Getting Ready for Adaptive RTSs

Laxmikant (Sanjay) Kale

http://charm.cs.illinois.edu

Parallel Programming Laboratory

Department of Computer Science

University of Illinois at Urbana Champaign

Overview

- Main exascale challenge is variability
  - Static and dynamic
  - Exacerbated by strong scaling requirements
  - Persistence is our [only?] friend
  - Good division of labor between “system” and app developer is essential
- My Mantra: Overdecomposition, migratability, asynchrony (Oma)
- Explain each concept briefly (what it is)
- Explain how it empowers RTS: Introspection and adaptivity
- Potential costs and how they can be mitigated: overhead, memory, algo overhead
  - Soln include considering node as a unit (so, have 8-16 work units per chunk)
- Show benefits apps:
  - Strong scaling via overdecomposition: NAMD 200+ us step
  - Asynchrony -> AMR
- What RTSs can do with this empowerment:
  - Ldb, FT, power/energy
  - Reconfigurability (apps/RTS) and runtime auto-tuning
- What can app developers do to get ready for exascale/arts
  - Note: our solution (OMA) was needed for dynamic irregular apps even on yesterday’s machines
    - Just that it needs to be applied to even regular apps
    - How charm++ meets exascale challenges already, almost
      - How we got so lucky: because of these irregular apps
  - What to do:
    - Explore overdecomposition in your apps
    - Create control points for runtime manipulation
    - Get used to words like “continuations”.. But we need only simpler versions of those

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

My Mantra

a

OM

My Mantra

Oh….Maybe the order doesn't matter

O M a

My Mantra

verdecomposition

OaM

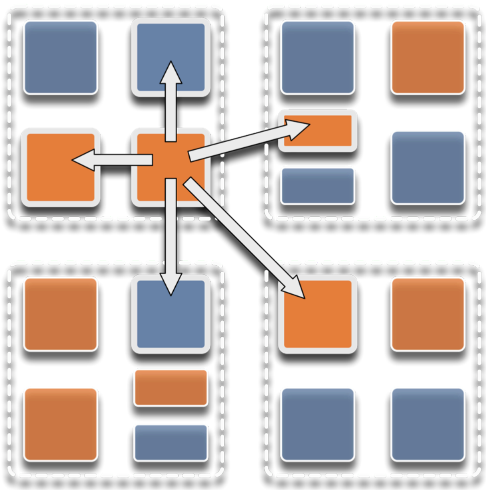
synchrony

igratability

PPL UIUC

Overdecomposition

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



Migratability

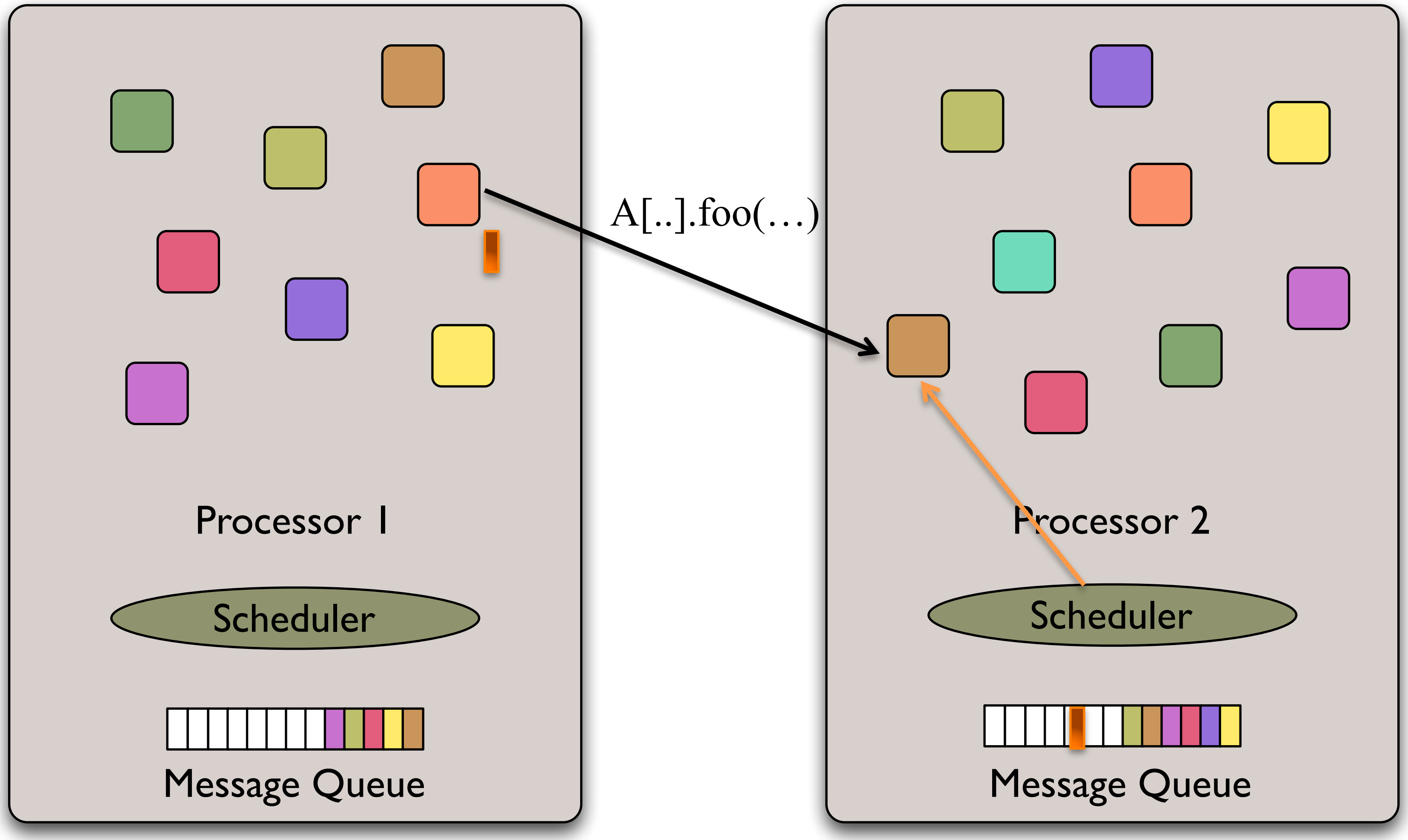
- Allow these work and data units to be migratable at runtime
  - i.e. the programmer or runtime, can move them
- Consequences for the app-developer
  - Communication must now be addressed to logical units with global names, not to physical processors
  - But this is a good thing
- Consequences for RTS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

PPL UIUC

Asynchrony: Message-Driven Execution

- Now:
  - 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

Message-driven Execution



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

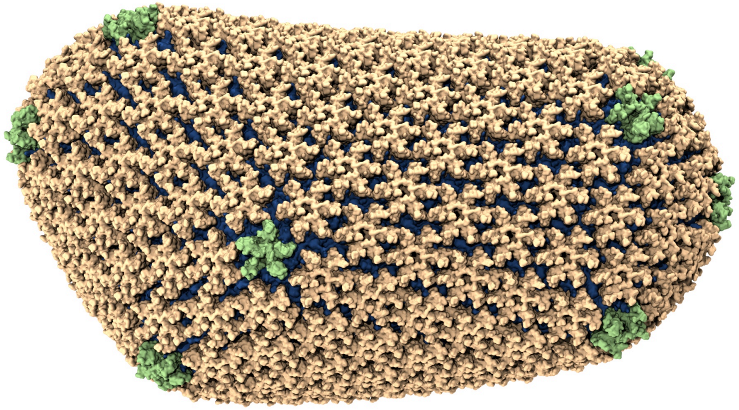
PPL UIUC

Application Examples
to
Demonstrate the Utility of

**Overdecomposition,
Migratability,
Asynchrony!**

NAMD: Biomolecular Simulations

- Collaboration with K. Schulten
- With over 45,000 registered users
- Scaled to most top US supercomputers
- In production use on supercomputers and clusters and desktops
- Gordon Bell award in 2002

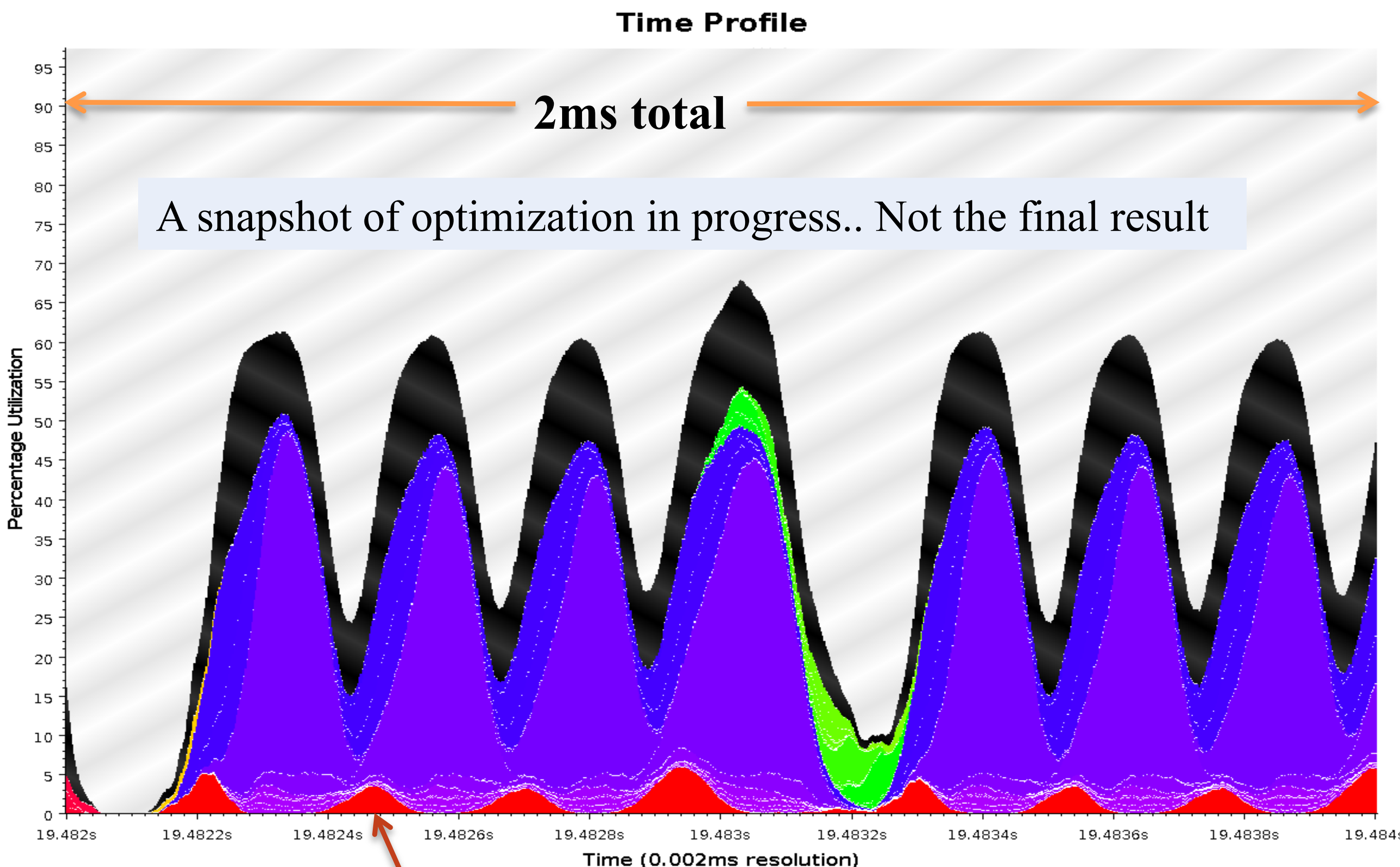


Recent success: Determination of the structure of HIV capsid by researchers including Prof Schulten

PPL UIUC

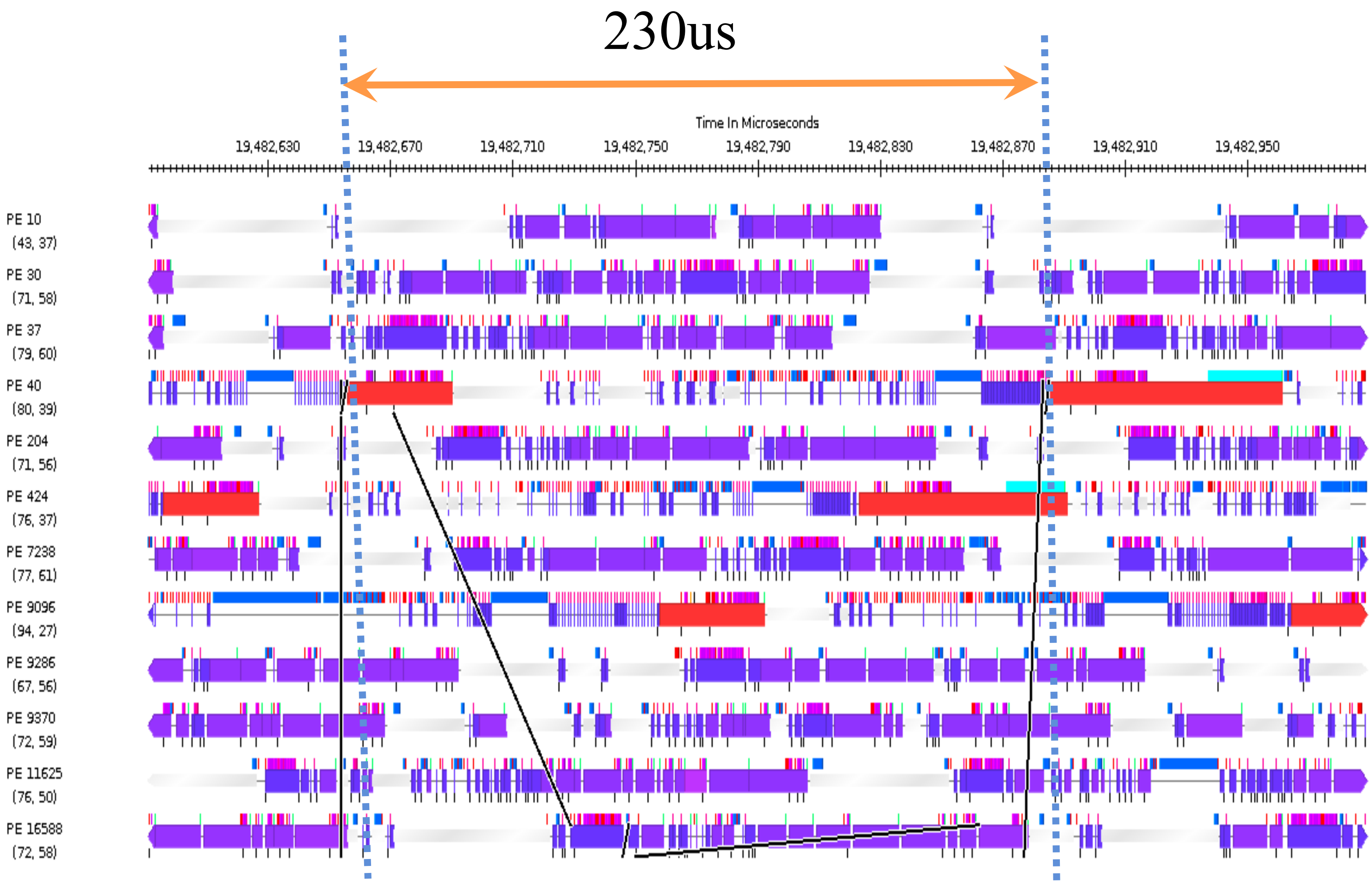
Time Profile of ApoA1 on Power7 PERCS

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



Overlapped steps, as a result of asynchrony

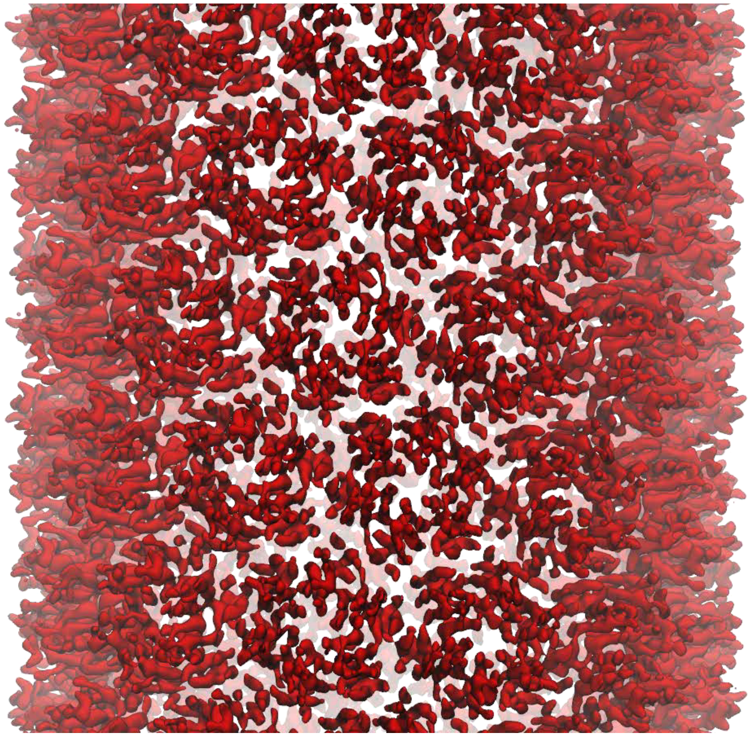
Timeline of ApoA1 on Power7 PERCS



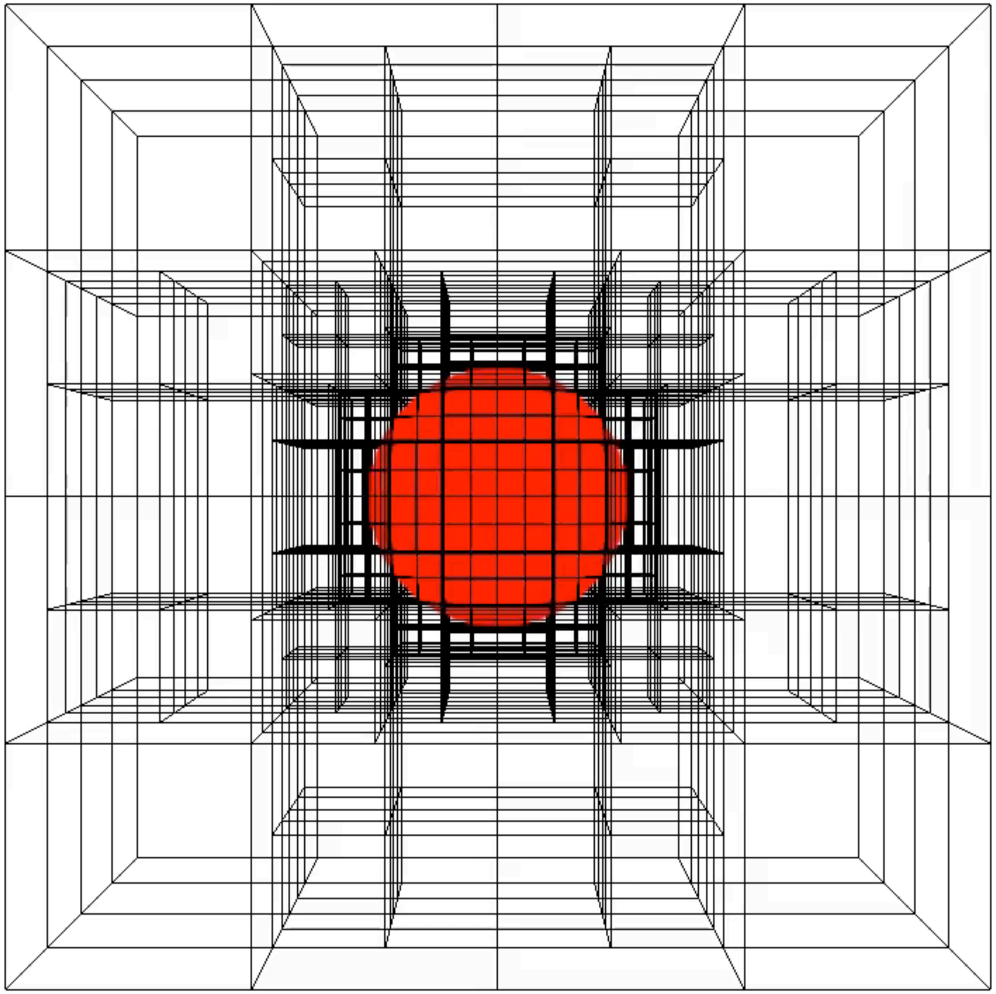
PPL
UIUC

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

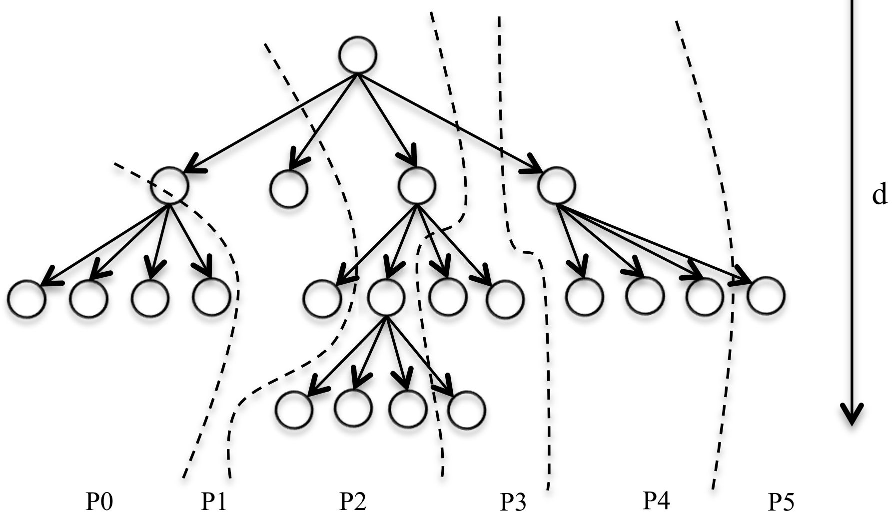


Structured AMR miniApp



Structured AMR

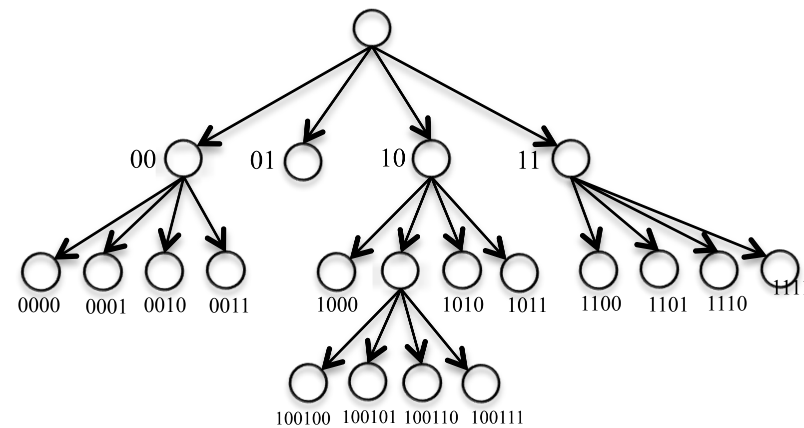
Typical MPI Approach



Process based

Contiguous blocks assigned to a process

Charm++ Approach

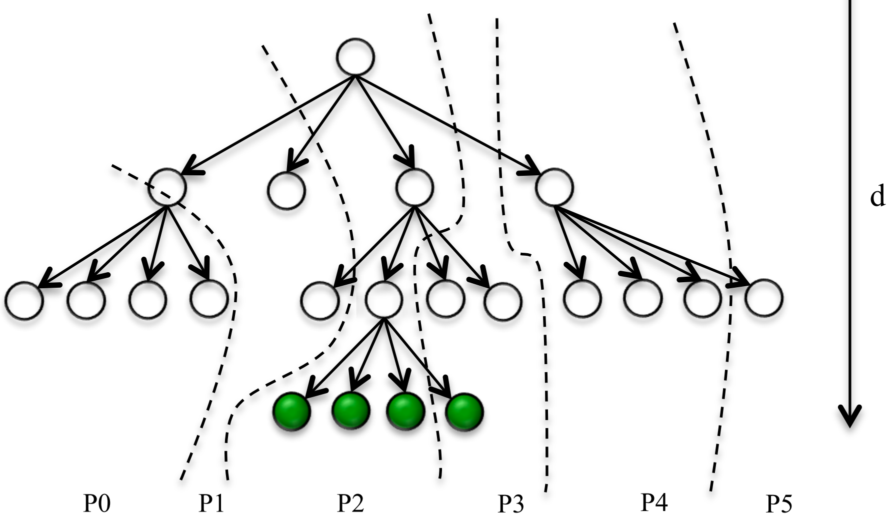


Object based

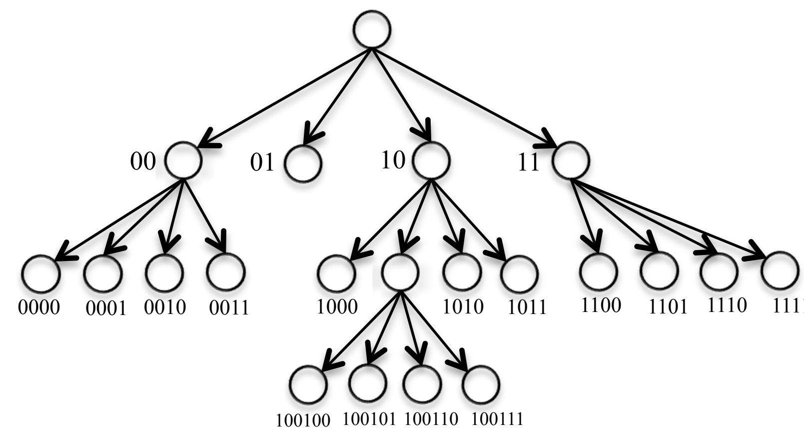
- Each block is an independent object
  - is the basic execution unit
  - can be mapped to any physical process
  - is uniquely addressable
  - is migratable

Structured AMR

Typical MPI Approach



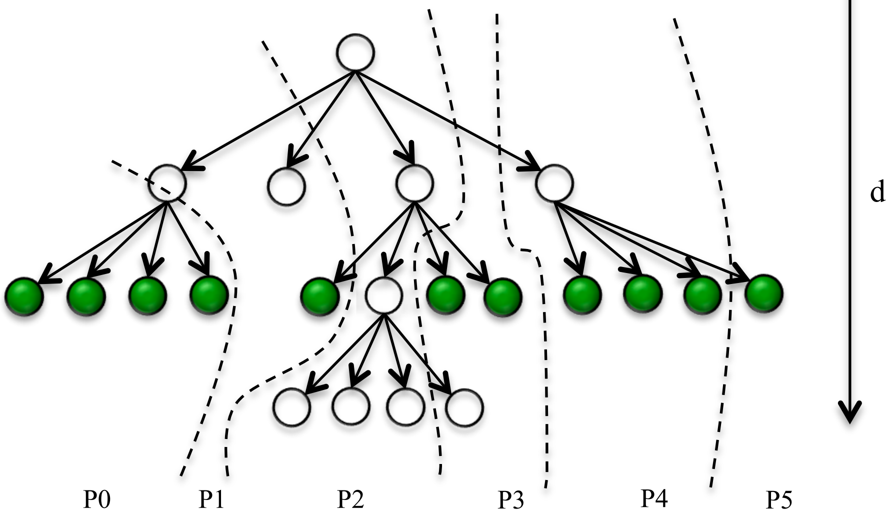
Charm++ Approach



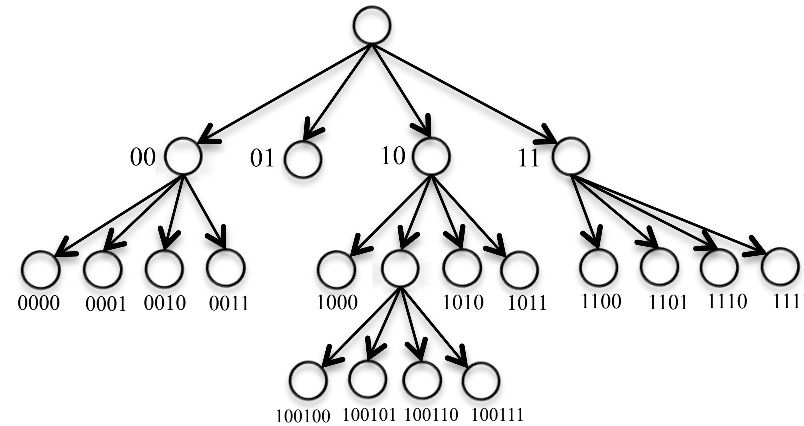
Mesh Restructuring

Structured AMR

Typical MPI Approach



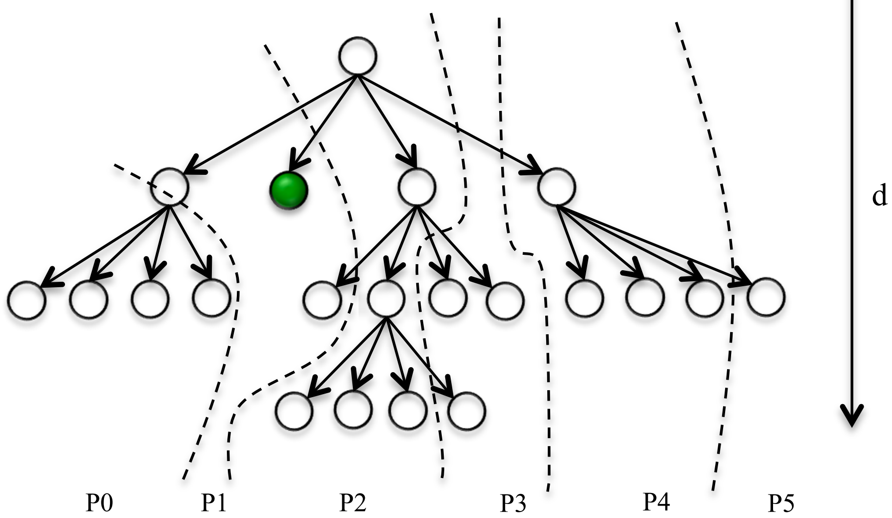
Charm++ Approach



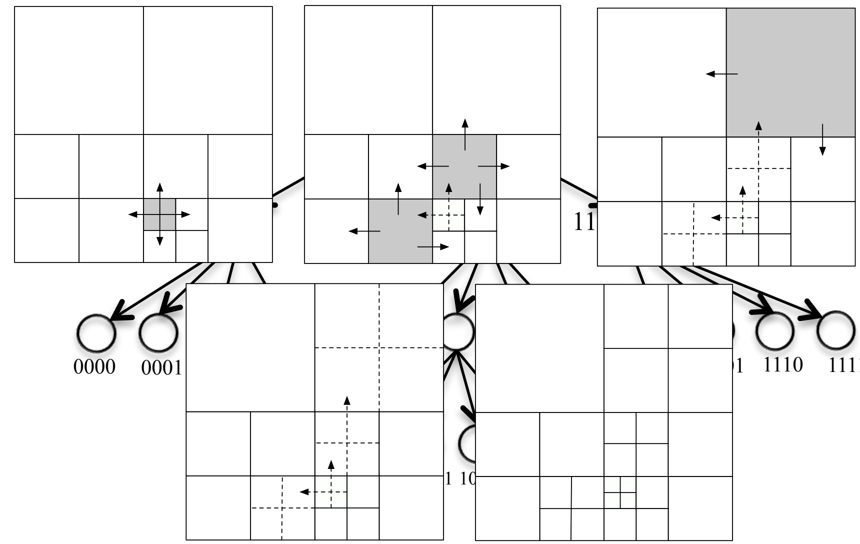
Mesh Restructuring

Structured AMR

Typical MPI Approach



Charm++ Approach



Mesh Restructuring

- Ripple Propagation Algorithm
  - Level-by-level
  - O(d) global reductions ≈ O(d*logP)

Synchronization overhead

- Tree-replication on each process
  - O(#blocks) memory per process

Memory overhead

Mesh Restructuring

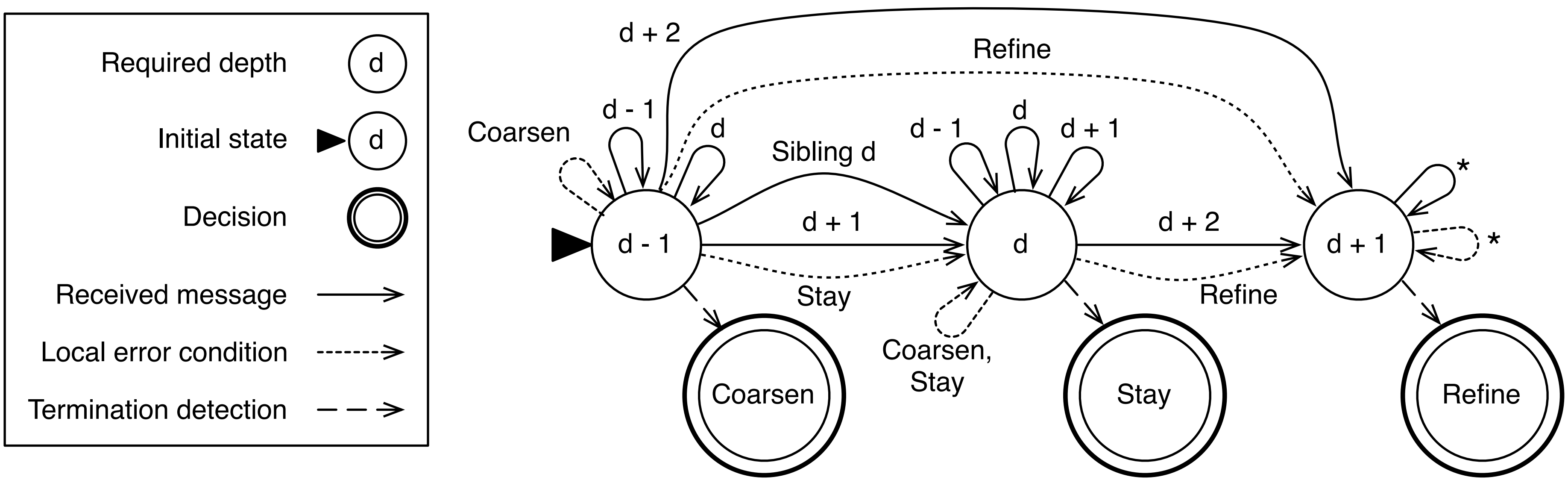
- Exchange messages with neighboring blocks
  - Update state using a state machine
  - Quiescence to detect global consensus

O(log P) time

- Blocks save current level of neighbors
  - O(#blocks/P) memory per process

O(#blocks/P) space

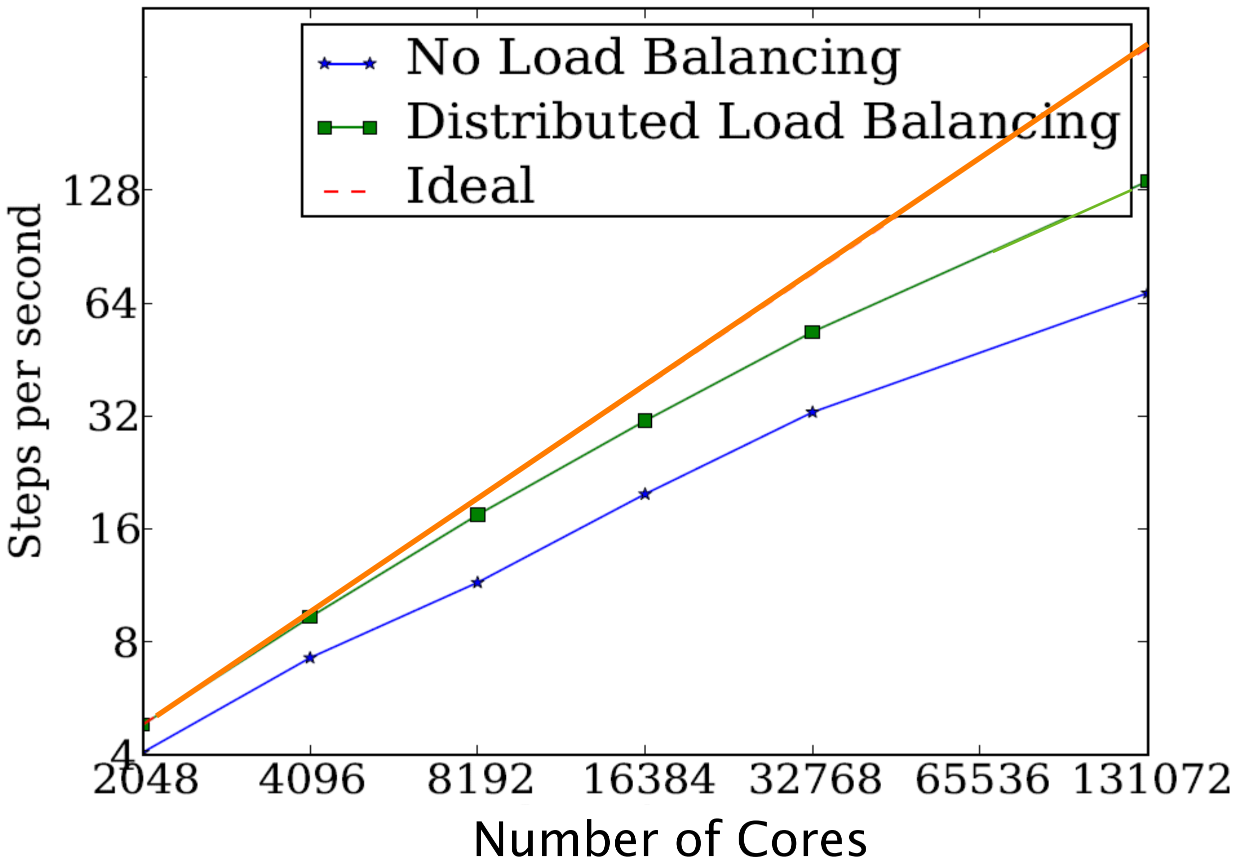
Structured AMR: State Machine



Structured AMR: Performance

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

Advection Benchmark
First order method in 3d-space



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

Exascale-like capabilities based on

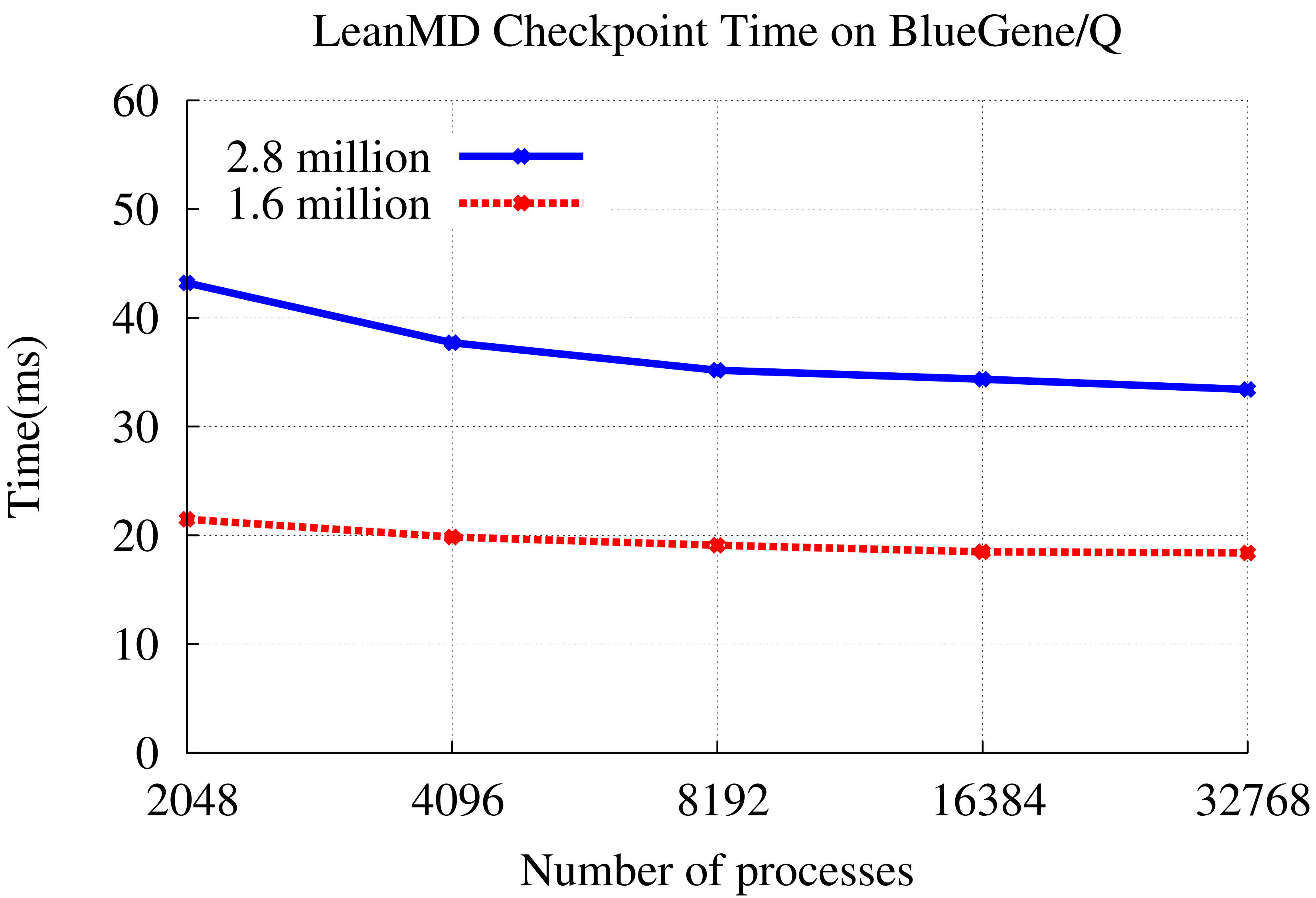
Overdecomposition, Migratability, Asynchrony!

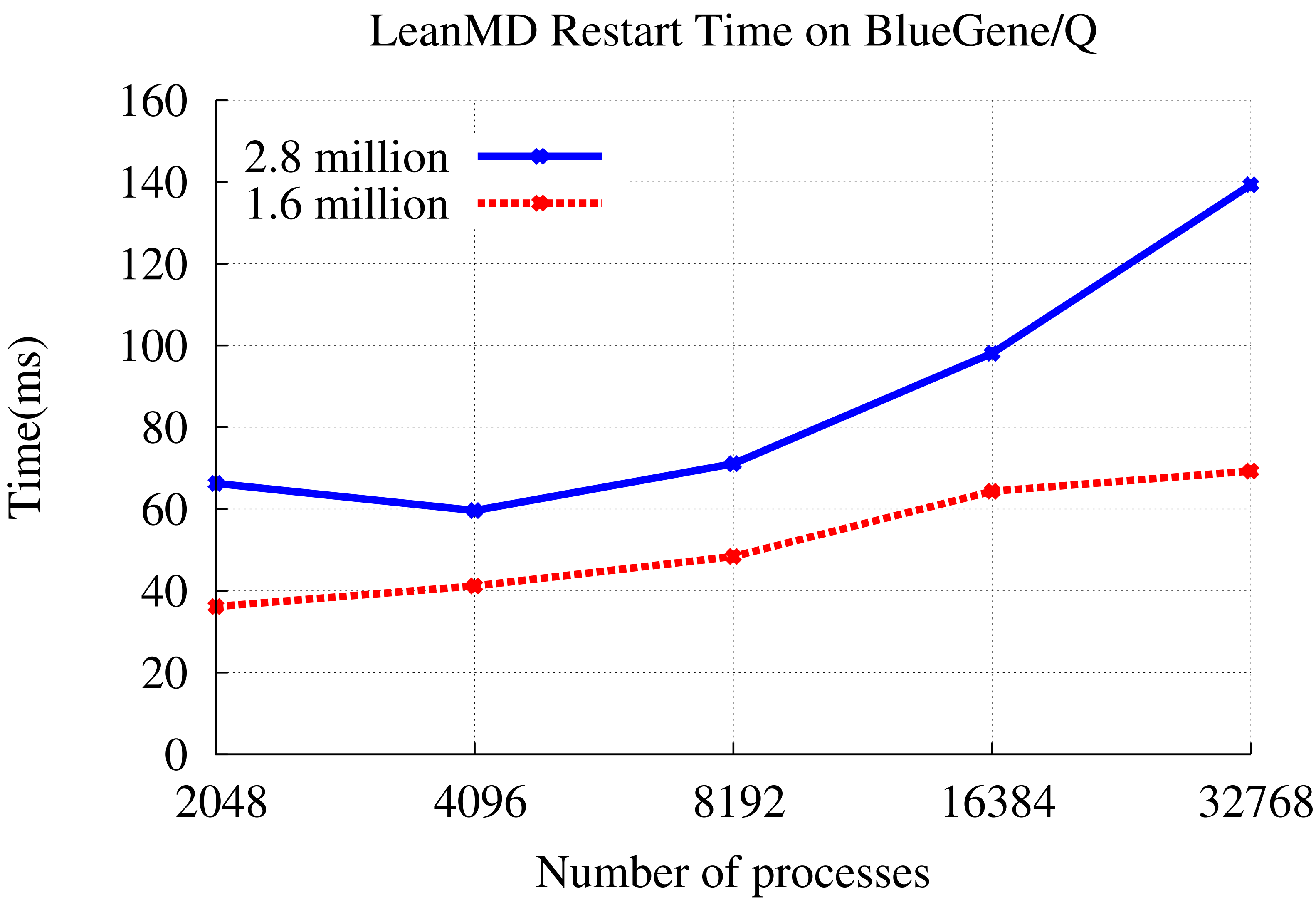
Fault Tolerance in Charm++/AMPI

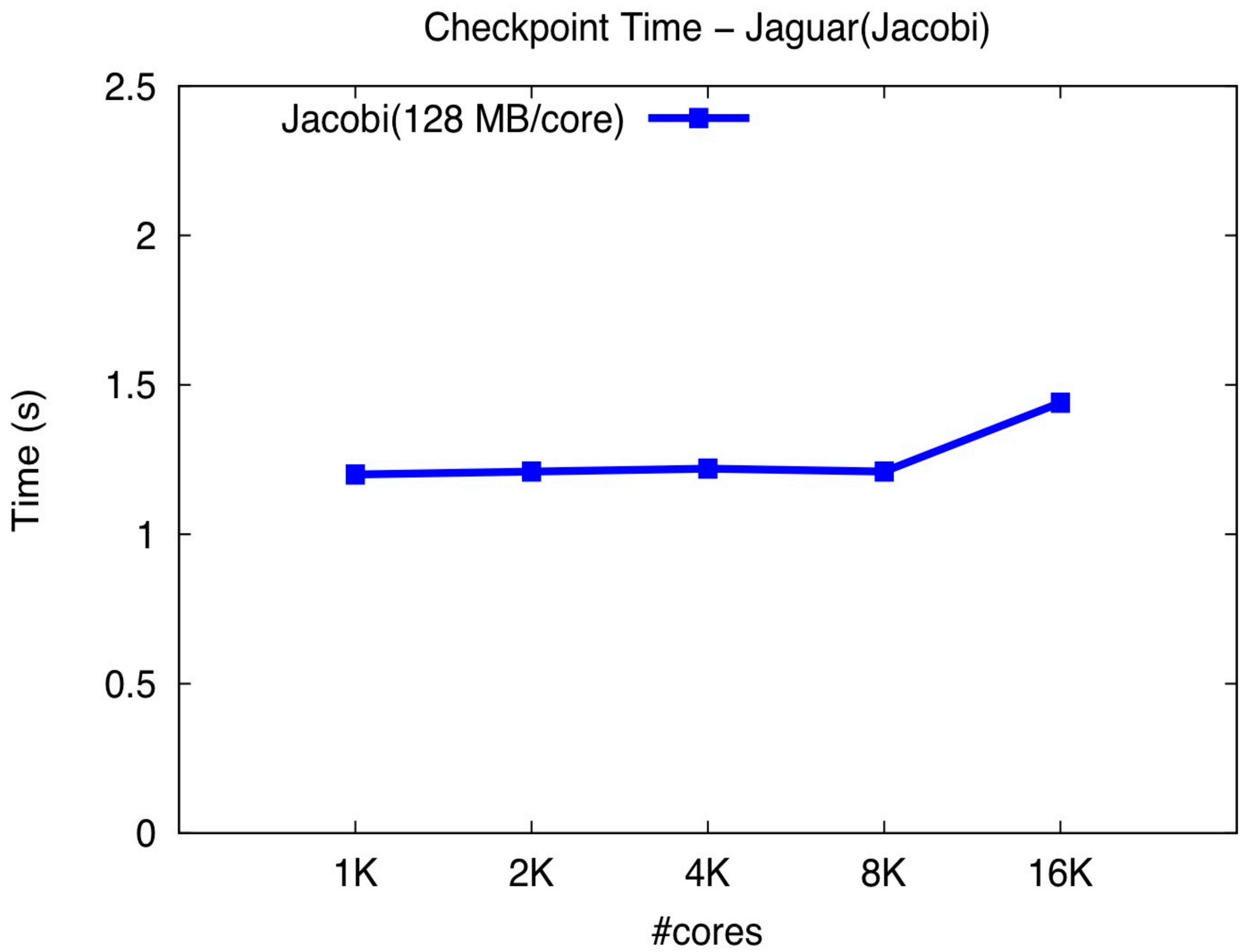
- Four approaches available:
  - Disk-based checkpoint/restart
  - In-memory double checkpoint w auto. restart
  - Proactive object migration
  - Message-logging: scalable fault tolerance
- Common Features:
  - Easy checkpoint: migrate-to-disk
  - Based on dynamic runtime capabilities
  - Use of object-migration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

In-local-storage Checkpoint/restart

- Is practical for many apps
  - Relatively small footprint at checkpoint time
- Very fast times…
- Demonstration challenge:
  - Works fine for clusters in production version of Charm++
  - For MPI-based implementations running at centers:
    - Scheduler does not allow jobs to continue on failure
    - Communication layers are not fault tolerant
  - Fault injection: dieNow(),
  - Spare processors


LeanMD Checkpoint Time on BlueGene/Q
2.8 million
1.6 million
Time(ms)
0
10
20
30
40
50
60
2048
4096
8192
16384
32768
Number of processes


LeanMD Restart Time on BlueGene/Q
Time(ms)
0
20
40
60
80
100
120
140
160
2.8 million
1.6 million
2048
4096
8192
16384
32768
Number of processes



Checkpoint Time – Jaguar(Jacobi)
Jacobi(128 MB/core)
2.5
2
1.5
1
0.5
0
Time (s)
1K
2K
4K
8K
16K
#cores

Extensions to fault recovery

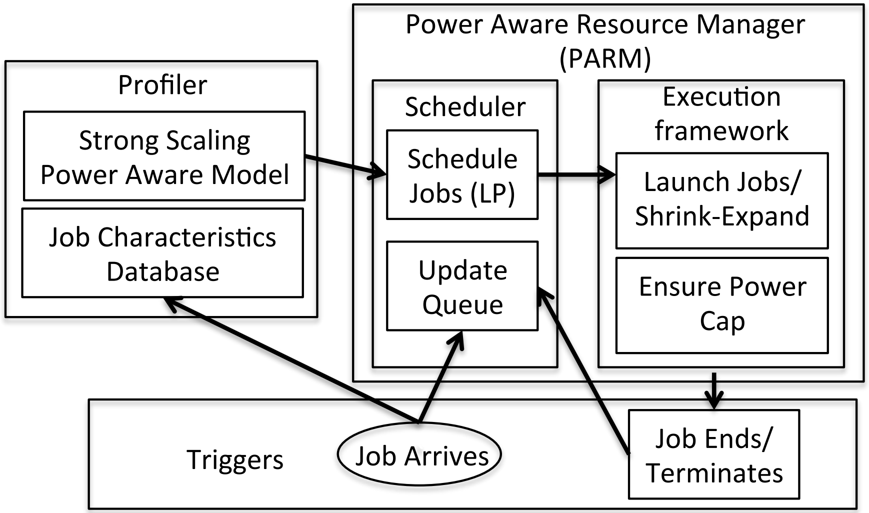
- Based on the same over-decomposition ideas
  - Use NVRAM instead of DRAM for checkpoints
    - Non-blocking variants
    - [Cluster 2012] Xiang Ni et al.
  - Replica-based soft-and-hard-error handling
    - As a “gold-standard” to optimize against
    - [SC 13] Xiang Ni, E. Meneses, N. Jain, et al.

Saving Cooling Energy

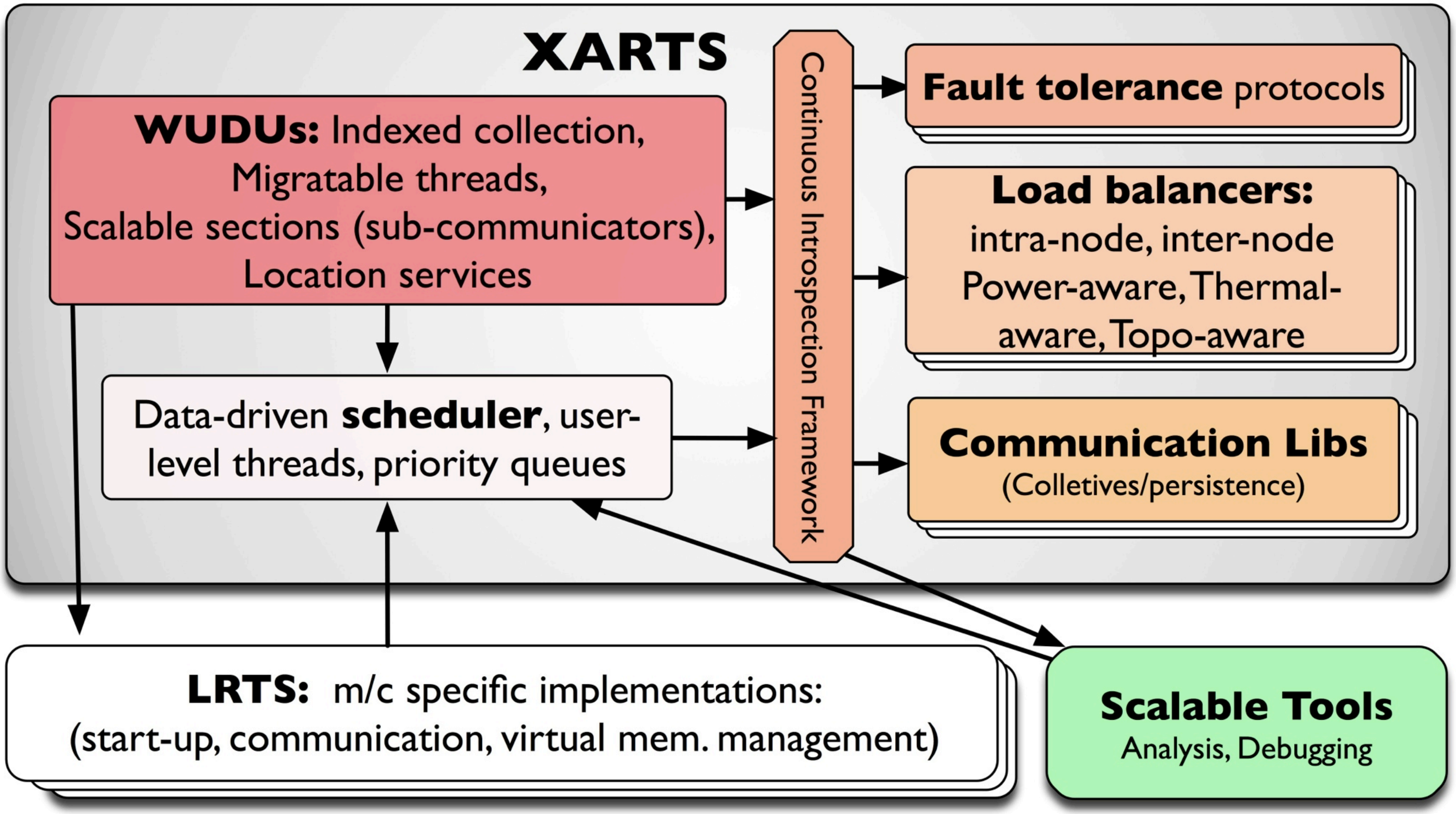
- Easy: increase A/C setting
  - But: some cores may get too hot
- So, reduce frequency if temperature is high (DVFS)
  - Independently for each chip
- *But,* this creates a load imbalance!
- No problem, we can handle that:
  - Migrate objects away from the slowed-down processors
  - Balance load using 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



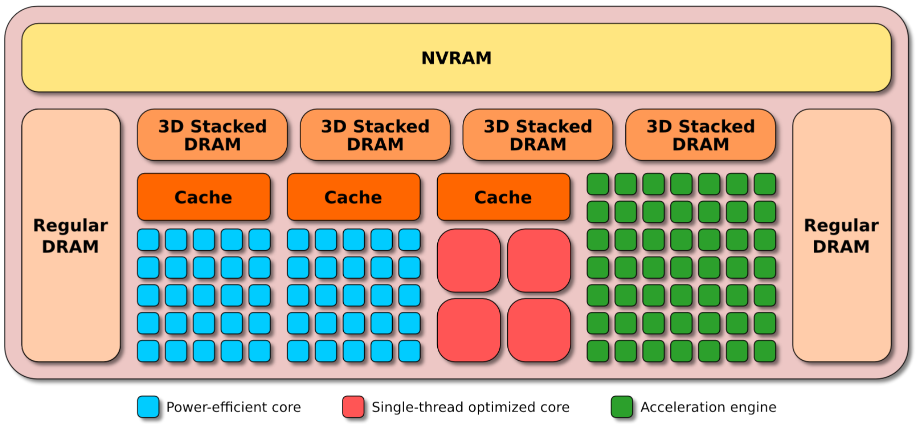
What Do RTSs Look Like: Charm++



ARGO

An Exascale Operating System and Runtime

Node



$9.7M ASCR DOE
3 year project, launched Aug 2013

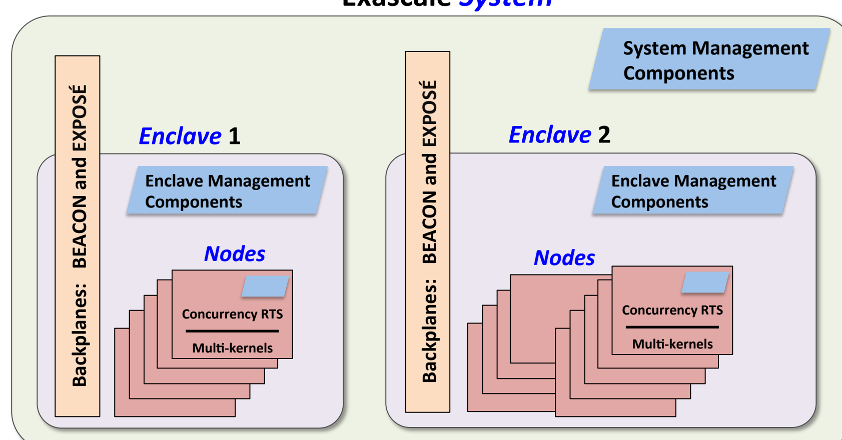
THE CREW OF THE ARGO:

Argonne National Laboratory;
Principle Investigator and Chief Architect: Pete Beckman
Chief Scientist: Marc Snir
P. Balaji, R. Gupta, K. Iskra, R. Thakur, K. Yoshii, F. Cappello
Boston University: J. Appavoo, O. Krieger
Lawrence Livermore National Laboratory:
M. Gokhale, E. Leon, B. Rountree, M. Schulz, B. Van Essen
Pacific Northwest National Laboratory: S. Krishnamoorthy, R. Gioiosa
University of Chicago: H. Hoffmann
University of Illinois at UC: L. Kale, E. Bohm, R. Venkataraman
University of Oregon: A. Malony, S. Shende, K. Huck
University of Tennesee Knoxville: J. Dongarra, G. Bosilca

Key Areas of Innovation:

- NodeOS/R
  - Core-specialization permits multiple, concurrent kernels
- Lightweight Concurrency
  - Embed fine-grained tasks and lightweight threads into OS for massive parallelism
- Backplane
  - Event, Control, and Performance backplanes to support global optimizations
- Global View
  - “Enclave” abstraction to allow global optimization of power, resilience, perf.

Exascale *System*

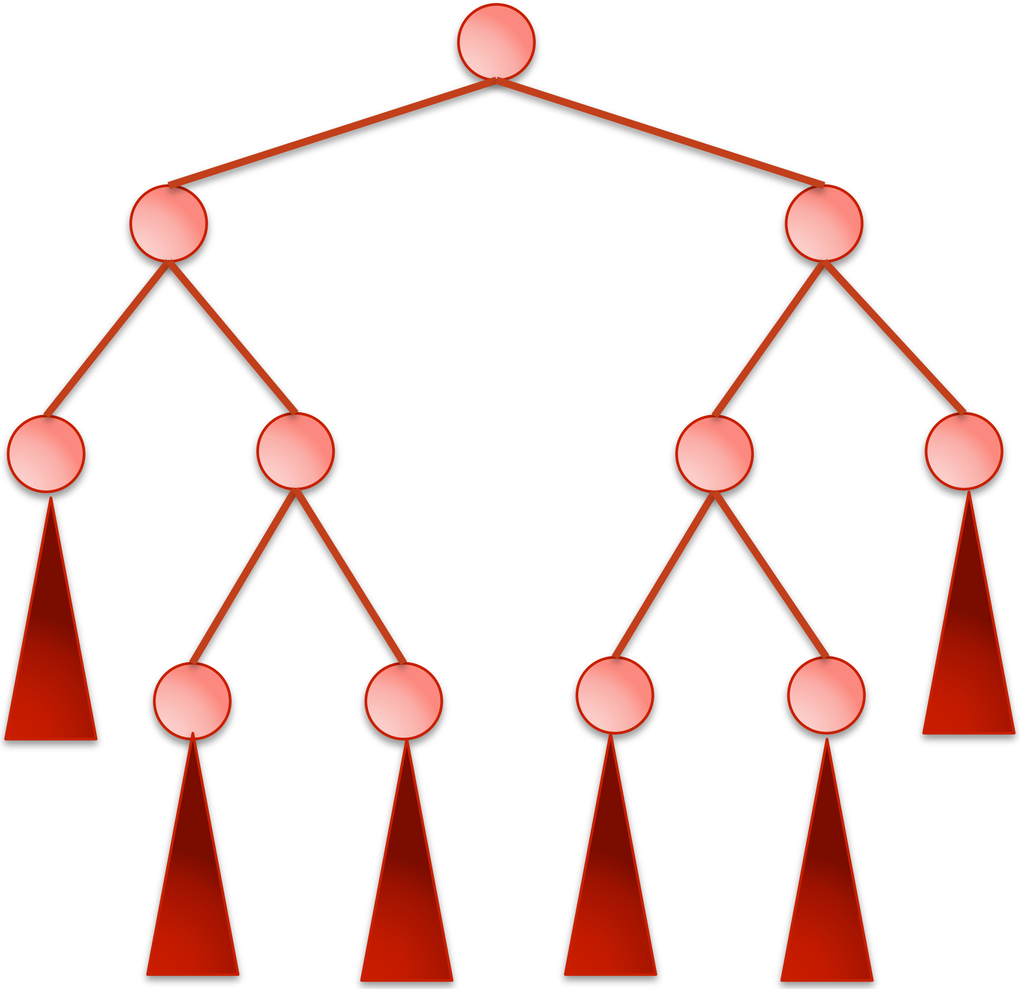


PPL UIUC

Allowing RTS to Reconfigure Apps

- We can push adaptivity further
  - With a collaboration between RTS and programmer
- The programmer:
  - Exposes some knobs (control-points) to the RTS
  - Describes their effects in a standard "language"
- The RTS:
  - Observes the runtime behavior,
  - Optimizes what it can without reconfiguration
  - When needed, asks app to reconfigure by choosing the right knob and direction

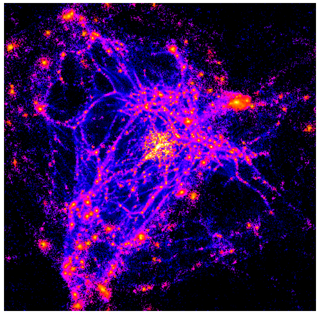
ChaNGa: Cosmology Simulation

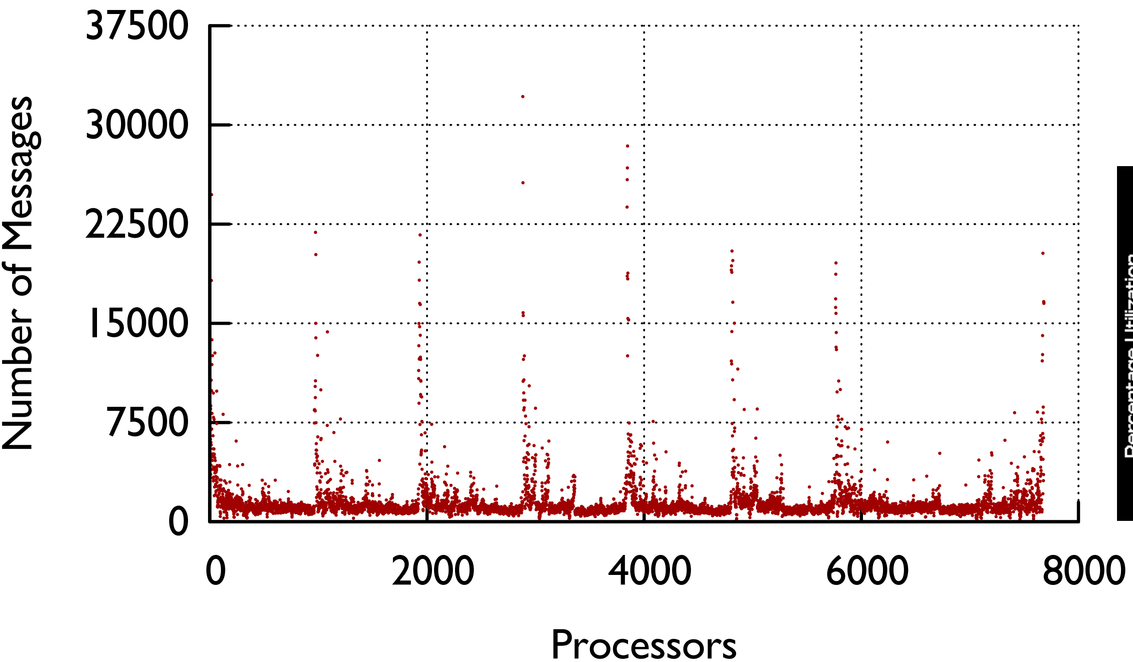


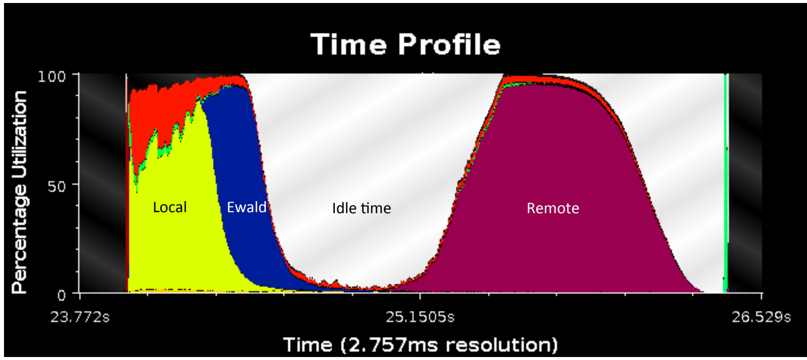
Collaboration with
Tom Quinn UW

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

Clustered Dataset – Dwarf



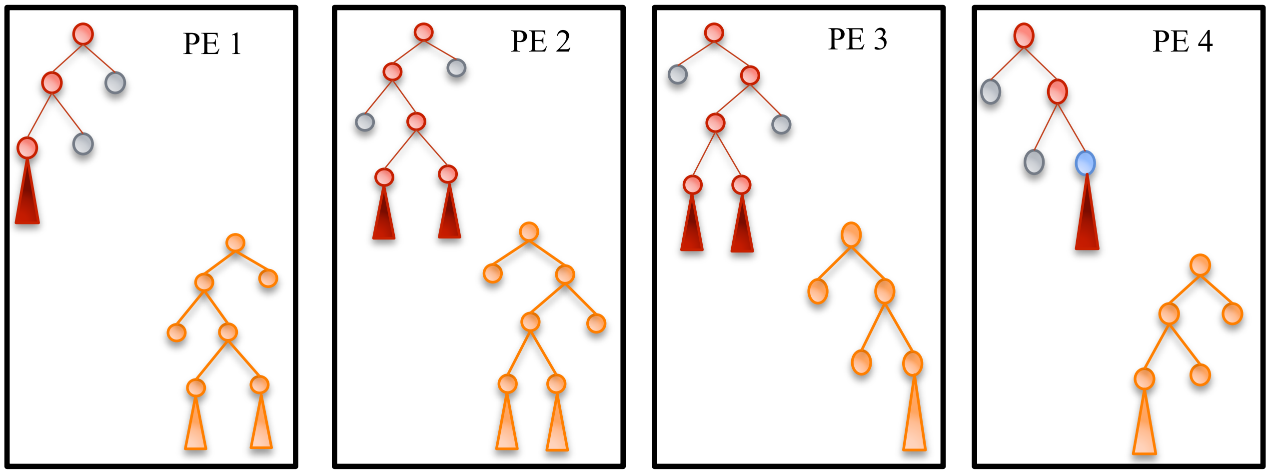




- Highly clustered
- Maximum request per processor: > 30K

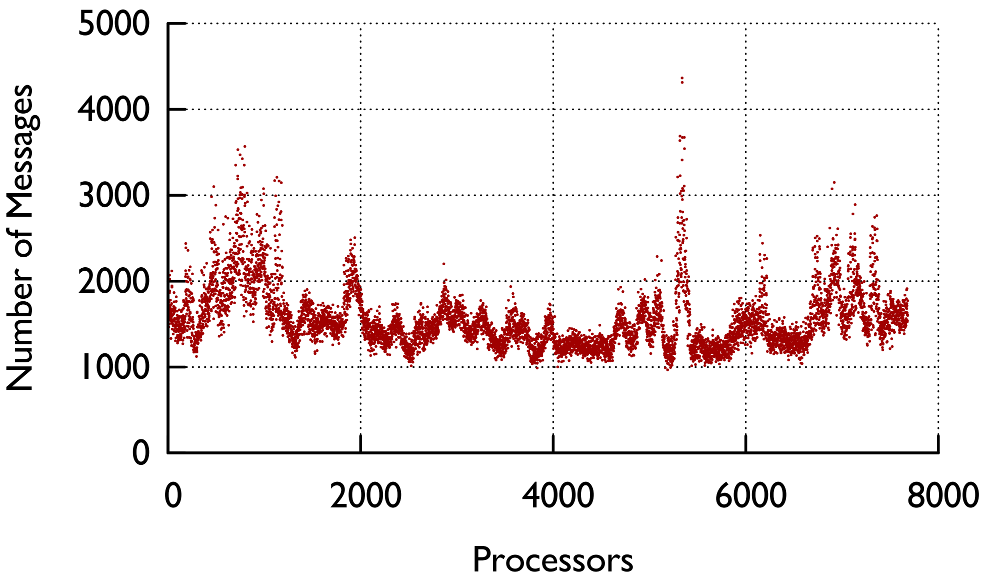
- Idle time due to message delay

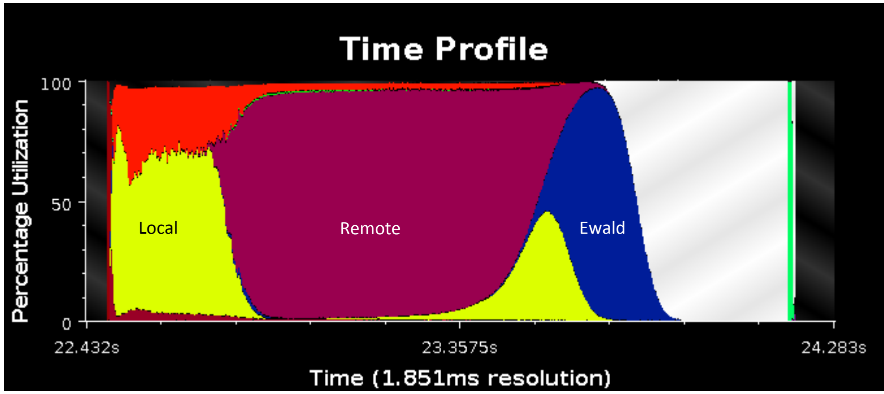
Solution: Replication

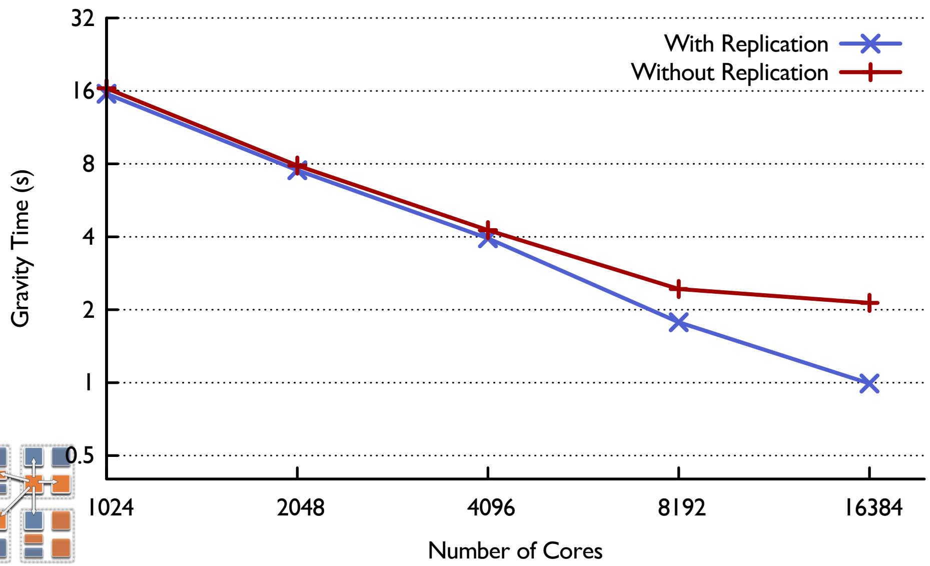


- Replicate tree nodes to distribute requests
- Requester randomly selects a replica

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

Control Point for Replication?

- This optimization can be turned into a control point via an abstraction
  - For data
    - That doesn't change during a phase, and
    - Is requested based on a key
  - The RTS can then observe and decide / tune
    - If replication is needed,
    - Which objects to replicate
    - Degree of replication
- It turns out to be of general use:
  - A cloth simulation, with collision detection, also can use it

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

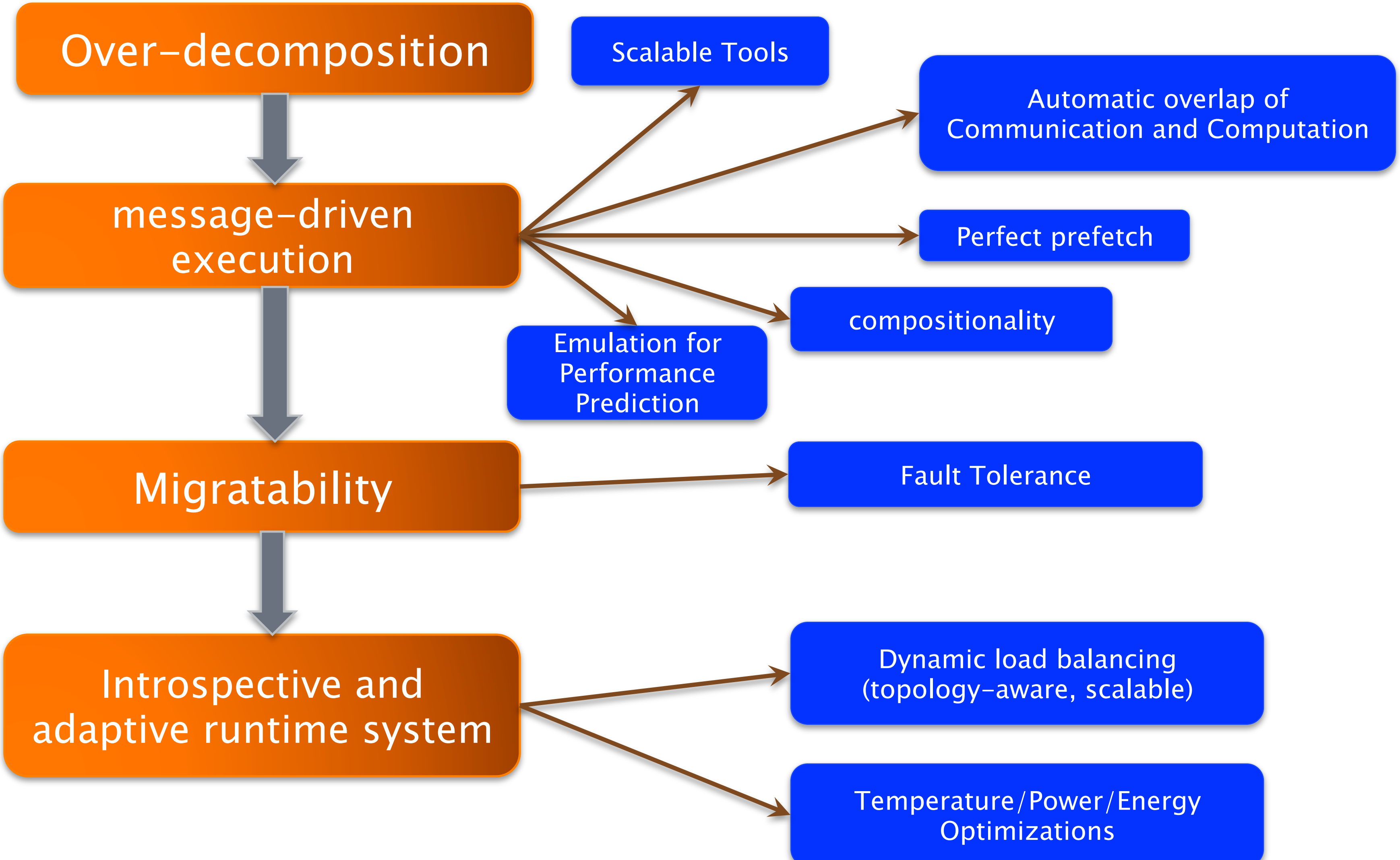
So, What are the Action Items

- Explore overdecomposition in your application
  - Without using any RTS
- Increase the asynchrony in your app
- Add migratability in small measures
  - But you will need to do some location management yourself
- Try coding a small module using an existing adaptive RTS
  - E.g. Charm++ modules work with MPI modules
- Create control points for runtime manipulation
- Get used to words like "continuations"..
  - But we need only simpler versions of those

Experiment with Languages/Libraries that support these concepts

- Programming models that exhibit some features
  - Charm++
  - Adaptive MPI
  - KAAPI
  - ProActive
  - FG-MPI (if it adds migration)
  - mpC
  - HPX (once it embraces migratability)
  - StarPU
  - ParSEC
  - CnC
  - MSA (multi-phase Shared arrays)
  - Charisma
  - Charj
  - Chapel: may be a higher level model
  - X10: has asynchrony, but not migratable units
- So, pick some of them to start experimenting w miniApps

Benefits in Charm++



Summary

- Adaptive Runtime Systems are coming
- Advice to Application developers
  - Get familiar with:
  - Do I need to repeat?
  - Overdecomposition, Migratability, Asynchrony
  - Experiment with new models that support these and are interoperable
    - E.g. Charm++ ☺

More info on Charm++:
http://charm.cs.illinois.edu

Charm++ workshop live webcast
http://charm.cs.illinois.edu/charmWorkshop
April 29-30 2014

Overdecomposition *Asynchrony* *Migratability*

PPL UIUC