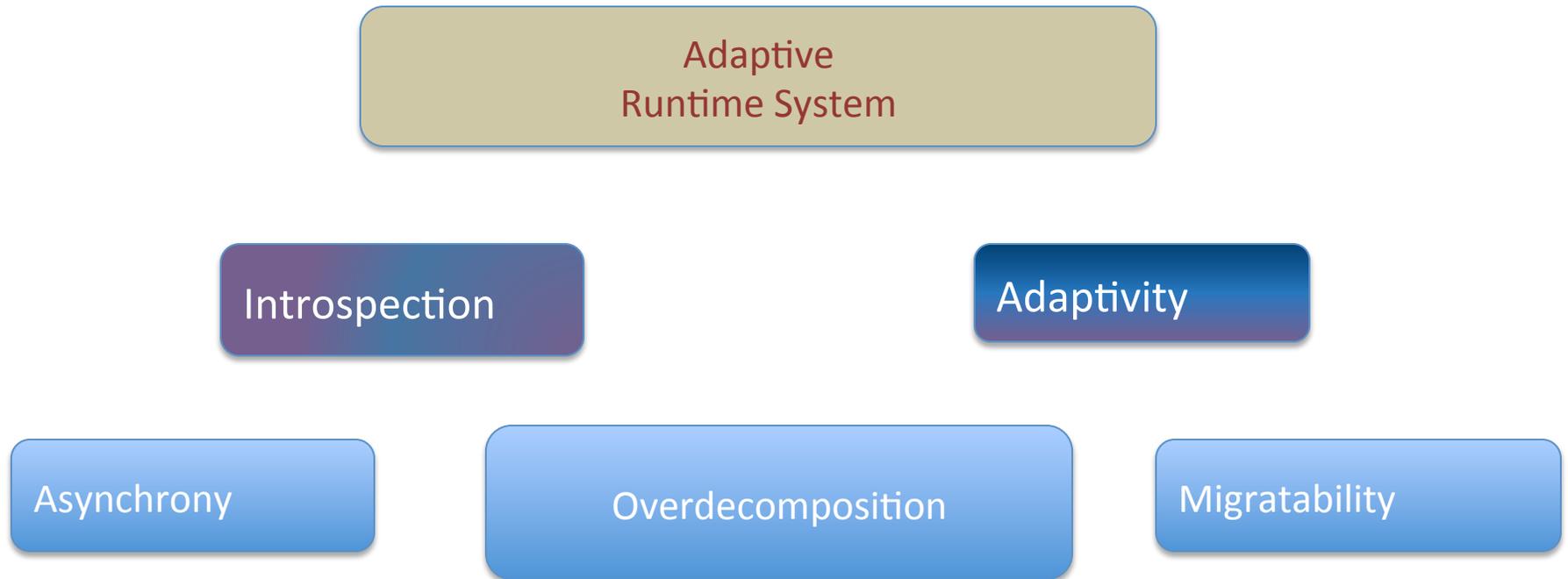








Empowering the RTS

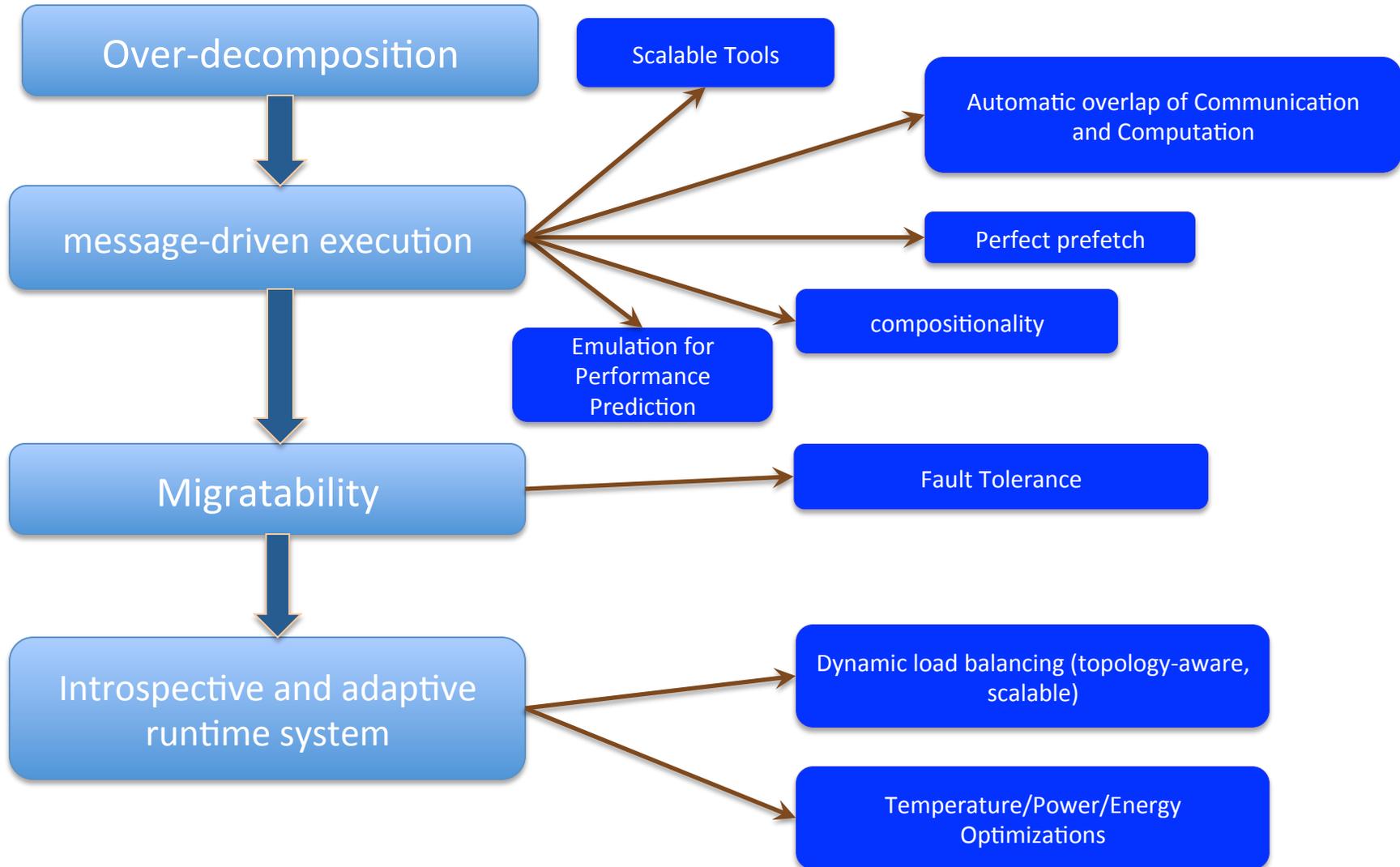


- The Adaptive RTS can:
 - Dynamically balance loads
 - Optimize communication:
 - Spread over time, async collectives
 - Automatic latency tolerance
 - Prefetch data with almost perfect predictability

Adaptive Runtime Systems

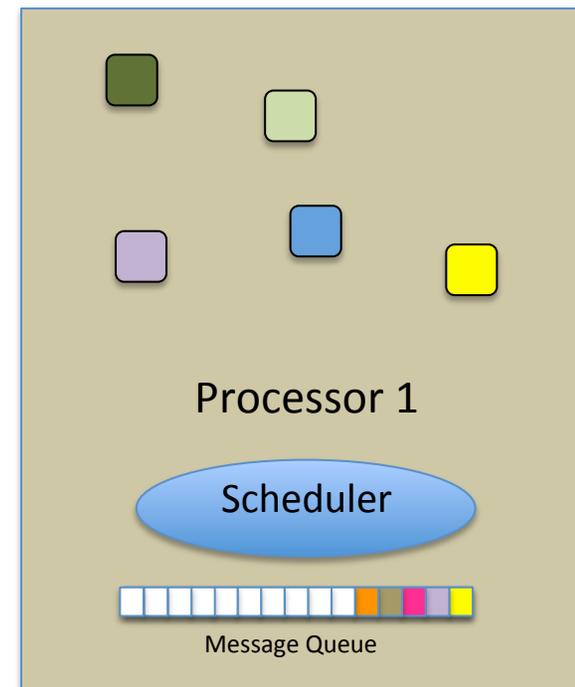
- Decomposing program into a large number of Objects empowers the RTS, which can:
 - Migrate Objects at will
 - Schedule tasks (Dependent Execution Blocks) at will
 - Instrument computation and communication at the level of these logical units
 - Object A communicates y bytes to B every iteration
 - Sequential Block S has a high cache miss ratio
 - Maintain historical data to track changes in application behavior
 - Historical => previous iterations
 - E.g., to trigger load balancing

Benefits in Charm++



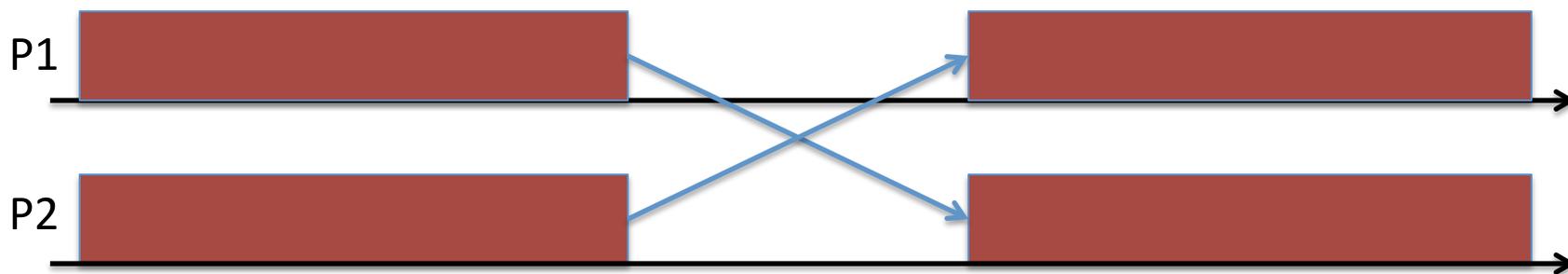
Utility for Multi-cores, Many-cores, Accelerators:

- Objects connote and promote locality
- Message-driven execution
 - A strong principle of prediction for data and code use
 - Much stronger than principle of locality
 - Can use to scale memory wall:
 - Prefetching of needed data:
 - into scratch pad memories, for example



Impact on communication

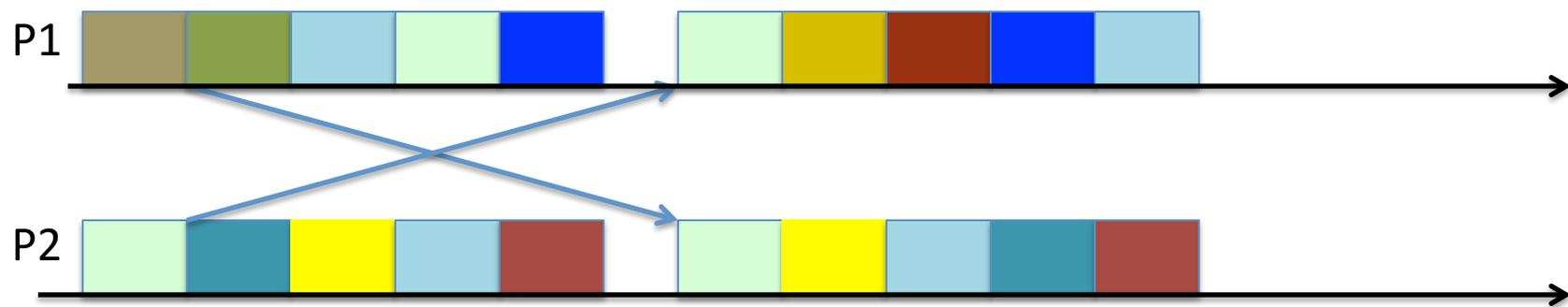
- Current use of communication network:
 - Compute-communicate cycles in typical MPI apps
 - So, the network is used for a fraction of time,
 - and is on the critical path
- So, current *communication networks are over-engineered for by necessity*



BSP based application

Impact on communication

- With overdecomposition
 - Communication is spread over an iteration
 - Also, adaptive overlap of communication and computation



Overdecomposition enables overlap

Empowering the RTS

Adaptive
Runtime System

Introspection

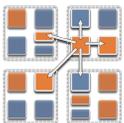
Adaptivity

Asynchrony

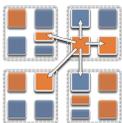
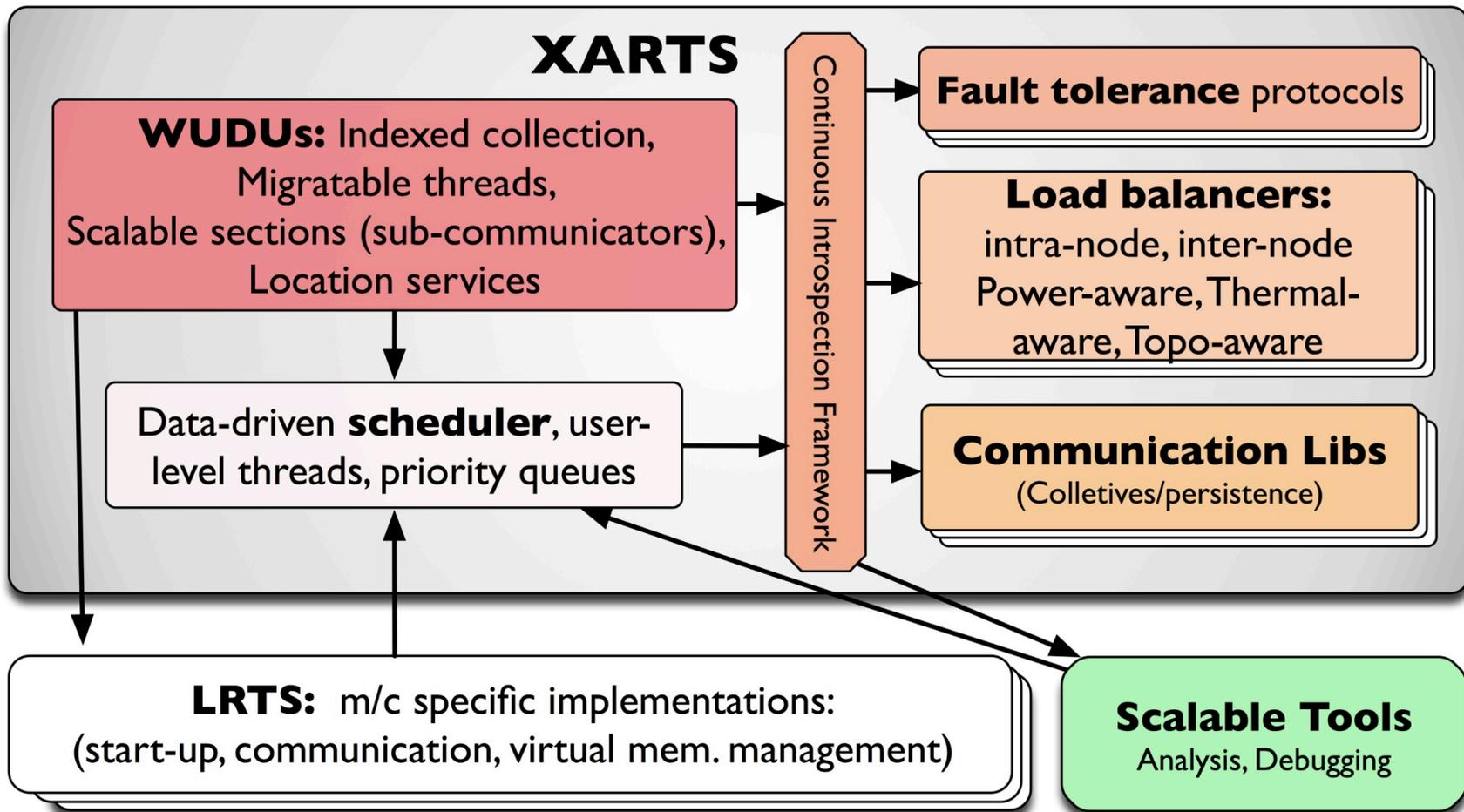
Overdecomposition

Migratability

- The Adaptive RTS can:
 - Dynamically balance loads
 - Optimize communication:
 - Spread over time, async collectives
 - Automatic latency tolerance
 - Prefetch data with almost perfect predictability



Charm++ RTS



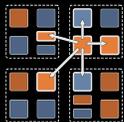
ChaNGa: Parallel Gravity

- Collaborative project (NSF)
 - with Tom Quinn, Univ. of Washington
- Gravity, gas dynamics
- Barnes–Hut tree codes
 - Oct tree is natural decomp
 - Geometry has better aspect ratios, so you “open” up fewer nodes
 - But is not used because it leads to bad load balance
 - Assumption: one-to-one map between sub-trees and PEs
 - Binary trees are considered better load balanced

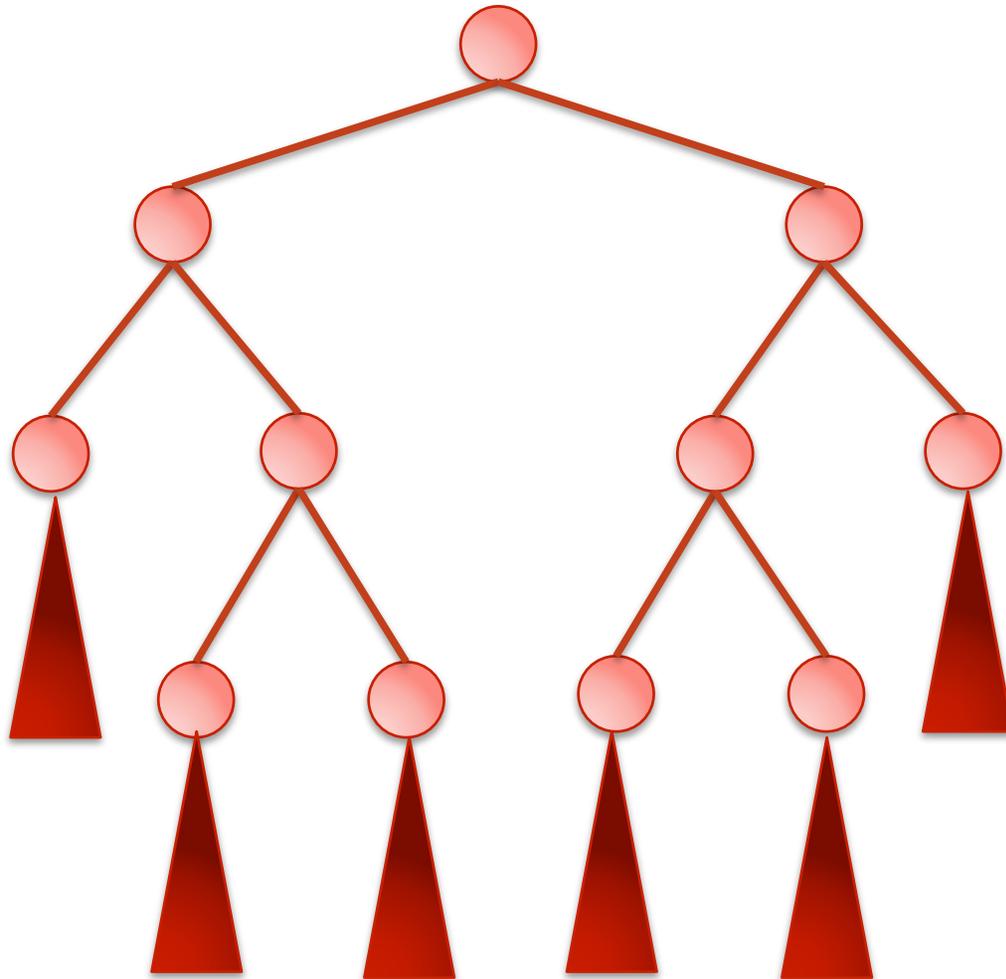
Evolution of Universe and Galaxy Formation



With Charm++: Use Oct-Tree, and let Charm++ map subtrees to processors

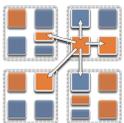


ChaNGa: Cosmology Simulation



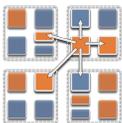
Collaboration with
Tom Quinn UW

- Tree: Represents particle distribution
- TreePiece: object/chares containing particles

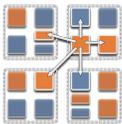
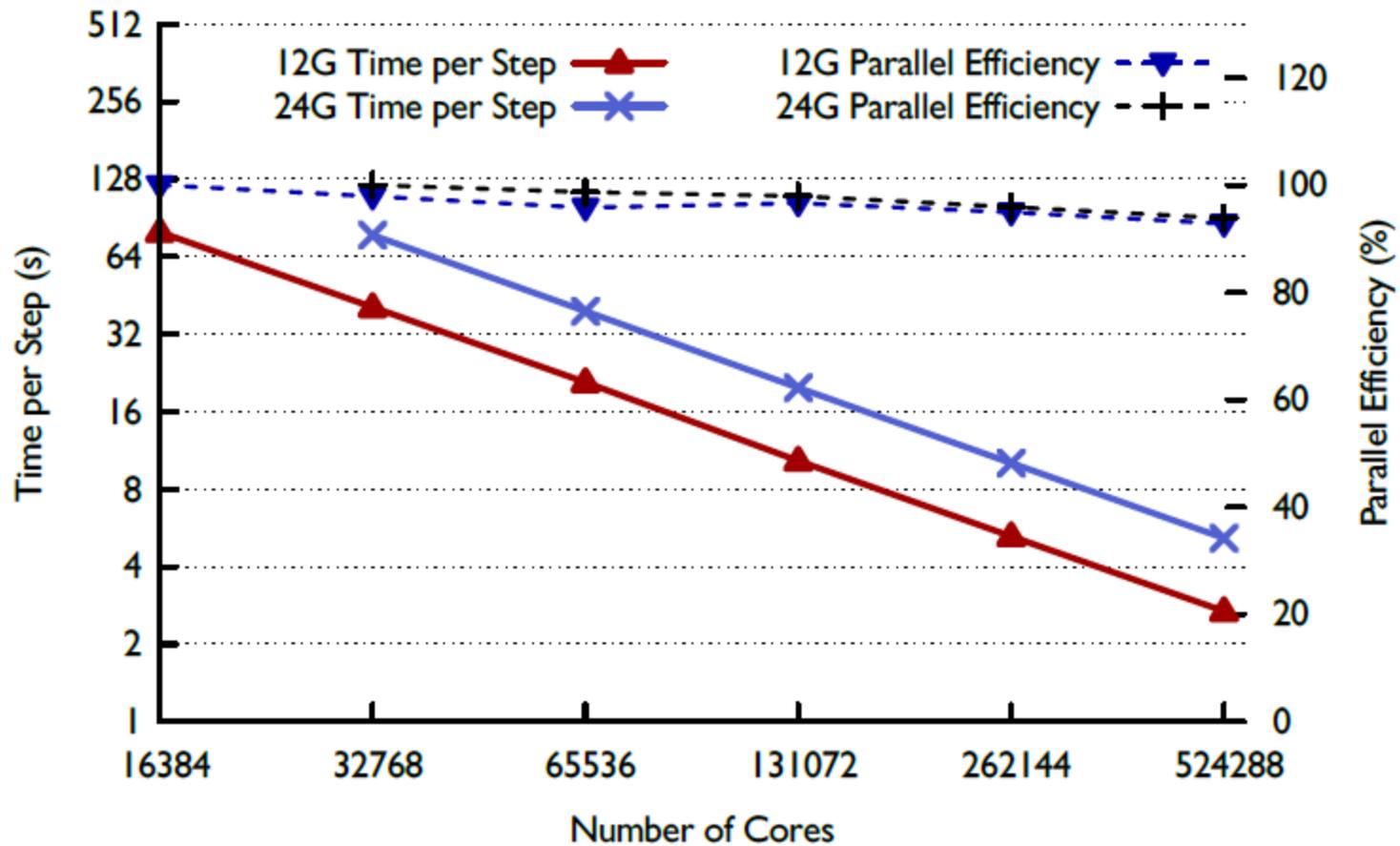


ChaNGa: Optimized Performance

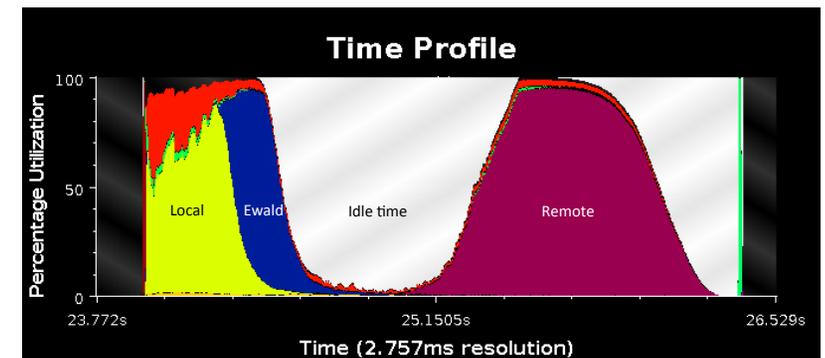
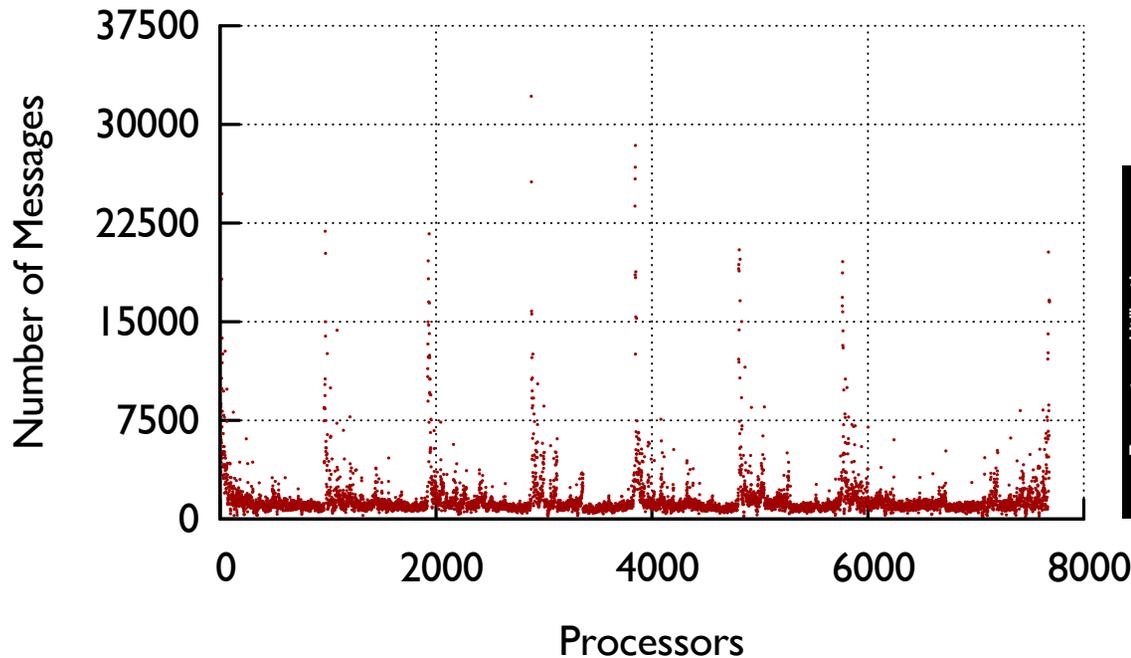
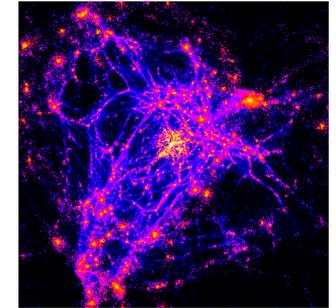
- Asynchronous, highly overlapped, phases
- Requests for remote data overlapped with local computations



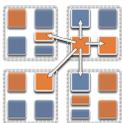
ChaNGa : a recent result



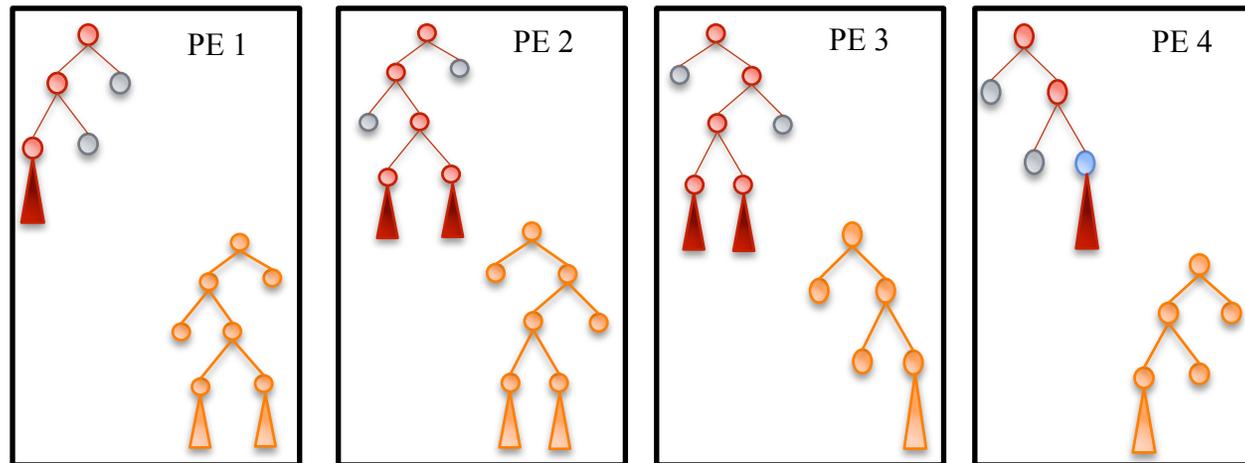
Clustered Dataset – Dwarf



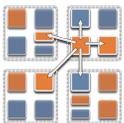
- Highly clustered
- Maximum request per processor: $> 30K$
- Idle time due to message delays
- Also, load imbalances: solved by Hierarchical balancers



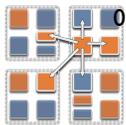
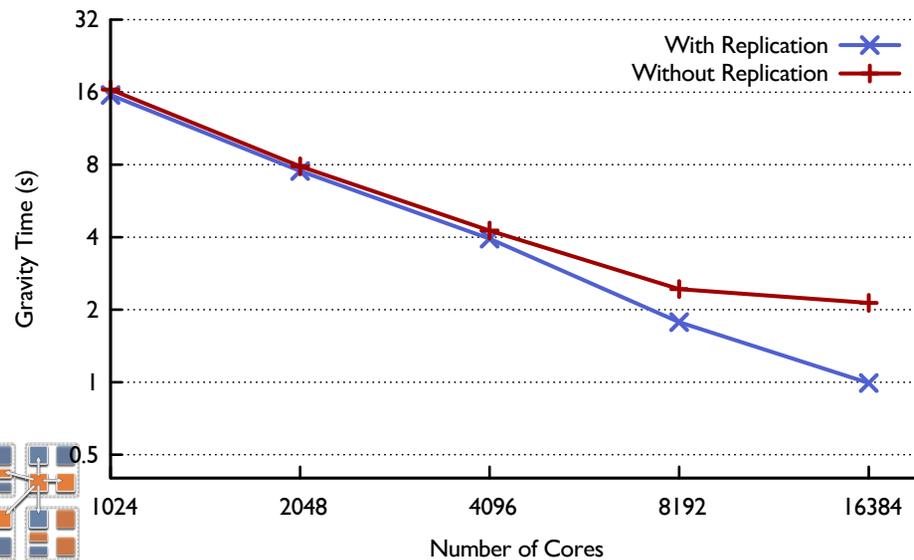
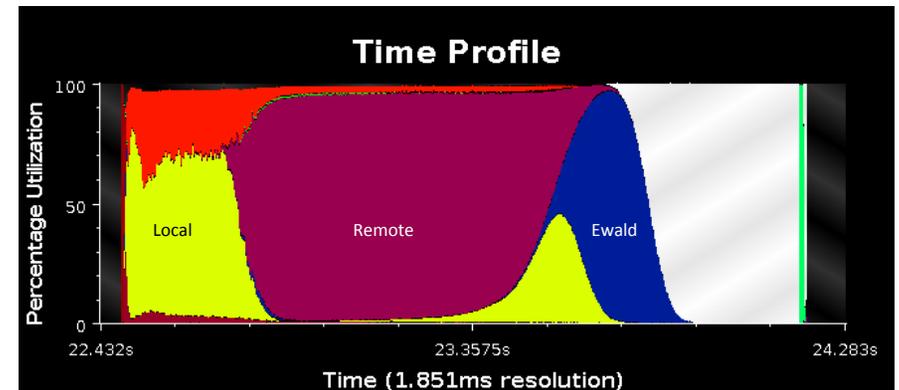
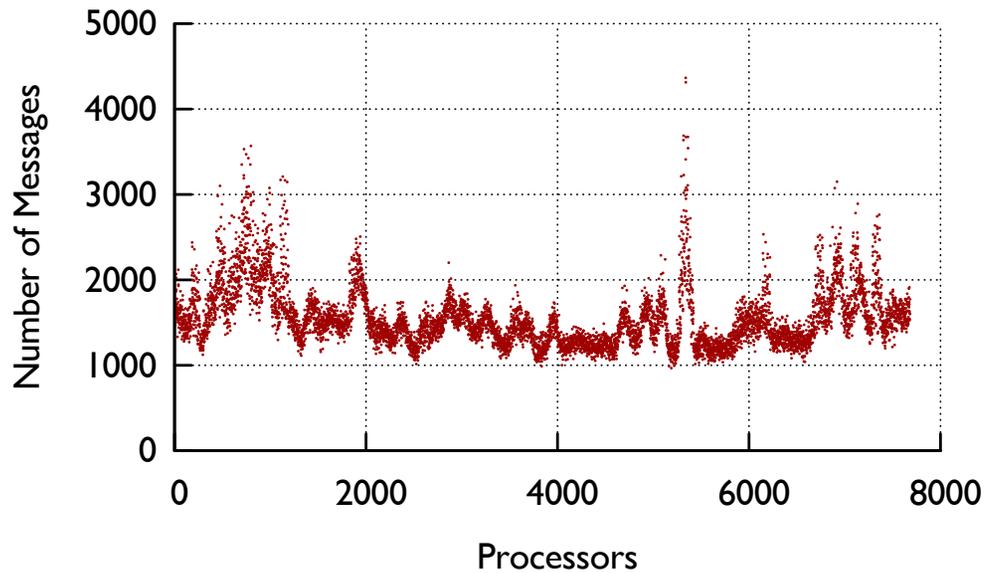
Solution: Replication



- Replicate tree nodes to distribute requests
- Requester randomly selects a replica



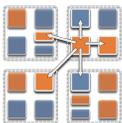
Replication Impact



- Replication distributes requests
- Maximum request reduced from 30K to 4.5K
- Gravity time reduced from 2.4 s to 1.7 s, on 8k

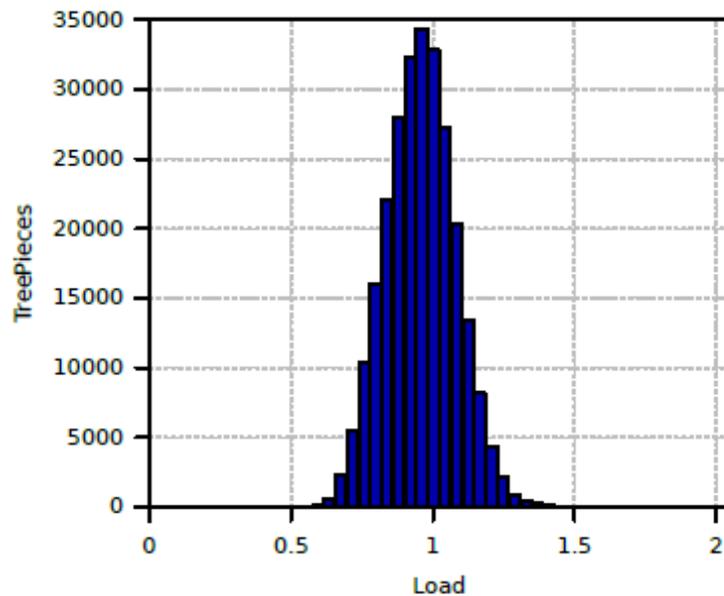
Multiple time-stepping!

- Our scientist collaborators suggest an algorithmic optimization:
 - Don't move slow-moving particles every step
 - i.e. don't calculate forces on them either
 - In fact, make many (say 5) categories (rungs) of particles based on their velocities
 - Rung sequence (with 5 rungs)
 - 4 3 4 2 4 3 4 1 4 3 4 2 4 3 4 0
 - Rung 0: all particles, Rung 4: fastest-moving particles
 - Each tree-piece object now presents a different load when different “rungs” are being calculated

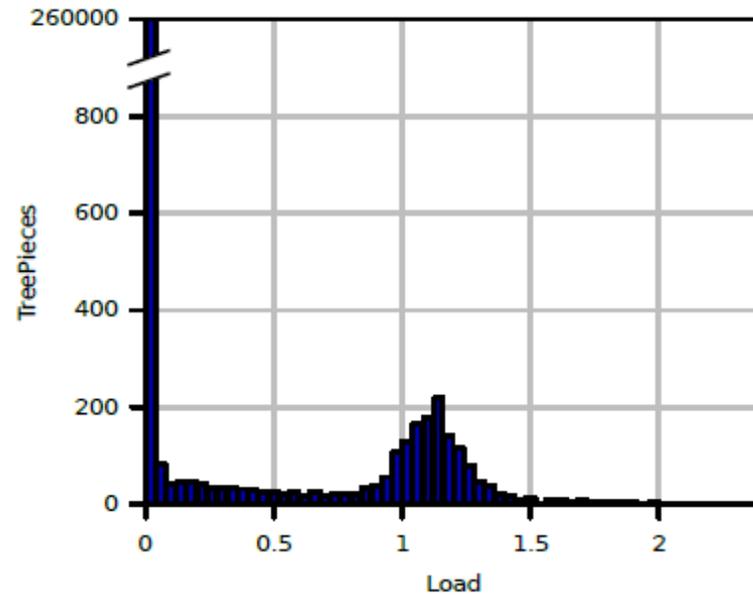


Multiple time-stepping!

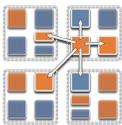
- Load (for the same object) changes across rungs
 - Yet, there is persistence within the same rung!
 - So, specialized phase-aware balancers were developed



(a) Rung 0

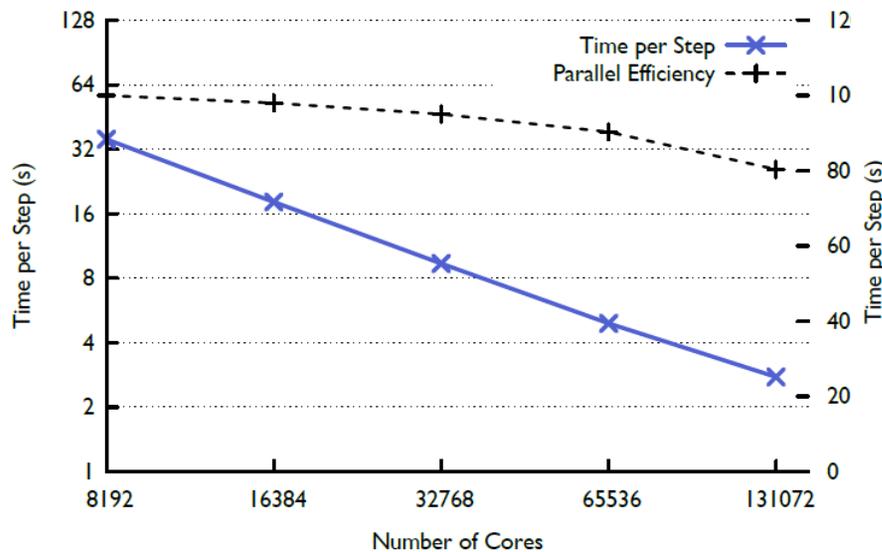


(b) Rung 4

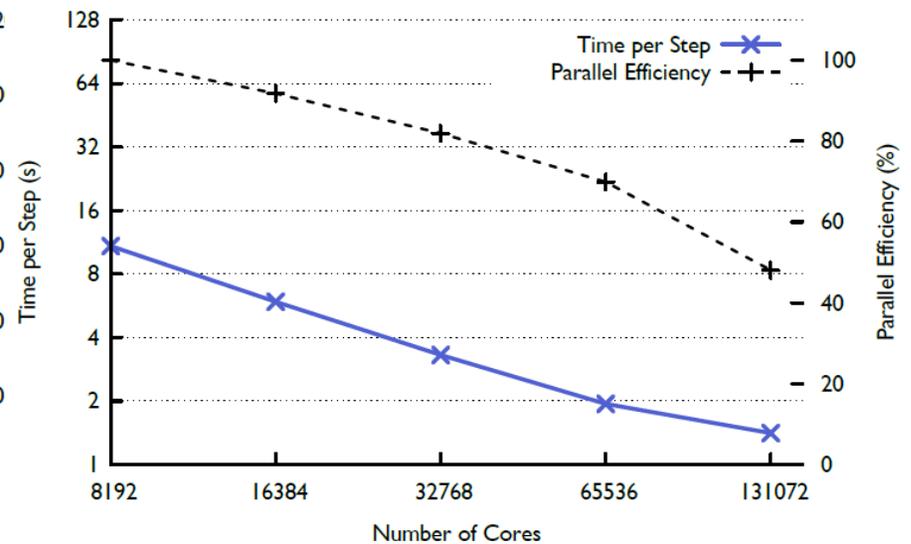


Multi-stepping tradeoff

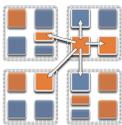
- Parallel efficiency is lower, but performance is improved significantly



Single Stepping

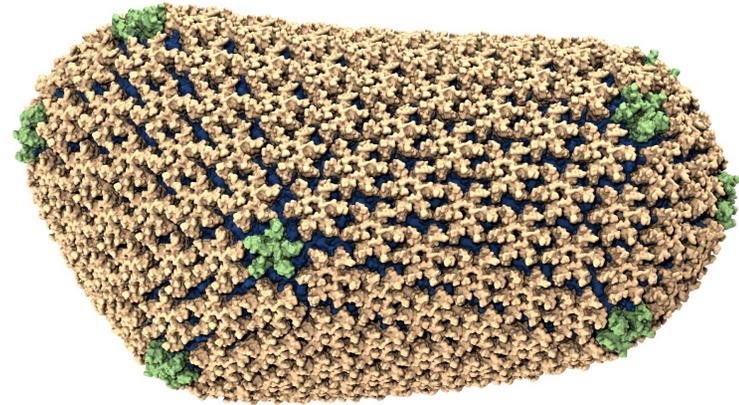


Multi Stepping

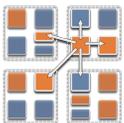


NAMD: Biomolecular Simulations

- Collaboration with K. Schulten
- With over 50,000 registered users
- Scaled to most top US supercomputers
- In production use on supercomputers and clusters and desktops
- Gordon Bell award in 2002

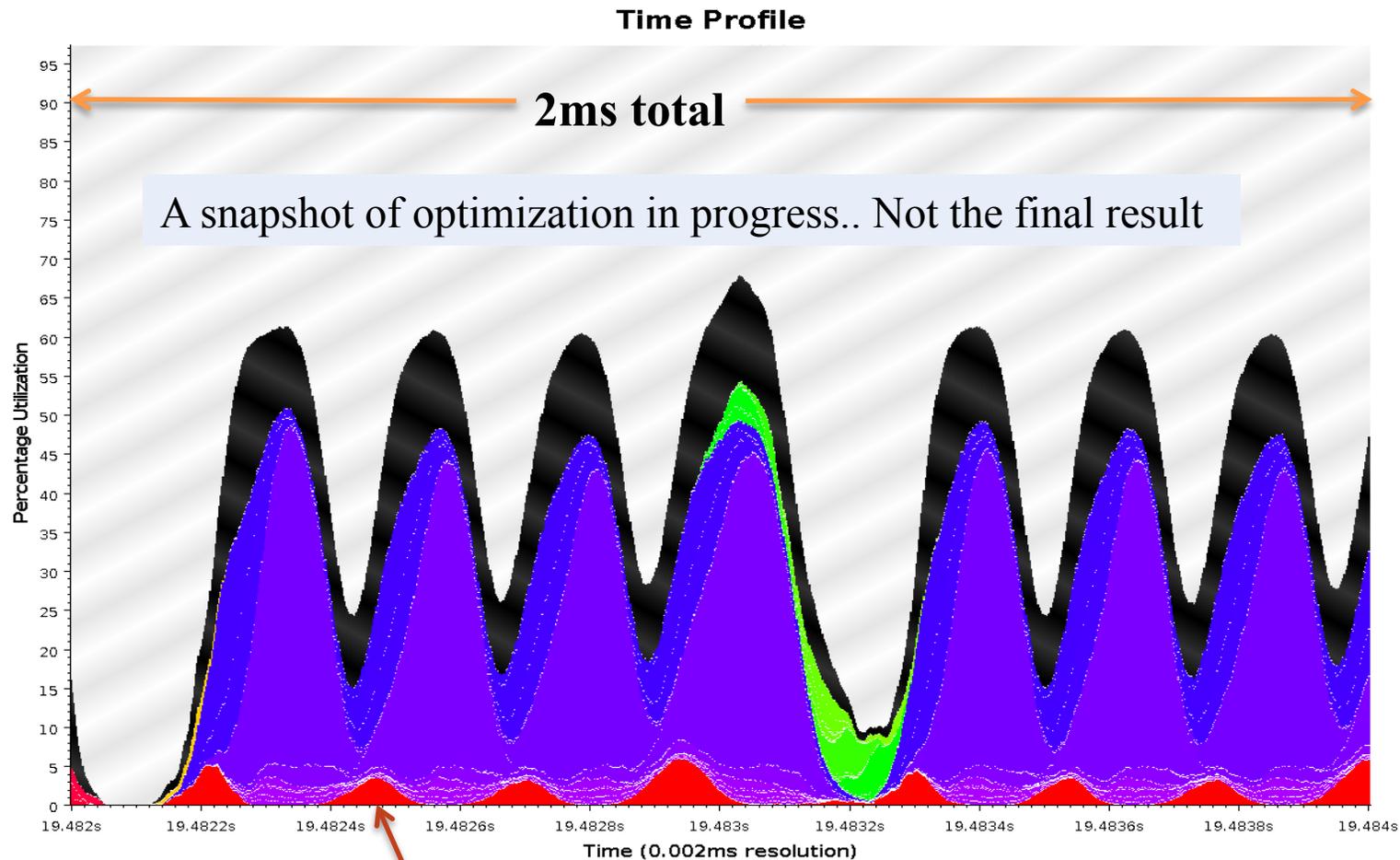


Recent success:
Determination of the
structure of HIV capsid
by researchers including
Prof Schulten

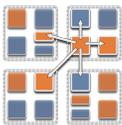


Time Profile of ApoA1 on Power7 PERCS

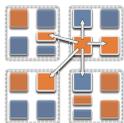
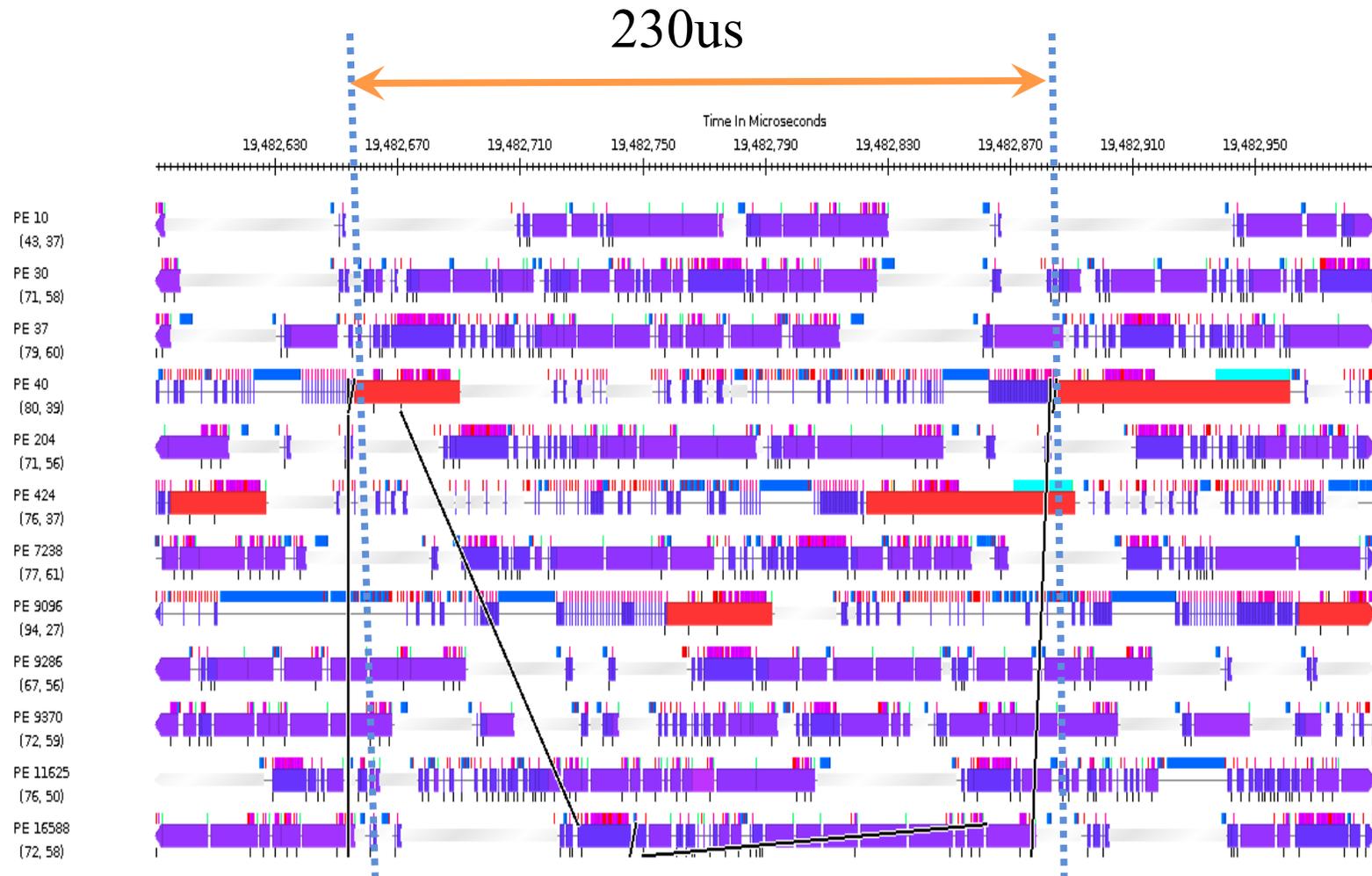
92,000 atom system, on 500+ nodes (16k cores)



Overlapped steps, as a result of asynchrony

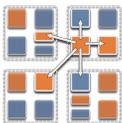
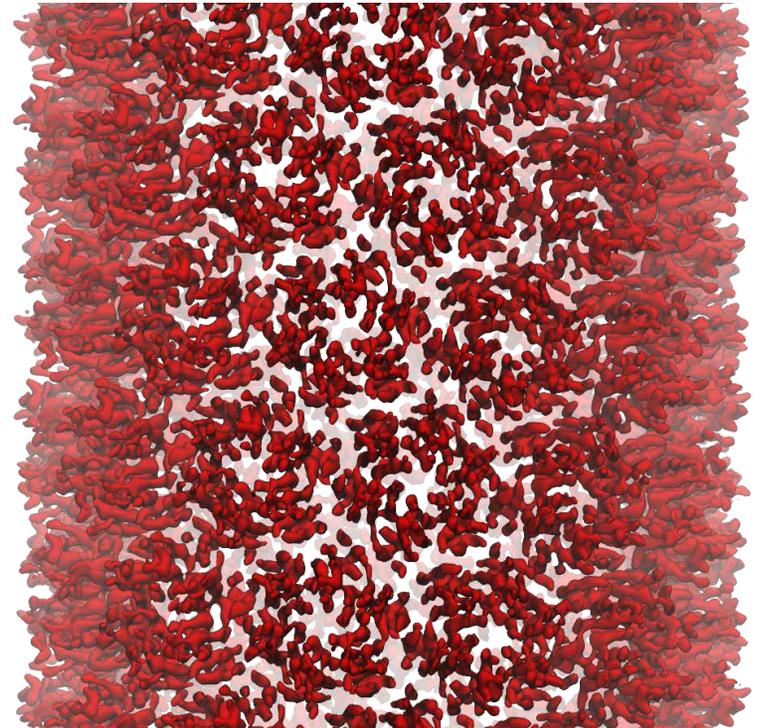


Timeline of ApoA1 on Power7 PERCS



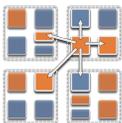
NAMD: Strong Scaling

- HIV Capsid was a 64 million atom simulation, including explicit water atoms
- Most biophysics systems of interests are 10M atoms or less... maybe 100M
- Strong scaling desired to billions of steps

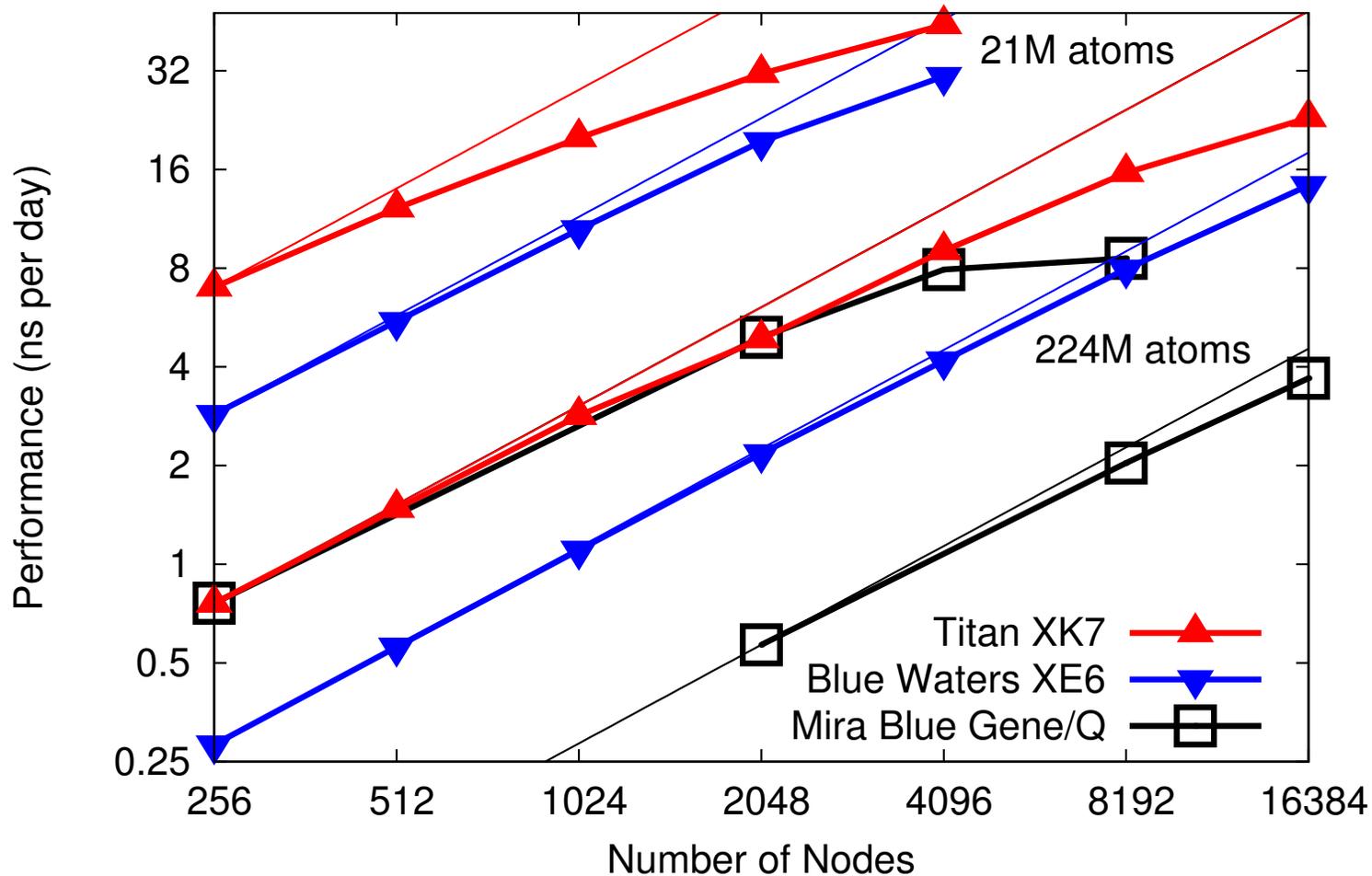


Enhancing Asynchrony in NAMD

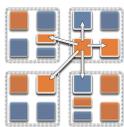
- Charm++ reductions are non-blocking
 - So, you *can* do other work while reduction is progressing through the system
- Synchronization:
 - NAMD, when used with a barostat (NPT ensemble), needs pressure from the current step to rescale volume
 - So, no other work was performed during reduction
- Enhancing asynchrony:
 - For strong scaling, the algebra was reworked to use the results of the reduction one step later
 - Overlapped reduction with an entire force computation step
 - 10% performance improvement on 16k nodes on Titan



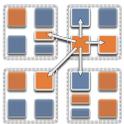
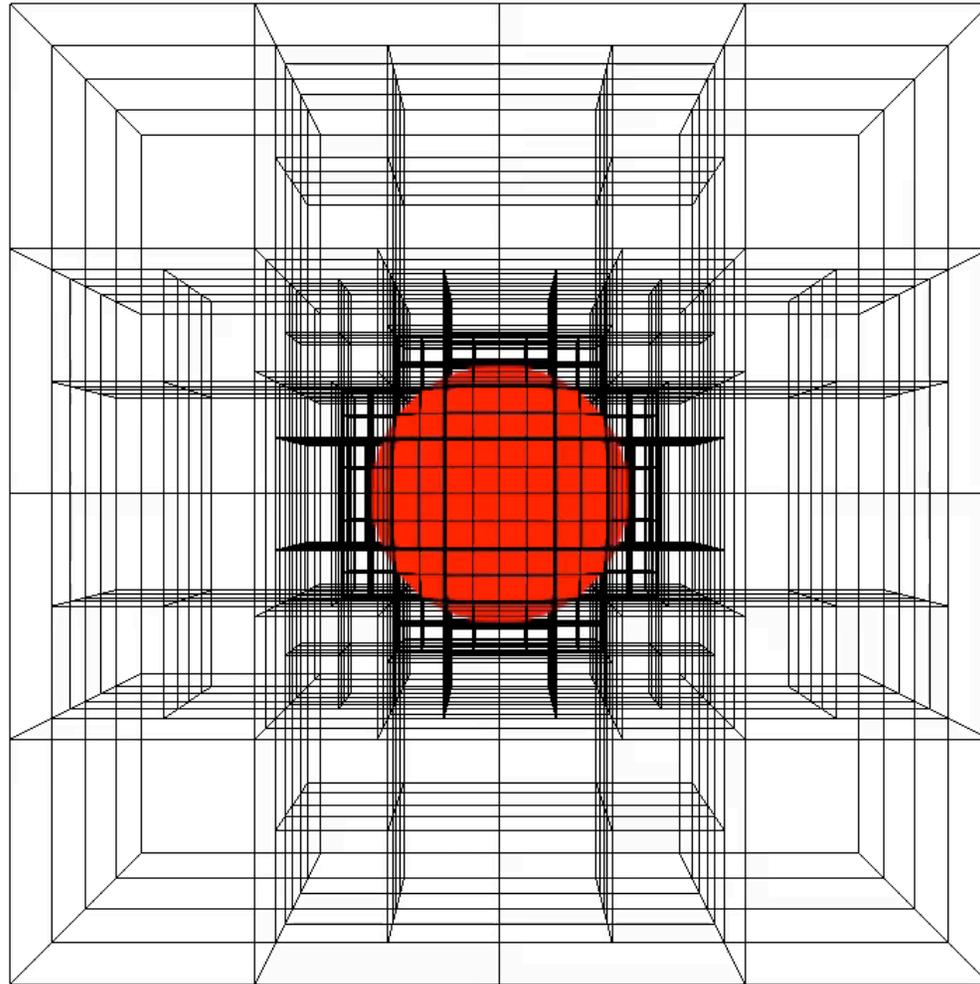
NAMD on Petascale Machines (2fs timestep with PME)



NAMD strong scaling on Titan Cray XK7, Blue Waters Cray XE6, and Mira IBM Blue Gene/Q for 21M and 224M atom benchmarks

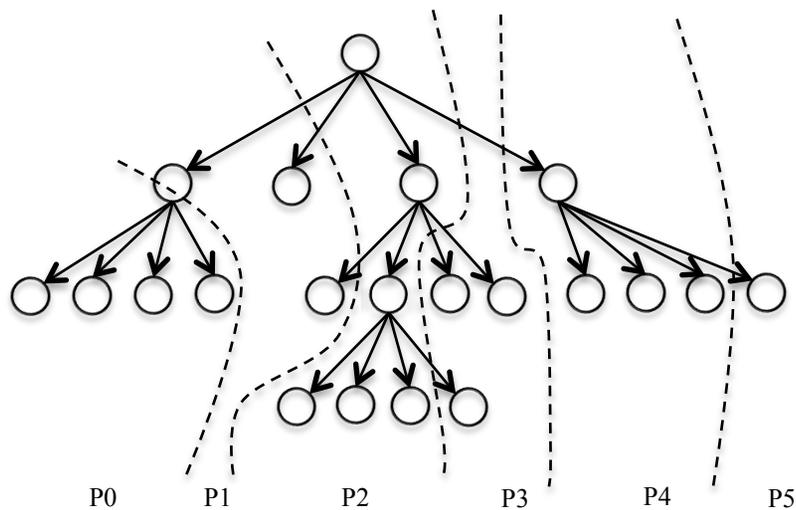


Structured AMR miniApp



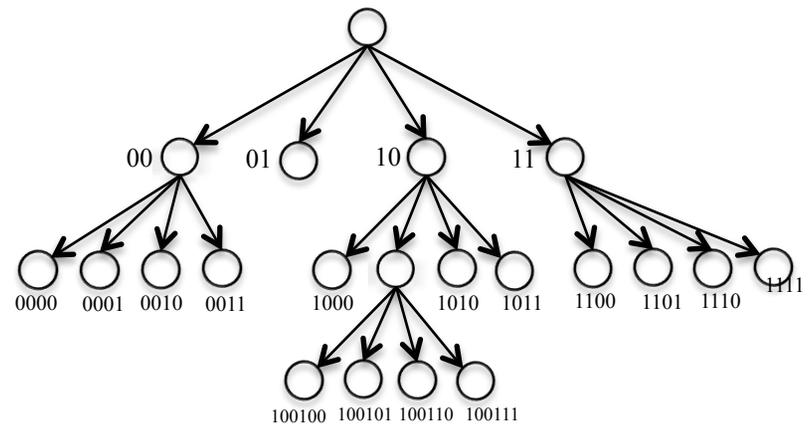
Structured AMR

Typical MPI Approach

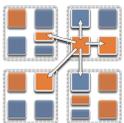


Process based
Contiguous blocks
assigned to a process

Charm++ Approach

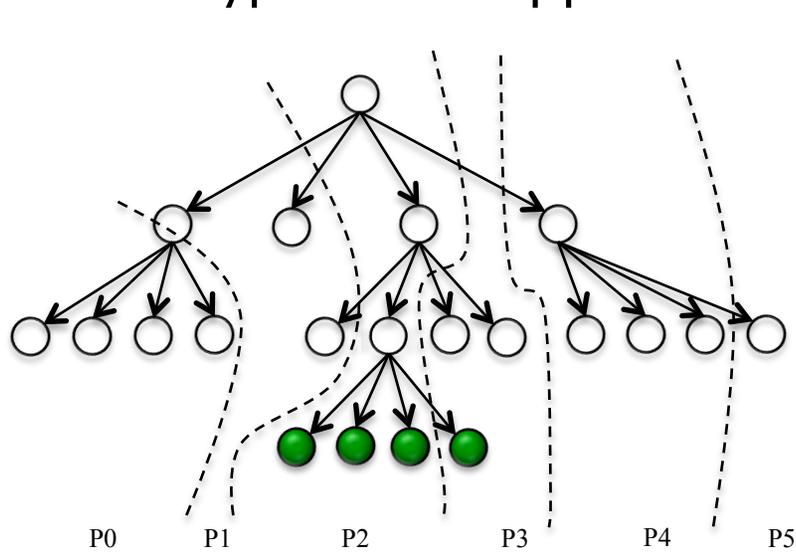


- Object based*
- Each block is an independent object
 - is the basic execution unit
 - can be mapped to any physical process
 - is uniquely addressable
 - is migratable

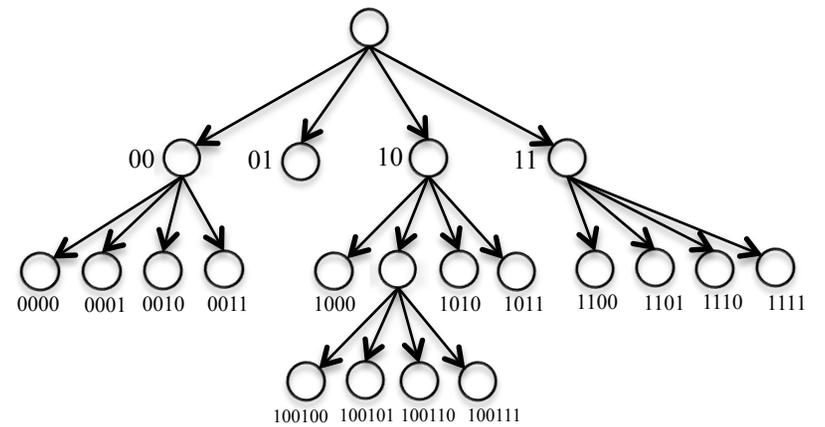


Structured AMR

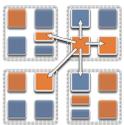
Typical MPI Approach



Charm++ Approach

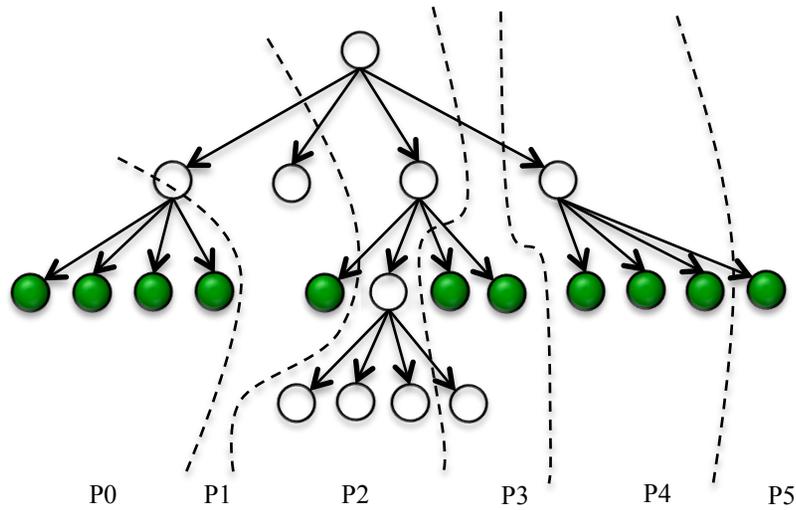


Mesh Restructuring

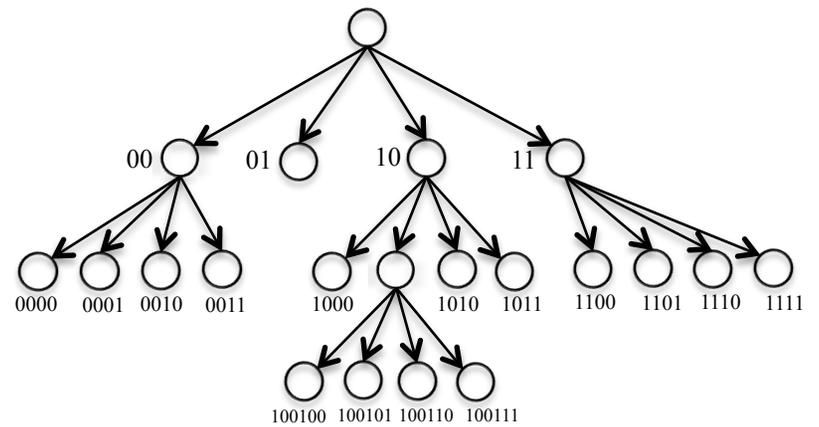


Structured AMR

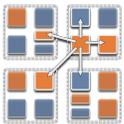
Typical MPI Approach



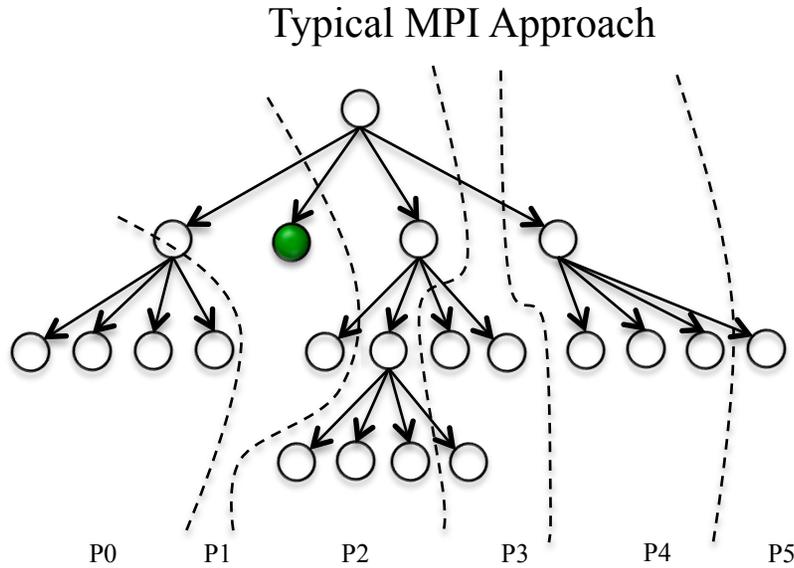
Charm++ Approach



Mesh Restructuring



Structured AMR

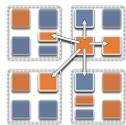


Mesh Restructuring

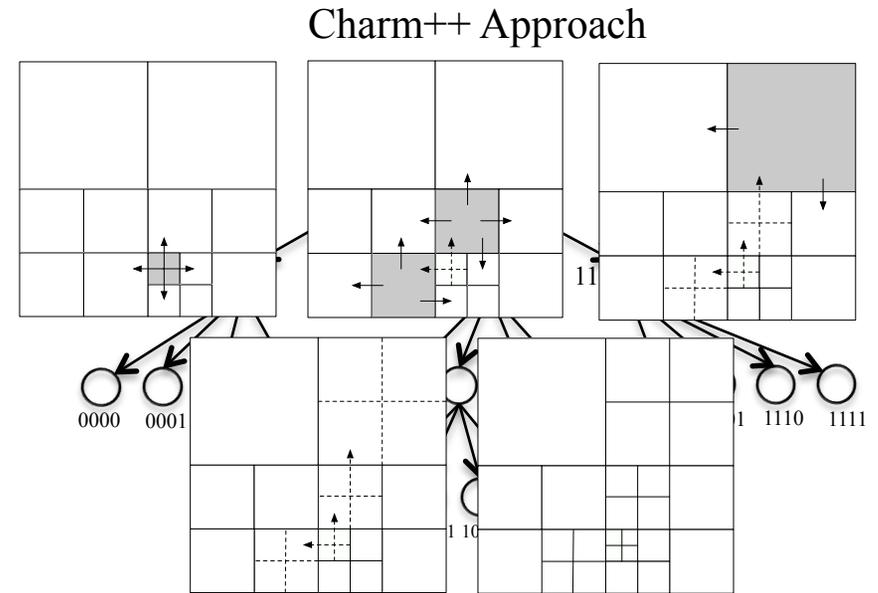
- Ripple Propagation Algorithm
 - Level-by-level
 - $O(d)$ global reductions $\approx O(d \cdot \log P)$
- Tree-replication on each process
 - $O(\#blocks)$ memory per process

Synchronization overhead

Memory overhead



d



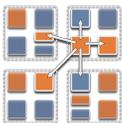
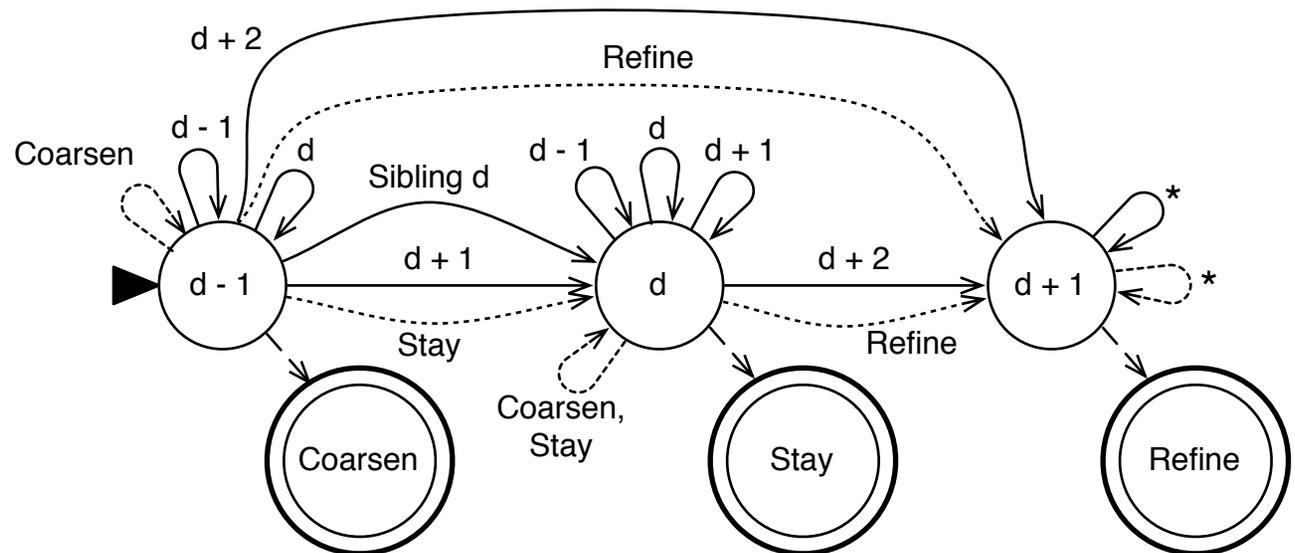
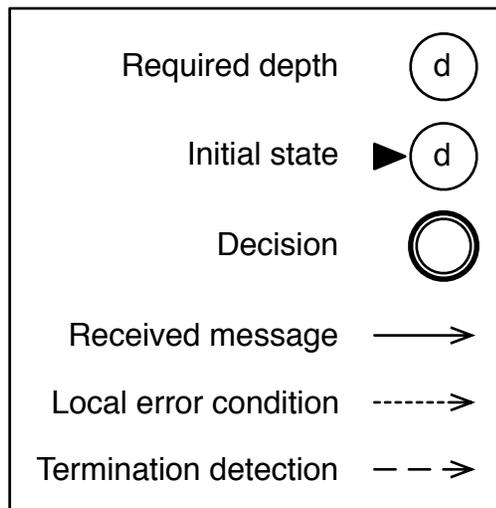
Mesh Restructuring

- Exchange messages with neighboring blocks
 - Update state using a state machine
 - Quiescence to detect global consensus
- Blocks save current level of neighbors
 - $O(\#blocks/P)$ memory per process

$O(\log P)$ time

$O(\#blocks/P)$ space

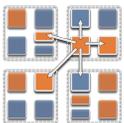
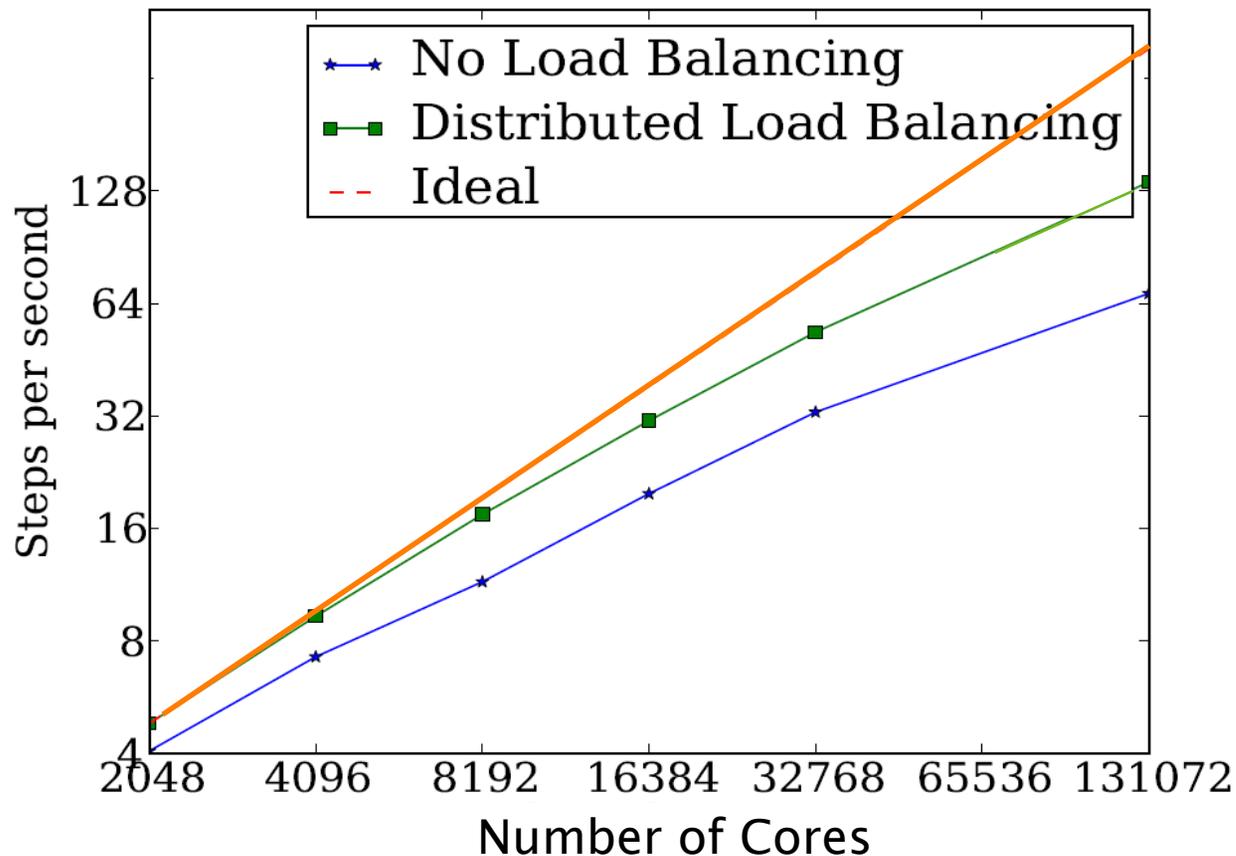
Structured AMR: State Machine



Structured AMR: Performance

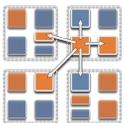
Testbed: IBM BG/Q Mira
Cray XK/6 Titan

Advection Benchmark
First order method in
3d-space

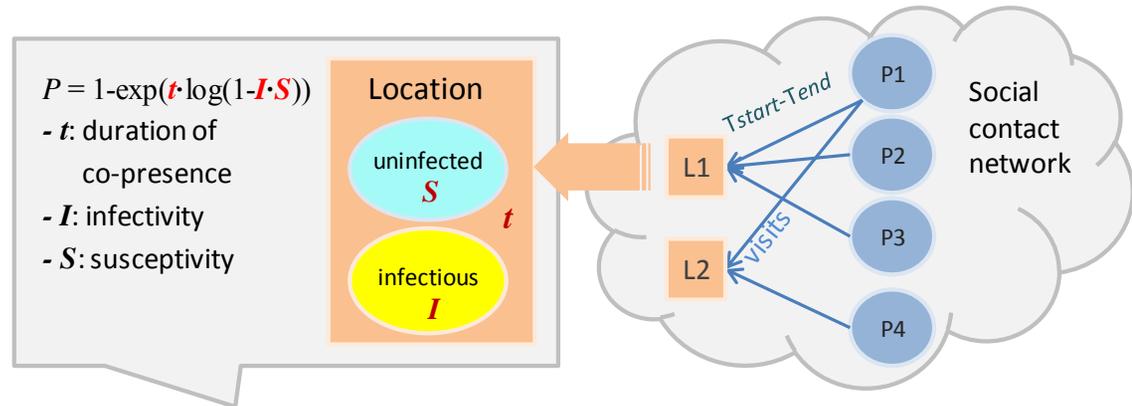
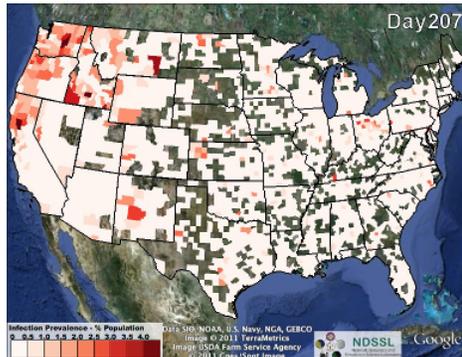


Episimdemics

- Simulation of spread of contagion
 - Code by Madhav Marathe, Keith Bisset, .. Vtech
 - Original was in MPI
- Converted to Charm++
 - Benefits: asynchronous reductions improved performance considerably

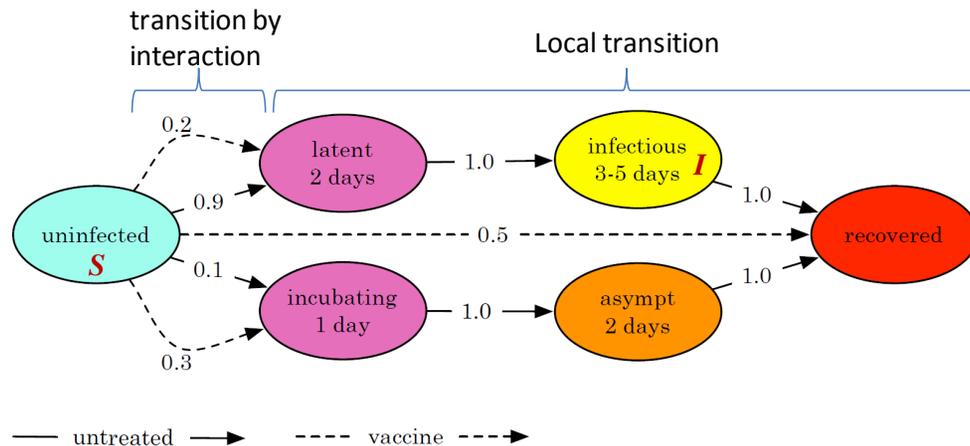


Simulating contagion over dynamic networks



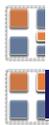
EpiSimdemics¹

- Agent-based
- Realistic population data
- Intervention²
- Co-evolving network, behavior and policy²

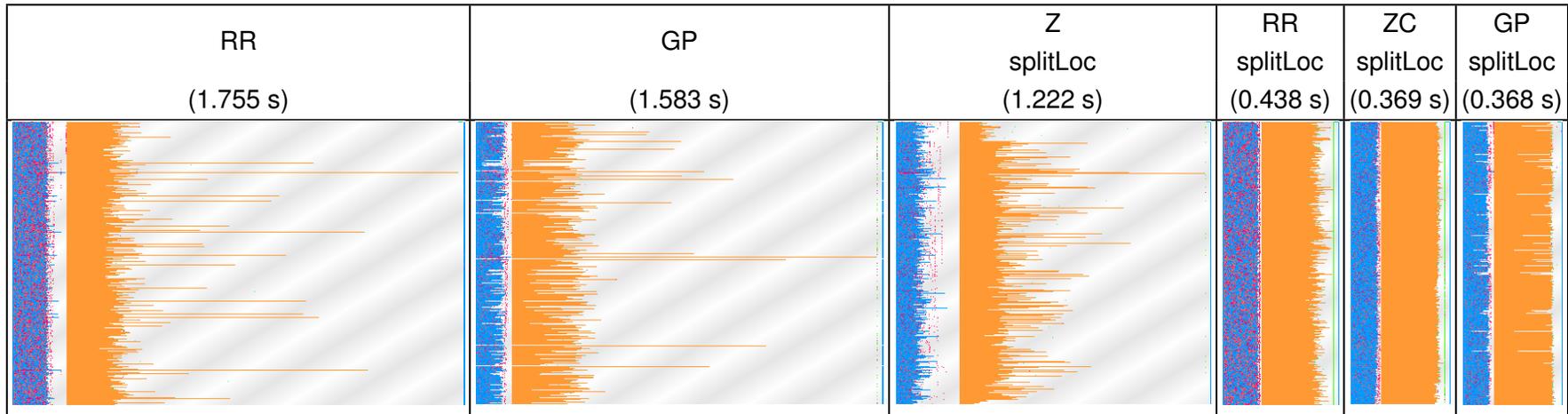


¹ C. Barrett et al., "EpiSimdemics: An Efficient Algorithm for Simulating the Spread of Infectious Disease over Large Realistic Social Networks," SC08

² K. Bisset et al., "Modeling Interaction Between Individuals, Social Networks and Public Policy to Support Public Health Epidemiology," WSC09.



Load distribution (Vulcan)



splitLoc: no peak in location computation
Z-splitLoc: no load balance

GP: shorter person phase
ZC-splitLoc: similar perf. w/ GP-splitLoc

- Blue: person computation
- Red: receiver's msg handling
- Orange: location computation

X-axis: **Time**

Y-axis: **Processor**

Timeline of an iteration from sampled subset of 332 cores of total 4K using Michigan data on Vulcan

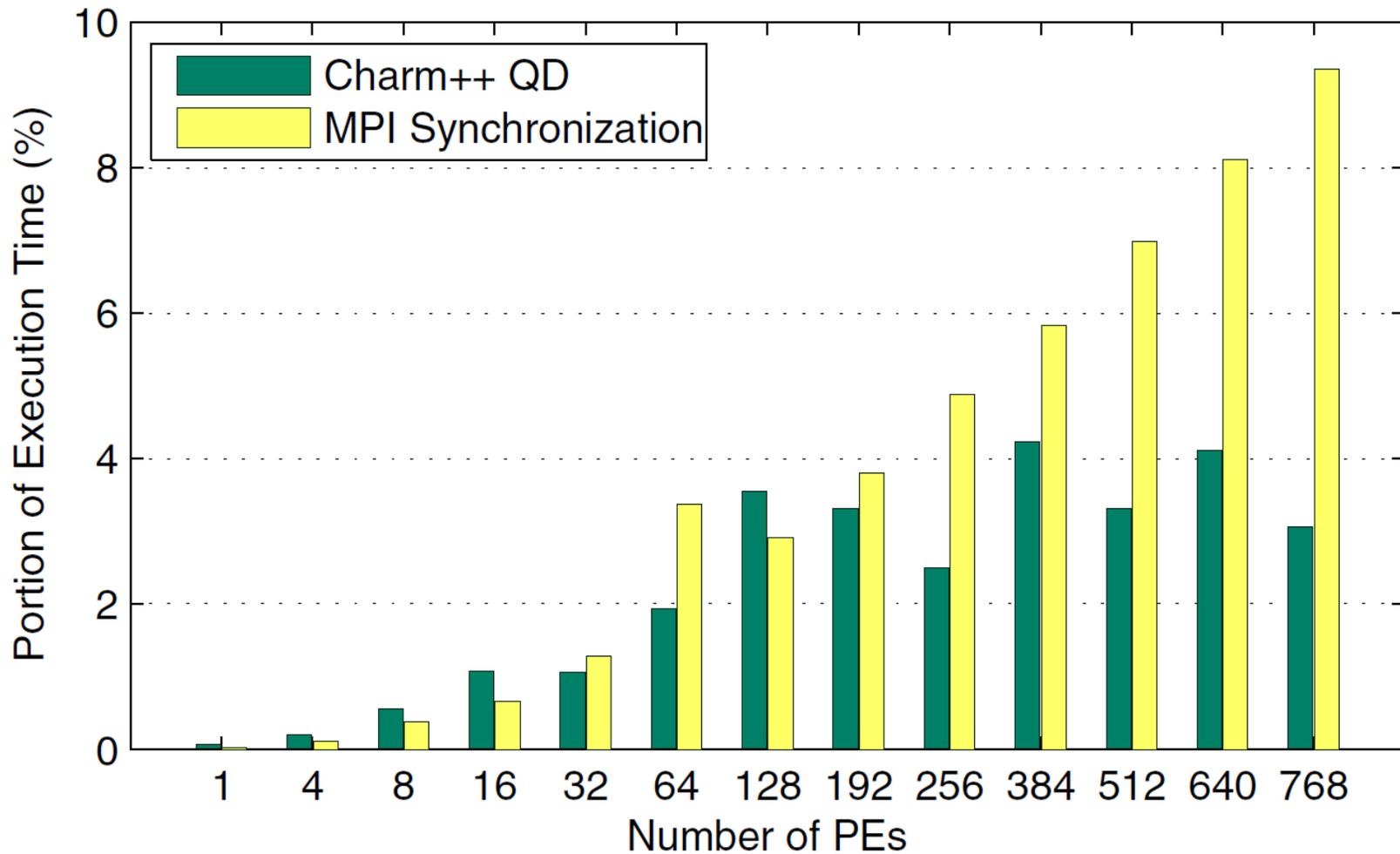
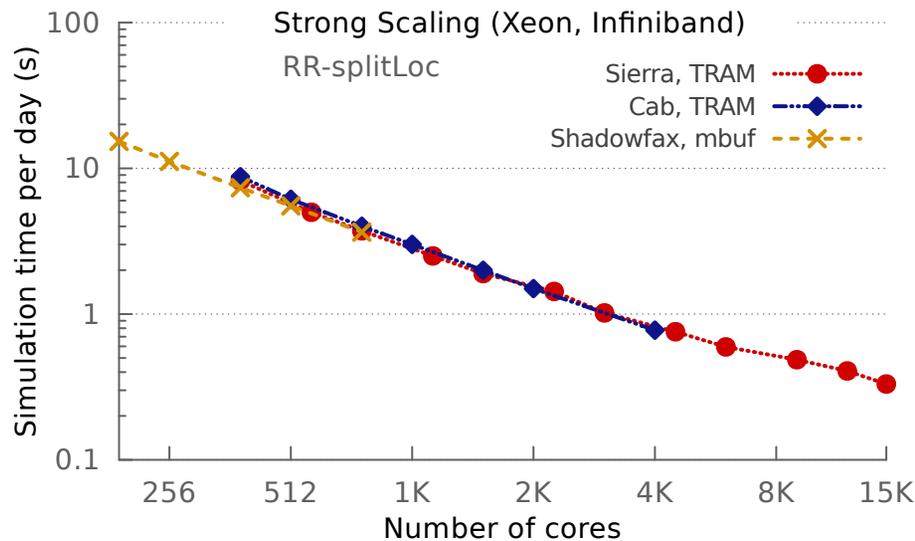
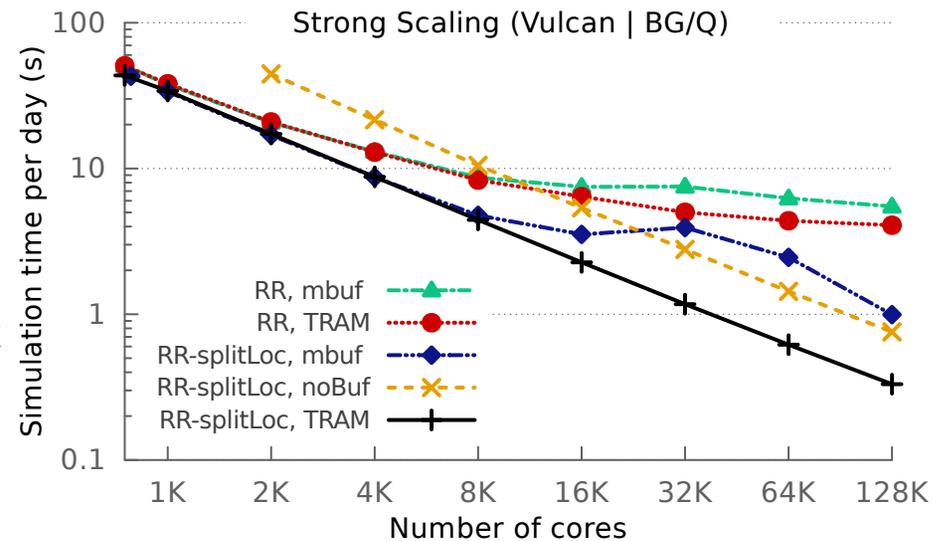
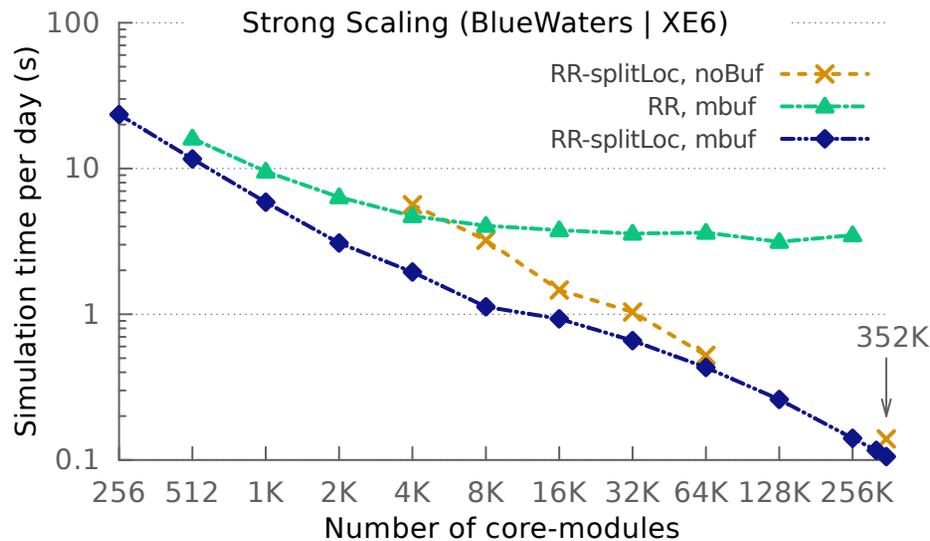


Figure 10. The synchronization cost using `contribute()` and QD method takes at most 4.23% of the total execution time while the MPI synchronization cost linearly increases up to 14.5% as the number of PEs used increases for simulating Arkansas population.

Strong scaling performance with the largest data set

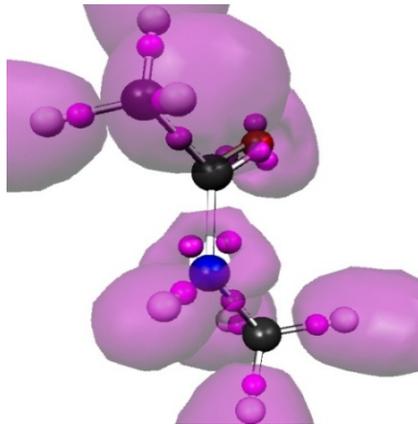


- Contiguous US population data
- **XE6: the largest scale (352K cores)**
- **BG/Q: good scaling up to 128K cores**
- Strong scaling helps timely reaction to pandemic

OpenAtom

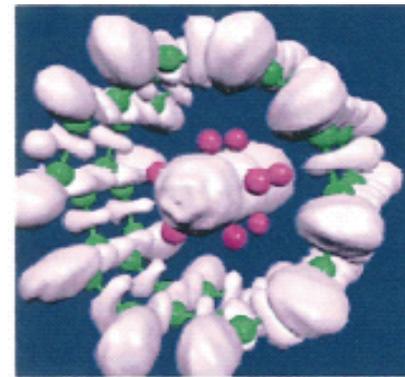
Car-Parinello Molecular Dynamics
NSF ITR 2001–2007, IBM, DOE, NSF

Molecular Clusters :

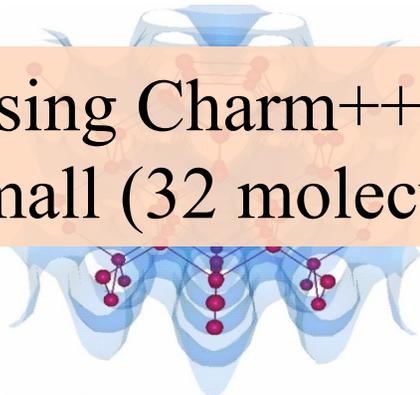


Recent NSF SSI-SI2 grant
With
G. Martyna (IBM)
Sohrab Ismail-Beigi

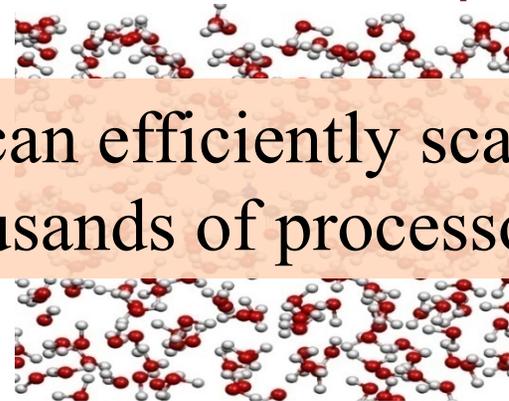
Nanowires:



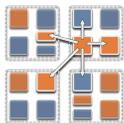
Semiconductor Surfaces:



3D-Solids/Liquids:

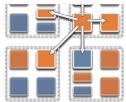
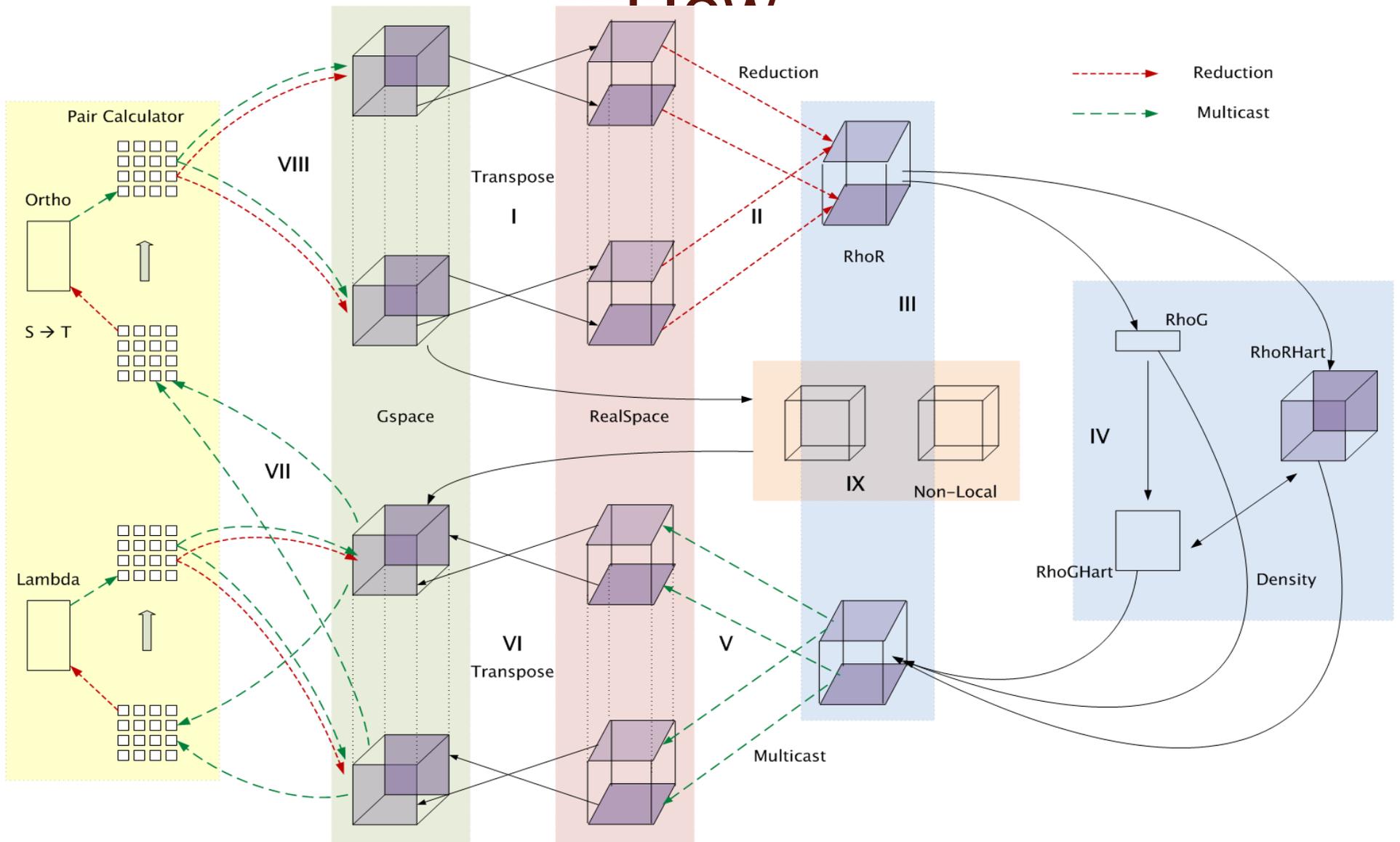


Using Charm++ virtualization, we can efficiently scale small (32 molecule) systems to thousands of processors

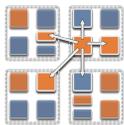
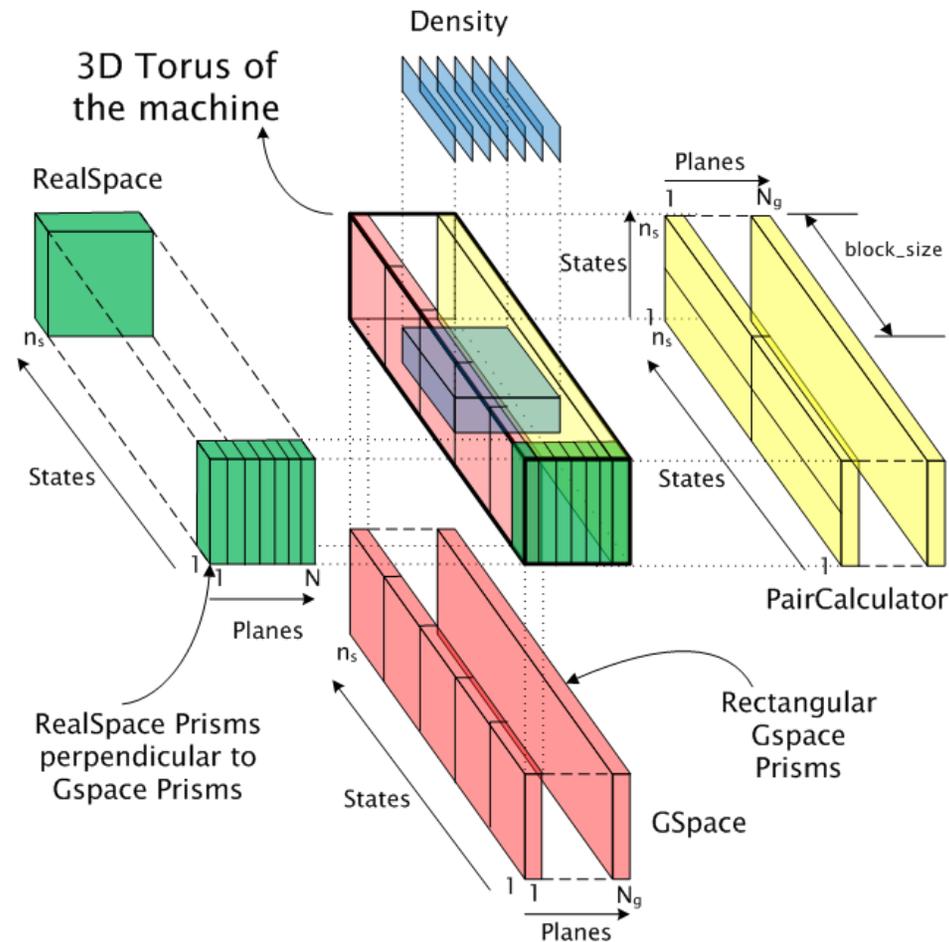


Decomposition and Computation

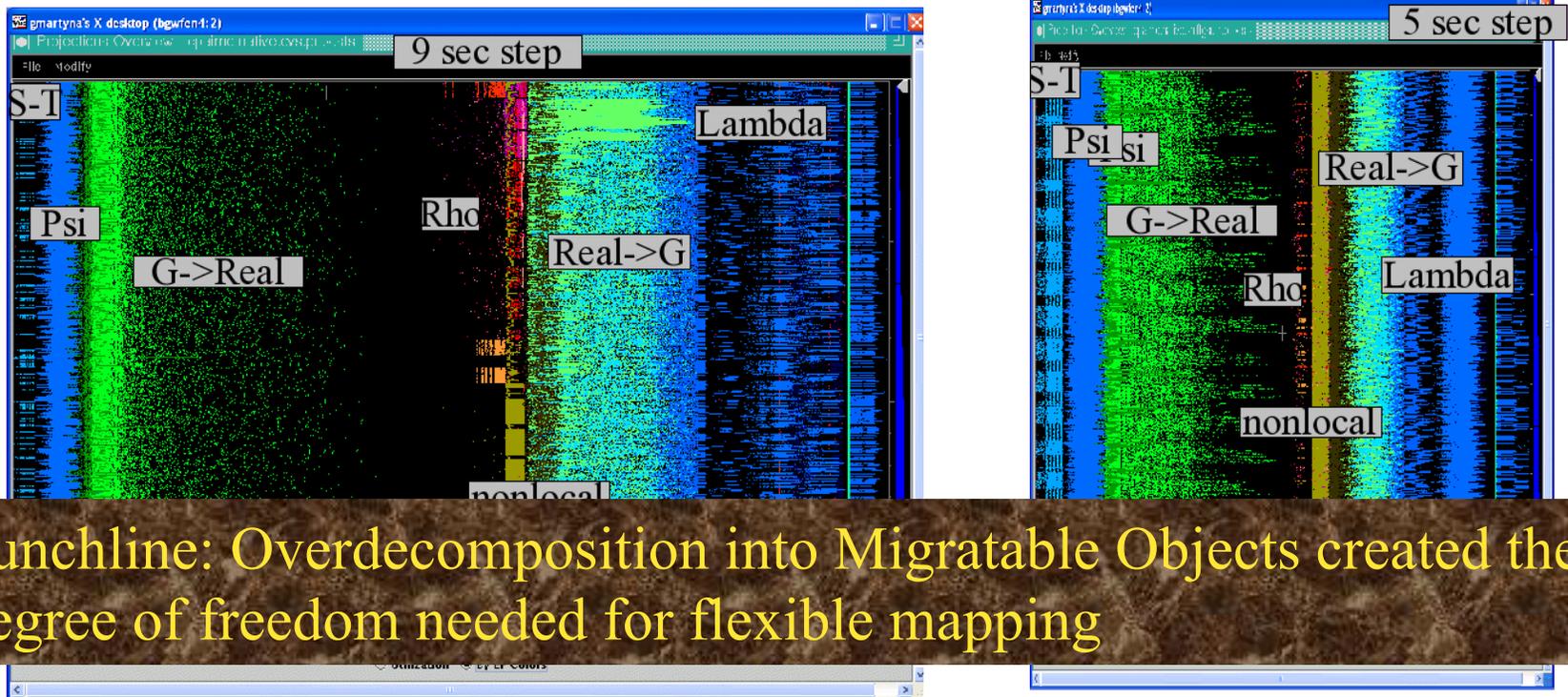
Flow



Topology Aware Mapping of Objects



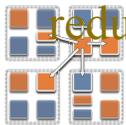
Improvements by topological aware mapping of computation to processors



Punchline: Overdecomposition into Migratable Objects created the degree of freedom needed for flexible mapping

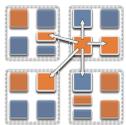
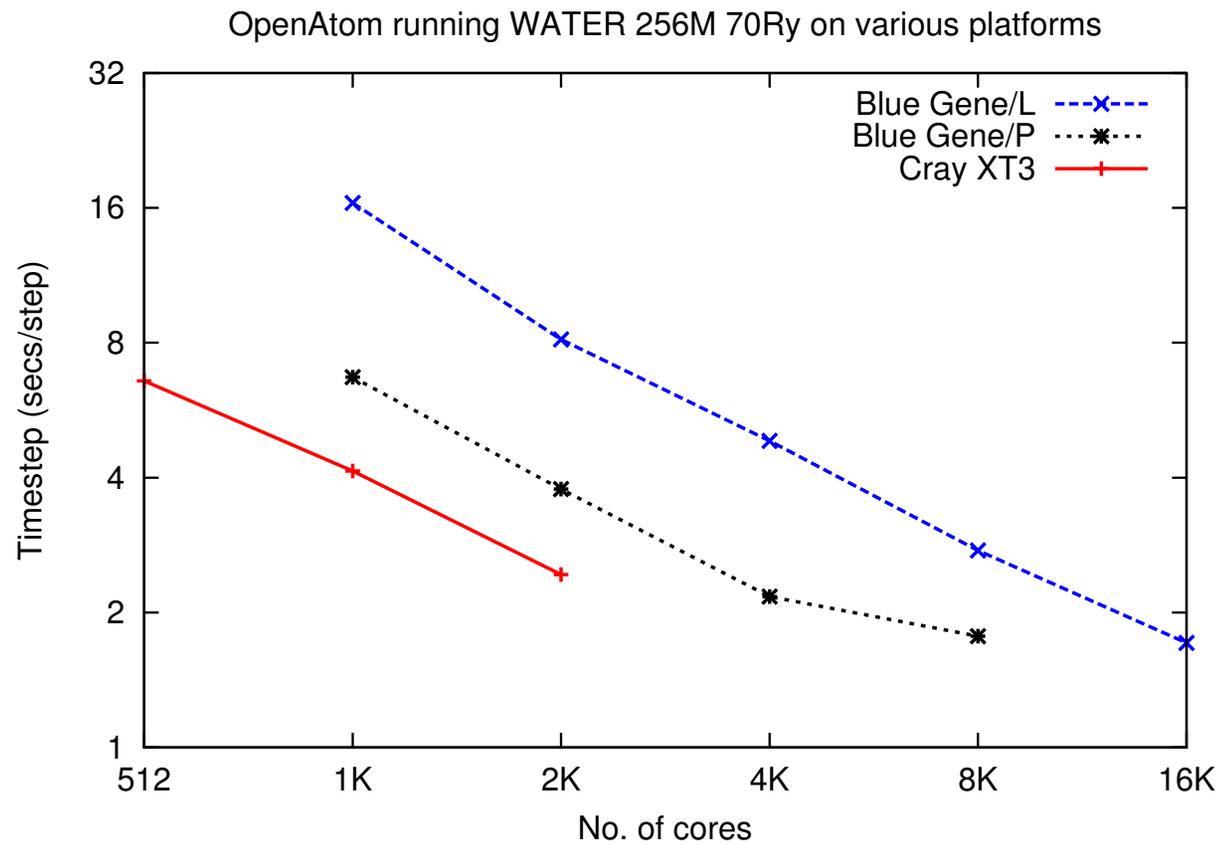
The simulation of the left panel, maps computational work to processors taking the network connectivity into account while the right panel simulation does not. The “black” or idle time processors spent waiting for computational work to arrive on processors is significantly

reduced at left. (256waters, 70R, on BG/L 4096 cores)



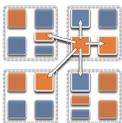
OpenAtom Performance Sampler

Ongoing work on:
K-points



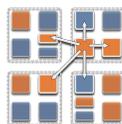
MiniApps

Mini-App	Features	Machine	Max cores
AMR	Overdecomposition, Custom array index, Message priorities, Load Balancing, Checkpoint restart	BG/Q	131,072
LeanMD	Overdecomposition, Load Balancing, Checkpoint restart, Power awareness	BG/P BG/Q	131,072 32,768
Barnes-Hut (n-body)	Overdecomposition, Message priorities, Load Balancing	Blue Waters	16,384
LULESH 2.02	AMPI, Over- decomposition, Load Balancing	Hopper	8,000
PDES	Overdecomposition, Message priorities, TRAM	Stampede	4,096



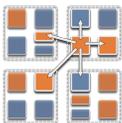
More MiniApps

Mini-App	Features	Machine	Max cores
1D FFT	Interoperable with MPI	BG/P BG/Q	65,536 16,384
Random Access	TRAM	BG/P BG/Q	131,072 16,384
Dense LU	SDAG	XT5	8,192
Sparse Triangular Solver	SDAG	BG/P	512
GTC	SDAG	BG/Q	1,024
SPH		Blue Waters	-



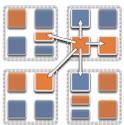
Where are Exascale Issues?

- I didn't bring up exascale at all so far..
 - Overdecomposition, migratability, asynchrony were needed on yesterday's machines too
 - And the app community has been using them
 - But:
 - On *some* of the applications, and maybe without a common general-purpose RTS
- The same concepts help at exascale
 - Not just help, they are necessary, and adequate
 - As long as the RTS capabilities are improved
- We have to apply overdecomposition to all (most) apps



A message of this talk

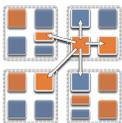
Intelligent, introspective, Adaptive Runtime Systems, developed for handling application's dynamic variability, already have features that can deal with challenges posed by exascale hardware



Fault Tolerance in Charm++ / AMPI

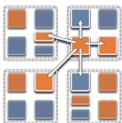
- Four approaches available:
 - Disk-based checkpoint/restart
 - In-memory double checkpoint w auto. restart
 - Proactive object migration
 - Message-logging: scalable fault tolerance
- Common Features:
 - Easy checkpoint: migrate-to-disk
 - Based on dynamic runtime capabilities
 - Use of object-migration
 - Can be used in concert with load-balancing schemes

Demo at Tech
Marketplace



Extensions to fault recovery

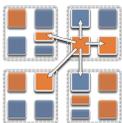
- Based on the same over-decomposition ideas
 - Use NVRAM instead of DRAM for checkpoints
 - Non-blocking variants
 - [Cluster 2012] Xiang Ni et al.
 - Replica-based soft-and-hard-error handling
 - As a “gold-standard” to optimize against
 - [SC 13] Xiang Ni, E. Meneses, N. Jain, et al.



Saving Cooling Energy

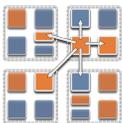
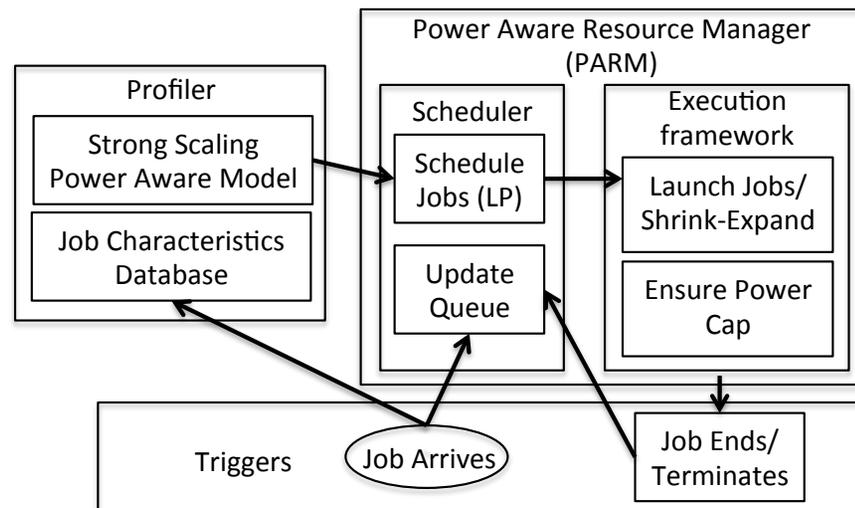
Demo at Tech
Marketplace

- Easy: increase A/C setting
 - But: some cores may get too hot
- So, reduce frequency if temperature is high (DVFS)
 - Independently for each chip
- *But*, this creates a load imbalance!
- No problem, we can handle that:
 - Migrate objects away from the slowed-down processors
 - Balance load using an existing strategy
 - Strategies take speed of processors into account
- Implemented in experimental version
 - SC 2011 paper, IEEE TC paper
- Several new power/energy-related strategies
 - PASA '12: Exploiting differential sensitivities of code segments to frequency change



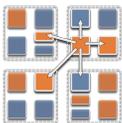
PARM: Power Aware Resource Manager

- Charm++ RTS facilitates malleable jobs
- PARM can improve throughput under a fixed power budget using:
 - overprovisioning (adding more nodes than conventional data center)
 - RAPL (capping power consumption of nodes)
 - Job malleability and moldability

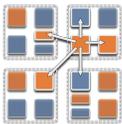


Costs of Overdecomposition?

- We examined the “Pro”s so far
- Cons and remedies:
- Scheduling overhead?
 - Not much at all
 - In fact get benefits due to blocking
- Memory in ghost layer increases
 - Fuse local regions with compiler support
 - Fetch one ghost layer at a time
 - Hybridize (pthreads/openMP inside objects/DEBs)
- Less control over scheduling?
 - i.e. too much asynchrony?
 - But can be controlled in various ways by an observant RTS/programmer
- For domain–decomposition based solvers, may increase number of iterations
 - You can lift it to node–level overdecomposition (use openMP inside)
 - Also, other ideas:
- Too radical and new?
 - Well, its working well for the past 10–15 years in multiple applications, via Charm++ and AMPI



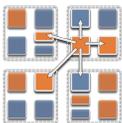
How can Application Developers get ready for Adaptive RTSs?



Its not that weird or new

- First, note:
 - The techniques I advocated were needed for dynamic irregular apps even on yesterday's machines
 - Just that they need to be applied to even regular apps
 - How Charm++ meets exascale challenges already, almost
 - How we got so lucky: because of these irregular apps

The adaptivity that was created via overdecomposition, migratability, & asynchrony, for dynamic applications, is also useful for handling machine variability at exascale



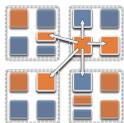
Summary

- Charm++ embodies an adaptive, introspective runtime system
- Many applications have been developed using it
 - NAMD, ChaNGa, Episimdemics, OpenAtom, ...
 - Many miniApps, and third-party apps
- Adaptivity developed for apps is useful for addressing exascale challenges
 - Resilience, power/temperature optimizations, ..

More info on Charm++:

<http://charm.cs.illinois.edu>

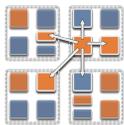
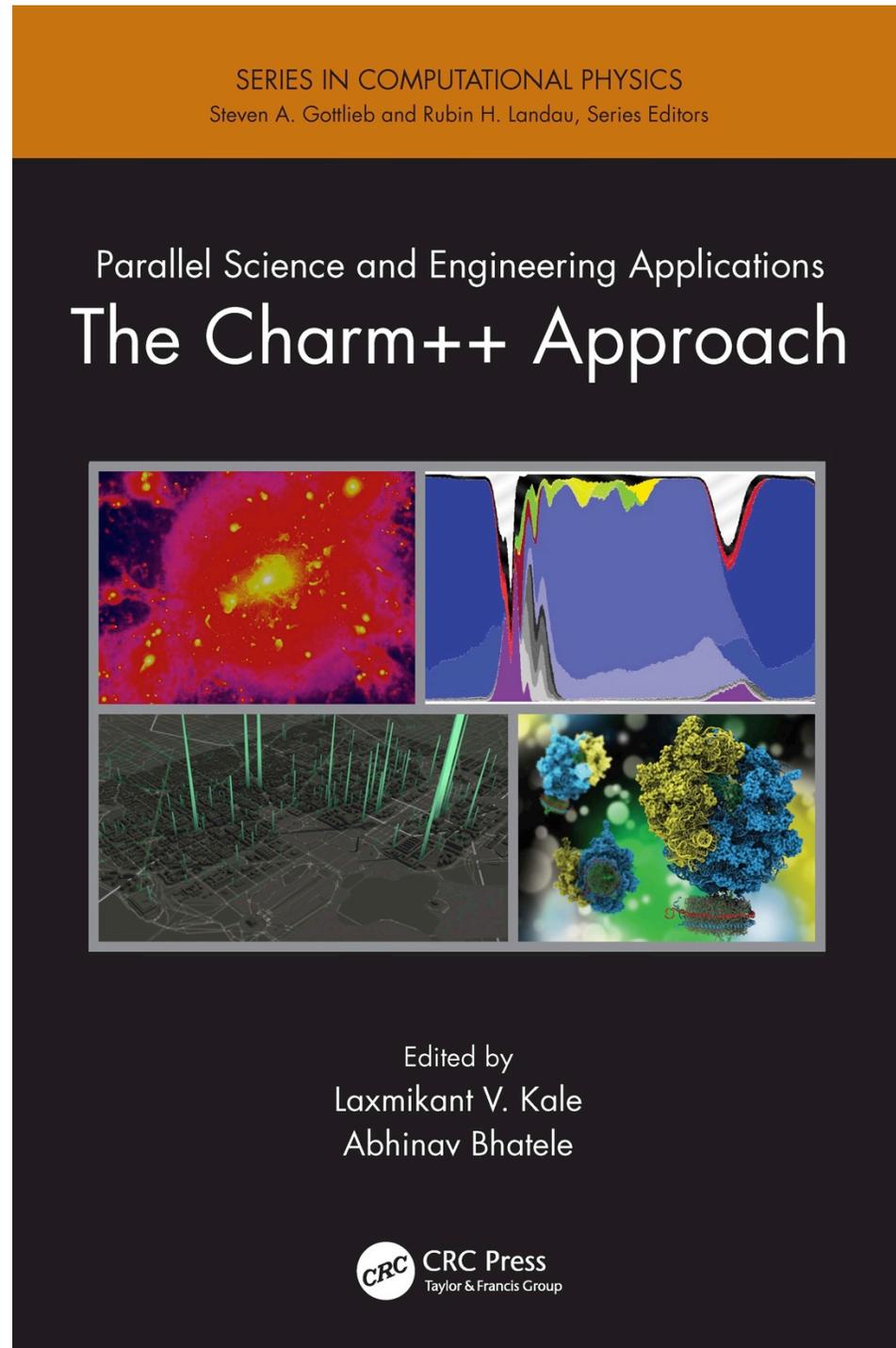
Including the miniApps



Overdecomposition Asynchrony Migratability

A recently published book surveys seven major applications developed using Charm++

More info on Charm++:
<http://charm.cs.illinois.edu>
Including the miniApps



7/16/14

70

