Programming Methodologies Beyond Petascale, based on adaptive runtime systems

Laxmikant (Sanjay) Kale

http://charm.cs.uiuc.edu

Parallel Programming Laboratory

Department of Computer Science

University of Illinois at Urbana Champaign

ILLINOIS

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

PARALLEL PROGRAMMING LAB

DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS

PPL UIUC

Summarizing the State of Art

- Petascale
  - Very powerful parallel computers are being built
  - Application domains exist that need that kind of power
- New generation of applications
  - Use sophisticated algorithms
  - Dynamic adaptive refinements
  - Multi-scale, multi-physics
- Challenge:
  - Parallel programming is more complex than sequential
  - Difficulty in achieving performance that scales to PetaFLOP/S and beyond
  - Difficulty in getting correct behavior from programs

PPL UIUC

Outline

- Characteristics of new generation of CSE apps
  - Climate modeling, weather prediction:
    - multi-physics, multiscale
    - Adaptively refine regions with "activity" – micro to macro transfer
  - Epidemiology , economics: model individual behavior and its impact on macroscopic factors
- Machine characteristics:
  - large-scale, heterogeneous, accelerators, interconnects,
- Challenges: resource management, compositionality

Our Guiding Principles

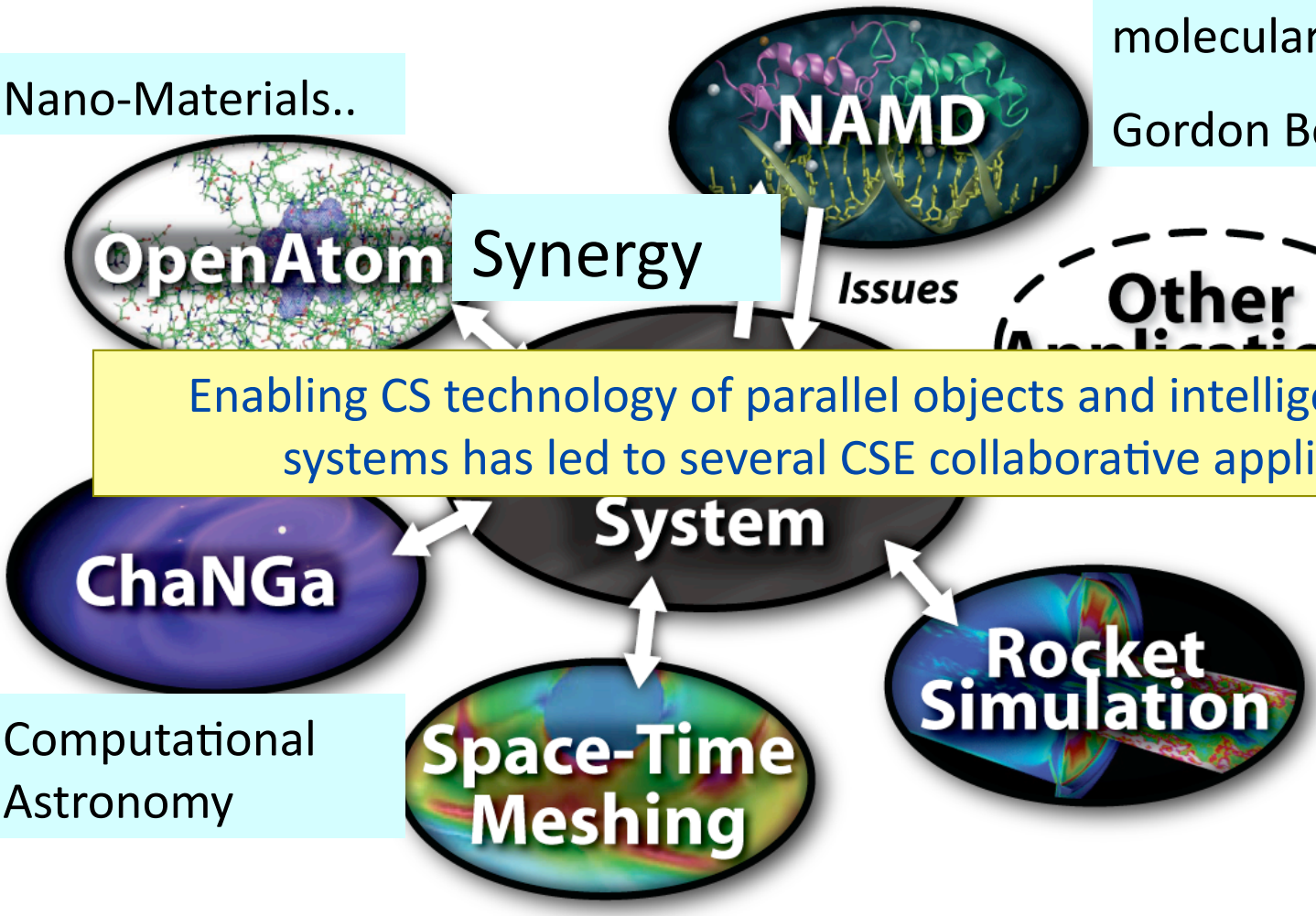
- No magic
  - Parallelizing compilers have achieved close to technical perfection, but are not enough
  - Sequential programs obscure too much information
- Seek an optimal division of labor between the system and the programmer
- Design abstractions based solidly on use-cases
  - Application-oriented yet computer-science centered approach

Charm++ and CSE Applications

Well-known Biophysics molecular simulations App

Gordon Bell Award, 2002

Nano-Materials..



Computational Astronomy

PPL UIUC

Technical challenges : outline

- Decomposition challenges
  - Multi-module, multi-paradigm, heterogenity, Compositionality
- Load balancing and resource management
  - Object based over-decomposition
- Communication challenges:
  - Asynchronous collectives, topology aware balancers,
- Scalable Performance analysis and debugging
- Pre-tuning for future machines
  - And also: tuning without access to the full machine
- Fault tolerance
- Languages: simplifying parallel programming
- Social challenges

PPL UIUC

Decomposition Challenges

- Current method is to decompose to processors
  - Problematic when you have multiple independently developed modules
  - Multi-paradigm: They may also be implemented using different paradigms, such as MPI, charm++, UPC, …
  - Even with a single module, deciding which processor does what work in detail is difficult at large scale
- Decomposition should be independent of number of processors
  - Berkeley white paper recently discovered this
  - Our design principle since early 1990's.

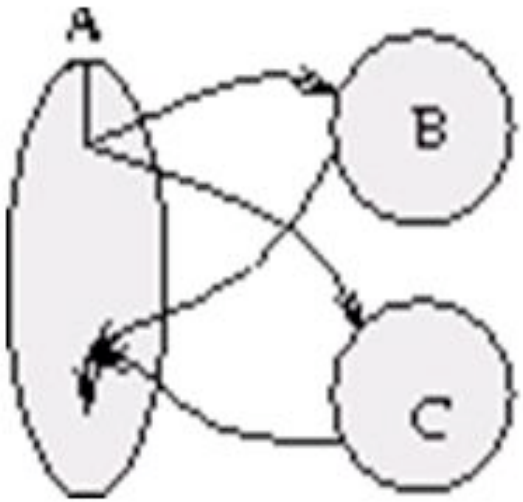
Challenge: Compositionality

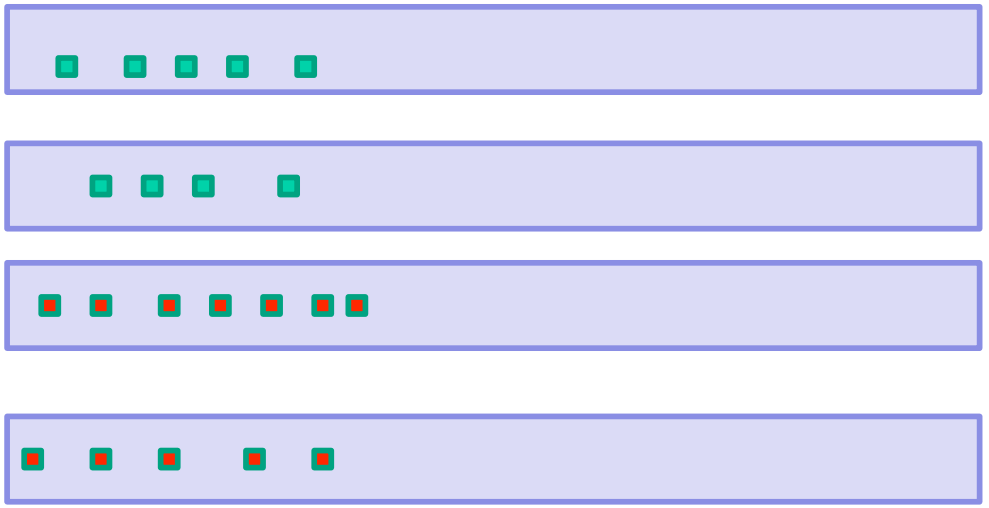
- It is important to support compositionality
  - For multi-module, multi-physics, multi-paradigm applications..
- What I mean by parallel composition
  - A || B where A and B are independently developed modules
  - A is parallel module by itself, and so is B
  - Programmers who wrote A were unaware of B and vice versa
- This is not supported by MPI
  - Developers support it by breaking abstraction boundaries
    - E.g. wildcard recvs in module A to process messages for module B
  - Nor by OpenMP implementations :

PPL UIUC

Without message-driven execution (and virtualization), you get either:

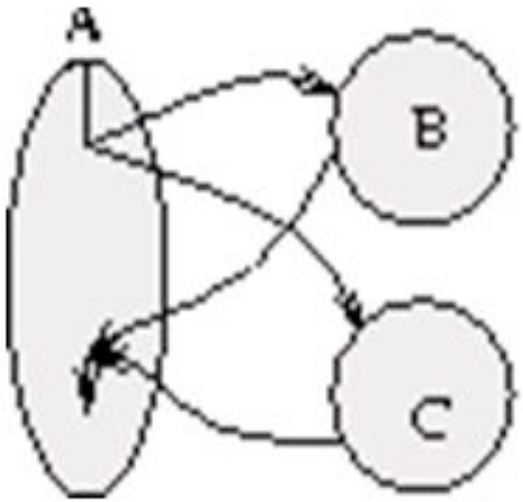
Space-division

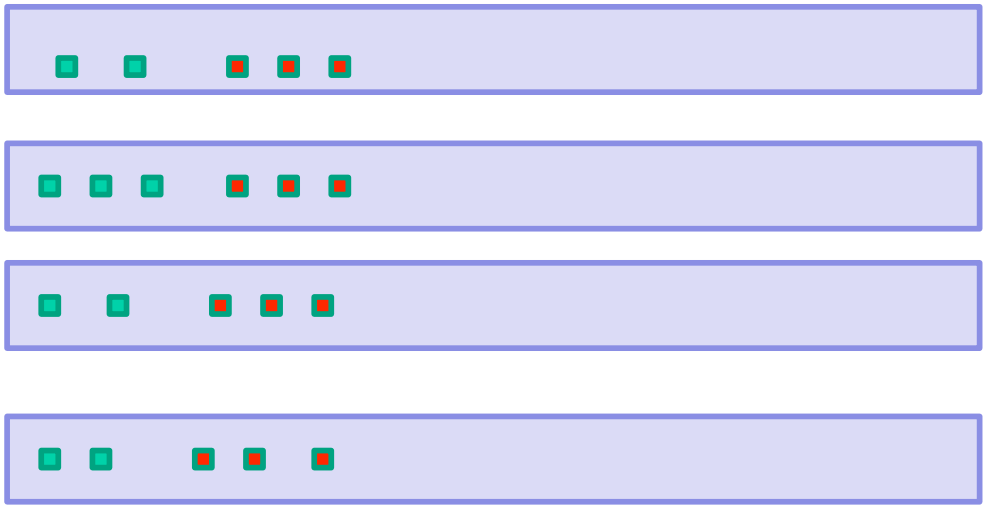




PPL UIUC

OR: Sequentialization





Challenge: heterogeneity

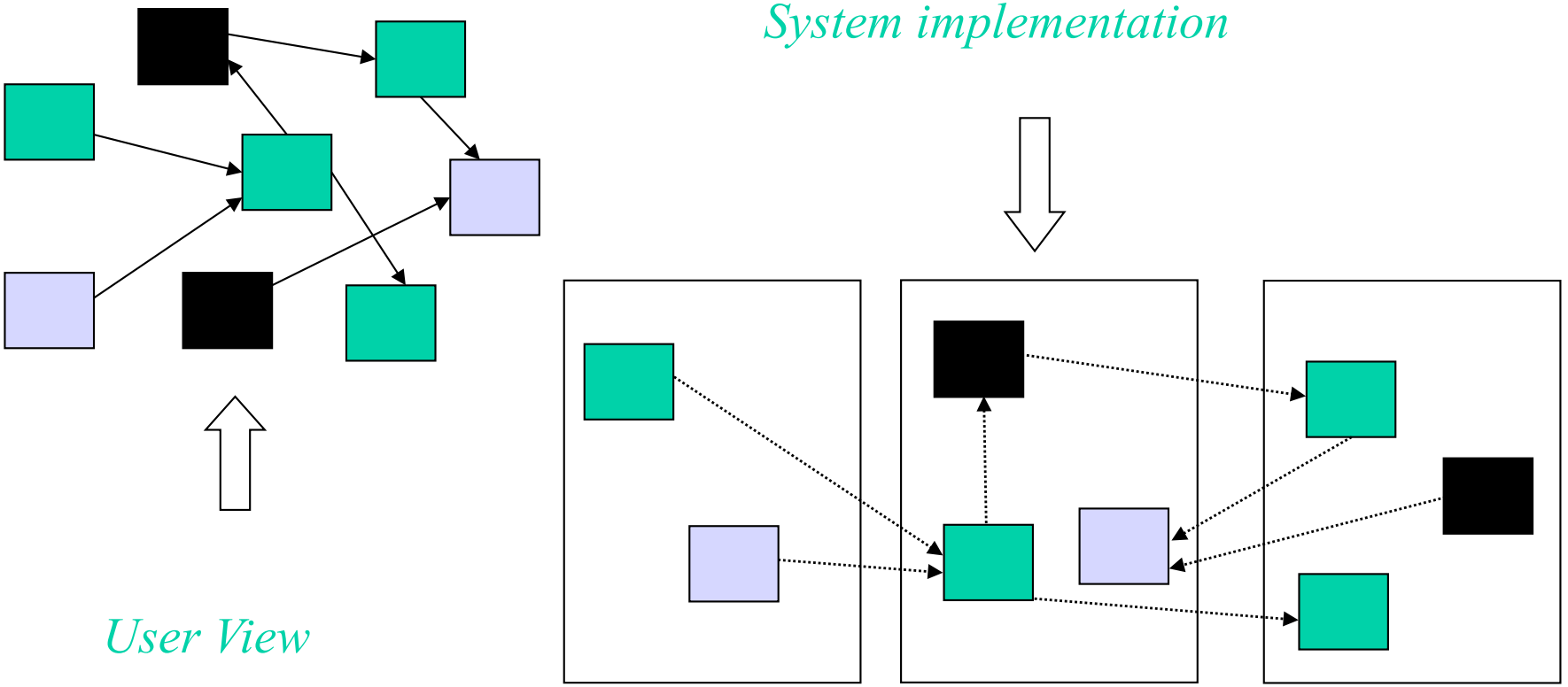
- Clusters of SMP nodes
  - With SMT to complicate matters further, and SSE
  - Do we need to force programmers to use hybrid model?
- Accelerators
  - Themselves heterogeneous
  - Cell, GPGPU, Larrabee, FPGAs, ..
- Objective:
  - try to insulate the programmer from complexity of managing accelerators
  - At least about which work-unit to run where

Object based over-decomposition

- Let
  - the programmer decompose computation into objects
    - Work units, data-units, composites
  - Let an intelligent runtime system assign objects to processors
  - RTS can change this assignment (mapping) during execution
- Locality of data references is a critical attribute for performance
- A parallel object can access only its own data
  - Asynchronous method invocation for accessing other's data

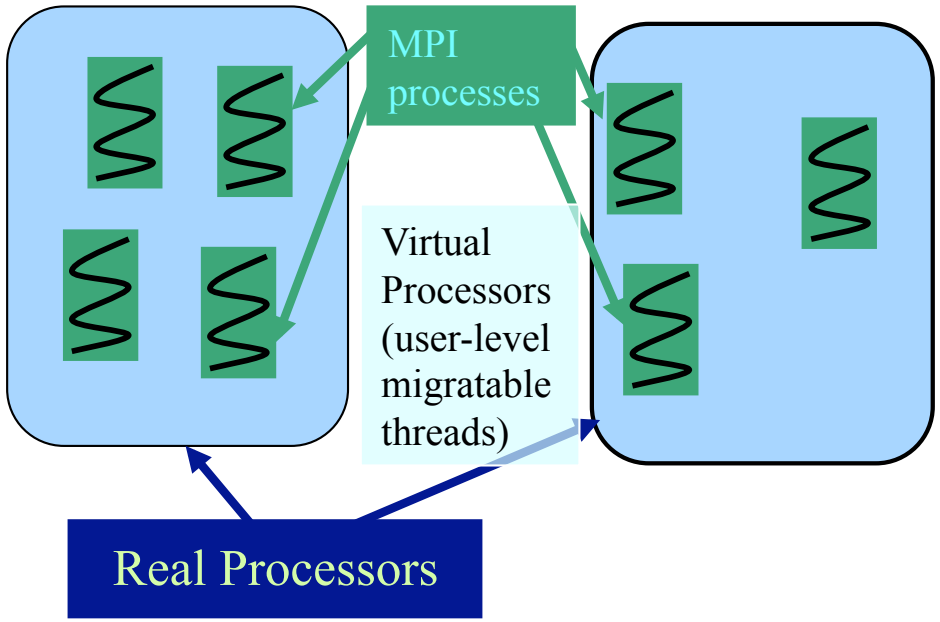
Object-based over-decomposition: Charm++

- Multiple "indexed collections" of C++ objects
- Indices can be multi-dimensional and/or sparse
- Programmer expresses communication between objects
  - with no reference to processors



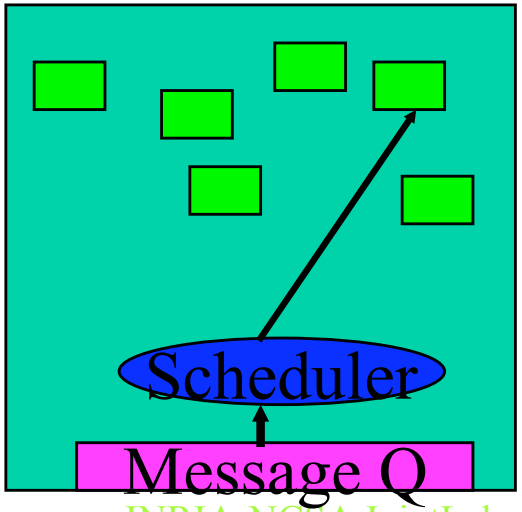
Object-based over-decomposition: AMPI

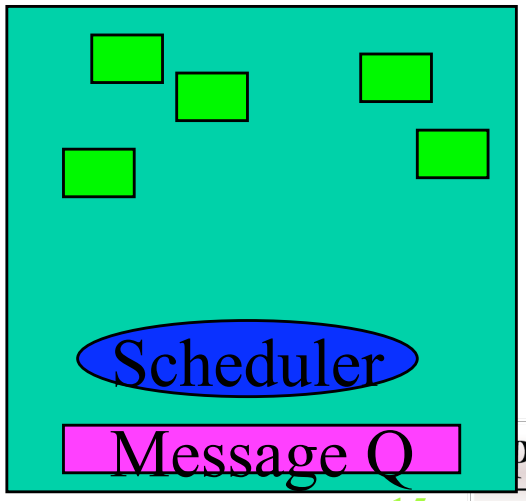
- Each MPI process is implemented as a user-level thread
- Threads are light-weight and migratable!
  - <1 microsecond context switch time, potentially >100k threads per core
- Each thread is embedded in a charm++ object (chare)



Why is it suitable for Multi-cores

- 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





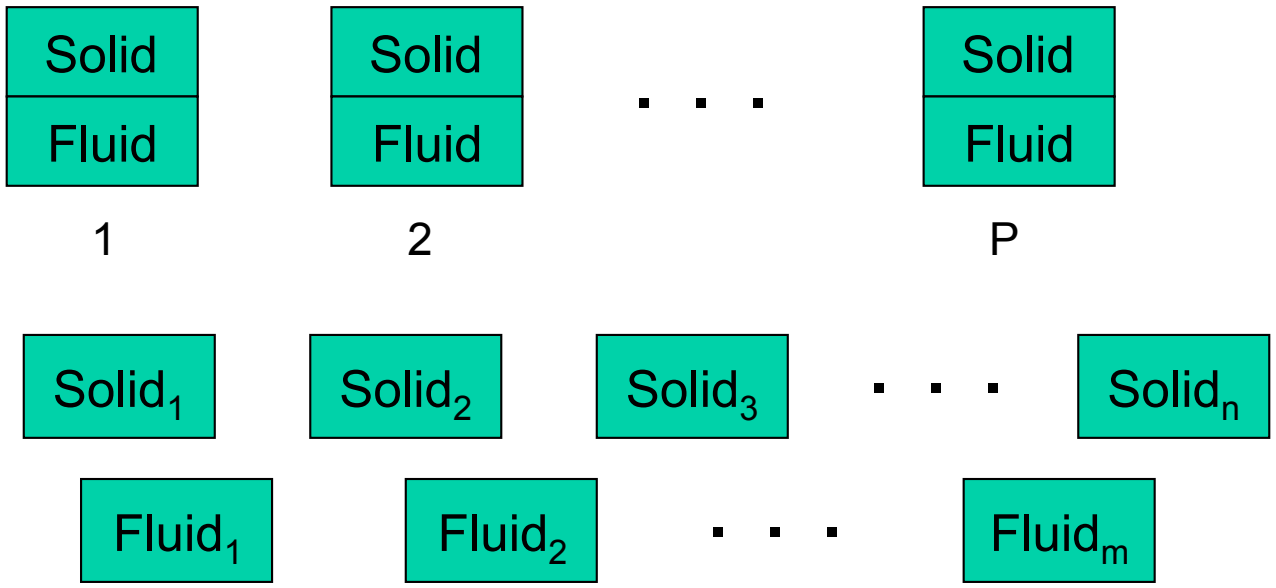
Parallel Decomposition and Processors

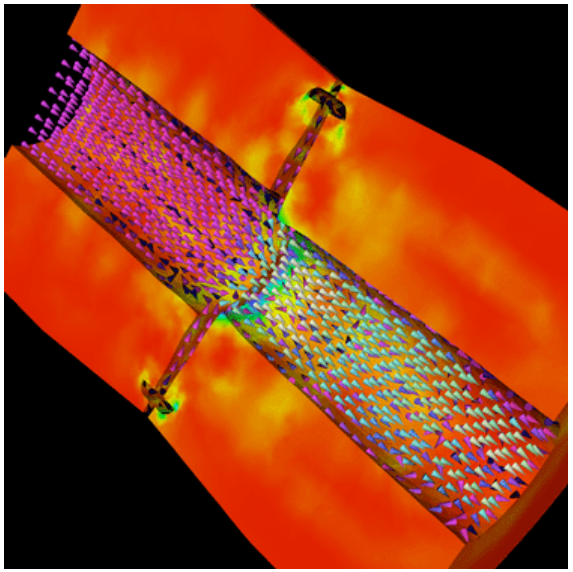
- MPI-style encourages
  - Decomposition into P pieces, where P is the number of physical processors available
  - If your natural decomposition is a cube, then the number of processors must be a cube
  - …
- Charm++/AMPI style "virtual processors"
  - Decompose into natural objects of the application
  - Let the runtime map them to processors
  - Decouple decomposition from load balancing

PPL UIUC

Decomposition independent of numCores

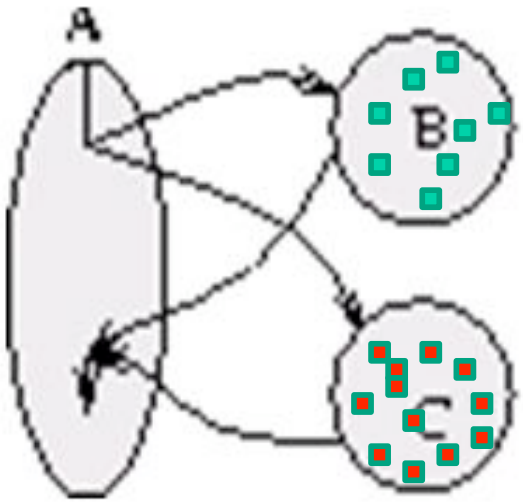
- Rocket simulation example under traditional MPI vs. Charm++/AMPI framework

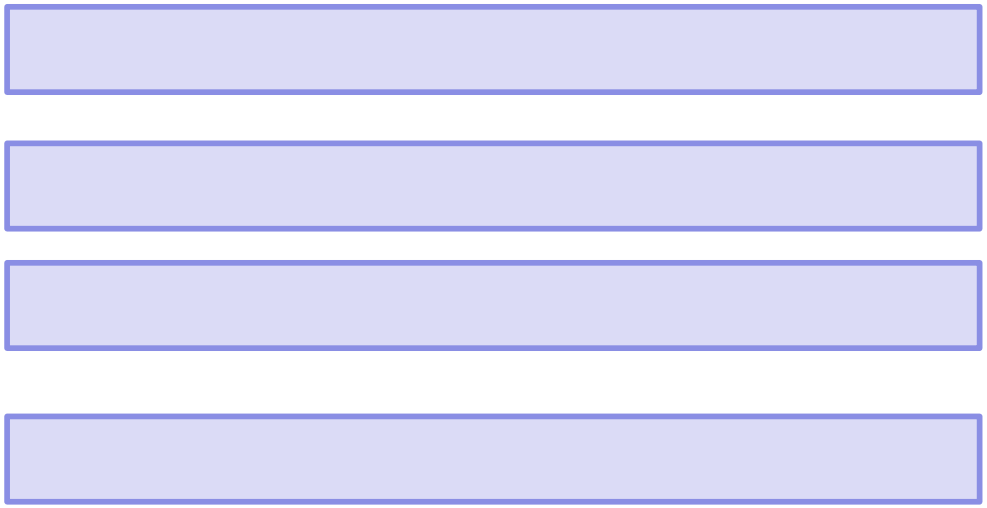




  - Benefit: load balance, communication optimizations, modularity

Parallel Composition: A1; (B || C); A2

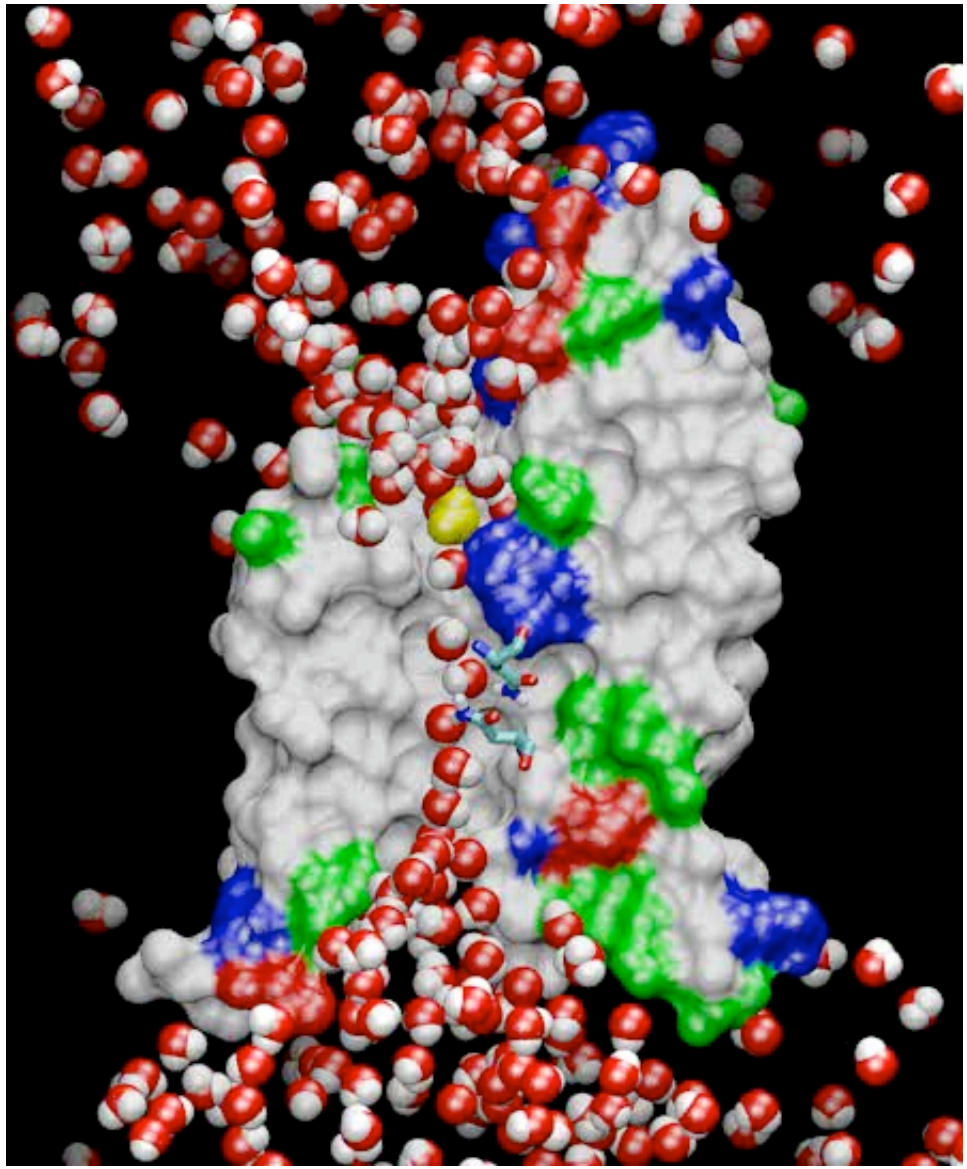




Recall: Different modules, written in different languages/paradigms, can overlap in time and on processors, without programmer having to worry about this explicitly

PPL UIUC

NAMD: A Production MD program

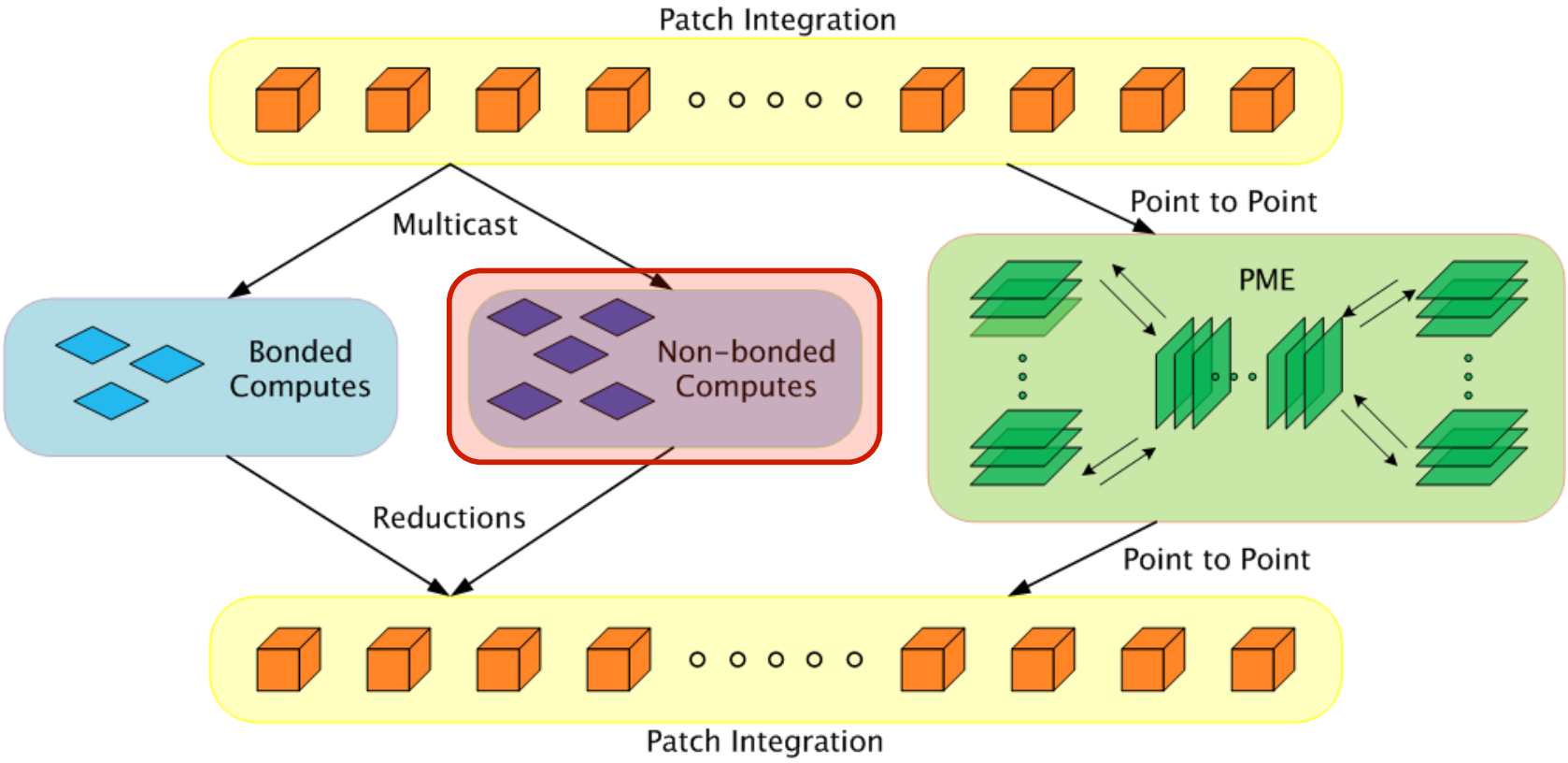


NAMD

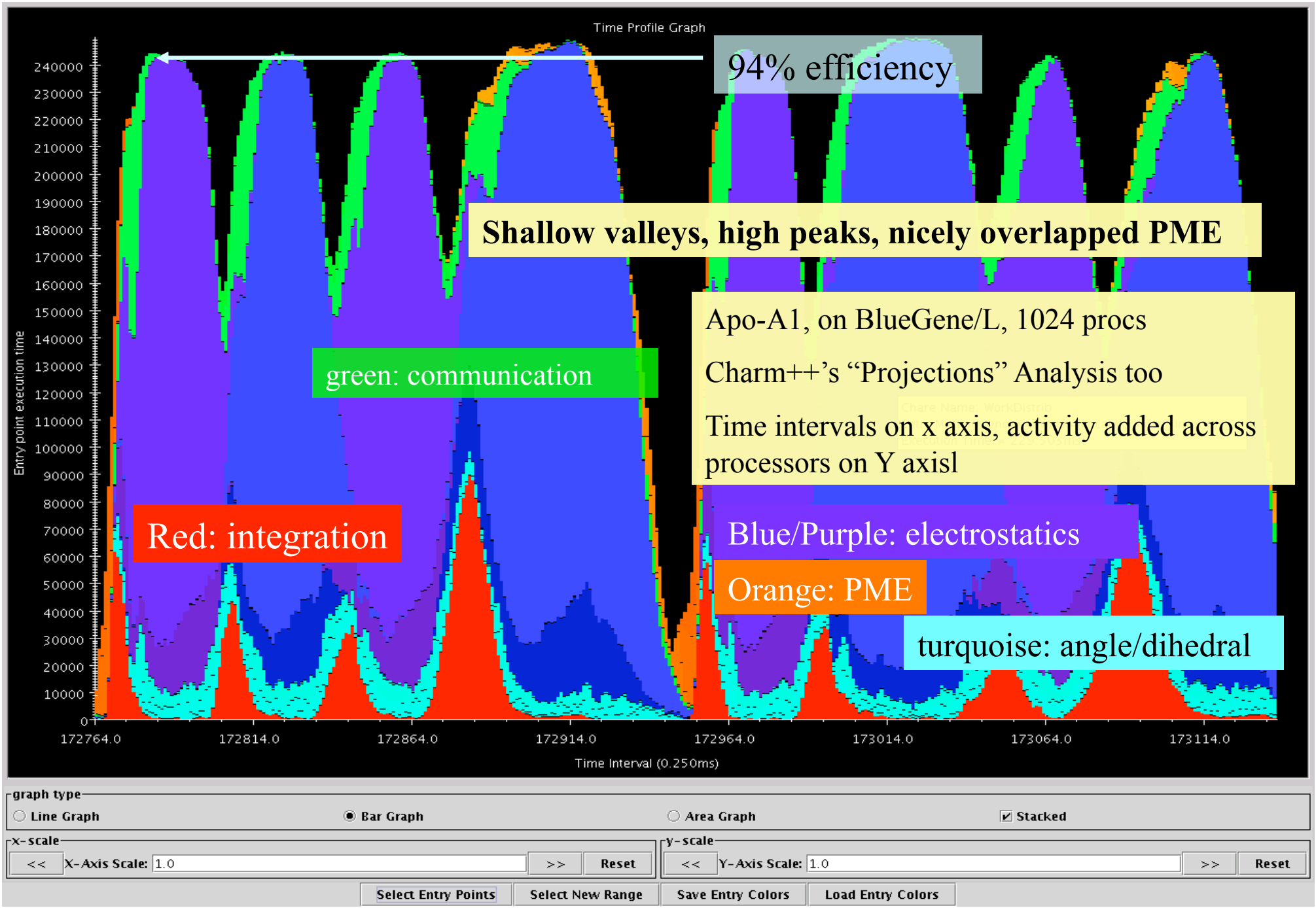
- Fully featured program
- NIH-funded development
- Distributed free of charge (~20,000 registered users)
- Binaries and source code
- Installed at NSF centers
- User training and support
- Large published simulations

PPL UIUC

Parallelization using Charm++

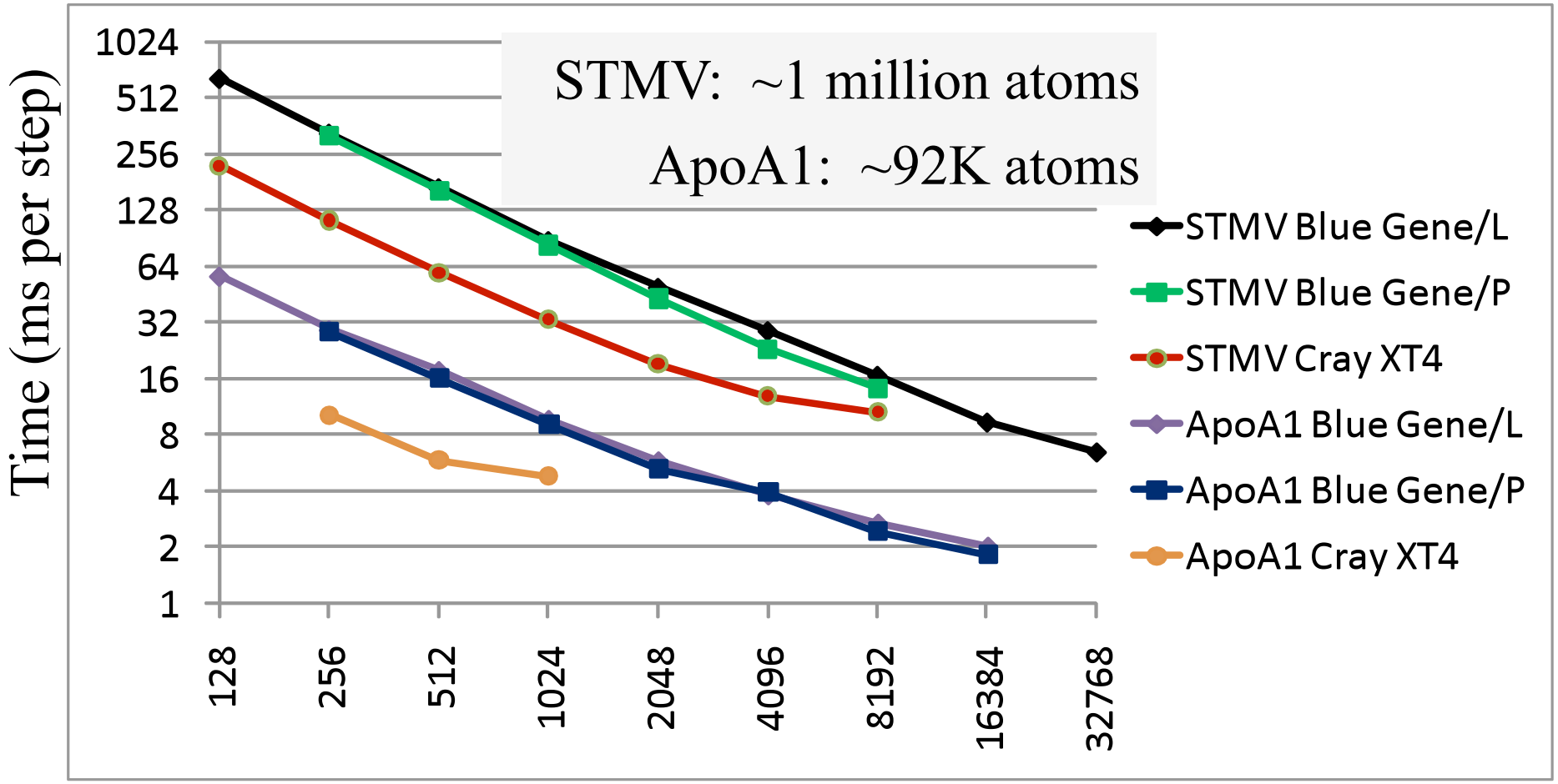


Bhatele, A., Kumar, S., Mei, C., Phillips, J. C., Zheng, G. & Kale, L. V. 2008 **Overcoming Scaling Challenges in Biomolecular Simulations across Multiple Platforms**. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, Miami, FL, USA, April 2008.


Time Profile Graph
94% efficiency
Shallow valleys, high peaks, nicely overlapped PME
Apo-A1, on BlueGene/L, 1024 procs
Charm++'s "Projections" Analysis too
Time intervals on x axis, activity added across processors on Y axisl
green: communication
Red: integration
Blue/Purple: electrostatics
Orange: PME
turquoise: angle/dihedral
Entry point execution time
Time Interval (0.250ms)
graph type
Line Graph
Bar Graph
Area Graph
Stacked
x-scale
X-Axis Scale: 1.0
y-scale
Y-Axis Scale: 1.0
Reset
Select Entry Points
Select New Range
Save Entry Colors
Load Entry Colors

Performance of NAMD

Blue Gene results based on work on DCMF many-to-many pattern by Sameer Kumar, IBM Research



PPL UIUC

Challenge: automated load balancing

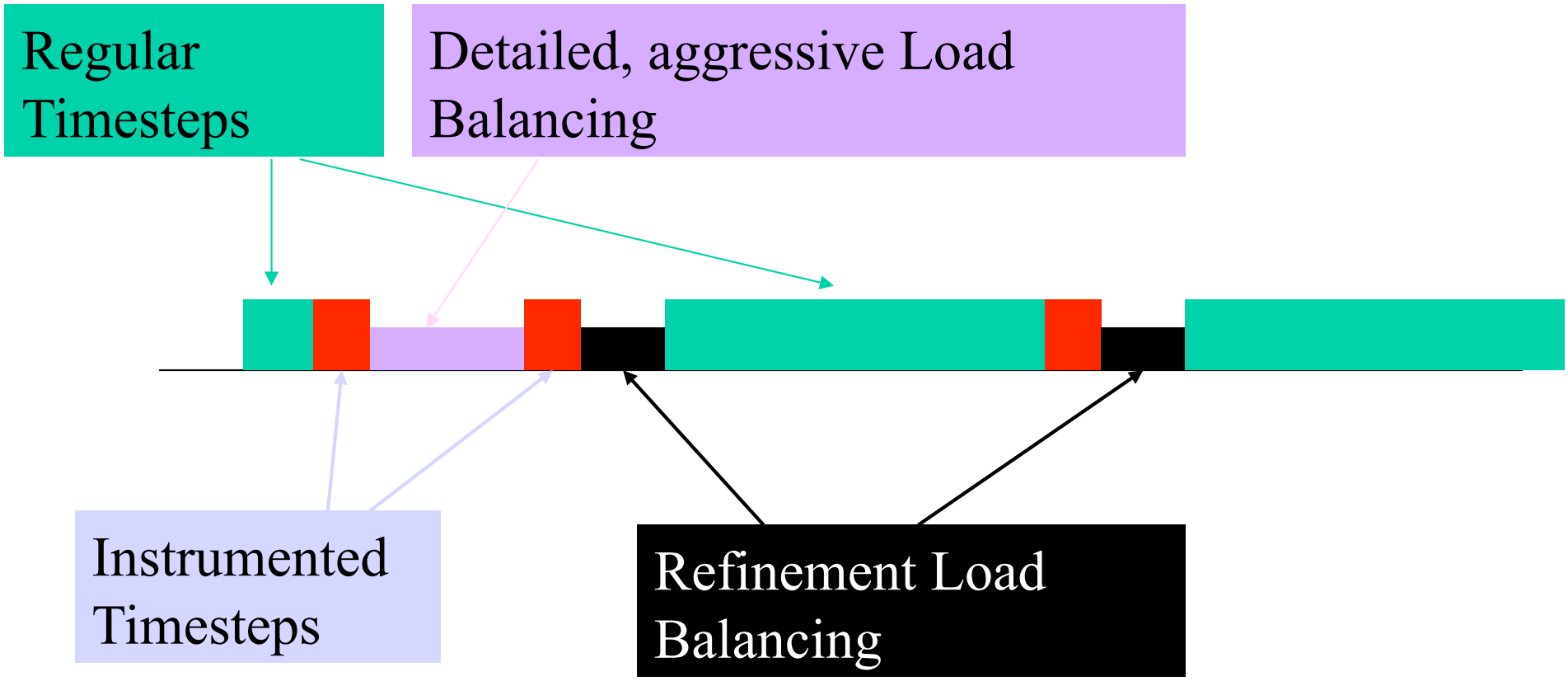
- Static
  - Irregular applications
  - Programmer shouldn't have to figure out ideal mapping
- Dynamic:
  - Applications are increasingly using adaptive strategies
  - Abrupt refinements
  - Continuous migration of work: e.g. particles in MD
- Challenges:
  - Performance limited by most overloaded processor
  - The chance that one processor is severely overloaded gets higher as #processors increases

Principle of persistence

Computational loads and communication patterns tend to persist, even in dynamic computations

So, recent past is a good predictor of near future

Measurement-based Load Balancing



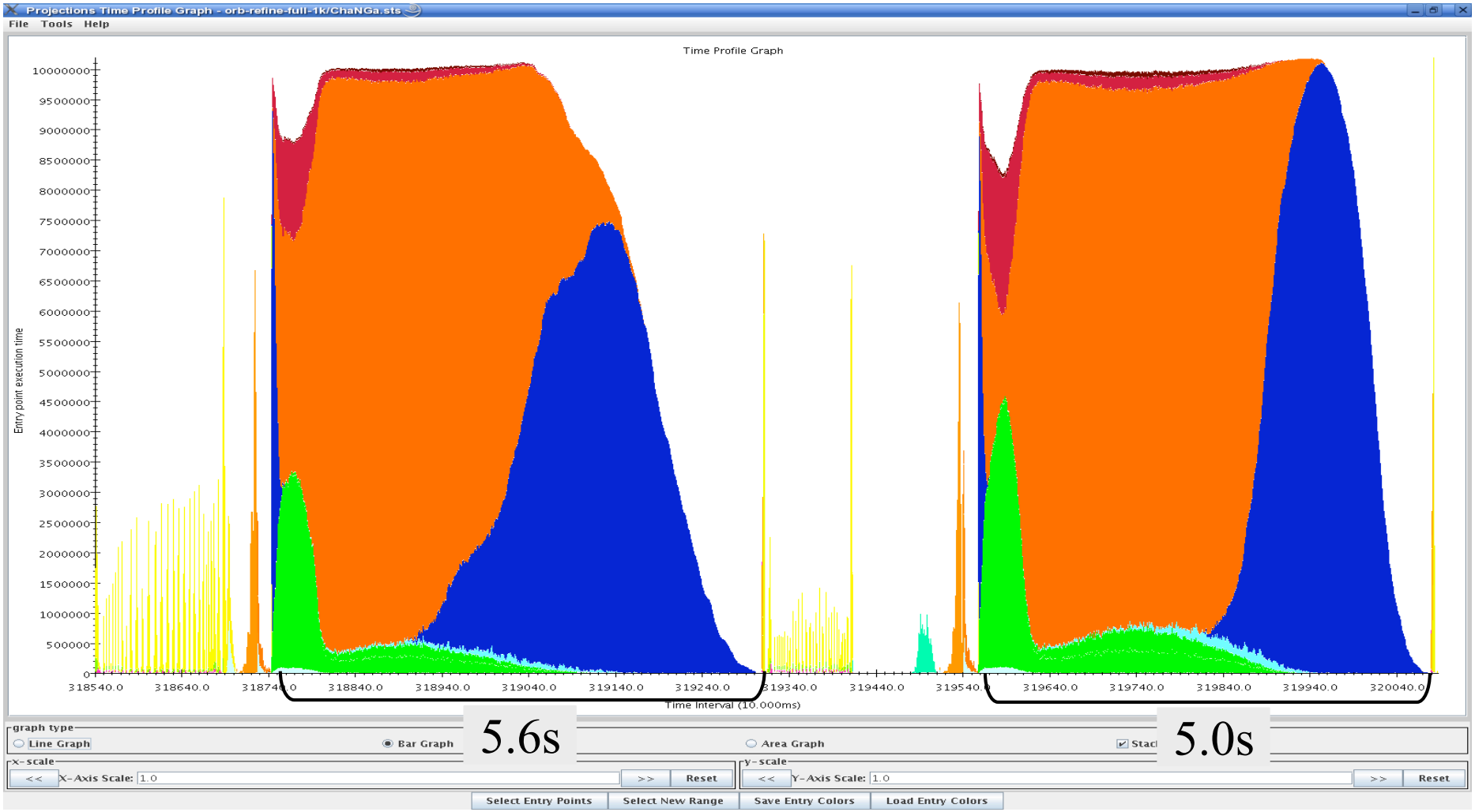
ChaNGa: Parallel Gravity

Evolution of Universe and Galaxy Formation

- 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

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

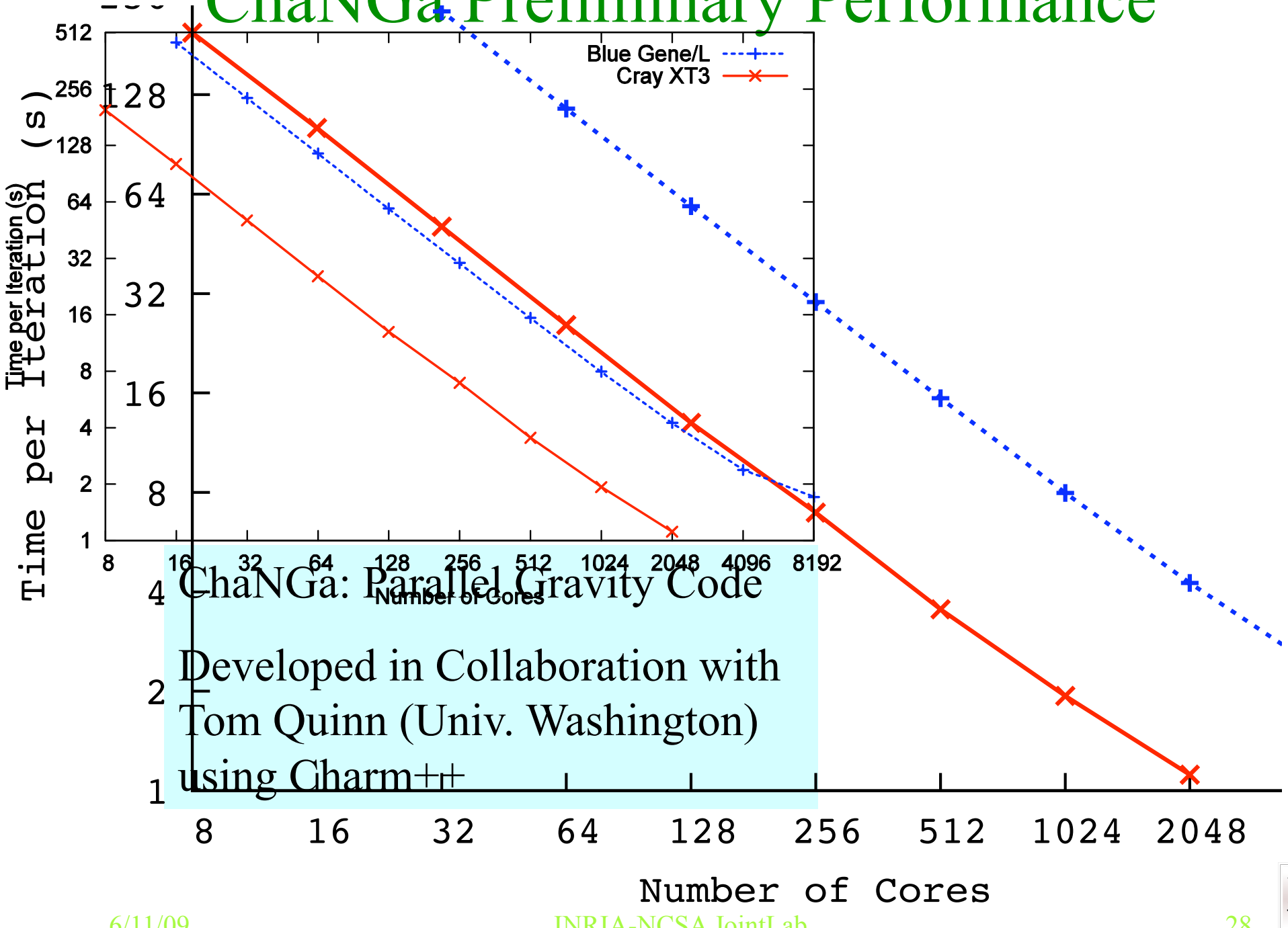
Load balancing with OrbRefineLB



Need sophisticated balancers and ability to choose the right ones automatically

PPL UIUC

ChaNGa Preliminary Performance



ChaNGa: Parallel Gravity Code

Developed in Collaboration with
Tom Quinn (Univ. Washington)
using Charm++

PPL UIUC

Load balancing for large machines: I

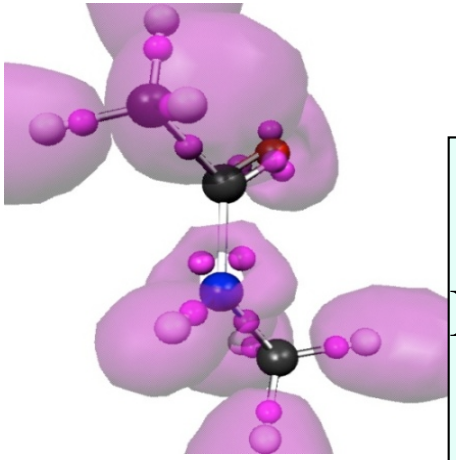
- Centralized balancers achieve best balance
  - Collect object-communication graph on one processor
  - But won't scale beyond tens of thousands of nodes
- Fully distributed load balancers
  - Avoid bottleneck but… Achieve poor load balance
  - Not adequately agile
- Hierarchical load balancers
  - Careful control of what information goes up and down the hierarchy can lead to fast, high-quality balancers
- Need for a universal balancer that works for all applications

Load balancing for large machines: II

- Interconnection topology starts to matter again
  - Was hidden due to wormhole routing etc.
  - Latency variation is still small
  - But bandwidth occupancy is a problem
- Topology aware load balancers
  - Some general heuristic have shown good performance
    - But may require too much compute power
  - Also, special-purpose heuristic work fine when applicable
  - Still, many open challenges

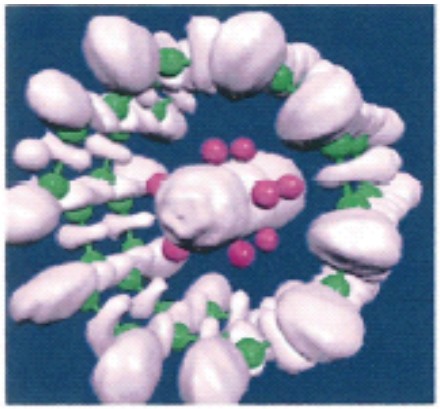
OpenAtom
Car-Parinello ab initio MD
NSF ITR 2001-2007, IBM

Molecular Clusters :

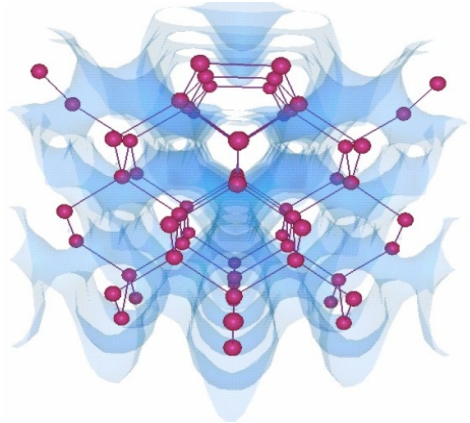


G. Martyna (IBM)
M. Tuckerman (NYU)
L. Kale (UIUC)

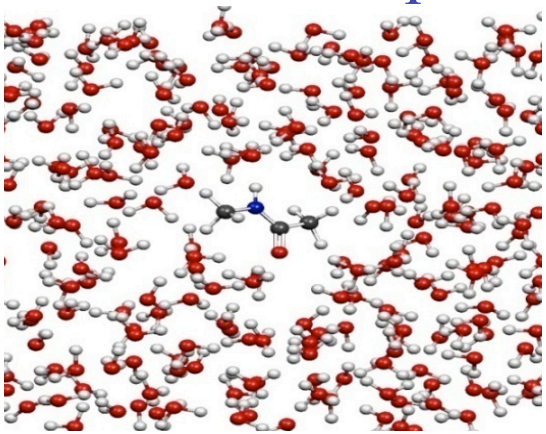
Nanowires:



Semiconductor Surfaces:



3D-Solids/Liquids:

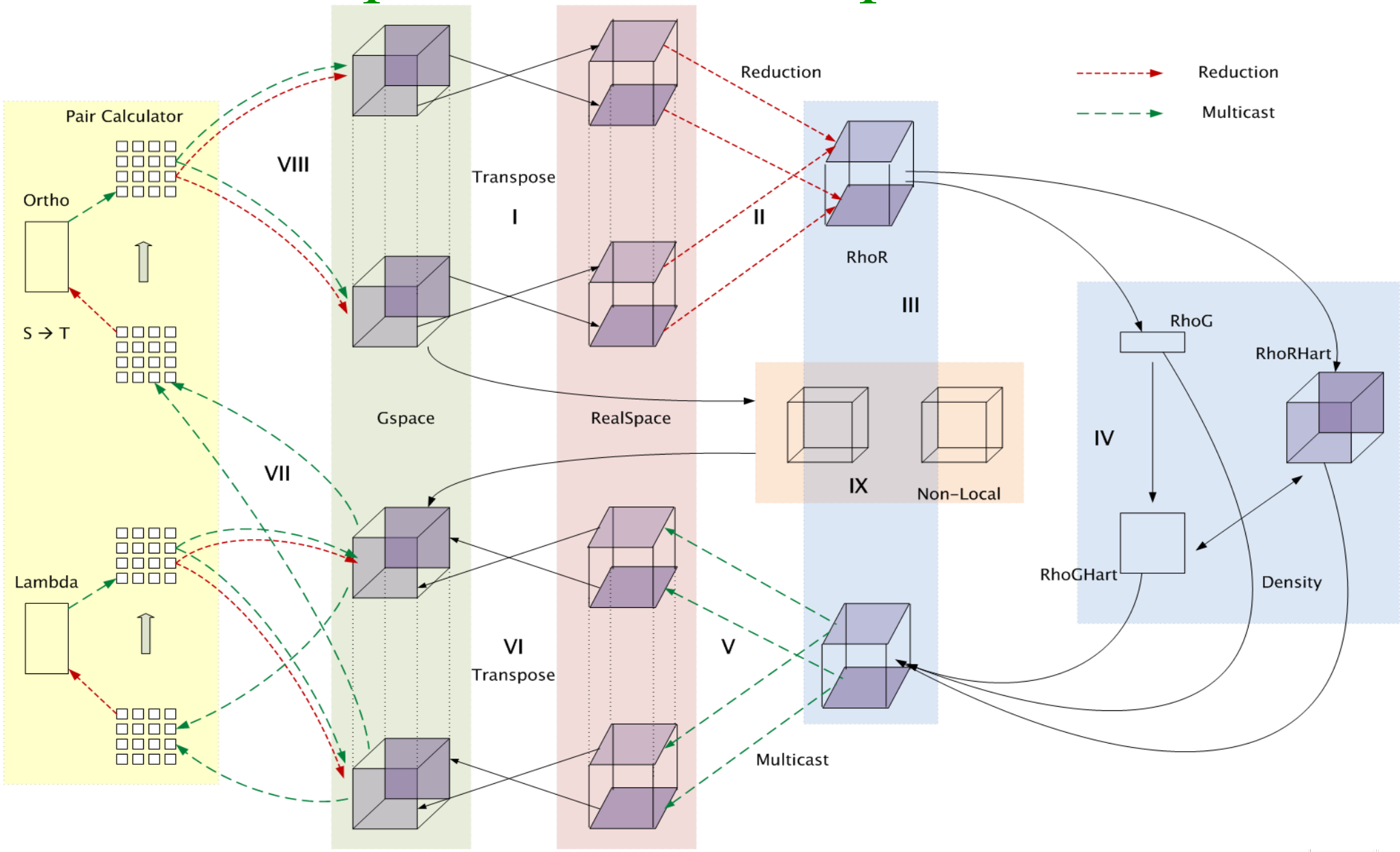


PPL UIUC

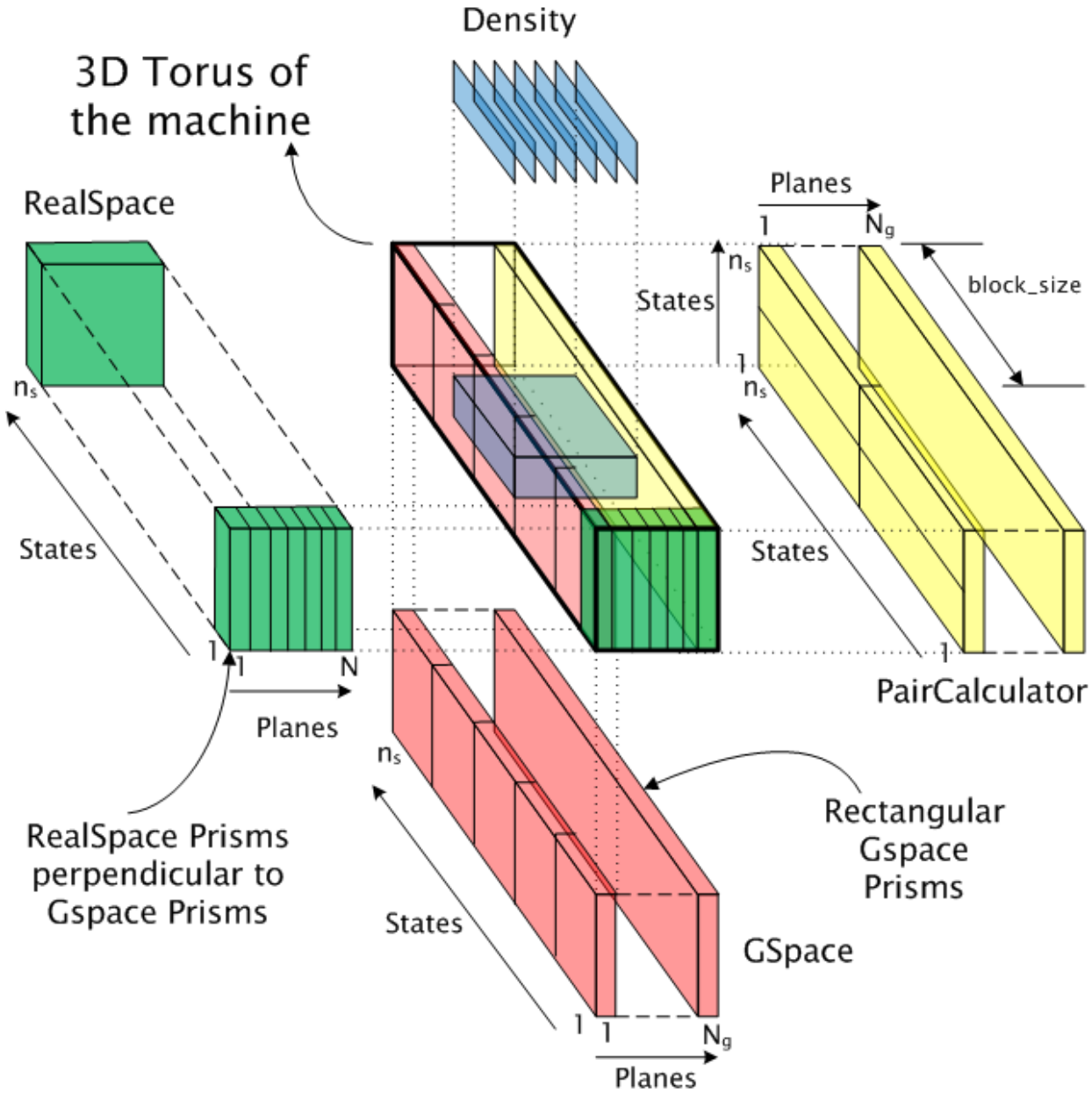
New OpenAtom Collaboration (DOE)

- Principle Investigators
  - G
    .
    Martyna (IBM TJ Watson)
  - M. Tuckerman (NYU)‖
  - L.V. Kale (UIUC)‖
  - K. Schulten (UIUC)‖
  - J. Dongarra (UTK/ORNL)‖
- Current effort focus
  - QMM
    M
    v
    i
    a integration with NAMD2
  - ORNL
    Cra
- A unique parallel decomposition of the Car-Parinello method.
- Using Charm++ virtualization, we can efficiently scale small (32 molecule) systems to thousands of processors

Decomposition and Computation Flow

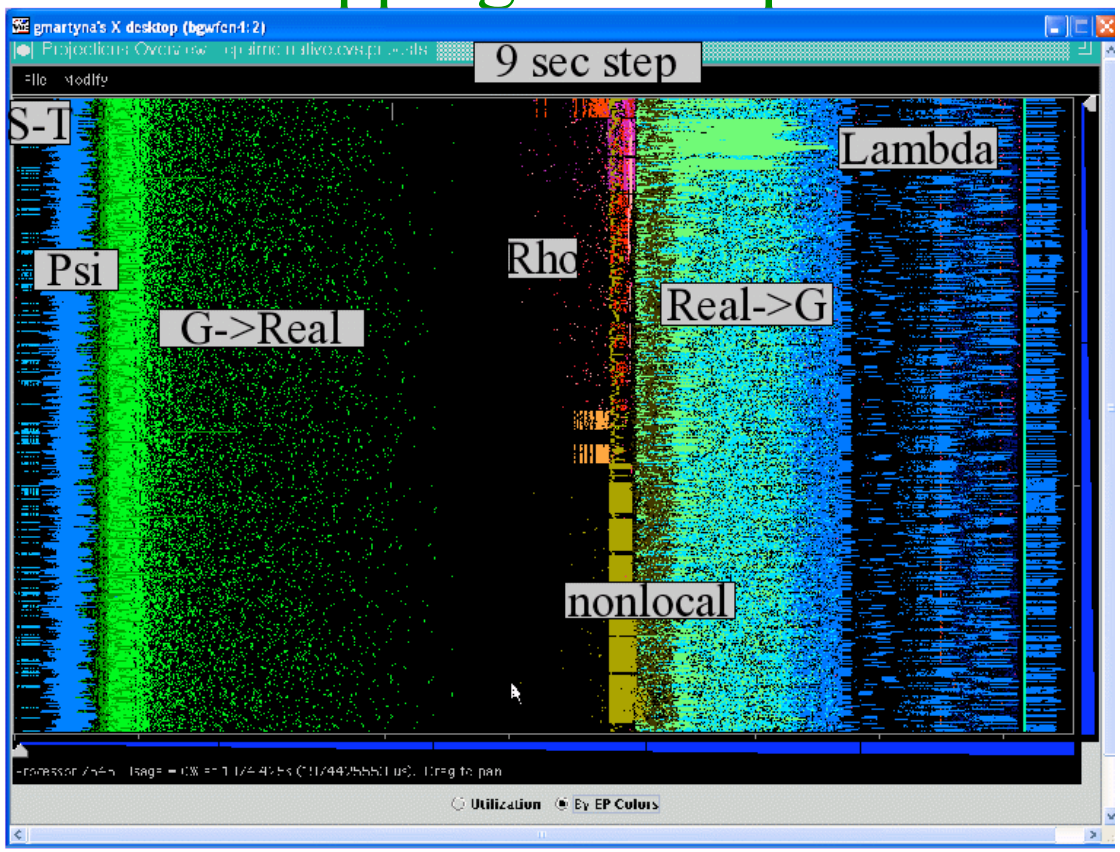


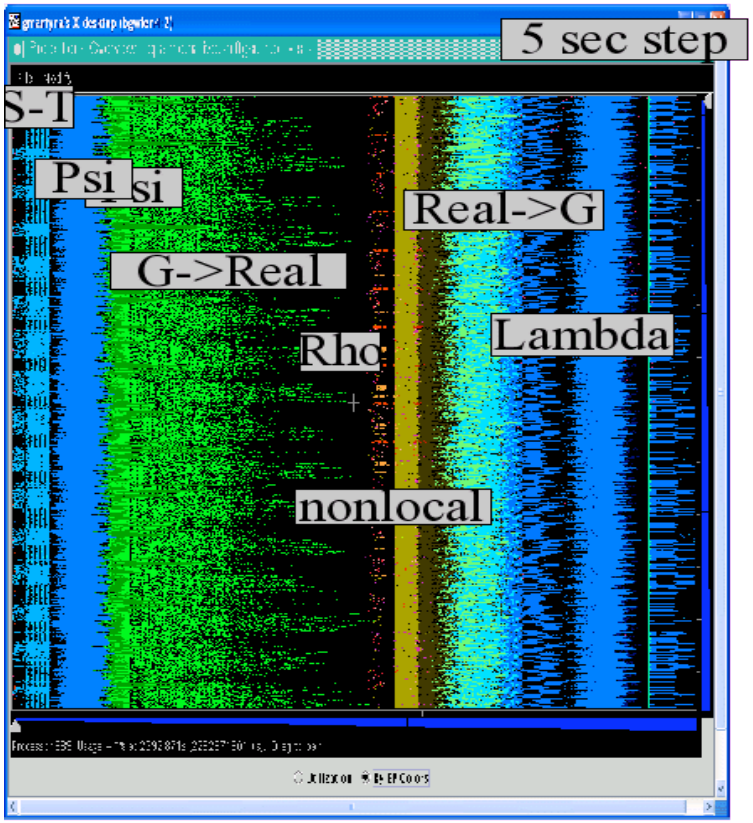
Topology aware mapping of objects



PPL UIUC

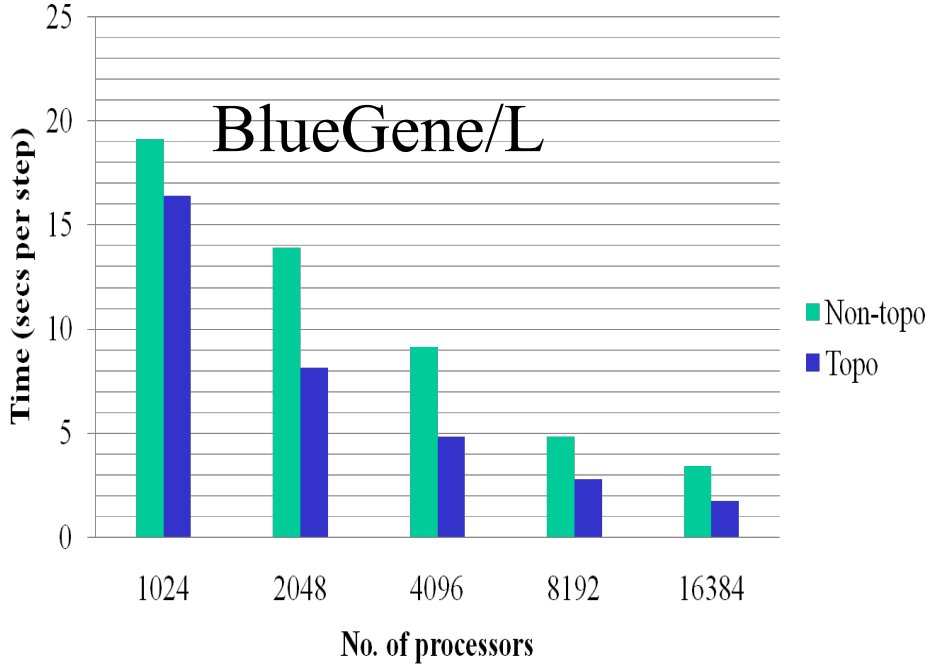
Improvements wrought by network topological aware mapping of computational work to processors

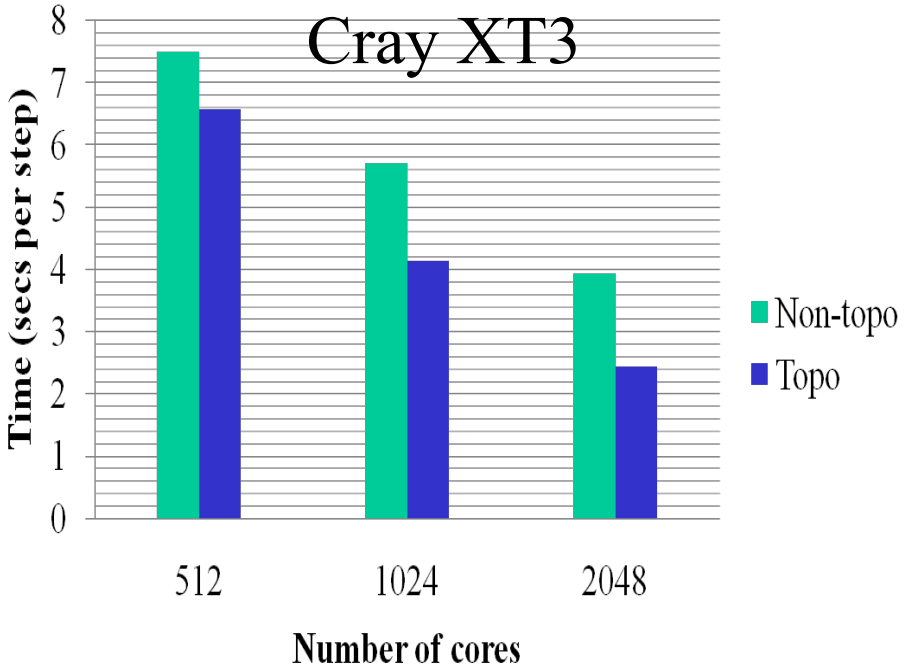




The simulation of the right panel, maps computational work to processors taking the network connectivity into account while the left panel simulation does not. The "black" or idle time processors spent waiting for computational work to arrive on processors is significantly reduced at left. (256waters, 70R, on BG/L 4096 cores)

OpenAtom: Topology Aware Mapping





Challenge:
Scalable Performance Analysis

See: Chee Wai Lee et al, HIPS 2008
(and Chee Wai's upcoming PhD thesis)

Challenge:
How to tune performance when you don't have access to the machine at scale

See: BigSim project papers at
http://charm.cs.uiuc.edu

BigSim leverages object-based virtualization to support such tuning

Raising the Level of Abstraction

- What we have seen so far, will help:
  - by automating resource management, FT, ..
- But: parallel interactions are still low-level
  - Programmer has to spend considerable efforts in spelling out interactions, and data decompositions, in particular
- Clearly, we need new programming models

Raising the Level of Abstraction

- Two metapoints:
  - Need for exploration (don't standardize too soon)
  - Interoperability
    - Allows a "beachhead" for novel paradigms
    - Long-term: Allow each module to be coded using the paradigm best suited for it
  - Interoperability requires concurrent composition
    - This may ***require*** message-driven execution

Simplifying Parallel Programming

- By giving up completeness!
  - A paradigm may be simple, but
    - not suitable for expressing all parallel patterns
    - yet, if it can cover a significant classes of patters (applications, modules), it is useful
  - *A collection of incomplete models, backed by a few complete ones, will do the trick*
- We must consider: why did many new models fail?
  - HPF could express many array operations succinctly
    - Implementations weren't always efficient,
    - but even more: everything couldn't be expressed in it, and it wasn't broadly interoperable

PPL UIUC

Simplifying Parallel Programming

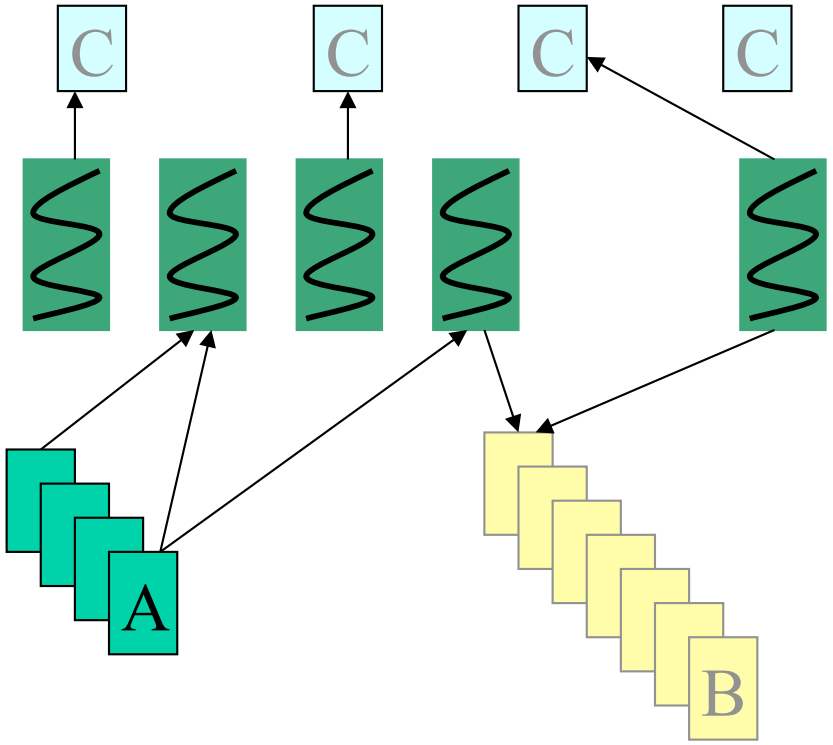
- Our own examples: both outlaw non-determinism
  - Multiphase Shared Arrays (MSA): restricted shared memory
    - LCPC '04
  - Charisma: Static data flow among collections of objects
    - LCR'04, HPDC '07

MSA: Multiphase Shared Arrays

Observations:

General shared address space abstraction is complex

Certain special cases are simple, and cover most uses



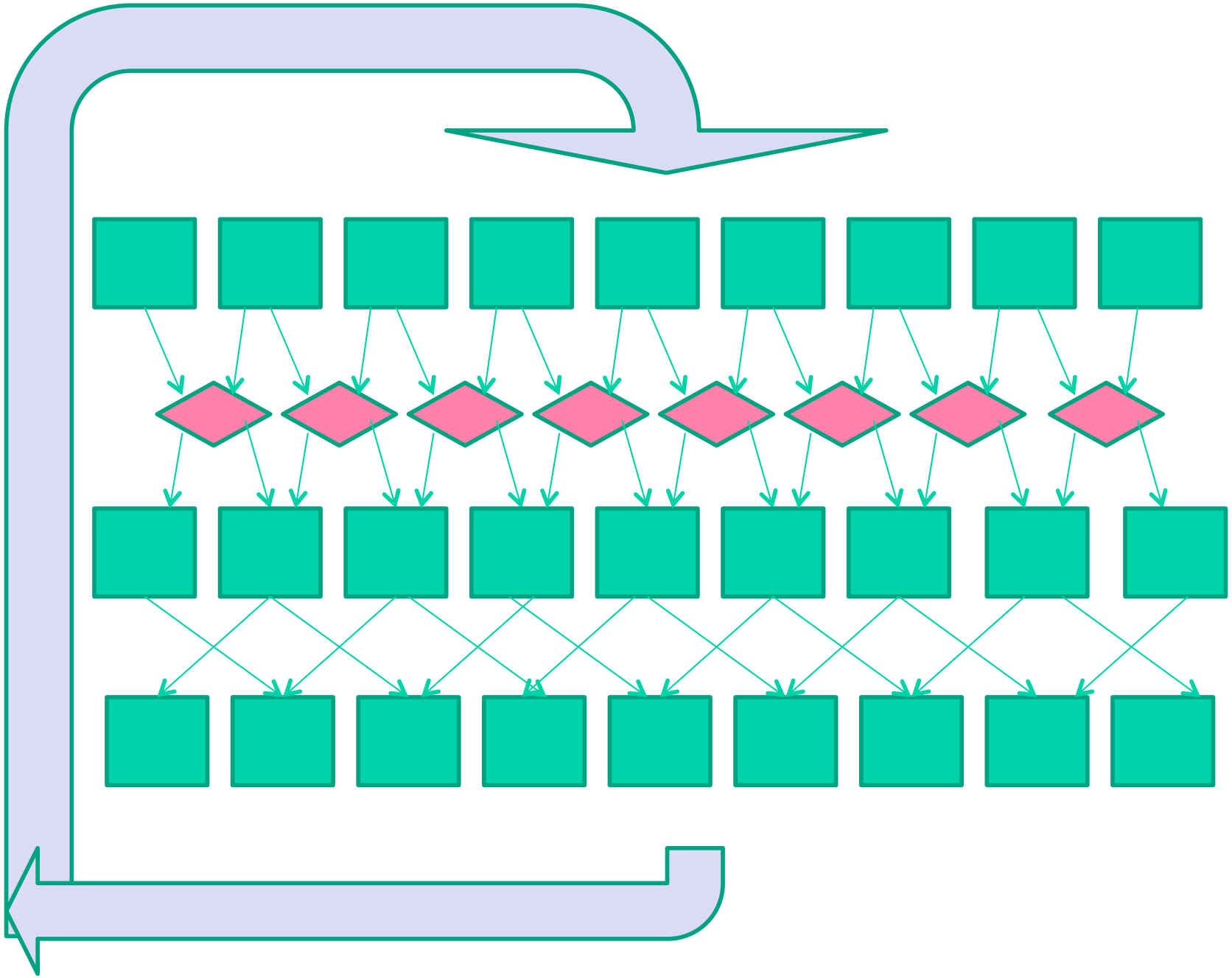
- In the simple model:
- A program consists of
  - A collection of Charm threads, and
  - Multiple collections of data-arrays
    - Partitioned into pages (user-specified)
- Each array is in one mode at a time
  - But its mode may change from phase to phase
- Modes
  - Write-once
  - Read-only
  - Accumulate
  - Owner-computes

PPL UIUC

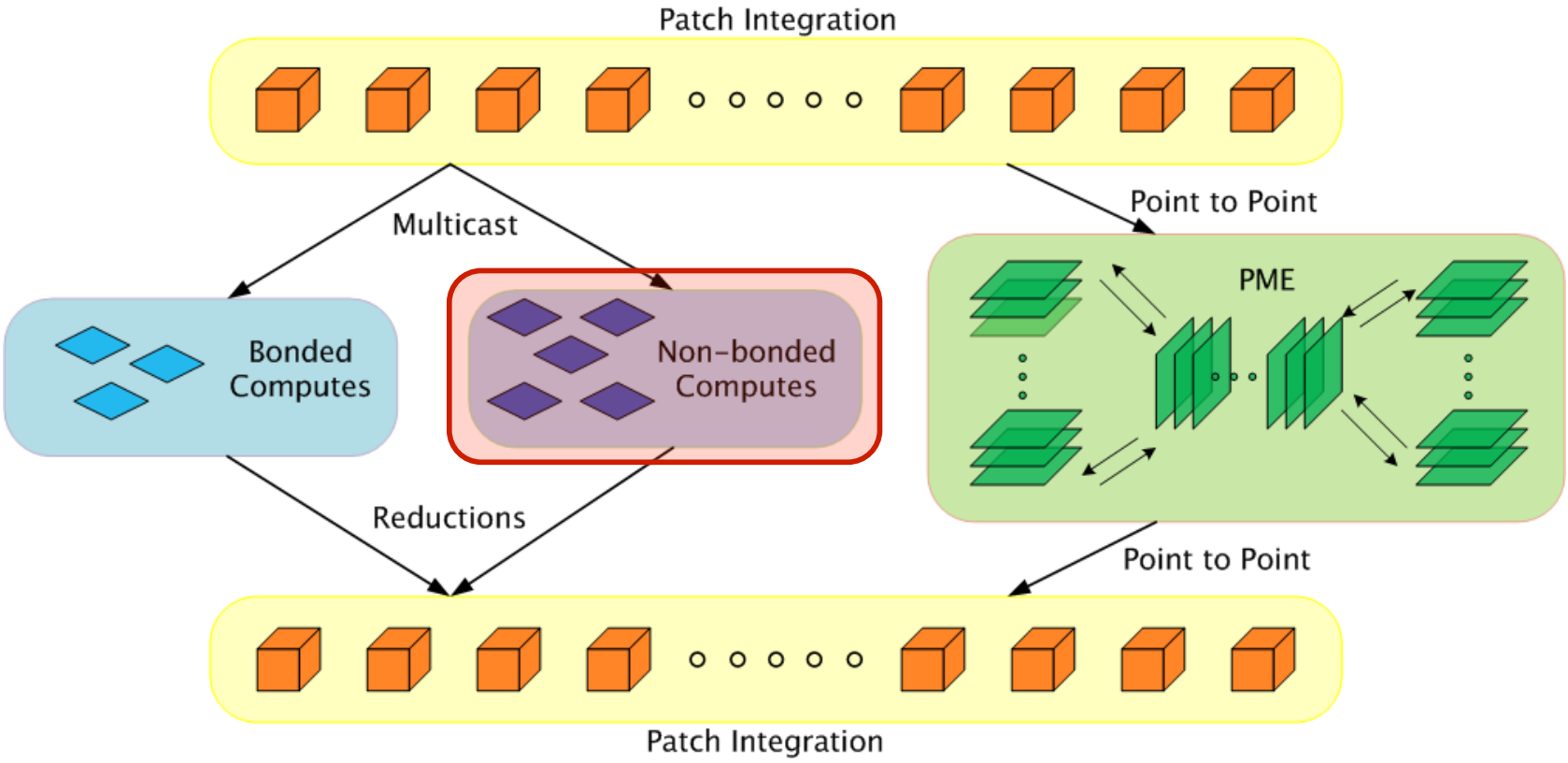
Charisma: Static Data Flow

Observation: many CSE applications or modules involve static data flow in a fixed network of entities

The amount of data may vary from iteration to iteration, but who talks to whom remains unchanged

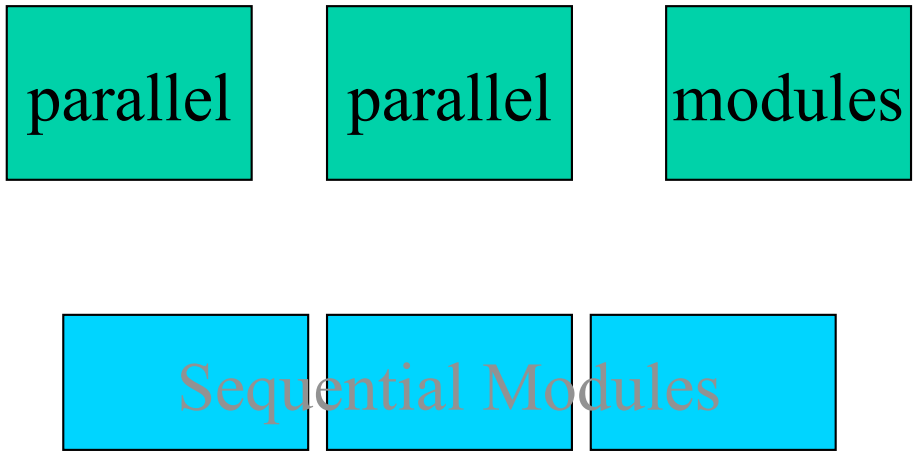


NAMD Uses Static Data Flow



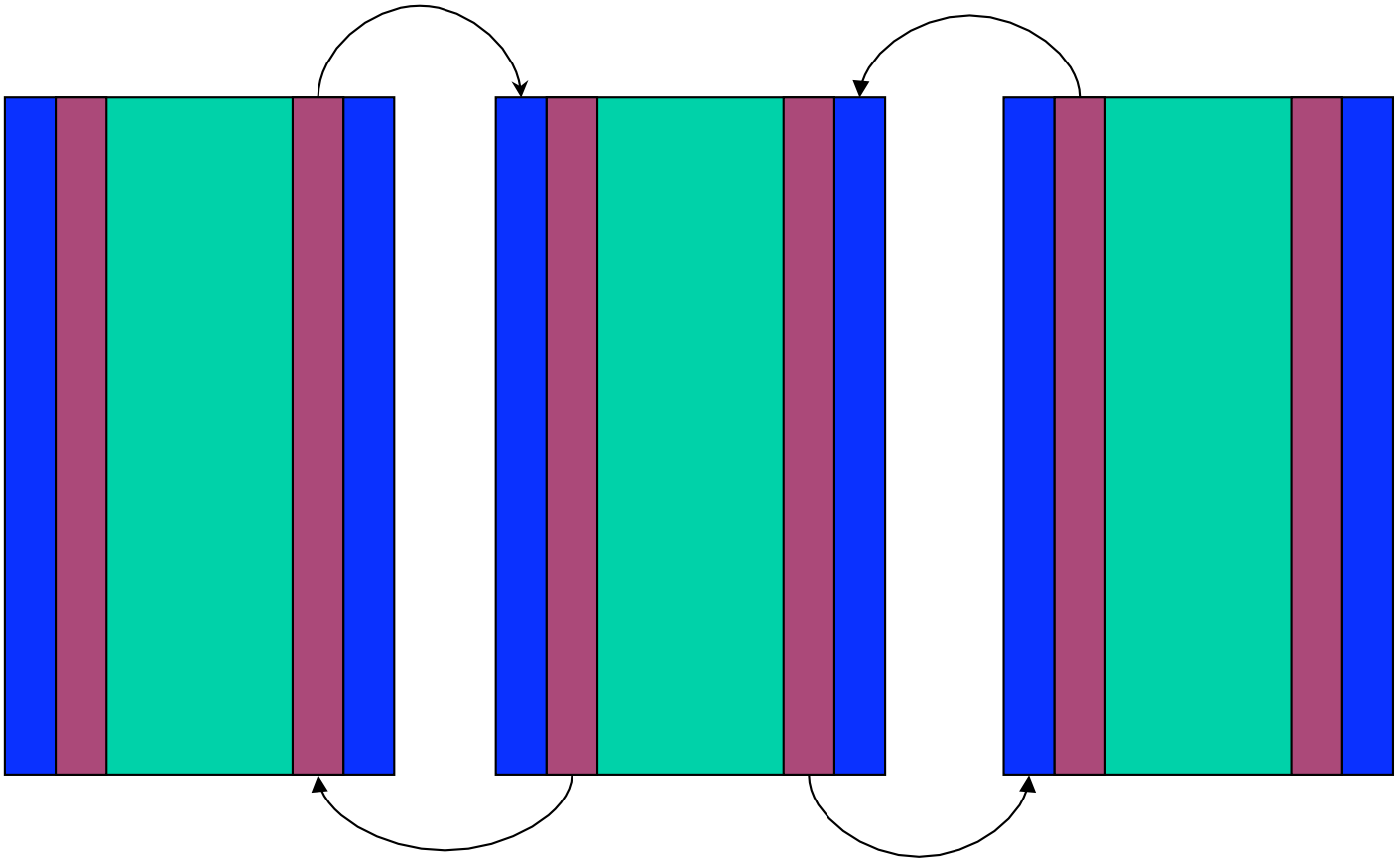
Bhatele, A., Kumar, S., Mei, C., Phillips, J. C., Zheng, G. & Kale, L. V. 2008 **Overcoming Scaling Challenges in Biomolecular Simulations across Multiple Platforms**. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, Miami, FL, USA, April 2008.

Charisma: Sequential-Parallel Separation



- Program consists of
  - Orchestration (.or) code
    - Chare arrays declaration
    - Orchestration with parallel constructs
    - Global flow of control
  - "Physics" code
    - Regular C++
    - User variables
    - Sequential methods

Charisma++ example (Simple)



```
while (e > threshold)
        forall i in J
          <+e, lb[i], rb[i]> :=  J[i].compute(rb[i-1],lb[i+1]);
```


PPL UIUC

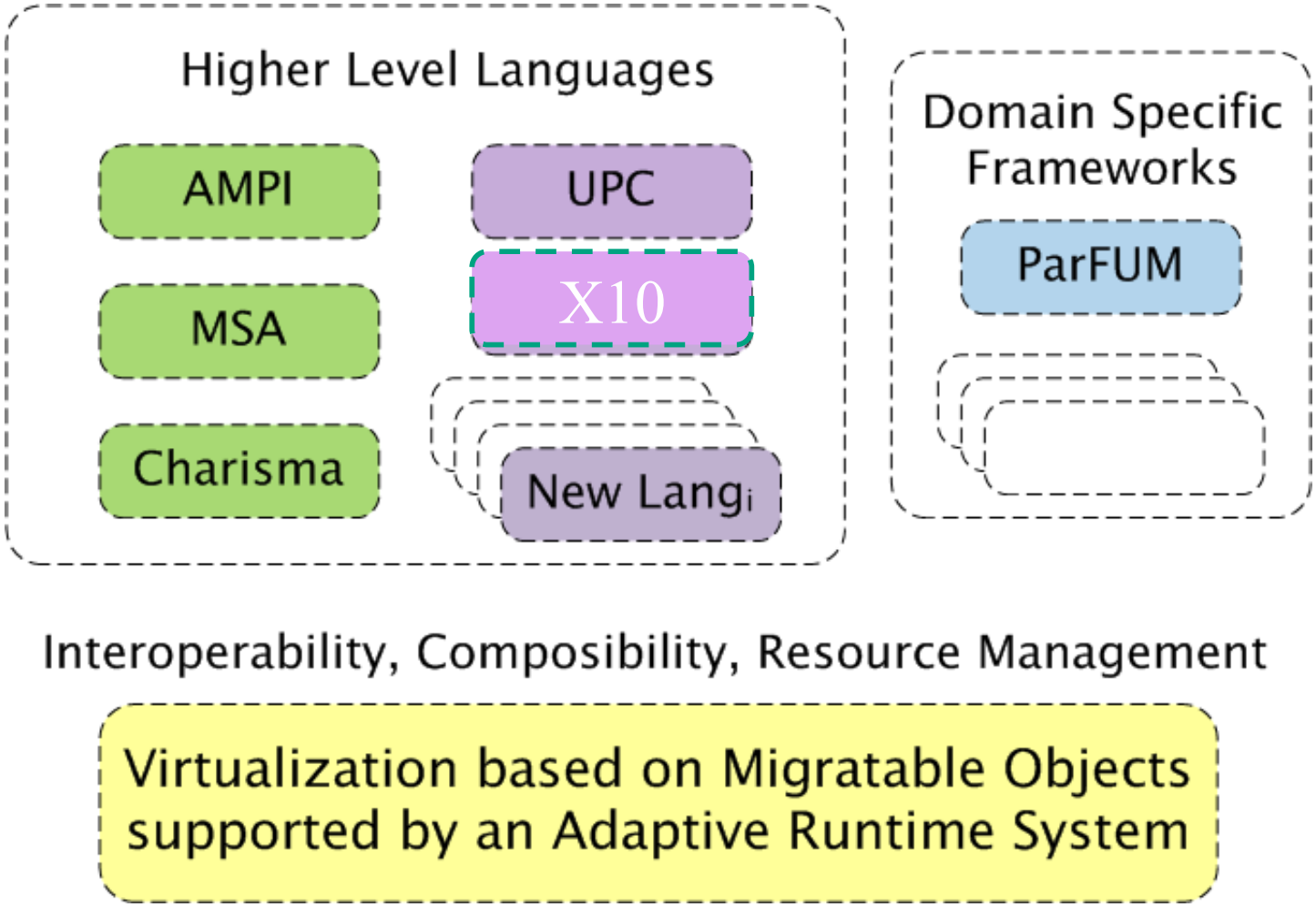
Mol. Dynamics with Spatial Decomposition

```
foreach i,j,k in cells
        <atoms[i,j,k]>:= cells[i,j,k].produceAtoms();
end-foreach
for iter := 0 to MAX_ITER
  foreach i,j,k,l,m,n in cellpairs
    <+forces[i,j,k]> :=
      cellpairs[i,j,k,l,m,n].coulombForces(atoms[i,j,k],atoms[l,m,n]);
  end-foreach

  foreach i,j,k in cells
        <atoms[i,j,k]> :=  cells[i,j,k].integrate(forces[i,j,k]);
  end-foreach
end-for
```


PPL UIUC

A View of an Interoperable Future



Summary

- Challenges in Resource management
  - Decomposition and Compositionality
  - Dynamic Load balancing
  - Fault Tolerance
  - Over-decomposition with Adaptive Runtime System helps
  - Scalable Performance analysis, and debugging
  - Early performance tuning (BigSim)
- Challenges in Raising the level of abstraction
  - Via "Incomplete yet simple" paradigms, and
  - domain-specific frameworks
  - Supported by an interoperable runtime system
- Exciting times ahead

More Info: http://charm.cs.uiuc.edu /

PPL UIUC