What Parallel HLLs Need

Laxmikant (Sanjay) Kale

http://charm.cs.illinois.edu

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

PARALLEL PROGRAMMING LAB
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS
PPL UIUC

Raising the Level of Abstraction

- Since parallel programming is challenging
  - Yes, admit it
- We need to increase productivity
  - Automating commonly needed functions
  - Raising the level of abstraction with Higher Level Programming Paradigms/Systems (HLS)
- HLPS:
  - What kinds of HLPS?

High Level Programming Systems

- Different ways of attaining “higher level”
  - Global view of data
  - Global view of control
  - Both
  - Simplified or specialized syntax
  - Safety properties
- But the largest benefit come from specialization
  - Domain specific languages
  - Domain specific Frameworks
  - Interaction-pattern specific languages

What do all the HLPS need?

- How can we facilitate development and use of such HLPS?
- Common Adaptive Runtime System
  - Resource management
  - Load balancing
  - Power energy and thermal optimization
  - Resilience, ..
- Interoperability
  - Since some of our HLPS are specialized, they are not "complete"
  - Have to interoperate with each other and with at least one complete language
- I will elaborate on these themes

Sanjay's Central Dogma: Overdecomposition

*Overdecomposition is **essential** for effective parallel programs,*
for computer performance and
for human productivity

What is overdecomposition?

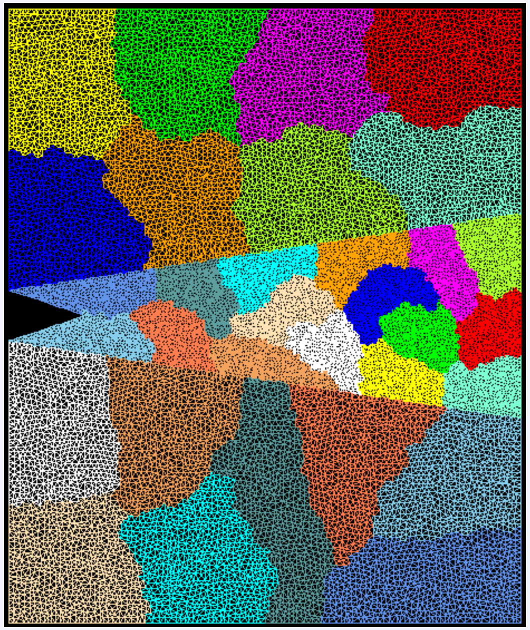
- Divide the computation into a large (but not too large) number of coarse pieces
  – Making decomposition independent of number of processors
- Not too large:
  – Making decomposition depend on the overhead:
  – Just large enough to amortize the overhead
- Express communication in terms of these pieces
  – Never addressing "the processors"
    - At least in the pure model

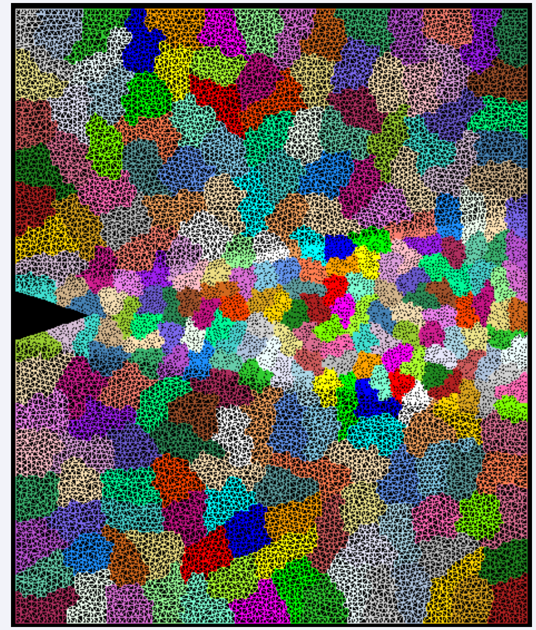
Grainsize

- Grainsize:
  - Rough definition: amount of computation per interaction: communication/scheduling event
- 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..





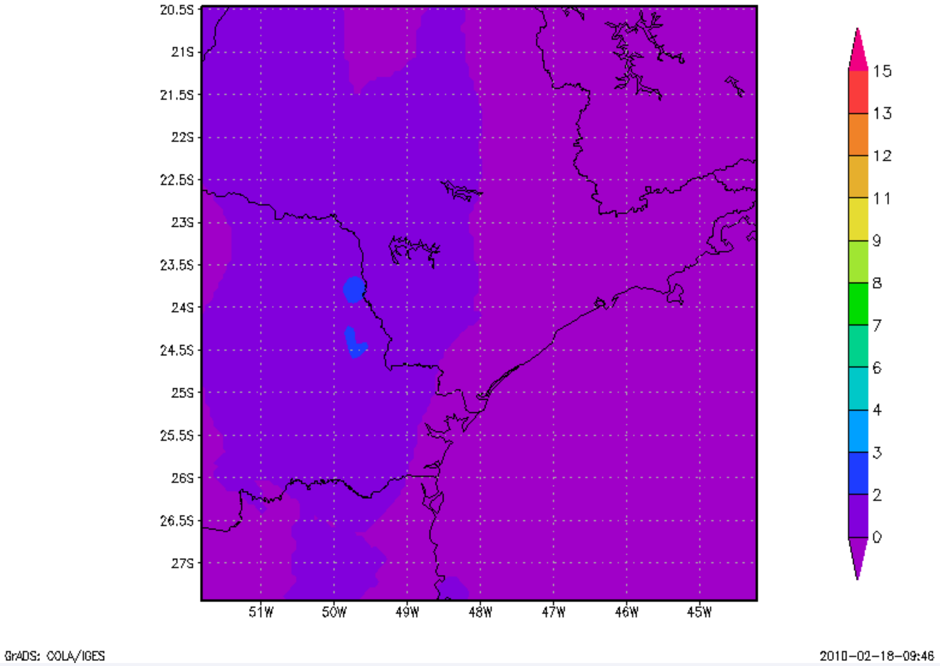
Decomposition into 16 chunks (left) and 128 chunks, 8 for each PE (right). The middle area contains cohesive elements. Both decompositions obtained using Metis. Pictures: S. Breitenfeld, and P. Geubelle

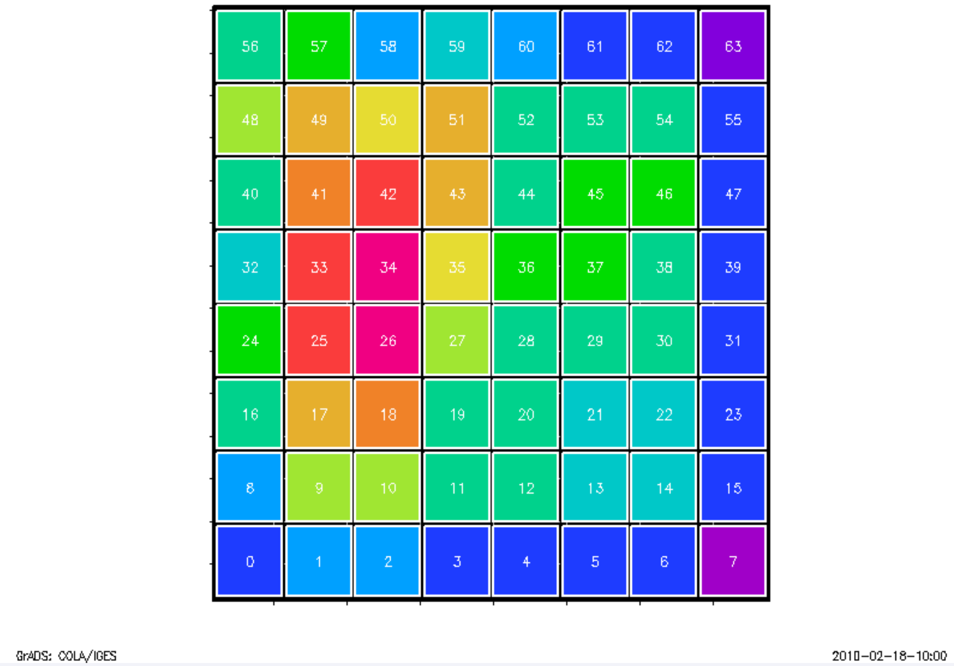
Grainsize example: NAMD

- 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, ..)

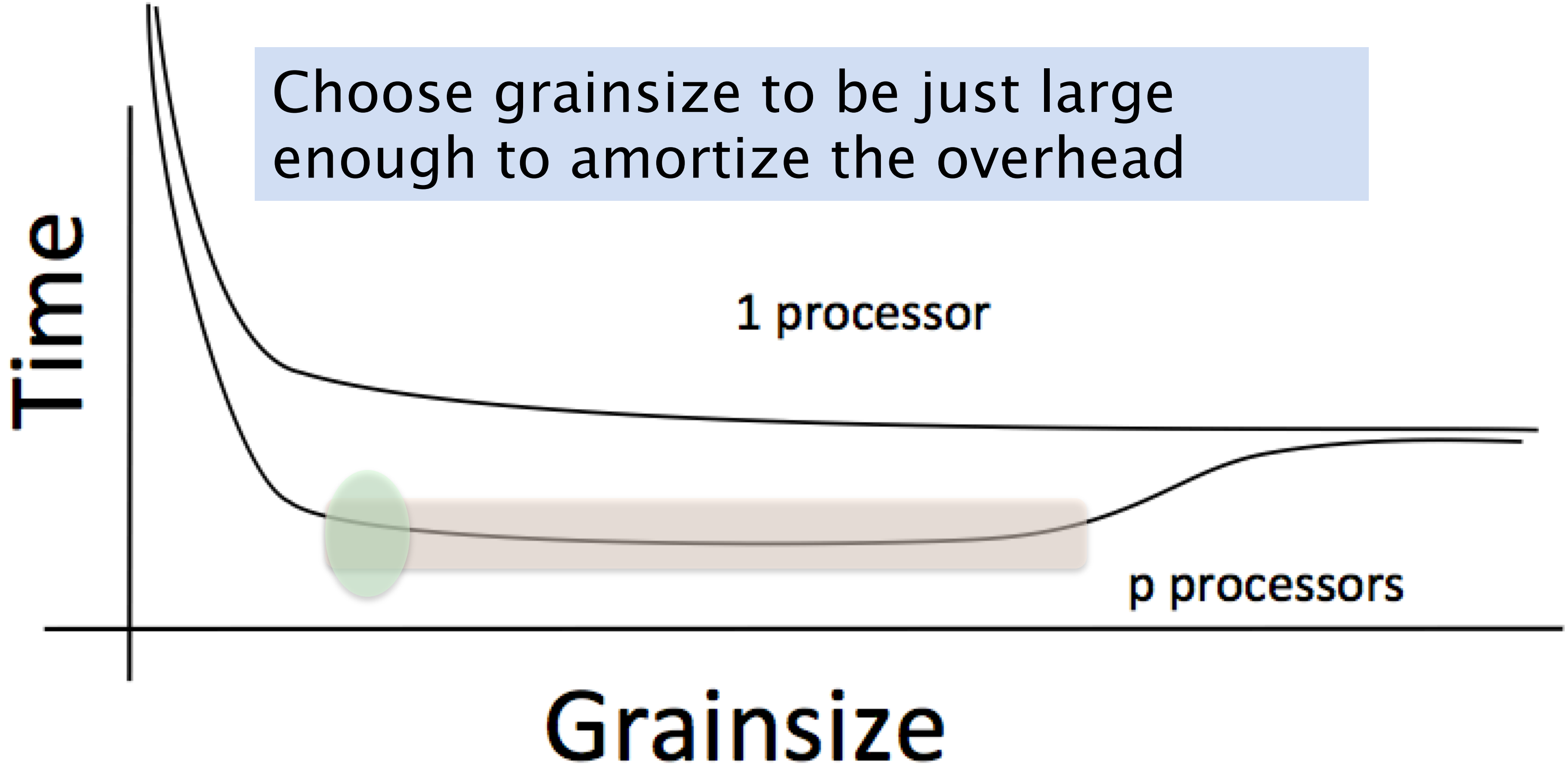




Instead of using 64 work units on 64 cores, used 1024 on 64

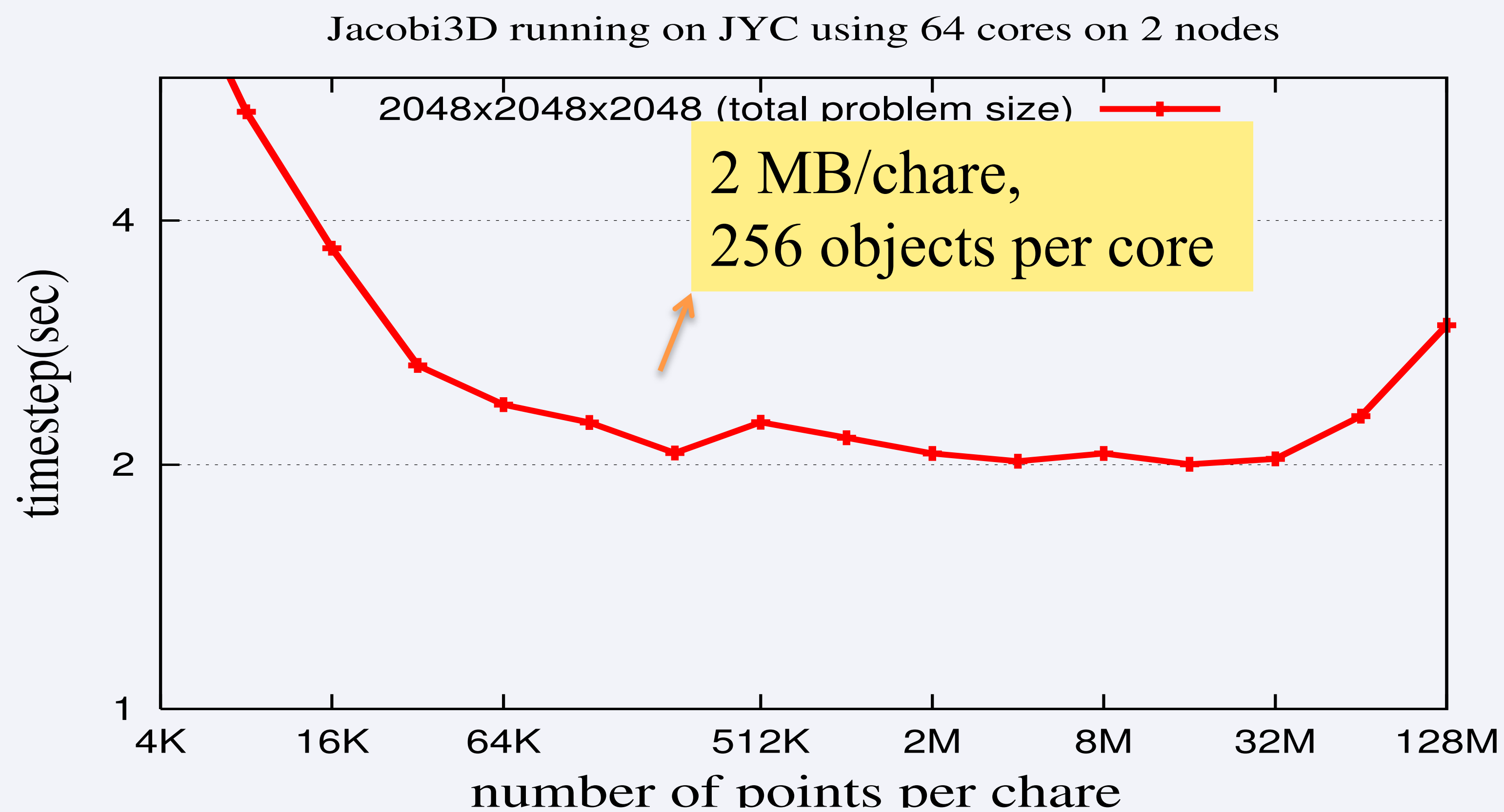
Working definition of grainsize :
amount of computation per remote interaction

Choose grainsize to be just large enough to amortize the overhead



Grainsize in a common setting

A 3D stencil computation



Restating: Over-decomposition

- Programmers decompose a computation into entities
  - Work units, data-units, composites
  - Into *coarse-grained* set of objects
  - Independent of number of processors
- The entities communicate with each other without reference to processors
  - So each entity is like a virtual processor by itself
- Let an intelligent runtime system assign these entities to processors
  - RTS can change this assignment during execution
  - Migratibility! An essential feature
- This empowers the control system
  - A large number of observables
  - Many control variables created

Sanjay's Central Dogma

Another sense in which this is my central dogma:
I and my research group have been exploring this idea for almost 20 years now!

> *Overdecomposition is **essential** for effective parallel programs, for computer performance and for human productivity*

Adaptive Control Systems

- To exercise adaptive control at runtime:
  - One needs a rich set of observables and control variables
- My group's research over the past 15-20 years:
  - Can be thought of as a quest to add more observables and control variables
  - Programming models, languages ,libraries, including:
    - Charm++, AMPI, Charisma, MSA, Charj,
- Now, I'd like to consolidate the experience and knowledge gained, and express it in a new *abstract programming model*

XMAPP

- XMAPP is an abstract programming model:
  - That means it characterizes a set of prog. models
- For a programming model to belong to this set, it must support
  - X: Overdecomposition
    - (as in: 8X objects than cores)
  - M: Migratability
  - A: Asynchrony
    - and Adaptivity, as a consequence of all the above
- So, XMAPP stands for:
  - Overdecomposition-based Migratibility, Asynchrony and Adaptivity in Parallel Programming

Members of XMAPP-class

- The programming models in XMAPP, or exhibit some features of it
  - Charm++
  - Adaptive MPI
  - KAAPI
  - ProActive
  - FG-MPI (if it adds migration)
  - HPX (once it embraces migratability)
  - ParSEC
  - CnC
  - MSA (multi-phase Shared arrays)
  - Charisma
  - Charj
  - DRMS (old abstraction from IBM research..)
  - Chapel: may be a higher level model
  - X10: has asynchrony, but not migratable units
  - Tascel

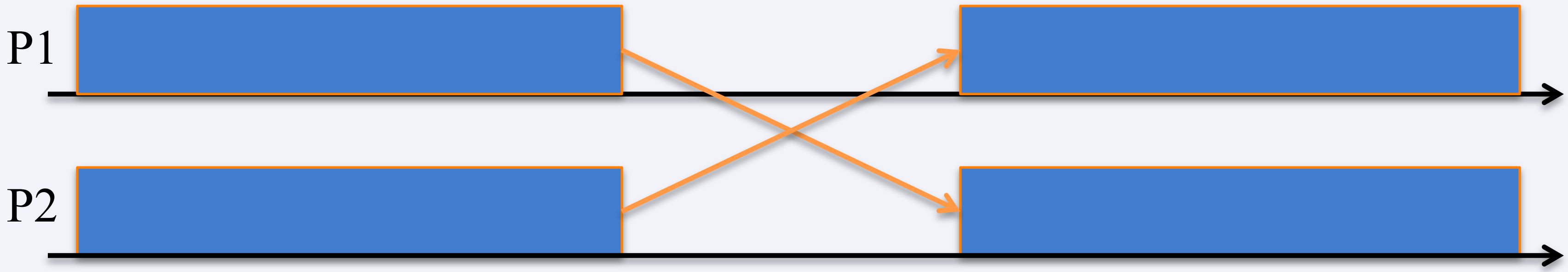
Also, general work on adaptivity is relevant: Trilinos, Hank Hoffman/UIC, …

HLPS and XMAPP

- To be able to use powerful adaptive runtime
  - Either it must belong to XMAPP class
  - Or it should compile/translate to an XMAPP class HLPS

Impact on communication

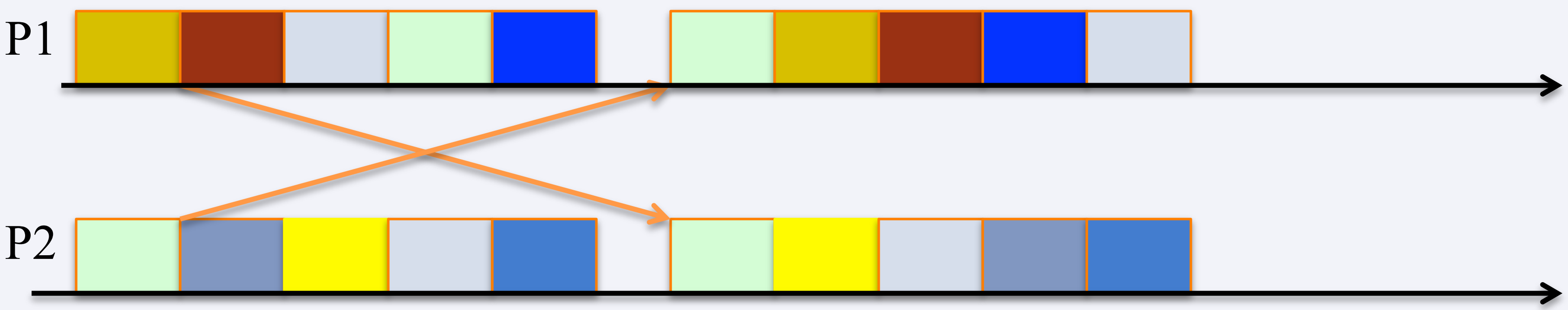
- Current use of communication network:
  - Compute-communicate cycles in typical MPI apps
  - So, the network is used for a fraction of time,
  - and is on the critical path
- So, current *communication networks are over-engineered for by necessity*



BSP based application

Impact on communication

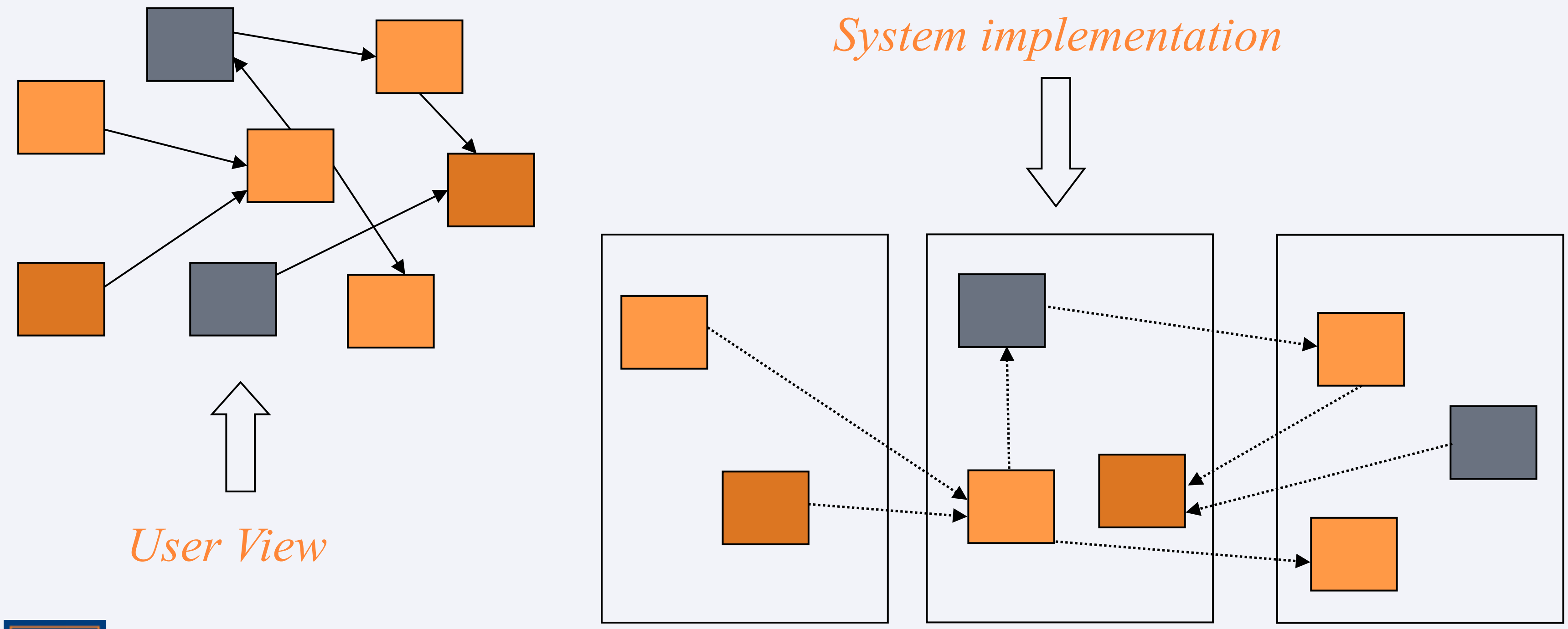
- With overdecomposition
  - Communication is spread over an iteration
  - Also, adaptive overlap of communication and computation

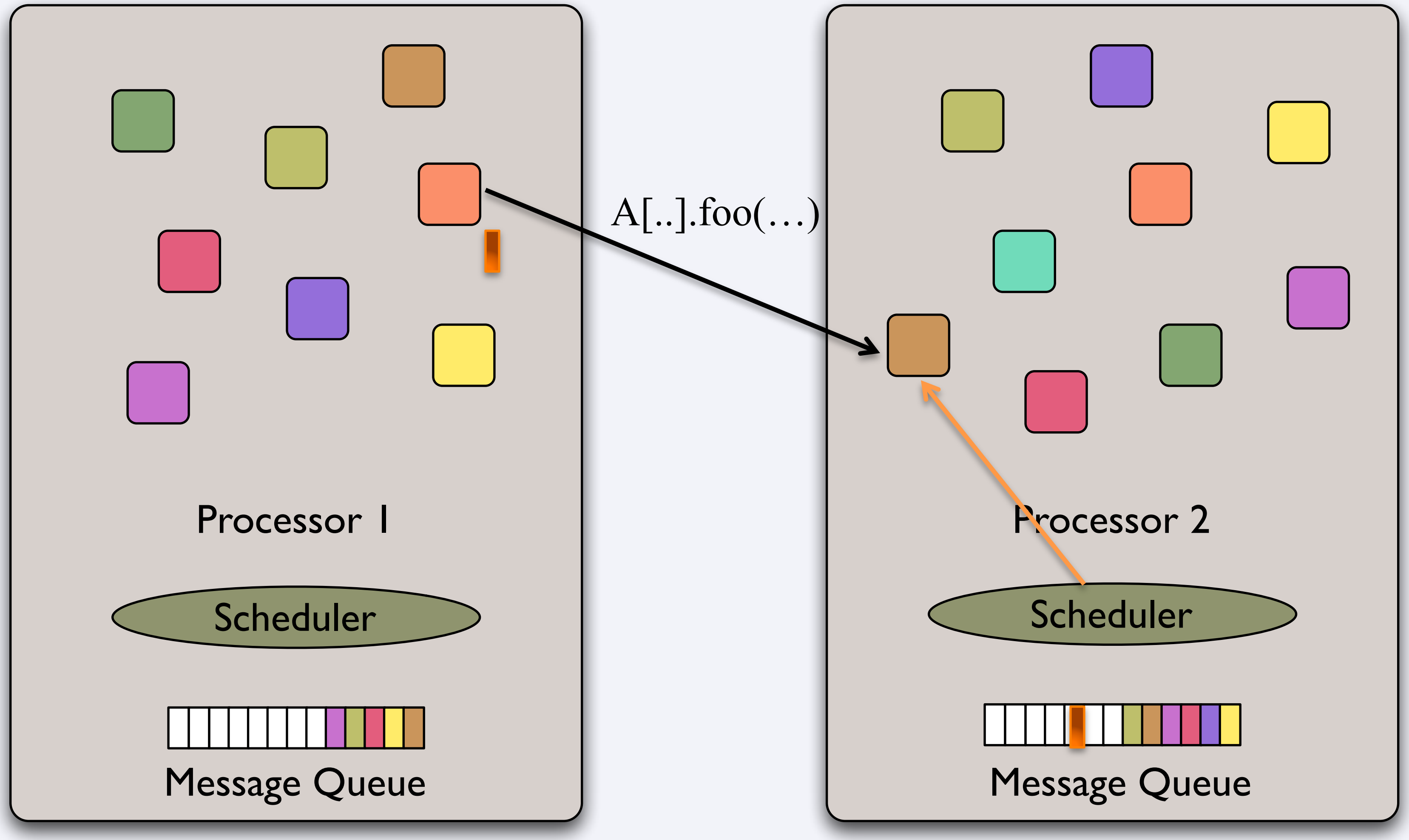


Overdecomposition enables overlap

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




A[..].foo(…)
Processor 1
Scheduler
Message Queue
Processor 2
Scheduler
Message Queue

Note the control points created

- Scheduling (sequencing) of multiple method invocations waiting in scheduler's queue
- Observed variables: execution time, object communication graph (who talks to whom)
- Migration of objects
  - System can move them to different processors at will, because..
- This is already very rich…
  - What can we do with that??

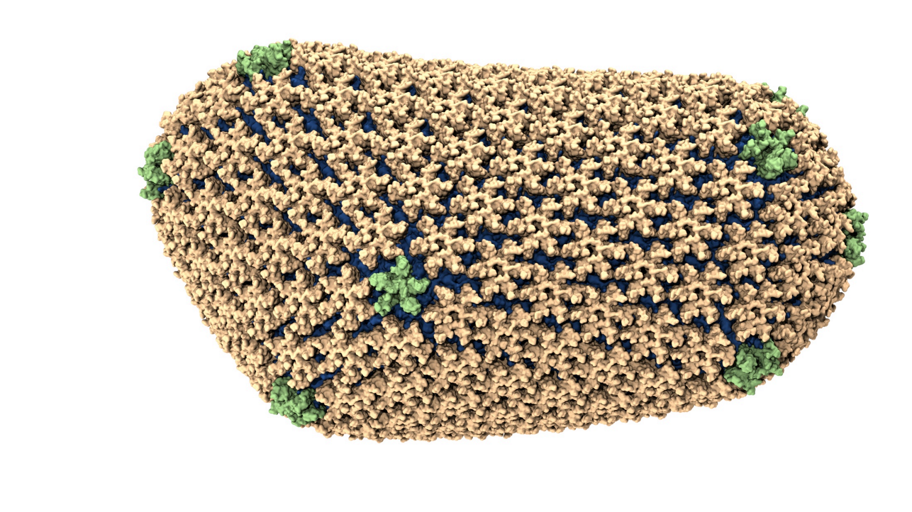
Optimizations Enabled/Enhanced by These New Control Variables

- Communication optimization
- Load balancing
- Meta-balancer
- Heterogeneous Load balancing
- Power/temperature/energy optimizations
- Resilience
- Shrink/Expand sets of nodes
- Application reconfiguration to add control points
- Adapting to memory capacity

XMAPP ideas and features have been demonstrated in full-scale production Charm++ applications

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



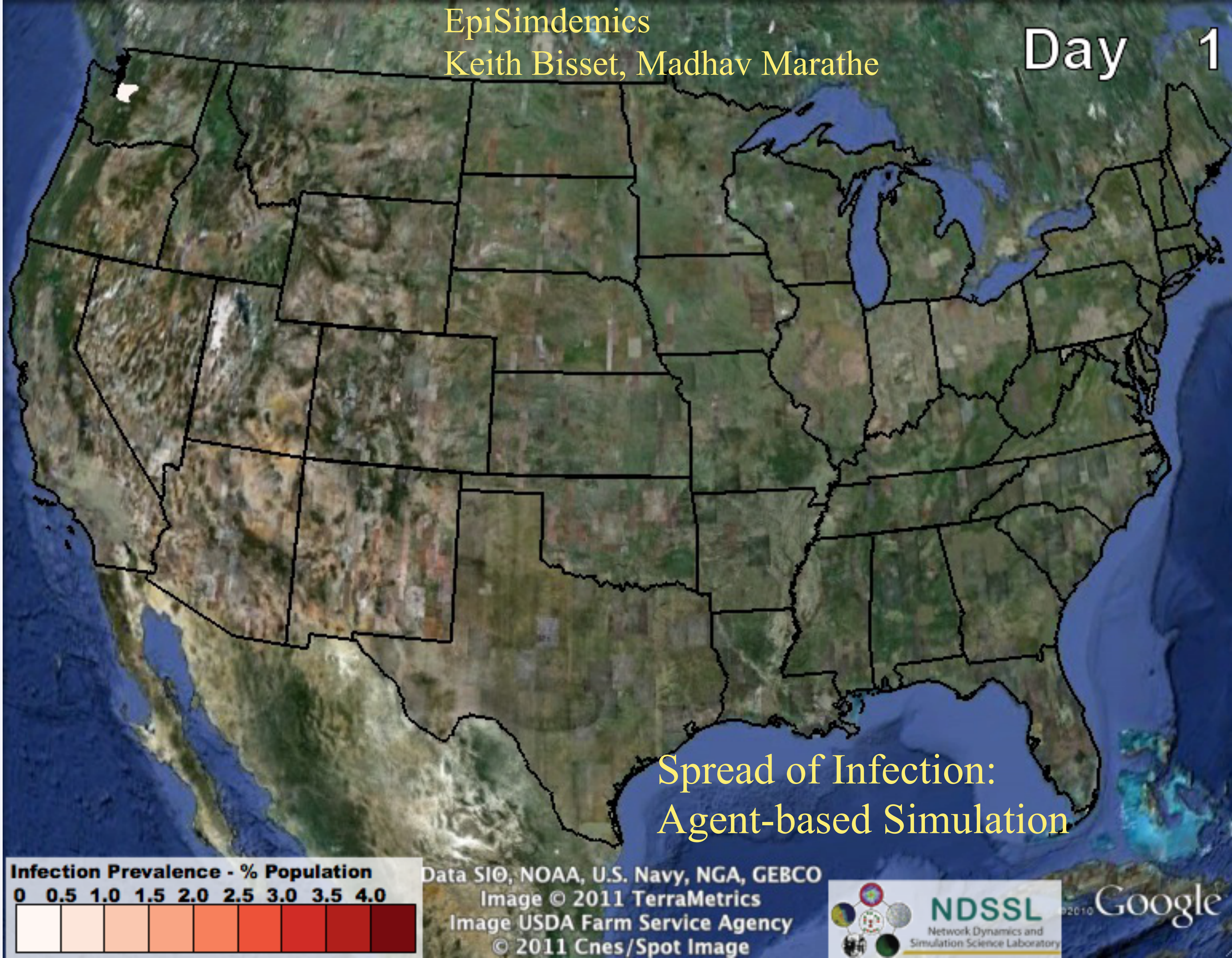
Recent success: Determination of the structure of HIV capsid by researchers including Prof Schulten

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

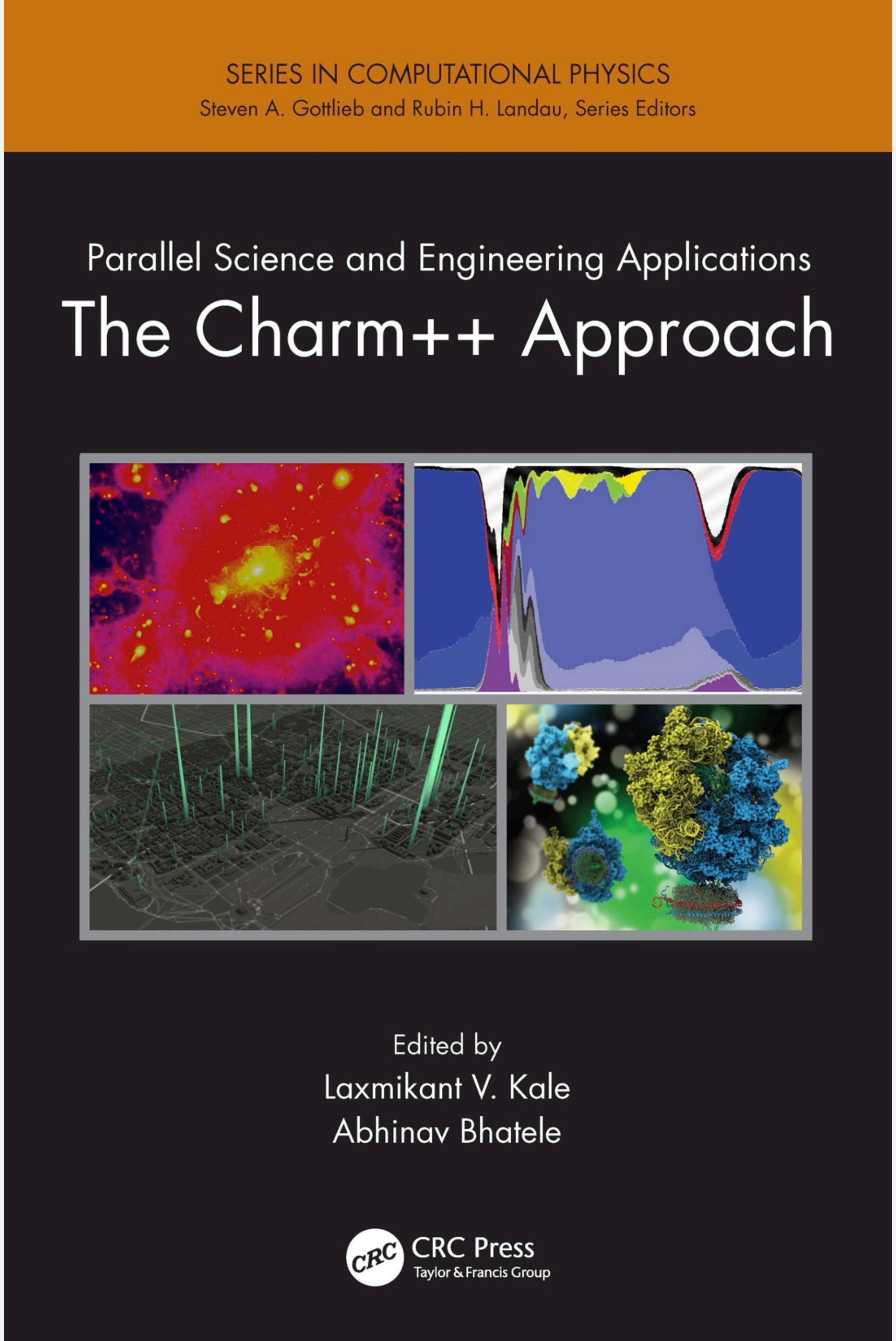
Evolution of Universe and Galaxy Formation

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


EpiSimdemics
Keith Bisset, Madhav Marathe
Day 1
Spread of Infection:
Agent-based Simulation
Infection Prevalence - % Population
0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image © 2011 TerraMetrics
Image USDA Farm Service Agency
© 2011 Cnes/Spot Image
NDSSL
Network Dynamics and Simulation Science Laboratory
Google

A just-published book
surveys seven major applications developed using Charm++

See booth#434
(CRC Press/
Taylor & Francis)



So, HLPS designers, IF you embrace overdecomposition, very powerful adaptive runtime techniques become feasible.

Moreover, these adaptive techniques are very much essential in the coming era of complex heterogenous and (yes) dynamic machines, and sophisticated and dynamic applications

High Level Programming Systems

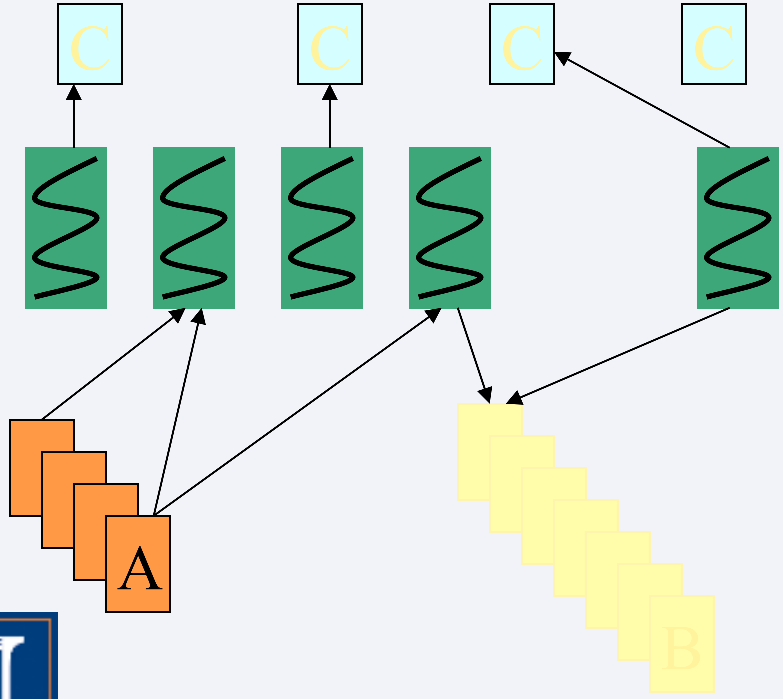
- Different ways of attaining "higher level"
  - Global view of data
  - Global view of control
  - Both
  - Simplified or specialized syntax
  - Safety properties
- But the largest benefits come from specialization
  - Domain specific languages
  - Domain specific Frameworks
  - Interaction-pattern specific languages

MSA: Multiphase Shared Arrays

Observations:

General shared address space abstraction is complex

Certain special cases are simple, and cover most uses

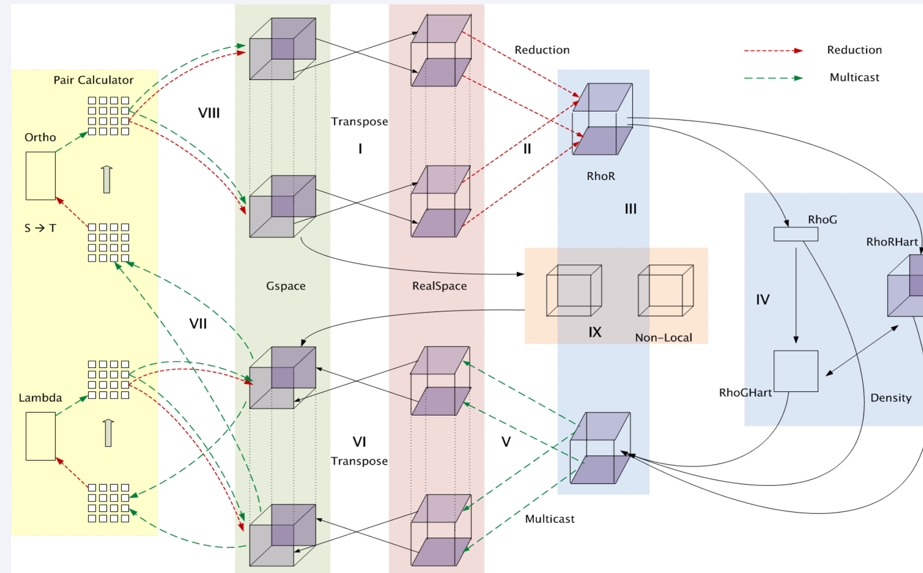


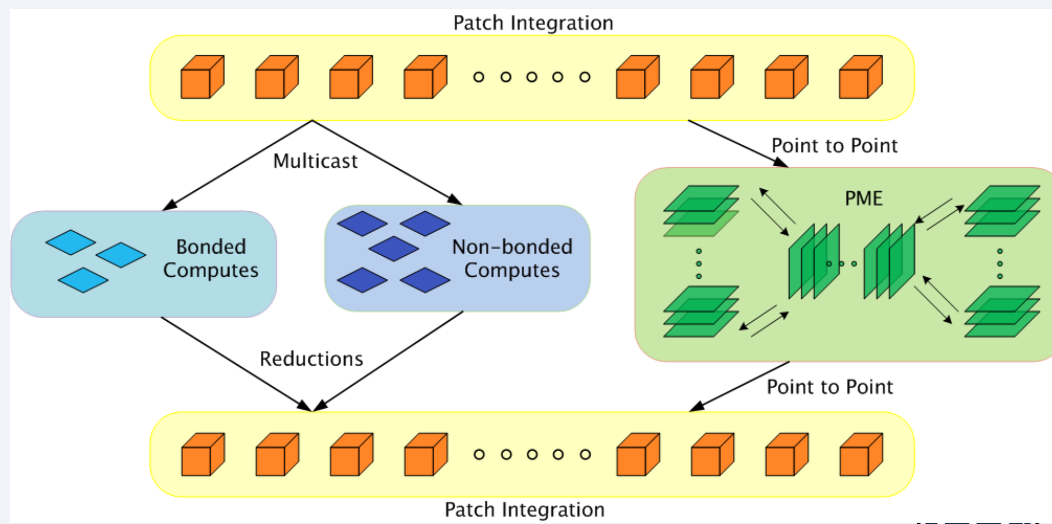
- In the simple model:
- A program consists of
  - A collection of Charm threads, and
  - Multiple collections of data-arrays
    - Partitioned into pages (user-specified)
- Each array is in one mode at a time
  - But its mode may change from phase to phase
- Modes
  - Write-once
  - Read-only
  - Accumulate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

Charisma: Static Data Flow

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

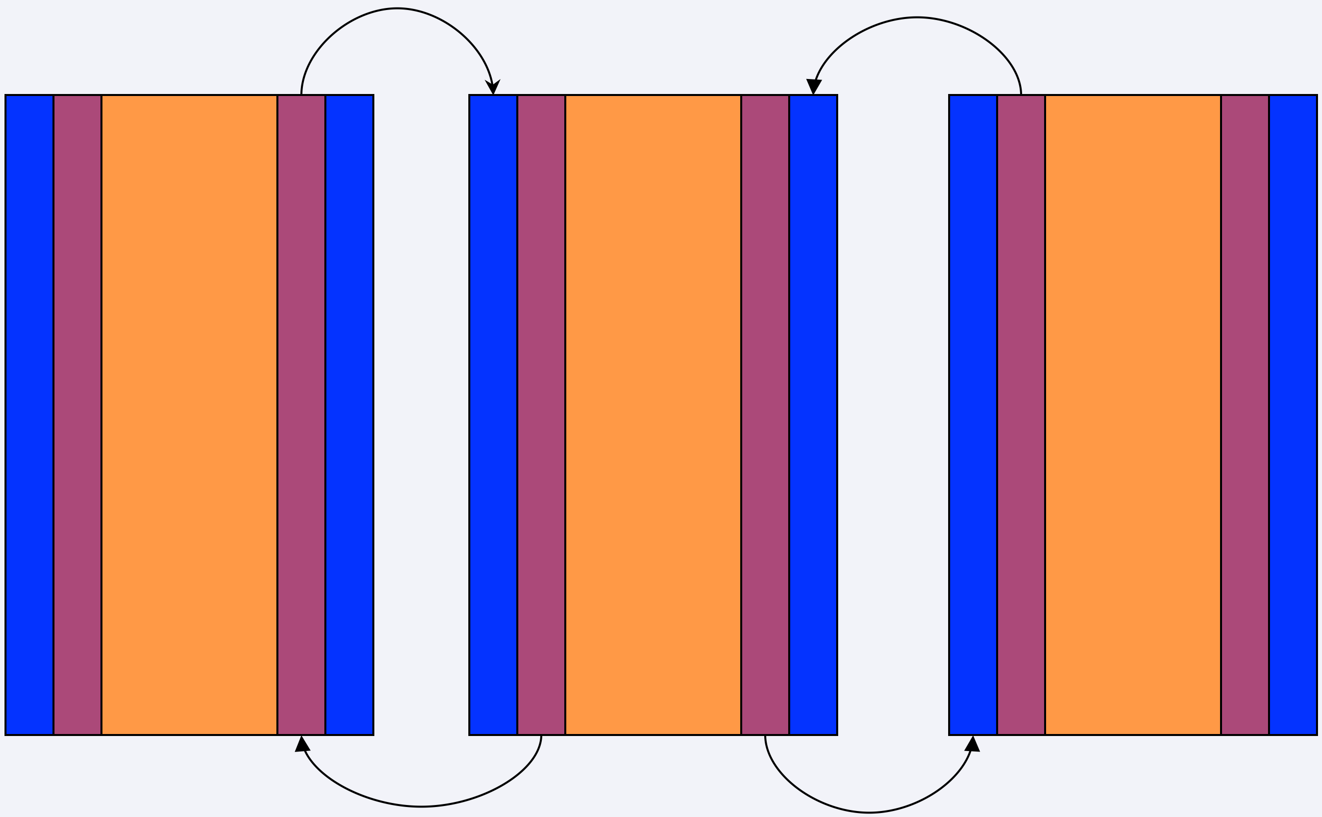
The amount of data may vary from iteration to iteration, but who talks to whom remains unchanged





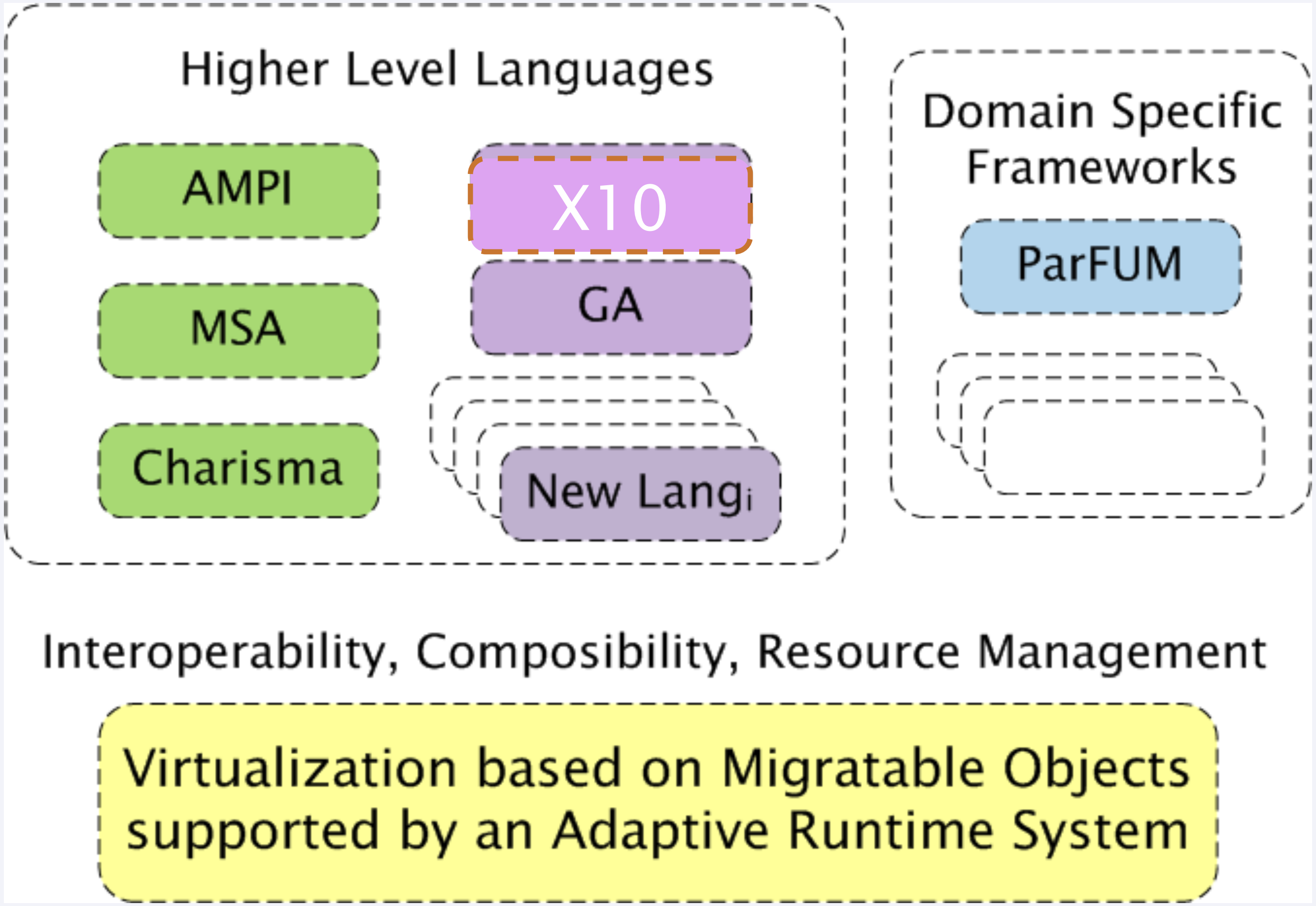
- *Arrays* of objects
- Global parameter space
  - Objects read from and write into it
- Clean division between
  - Parallel (orchestration) code
  - Sequential methods

Charisma++ example (Simple)



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

A View of an Interoperable Future



Interoperability

- So far:
  - One can write an application in one of several "languages", and have it use the same ARTS
- Interoperability requires
  - Allowing composing applications using modules written in different HLPS
  - So that they co-exist efficiently
  - So that they can exchange control and data uniformly
- For this:
  - we have to look at how HLPSs view a "processor" and how control transfers among program units

Implicit vs explict control transfer

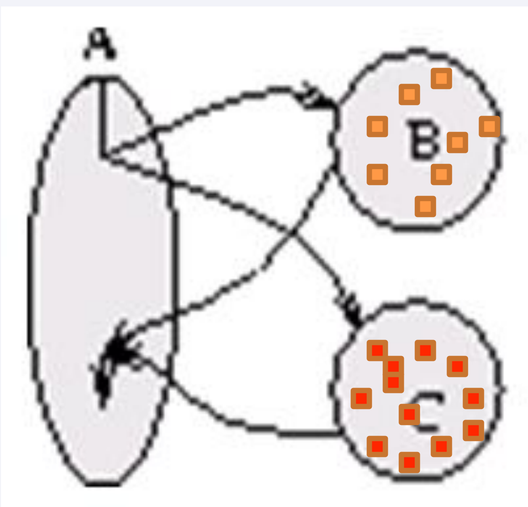
- Examples will illustrate this:
  - MPI (explicit): control transfer as directed by the programmer
  - Charm++ (implicit): control transfers as dictated by the message-driven scheduler at runtime
- Interoperability and control transfer regimes:
  - Within explicit HLPS (MPI/UPC/..)
  - Within implicit HLPS
    - Charm++/MSA/Charisma/.., all XMAPP HLPS
  - Across explicit and implicit HLPS

Explicit control transfer regime

- The interoperation itself is relatively easy:
  – As long as a common low-level runtime is agreed on, such as GASNET or Portals, …
- Typically:
  – Boils down to using one of several communication mechanisms
    - Send/recv, CAF style remote accesses, upc get/put
  – Still, leaves engineering issues to solve

Implicit regimes support parallel composition

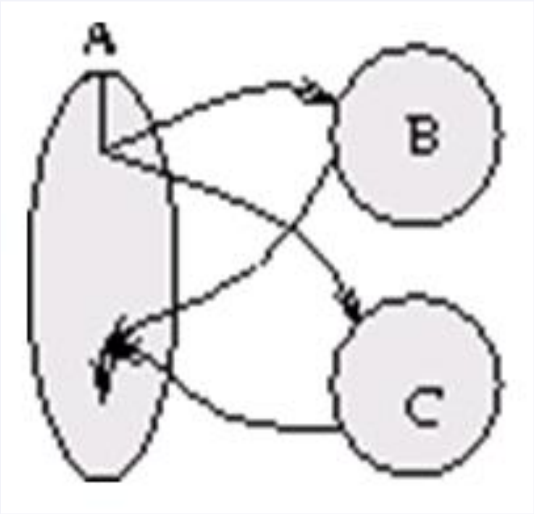
Parallel Composition: A1; (B || C); A2



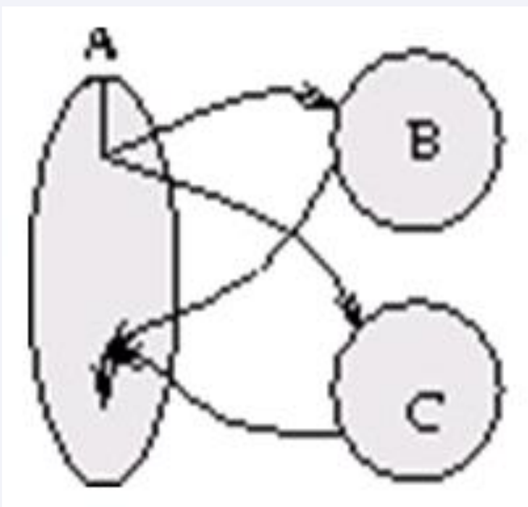
Recall: Different modules, written in different languages or paradigms, can overlap in time and on processors, without programmer having to worry about this explicitly

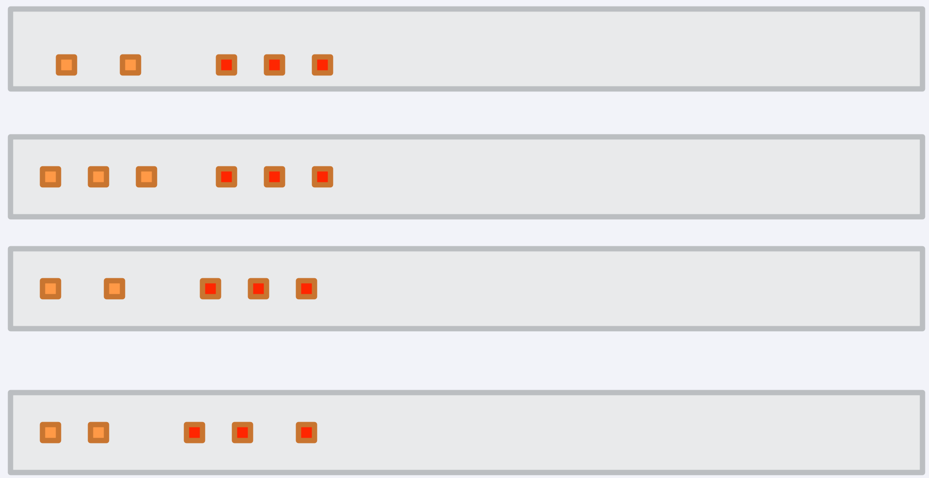
Without message-driven execution (and virtualization), you get either:

Space-division



OR: Sequentialization





Data transfer across modules

- For implicit regimes:
  - How to transfer data?
  - Programmer doesn't know where the sender or the receiver is (migratability)
  - Programmer doesn't know how to address the entities of the other module
    - Or else, we have libraries with L^2 interfaces!

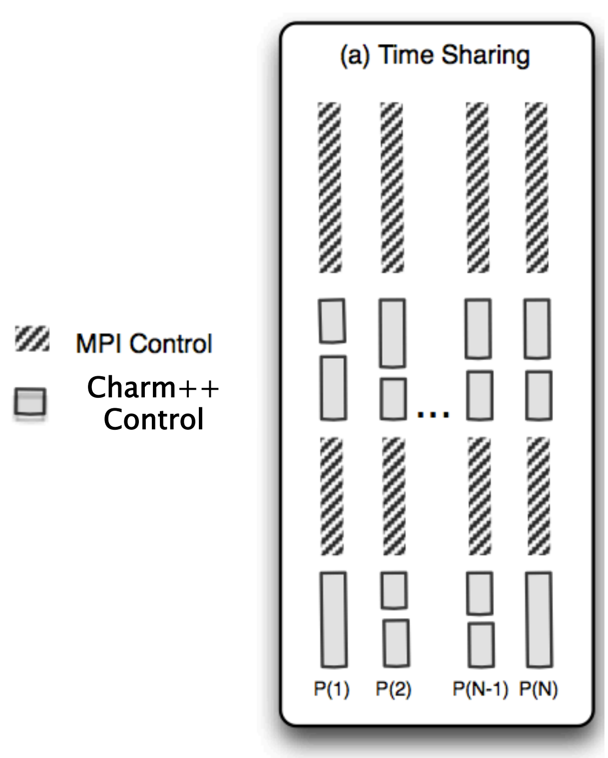
Data Transfer Solutions:

- Use MSA as a common medium!
  - Module1 deposits in an MSA, module 2 picks up data from MSA
  - MSA is then accepted as a common data transfer protocol by all libraries
- Use processor-level concentration
  - Deposit data to local processor
  - The other modules processor-level entities grab the data and redistribute it as needed
  - May need extensions in the HLPS
    - Or use low-level escape valvle for this purpose

Mixing Implicit and Explicit control

- Implicit HLPS have a message-driven scheduler
  - Buried deep inside its runtime
- The solution:
  - Expose the scheduler!
  - Make it a callable function

Charm++ interoperates with MPI



Interoperability: Recent experience

- Nikhil Jain extended Charm++ to facilitate interoperable libraries

Is Interoperation Feasible in Production Applications?

Application	Library	Productivity	Performance
CHARM in MPI (on Chombo)	HistSort in Charm++	195 lines removed	48x speed up in Sorting
EpiSimdemics	MPI IO	Write to single file	256x faster input
NAMD	FFTW	280 lines less	Similar performance
Charm++'s Load Balancing	ParMETIS	Parallel graph partitioning	Faster applications

Recap

- High Level Programming Systems need:
  - A common adaptive runtime system as a base
    - Should generate migratable work/data units, at the backend, to leverage most powerful runtime techniques
    - These necessitate implicit transfer of control
      - message-driven execution
  - Interoperation
    - Support and abstractions for interoperation and data-exchange across multi-paradigm boundaries
    - Challenging when implicit-control modules are involved
    - Showed some techniques that are useful, but more are needed
  - Message to HLPS developers:
    - Use an adaptive runtime system, such as Charm++, to build upon

Recap

- High Level Programming Systems need:
  - A common adaptive runtime system as a base
    - Should generate migratable work/data units, at the backend, to leverage most powerful runtime techniques
    - These necessitate implicit transfer of control
      - message-driven execution
  - Interoperation
    - Support and abstractions for interoperation and data-exchange across multi-paradigm boundaries
    - Challenging when implicit-control modules are involved
    - Showed some techniques that are useful, but more are needed
  - Message to HLPS developers:
    - Use an adaptive runtime system, such as Charm++, to build upon

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

See you at Charm++ BOF at
Tues 5:30-7:00, Rm 702-706

I am looking for a postdoc
and/or a research programmer