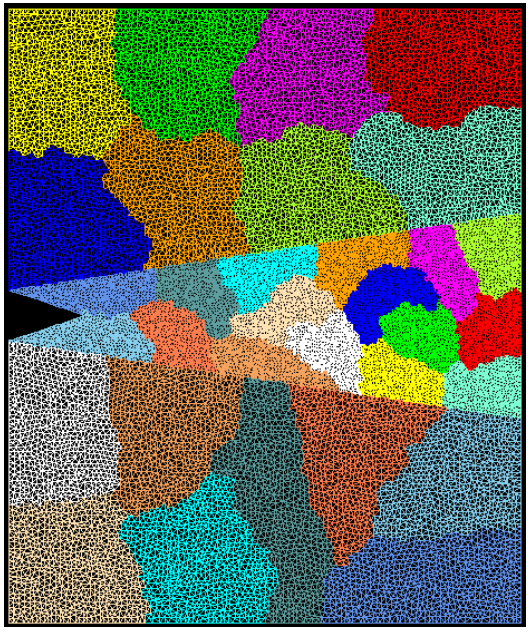
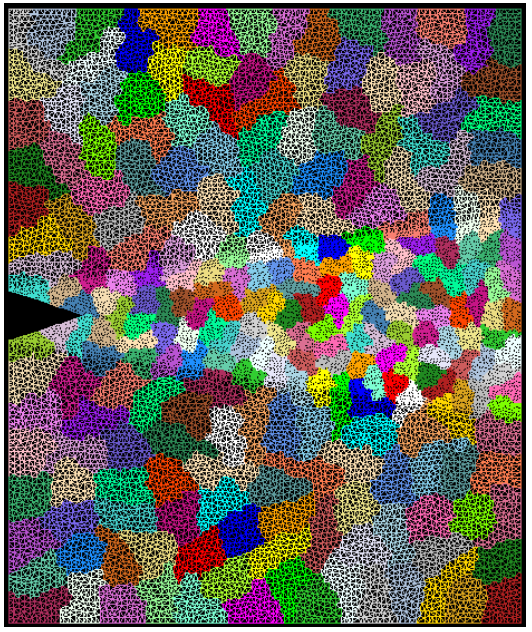
Grainsize

- Charm++ philosophy:
  - let the programer decompose their work and data into coarse-grained entities
- It is important to understand what I mean by coarse-grained entities
  - You don't write sequential programs that some system will auto-decompose
  - You don't write programs when there is one object for each *float*
  - You consciously choose a grainsize, BUT choose it independent of the number of processors
    - Or parameterize it, so you can tune later

Crack Propagation

This is 2D, circa 2002…
but shows over-decomposition for unstructured meshes..





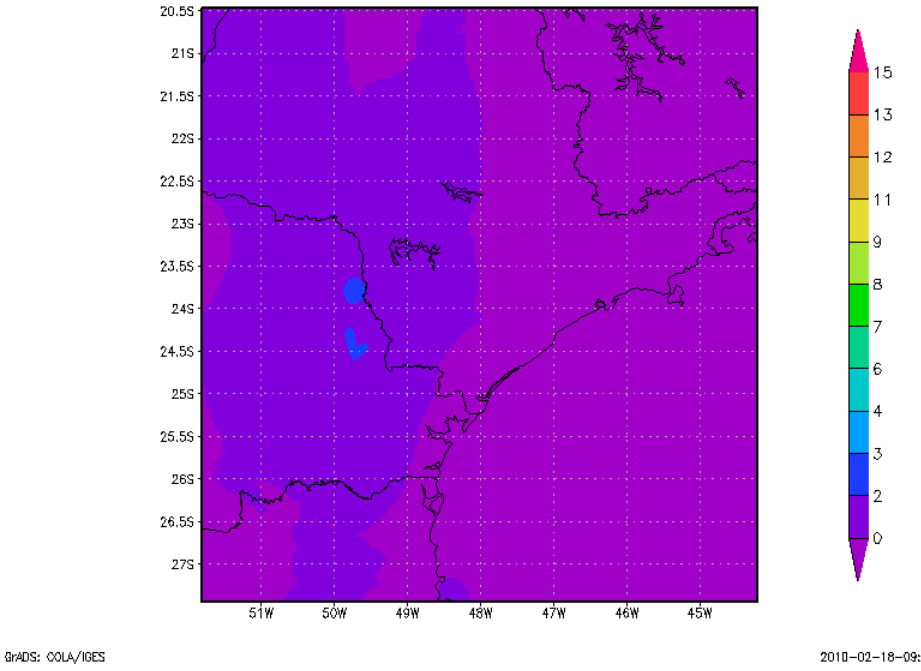
Decomposition into 16 chunks (left) and 128 chunks, 8 for each PE (right). The middle area contains cohesive elements. Both decompositions obtained using Metis. Pictures: S. Breitenfeld, and P. Geubelle

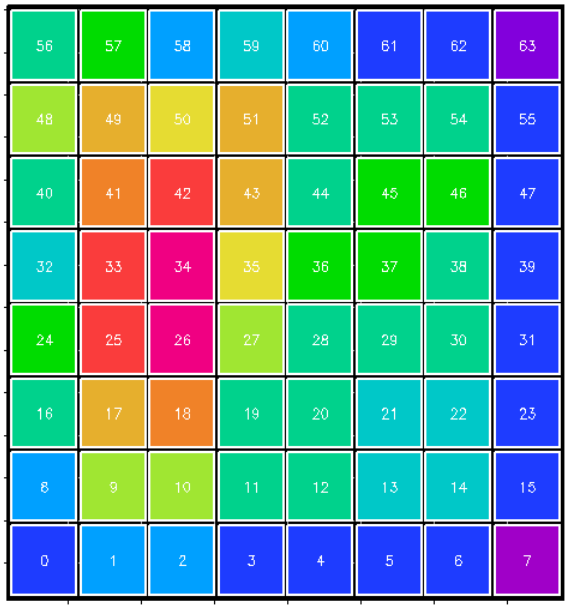
Grainsize example: NAMD

- High Performing examples: (objects are the work-data units in Charm++)
- On Blue Waters, 100M atom simulation,
  - 128K cores (4K nodes), 5,510,202 objects
- Edison, Apoa1(92K atoms)
  - 4K cores , 33124 objects
- Hopper, STMV, 1M atoms,
  - 15,360 cores, 430,612 objects

Grainsize: Weather Forecasting in BRAMS

- Brams: Brazillian weather code (based on RAMS)
- AMPI version (Eduardo Rodrigues, with Mendes , J. Panetta, ..)

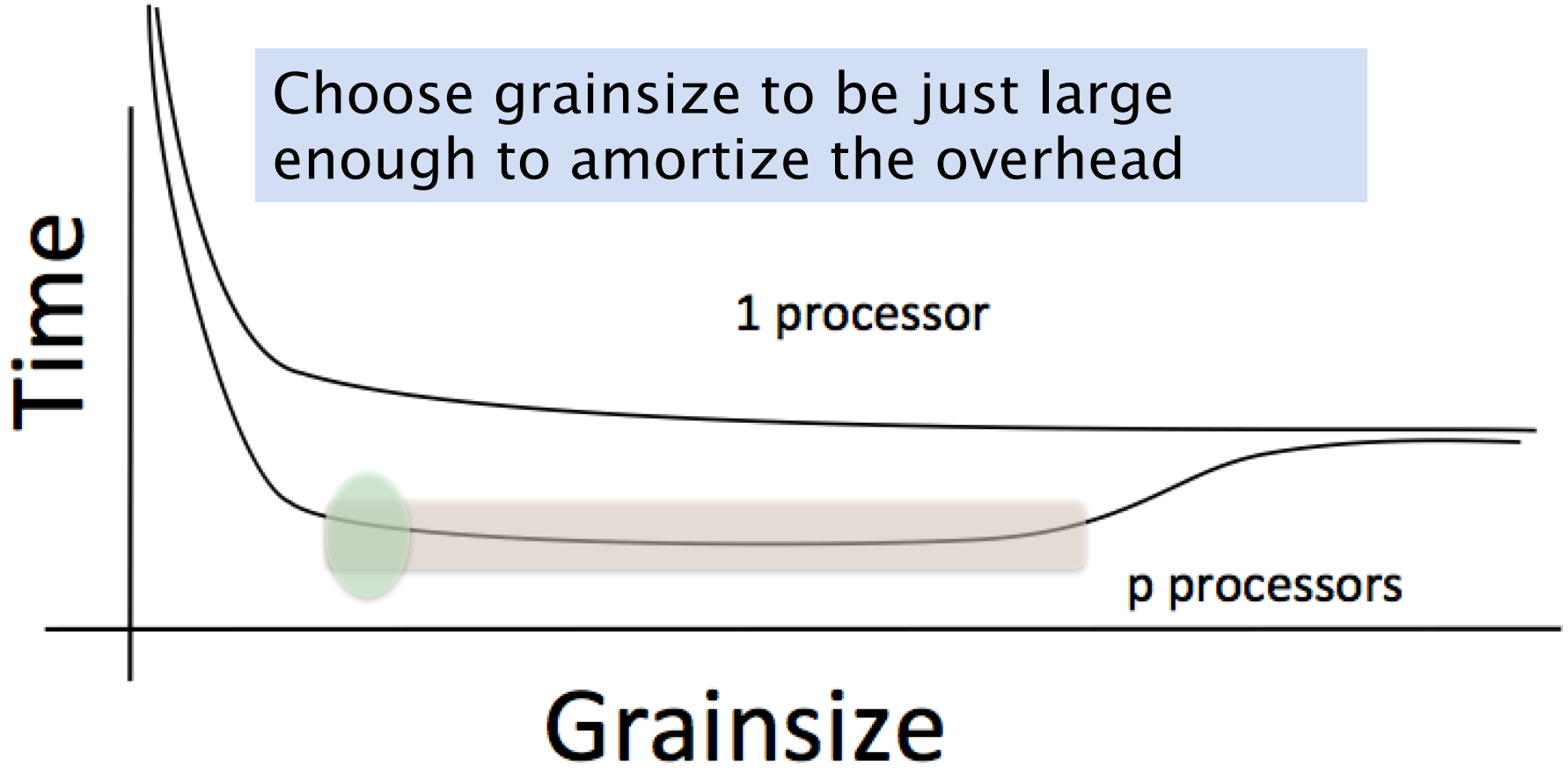




Instead of using 64 work units on 64 cores, used 1024 on 64

PPL UIUC

Working definition of grainsize :
amount of computation per remote interaction



PPL
UIUC

Grainsize and Overhead

- What is the ideal grainsize?
- Should it depend on the number of processors?

$$T_1 = T\left(1 + \frac{v}{g}\right)$$

$$T_p = \max\left\{g, \frac{T_1}{P}\right\}$$

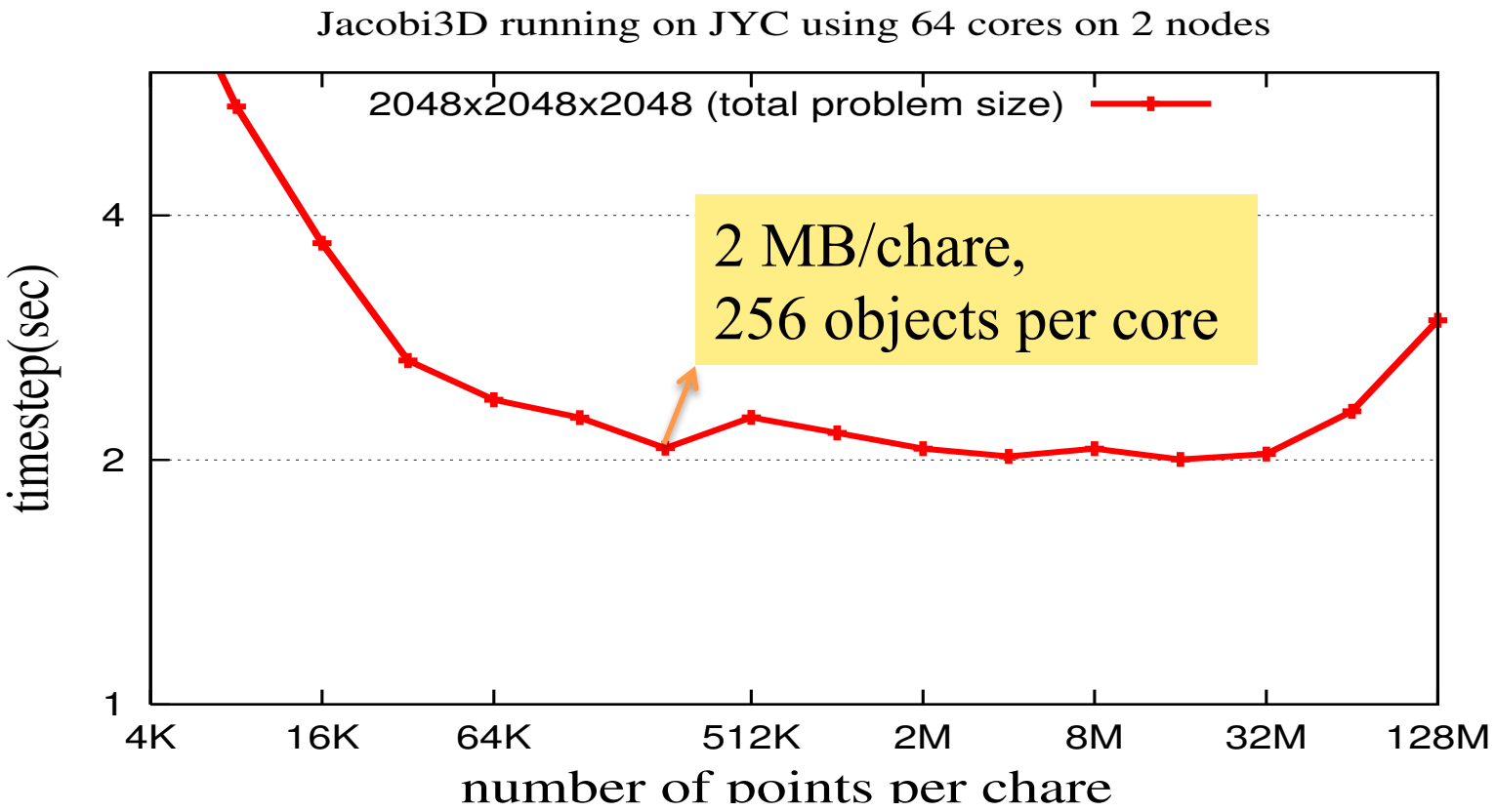
$$T_p = \max\left\{g, \frac{T*\left(1 + \frac{v}{g}\right)}{P}\right\}$$

V: overhead per message

Tp: p processor completion time

G: grainsize (computation per message)

Grainsize in a common setting



PPL UIUC

Rules of thumb for grainsize

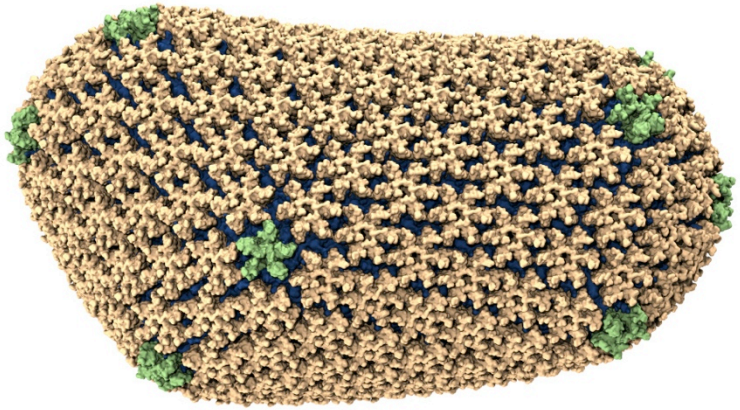
- Make it as small as possible, as long as it amortizes the overhead
- More specifically, ensure:
  - *Average* grainsize is greater than k•v (say 10v)
  - No single grain should be allowed to be too large
    - Must be smaller than T/p, but actually we can express it as
      - Must be smaller than k•m•v (say 100v)
- Important corollary:
  - You can be at close to optimal grainsize without having to think about P, the number of processors

Charm++ Applications as case studies

Only brief overview today

NAMD: Biomolecular Simulations

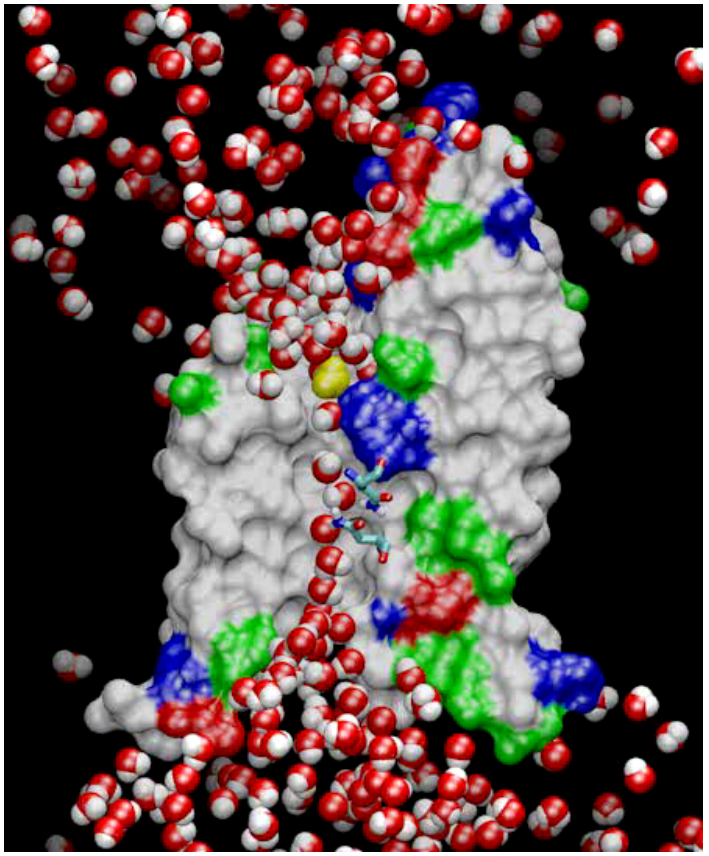
- Collaboration with K. Schulten
- With over 50,000 registered users
- Scaled to most top US supercomputers
- In production use on supercomputers and clusters and desktops
- Gordon Bell award in 2002



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

Molecular Dynamics: NAMD

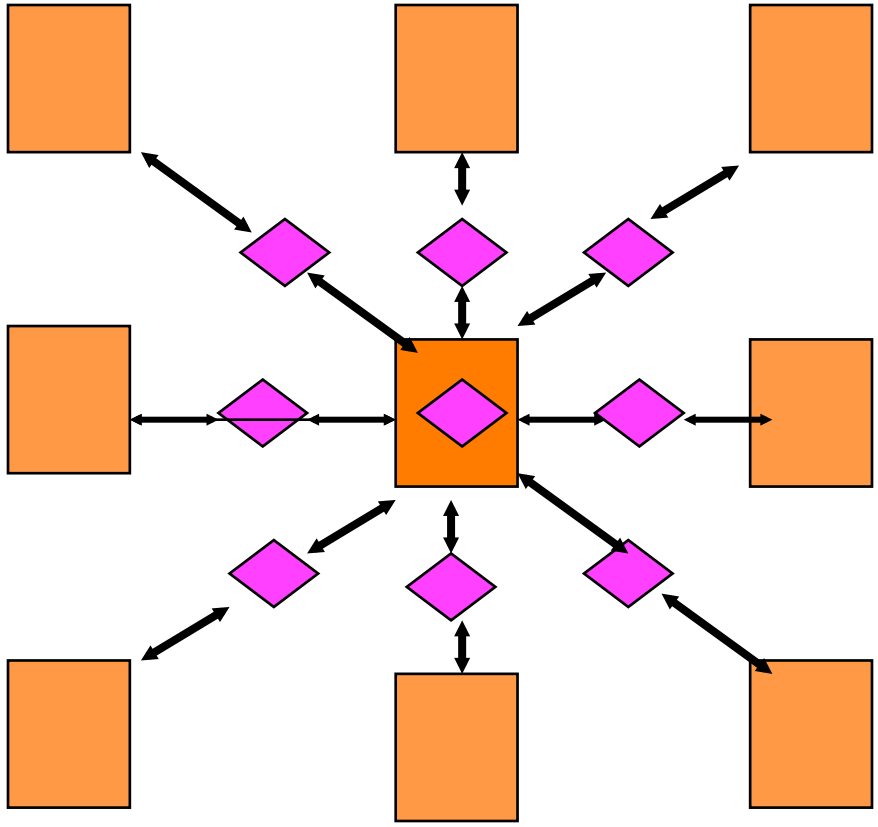
- Collection of [charged] atoms
  - With bonds
  - Newtonian mechanics
  - Thousands to millions atoms
- At each time-step
  - Calculate forces on each atom
    - Bonds
    - Non-bonded: electrostatic and van der Waal's
      - Short-distance: every timestep
      - Long-distance: using PME (3D FFT)
      - Multiple Time Stepping : PME every 4 timesteps
  - Calculate velocities
  - Advance positions



Challenge: femtosecond time-step, millions needed!

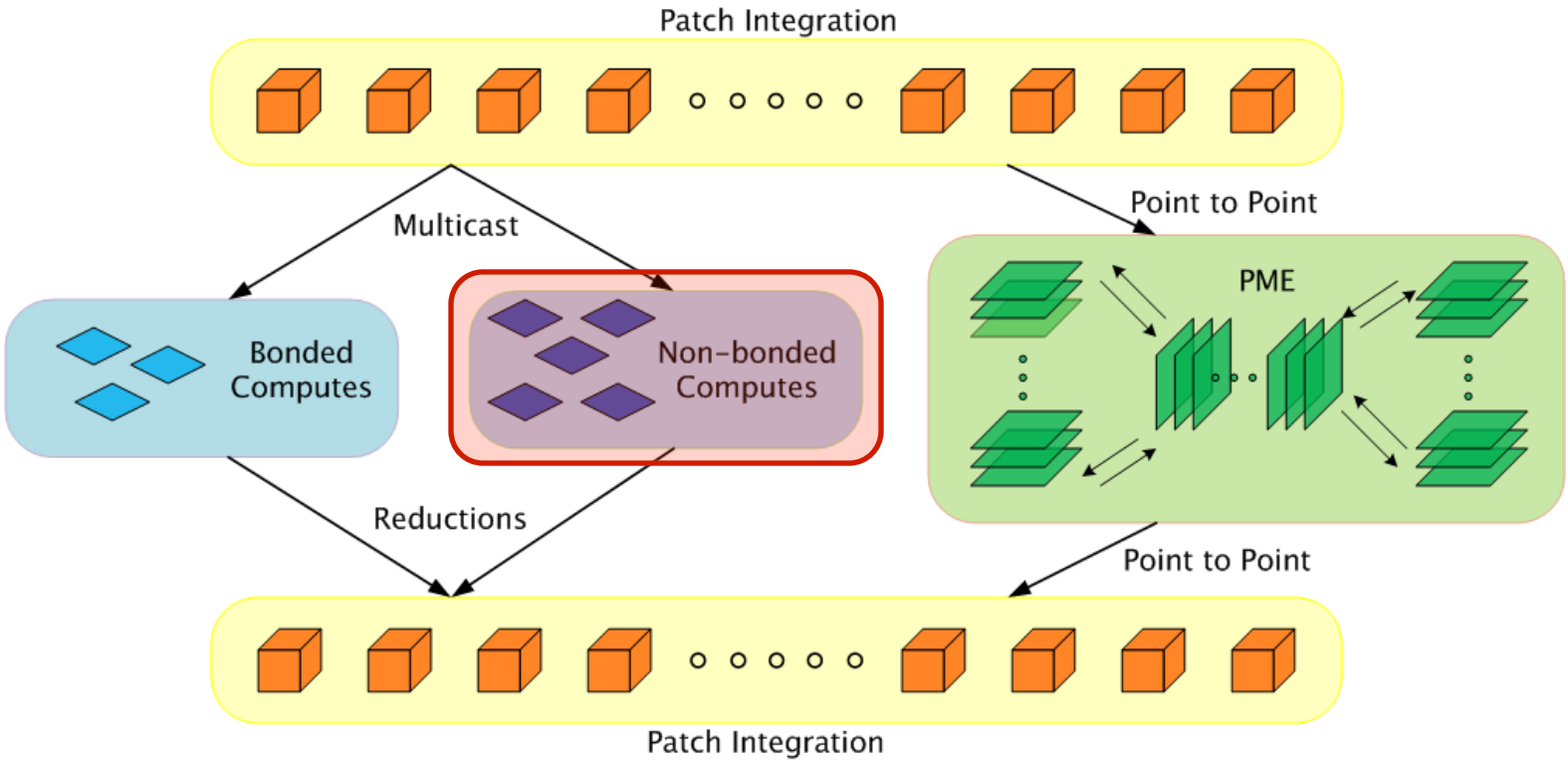
Hybrid Decomposition

Object Based Parallelization for MD: Force Decomp. + Spatial Decomp.



- We have many objects to load balance:
  - Each diamond can be assigned to any proc.
  - Number of diamonds (3D):
  - 14·Number of Cells

Parallelization using Charm++

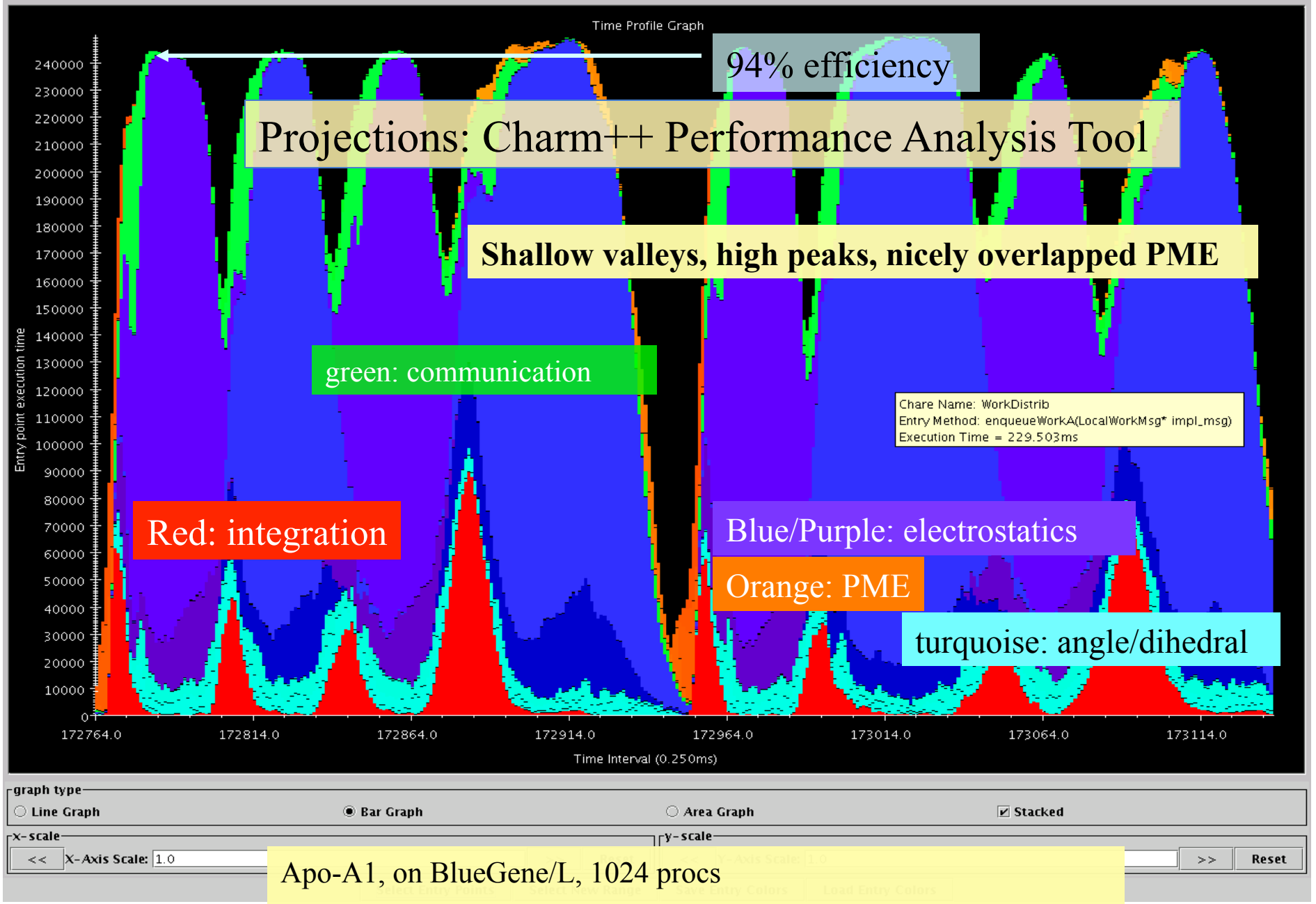


Sturdy design!

- This design,
  - done in 1995 or so, running on 12 node HP cluster
- Has survived
  - With minor refinements
- Until today
  - Scaling to 500,000+ cores on Blue Waters!
  - 300,000 Cores of Jaguar, or BlueGene/P

We are developing the NAMD project, for instance, on a cluster of twelve HP 735/125 workstations connected with an ATM, or asynchro-

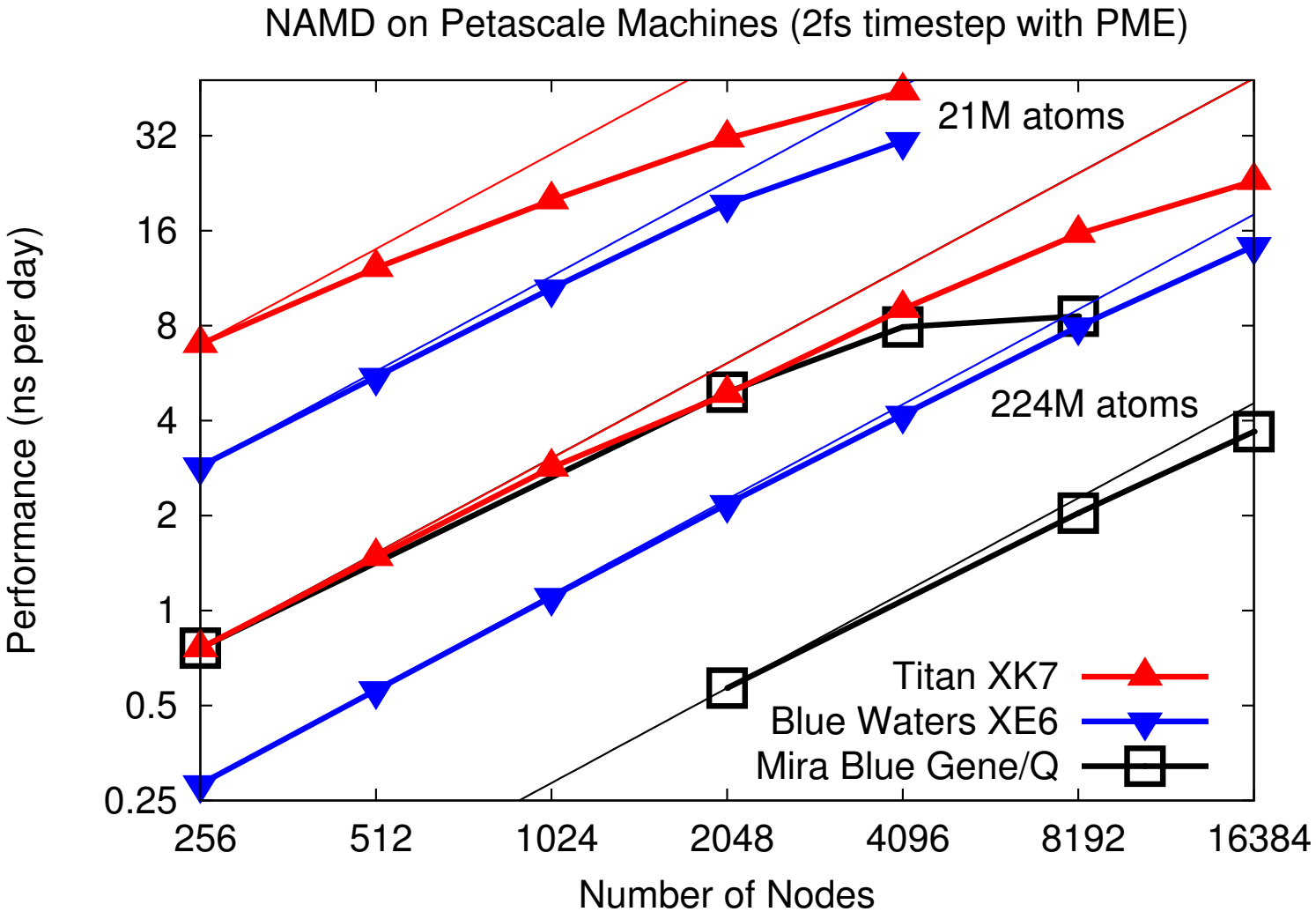
1993



Apo-A1, on BlueGene/L, 1024 procs

Time intervals on X axis, activity added across processors on Y axis

PPL UIUC



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

PPL UIUC

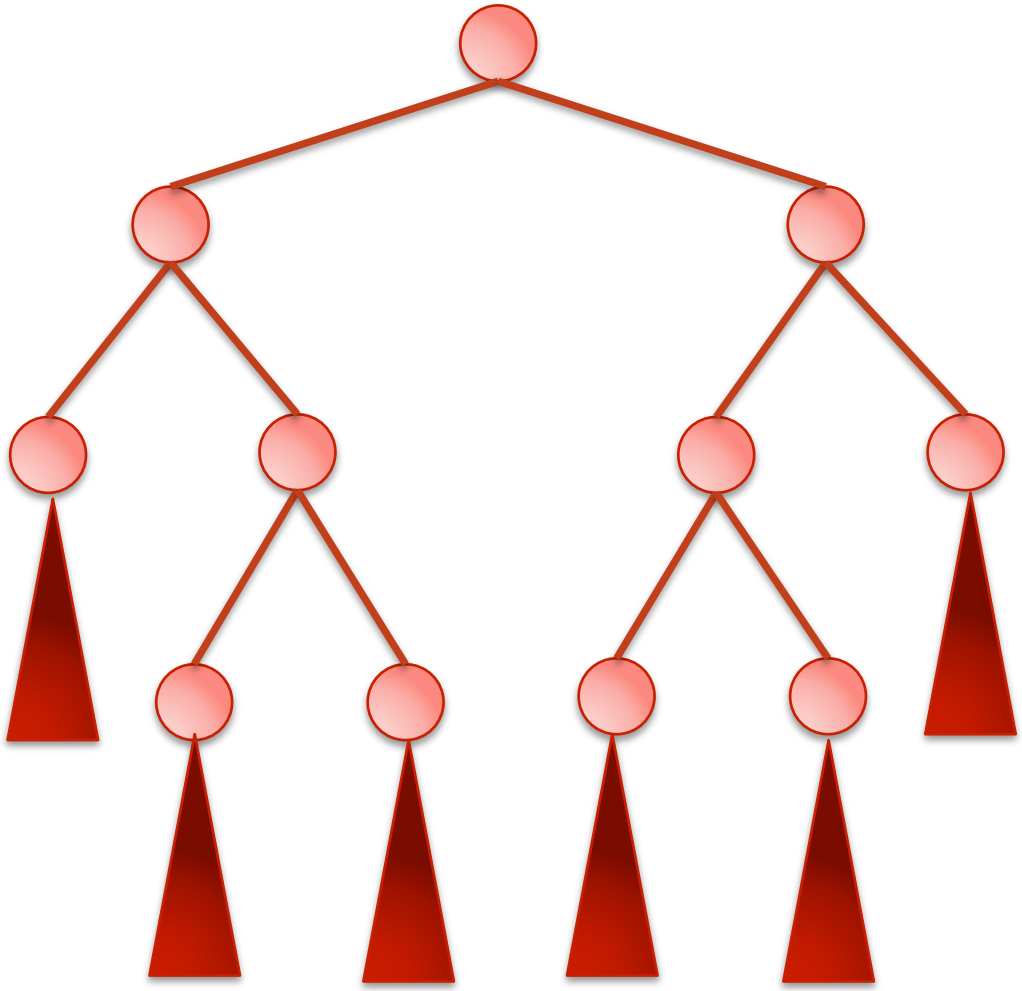
ChaNGa: Parallel Gravity

- 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

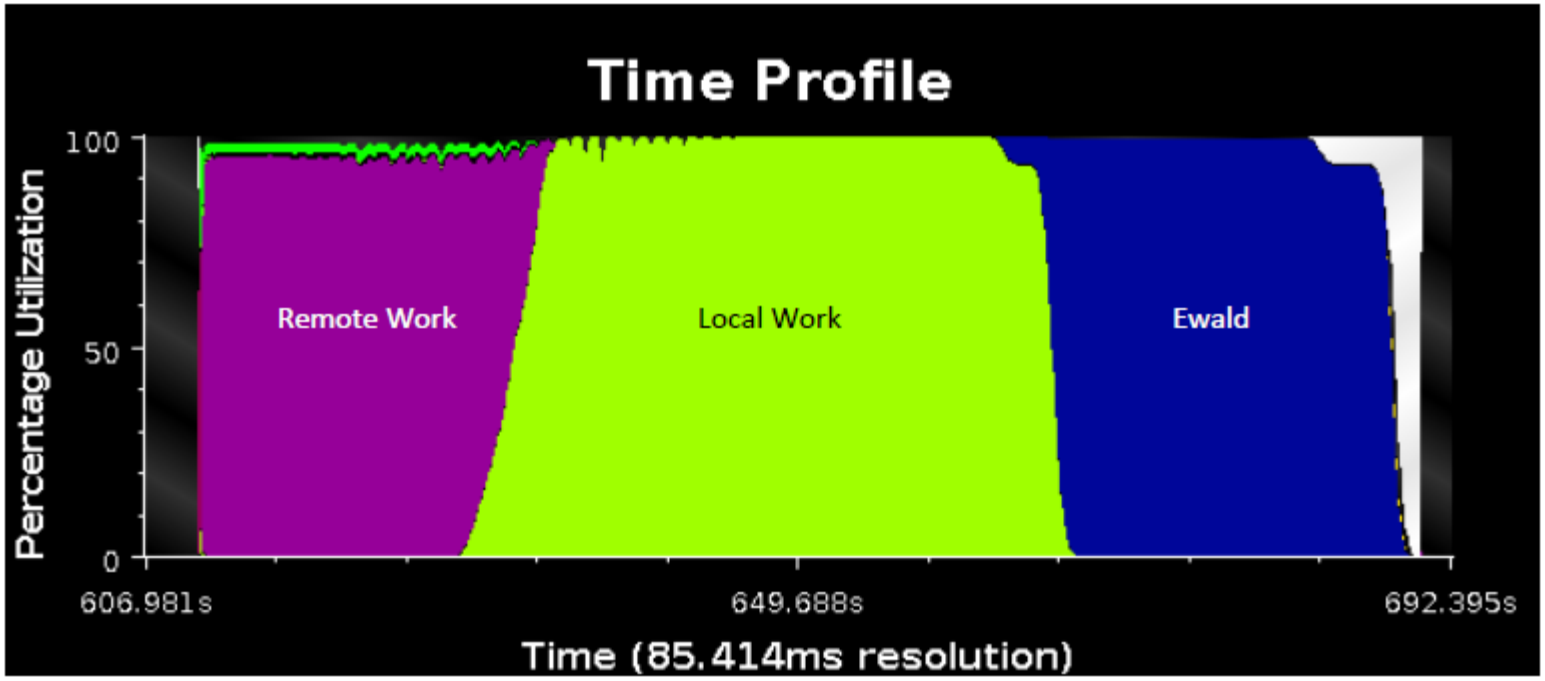


Collaboration with
Tom Quinn UW

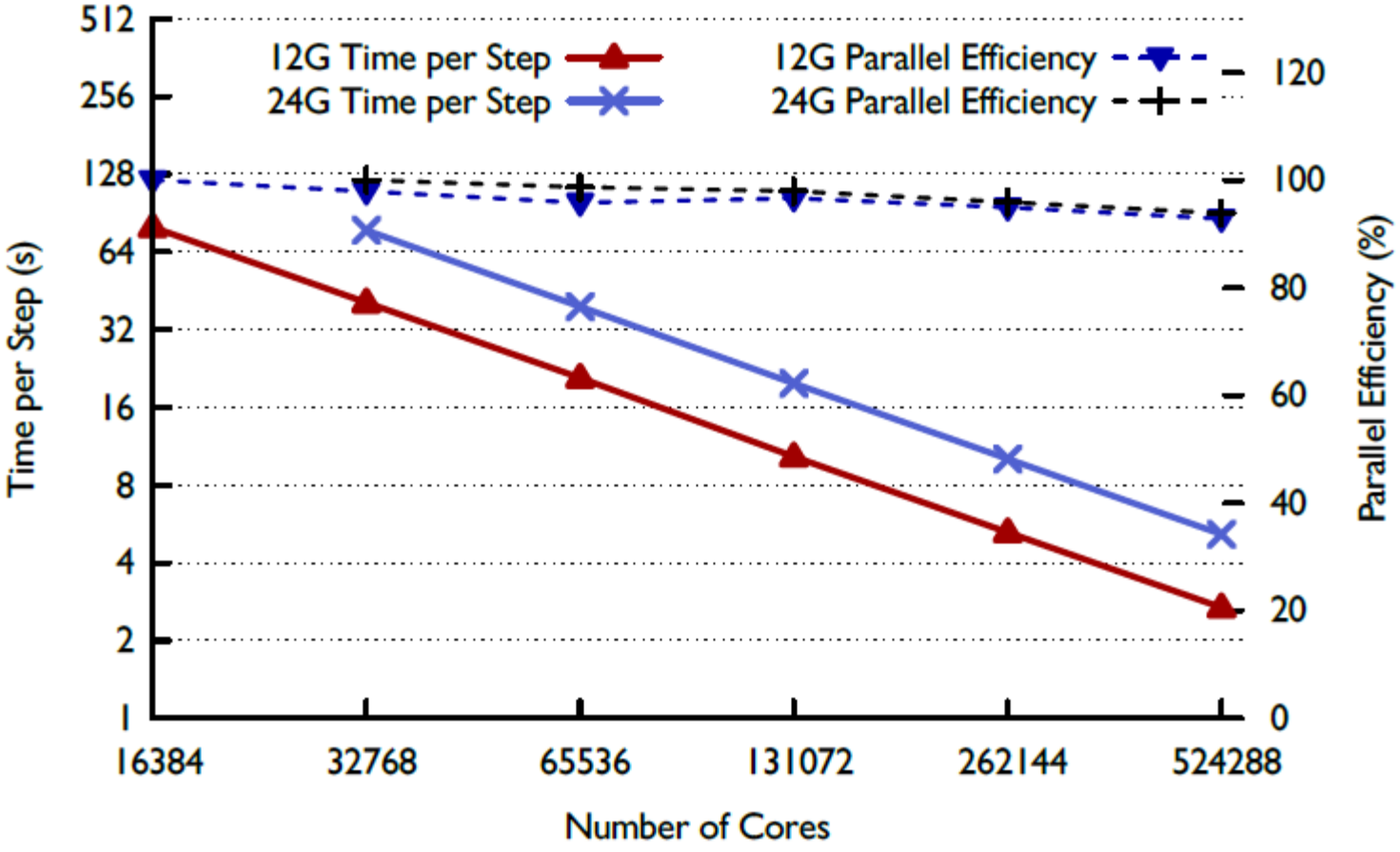
- Tree: Represents particle distribution
- TreePiece: object/ chares containing particles

ChaNGa: Optimized Performance

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



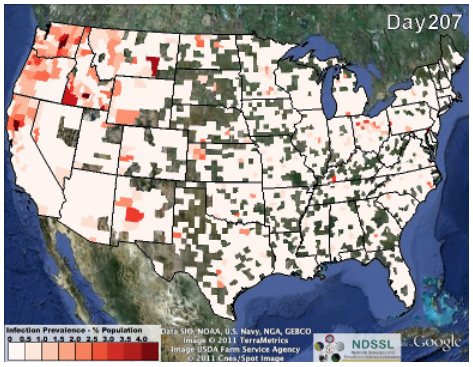
ChaNGa : a recent result

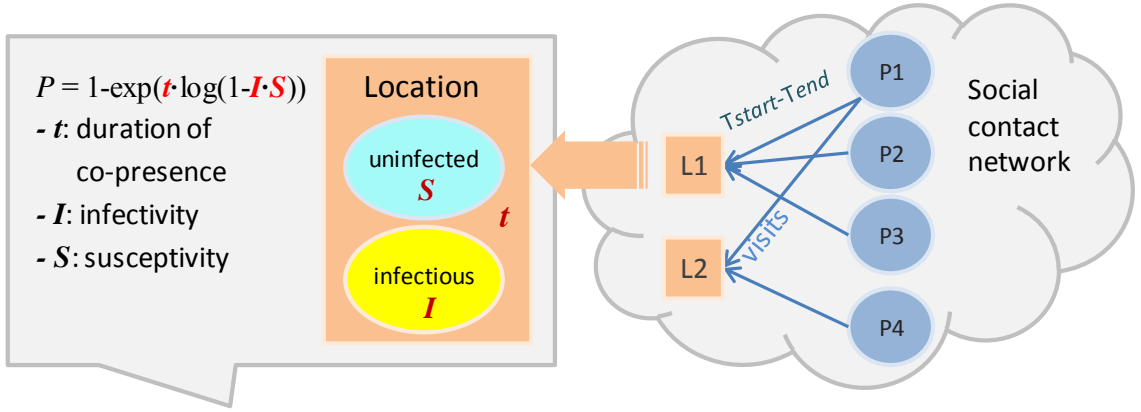


Episimdemics

- 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

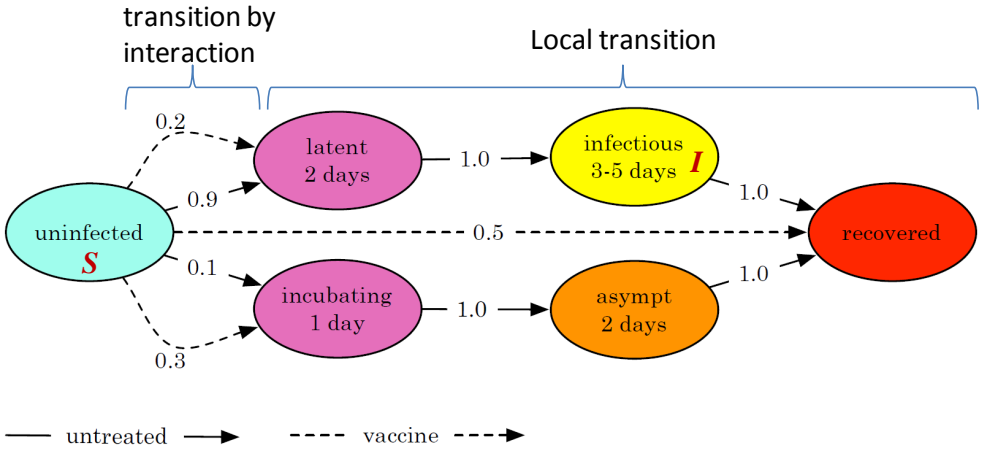
Simulating contagion over dynamic networks





EpiSimdemics[1]

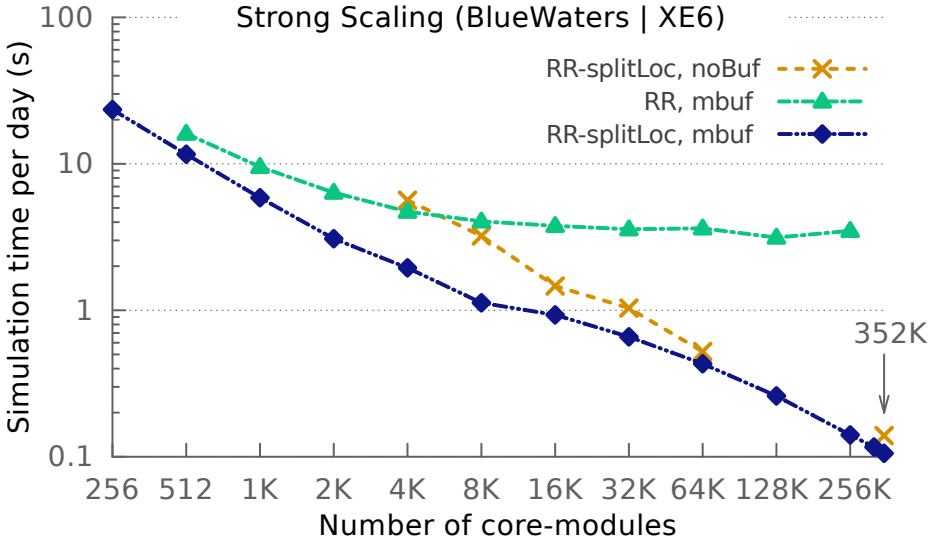
- Agent-based
- Realistic population data
- Intervention[2]
- Co-evolving network, behavior and policy[2]

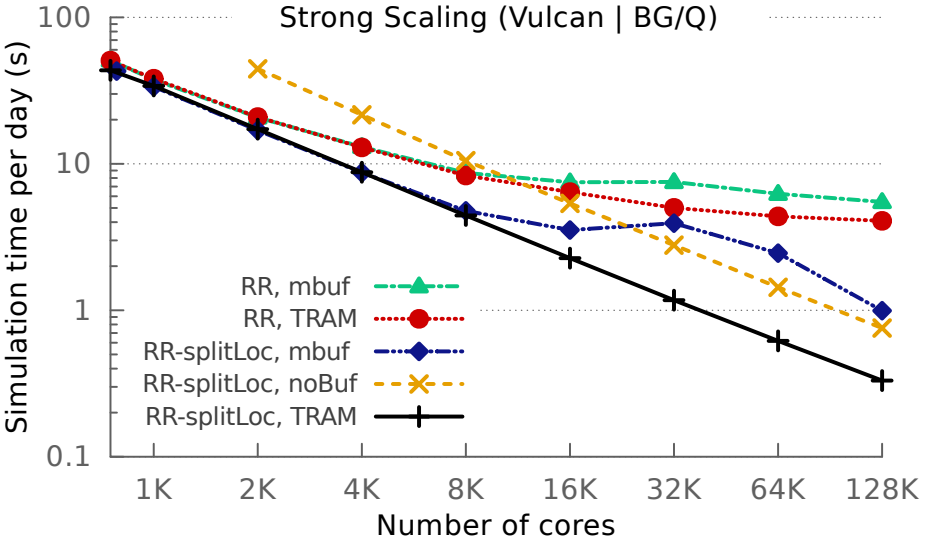


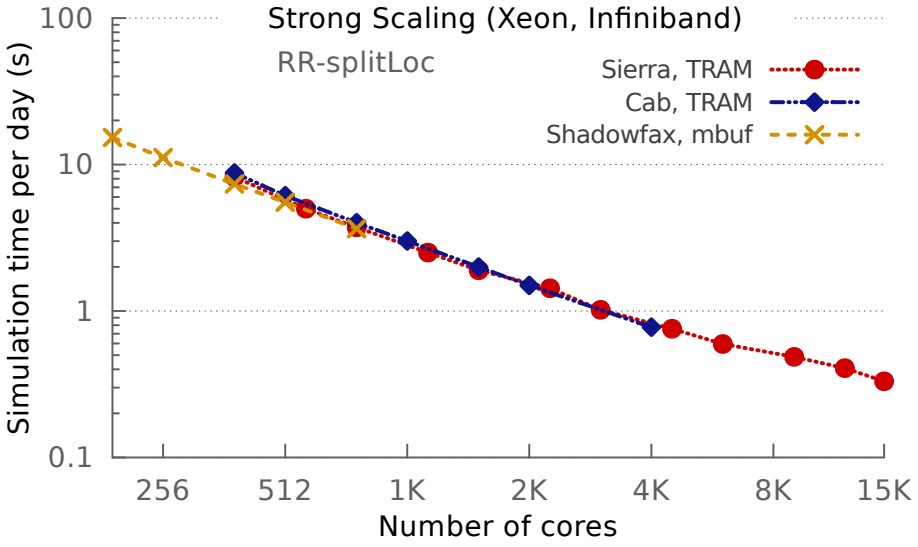
[1] C. Barrett et al.,"EpiSimdemics: An Efficient Algorithm for Simulating the Spread of Infectious Disease over Large Realistic Social Networks," SC08

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

Strong scaling performance with the largest data set







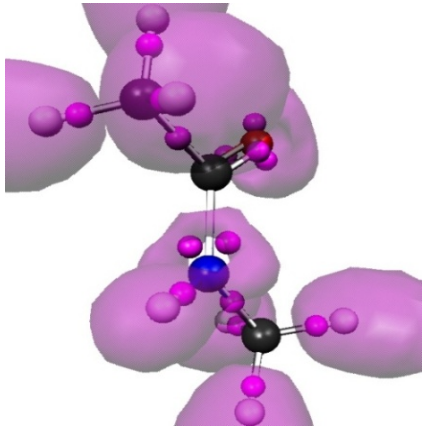
- Contiguous US population data
- **XE6**: **the largest scale** (**352K cores**)
- **BG/Q**: good scaling up to 128K cores
- Strong scaling helps timely reaction to pandemic

OpenAtom

Car-Parinello Molecular Dynamics

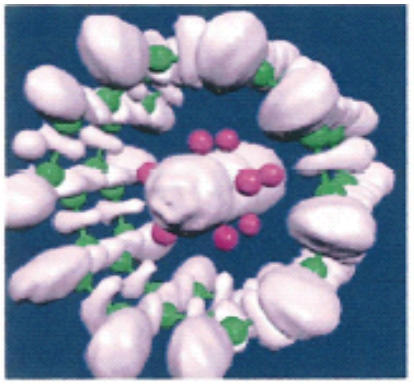
NSF ITR 2001-2007, IBM, DOE,NSF

Molecular Clusters :

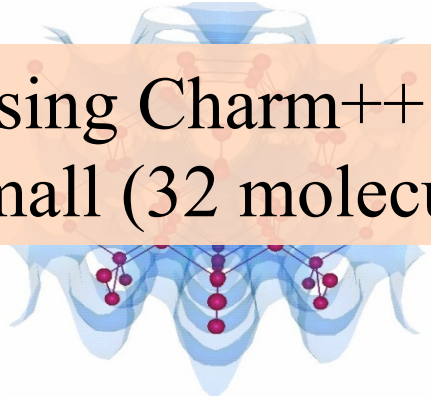


Recent NSF SSI-SI2 grant
With
G. Martyna (IBM)
Sohrab Ismail-Beigi

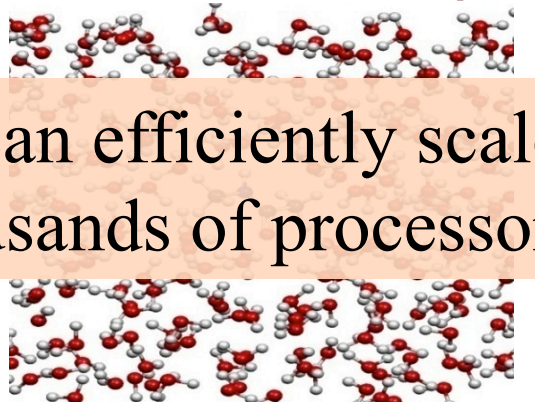
Nanowires:



Semiconductor Surfaces:

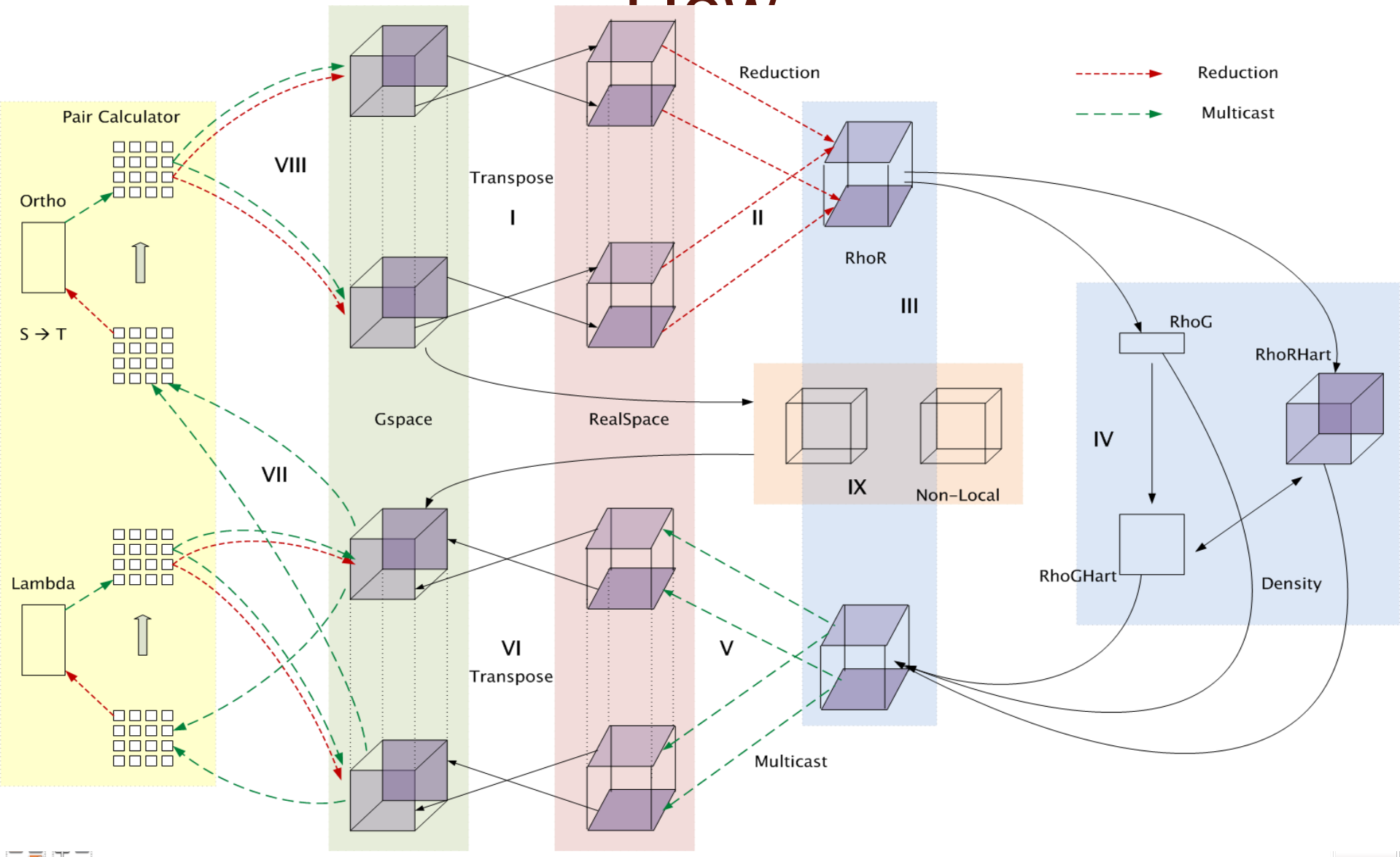


3D-Solids/Liquids:

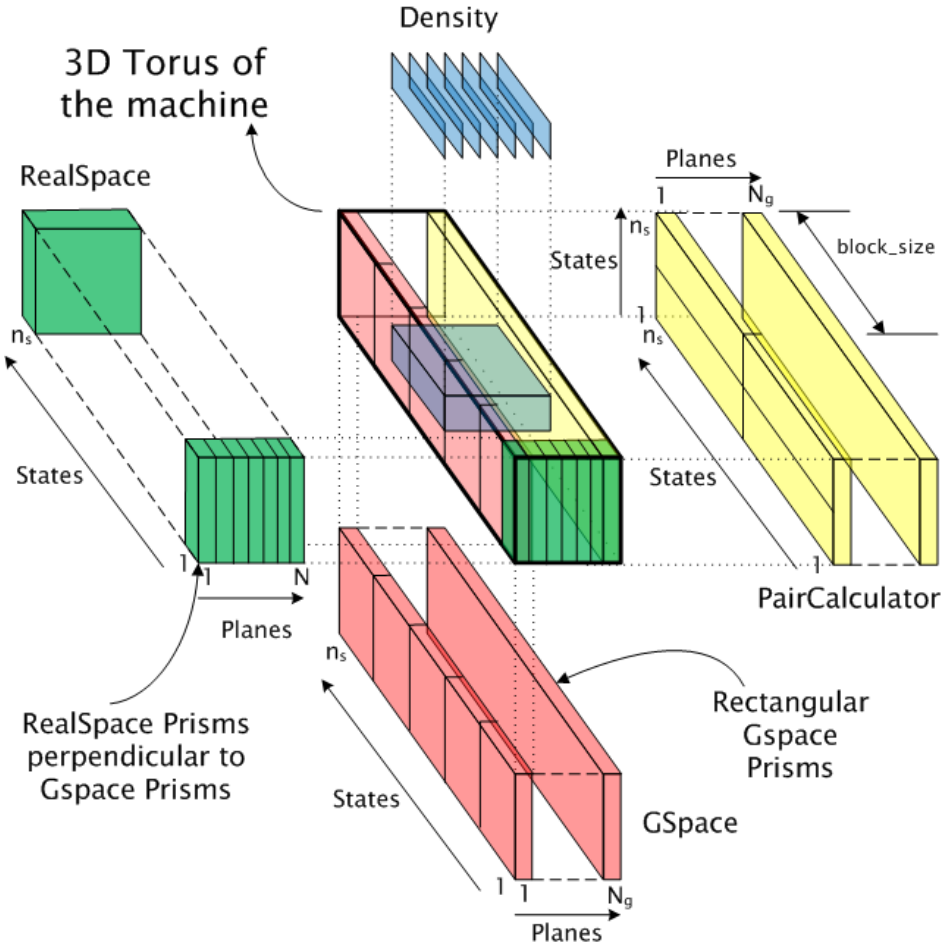


Using Charm++ virtualization, we can efficiently scale small (32 molecule) systems to thousands of processors

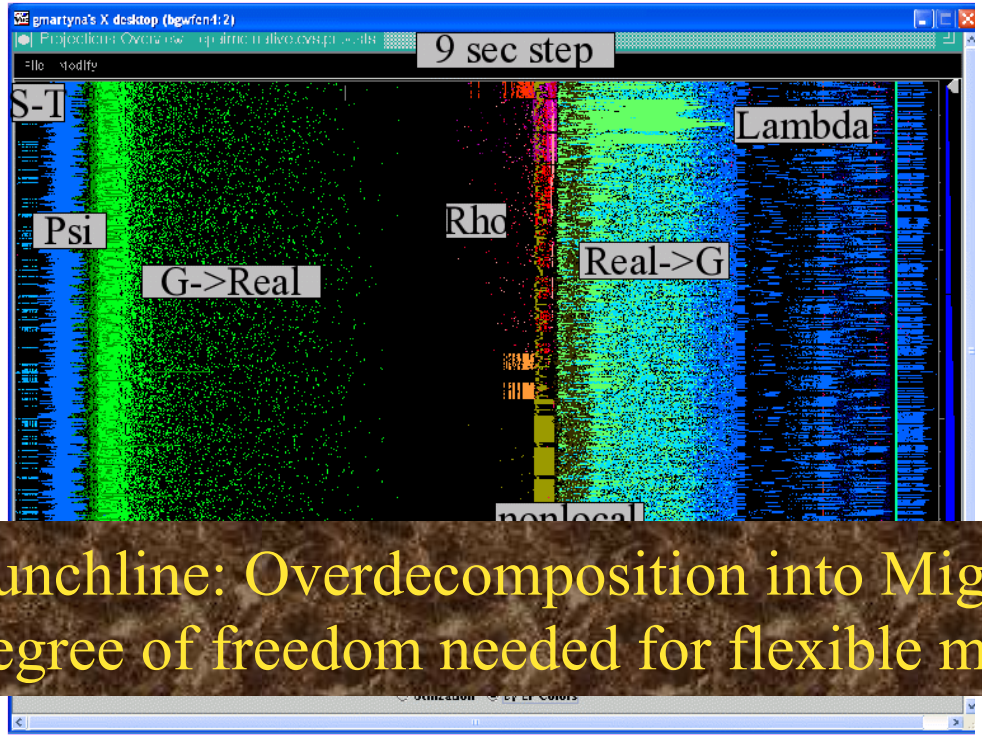
Decomposition and Computation Flow

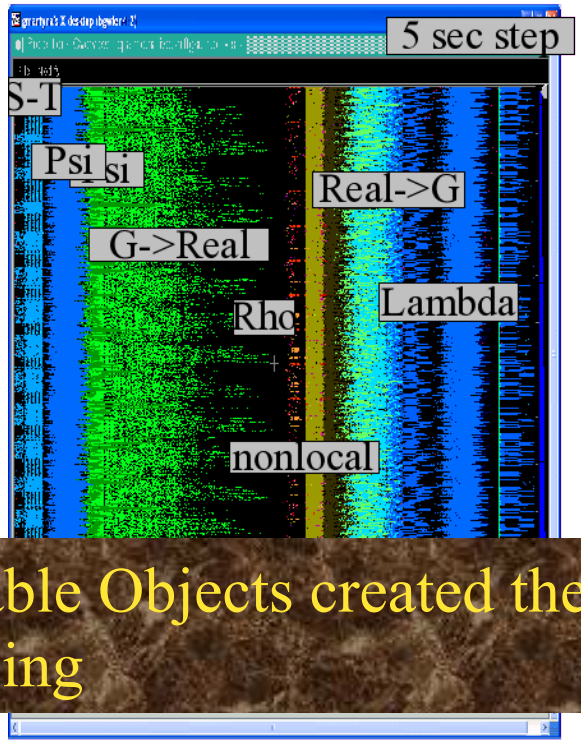


Topology Aware Mapping of Objects



Improvements by topological aware mapping of computation to processors





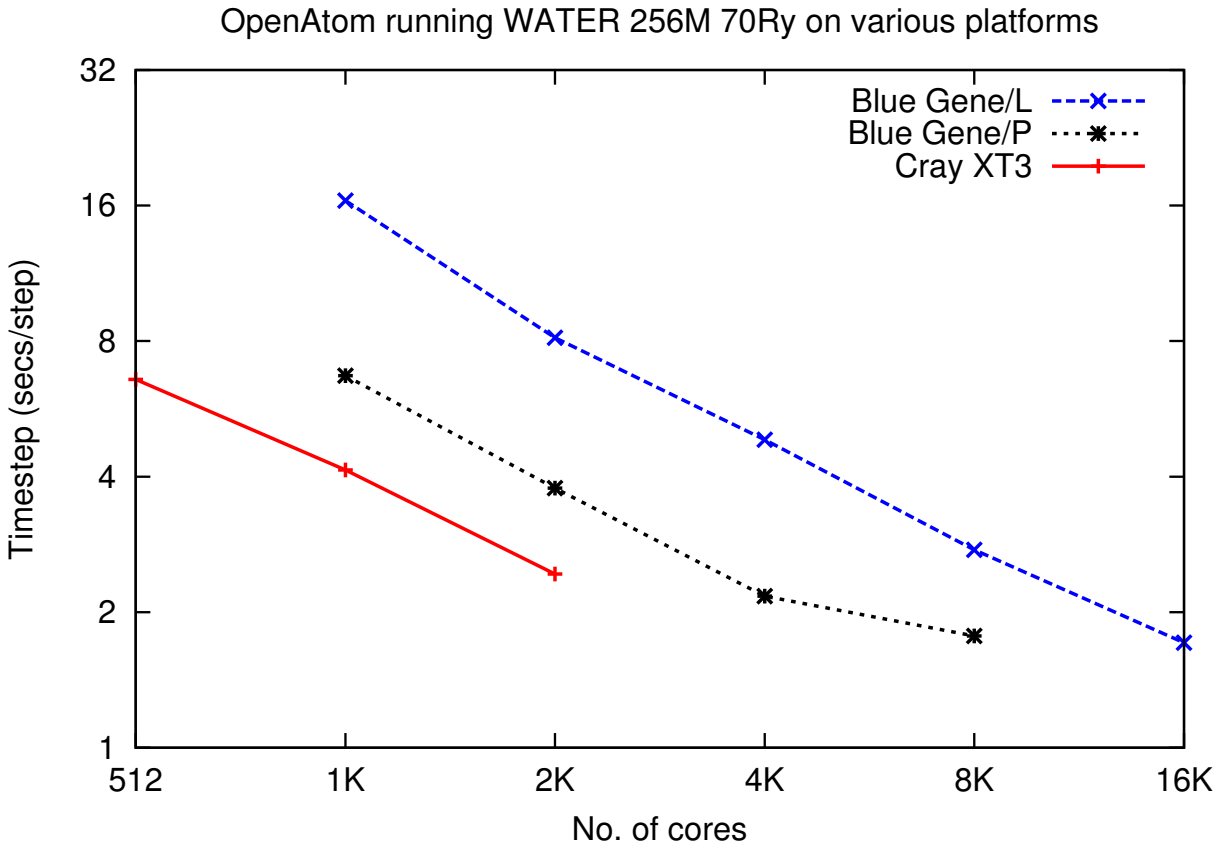
Punchline: Overdecomposition into Migratable Objects created the degree of freedom needed for flexible mapping

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)

PPL UIUC

OpenAtom Performance Sampler

Ongoing work on:
K-points



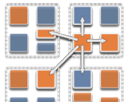
PPL UIUC

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

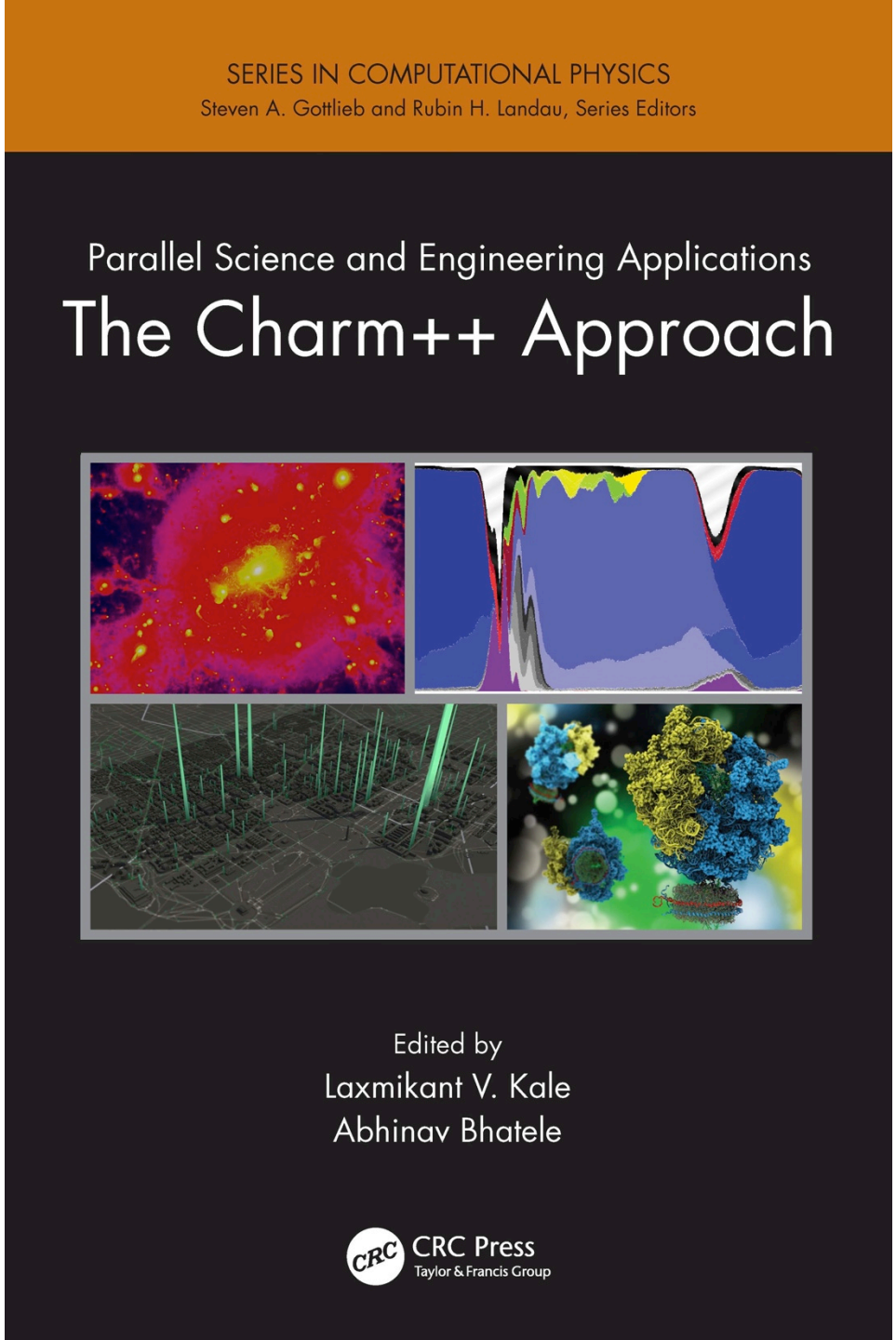
More MiniApps

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	-



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

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



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

Relevance to Exascale

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

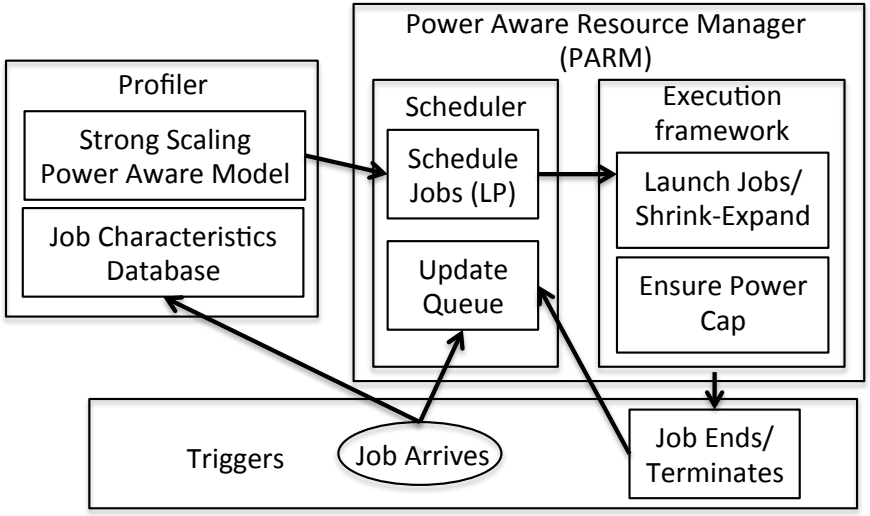
Saving Cooling Energy

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



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*