A Unified Runtime Infrastructure for Exascale Programming Models

Pavan Balaji

Argonne National Laboratory

(with input from others at Argonne, Oak Ridge, Lawrence Berkeley, Pacific Northwest, Lawrence Livermore, Sandia, UIUC, UH, OSU, Rice, IBM and Cray)

Presentation Layout

- **State of Programming Models**
- Application Requirements for the Exascale Era
- The Vision for a Unified Programming Infrastructure
- Technical Challenges
- Concluding Remarks

Current State of Programming Models

- Three broad categories
  - High-level Compilers/Languages
    - UPC, Chapel, CAF, …
  - High-level Libraries
    - Global data space models (Global Arrays, Global Trees) and Global computation space models (ADLB, Scioto, Charm++)
  - Low-level Runtime Systems
    - MPI, ARMCI, GASNET, OSPRI, accelerator models (OpenCL, CUDA), …
- Each model provides unique capabilities, but comes with its set of challenges as well

Application Challenges for Exascale Systems

- Applications have so far relied on more-or-less a single model
  - Most applications use MPI (either directly or through high-level domain-specific libraries)
  - Some applications use alternate models such as Global Arrays (NWChem) or UPC (NSA applications)
- As we move forward to exascale, applications will need more!
  - While the programming models that exist today lack capabilities to handle exascale challenges, we are not yet at a point where we need a completely new model
    - Each model has its flaws, but each model has its strengths too
    - Each model is very good at the set of things it is built for
  - Instead of redesigning a completely new programming model, we should leverage the strengths of the different models

Cherry-picking Programming Models

- Multi-model programming
  - For multimodule applications primarily based on MPI, how can a new module be written using alternate models such as UPC or CAF in such a way that it can interoperate with the rest of the application?
  - How can an application written in Cray Chapel or IBM X10 utilize math libraries written in MPI, such as PETSc, that have had close to a hundred man-years of development invested in them?
  - Can an MPI application directly move data from a local accelerator device to another physical node without explicitly using accelerator programming models to stage data locally before using MPI to move it outside the node?
  - If you have an ADLB application using work stealing and task migration, can it interact with Global Arrays to provide a globally accessible data region?

Presentation Layout

- State of Programming Models
- **Application Requirements for the Exascale Era**
- The Vision for a Unified Programming Infrastructure
- Technical Challenges
- Concluding Remarks

Application Requirements for the Exascale Era

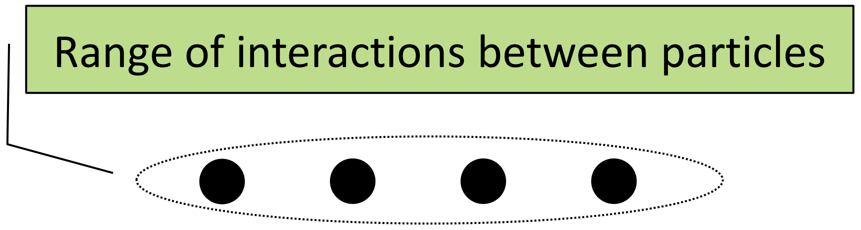
- Applications need to deal with two dimensions of problems:
  - The science they are trying to solve is becoming more complex (hence the need for exascale computing)
    - More data requirements, more computation
  - Hardware architectures are becoming more complex (hierarchical architectures, heterogeneous systems)
    - Current machines cannot just scale up because of cost and power constraints
- Current computation and communication methodologies used by applications cannot just migrate to exascale architectures
  - Too many variables here; everything will not magically scale

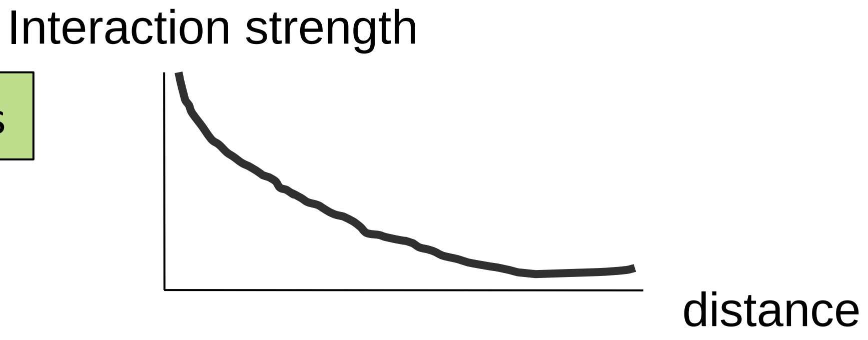
N-Body Coulomb Interactions

- Current applications have been looking at small-to-medium molecules consisting of 20-100 atoms
  - Amount of computation per data element is reasonably large, so scientists have been reasonably successful decoupling computation and data movement
- For exascale systems, scientists want to study molecules of the order of a 1000 atoms or larger
  - Coulomb interactions between the atoms is much stronger in the problems today than what we expect for exascale-level problems
  - Larger problems will need to support short-range and longer-range components of the coulomb interactions (possibly using different solvers)
    - Diversity in the amount of computation per data element is going to increase substantially
    - Regularity of data and/or computation would be substantially different

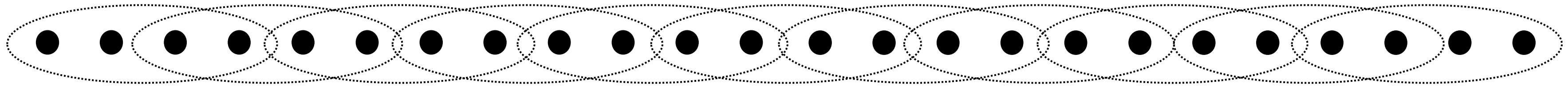
Quantum mechanical interactions are near-sighted (Walter Kohn)

Traditional quantum chemistry studies lie within the nearsighted range where interactions are dense:





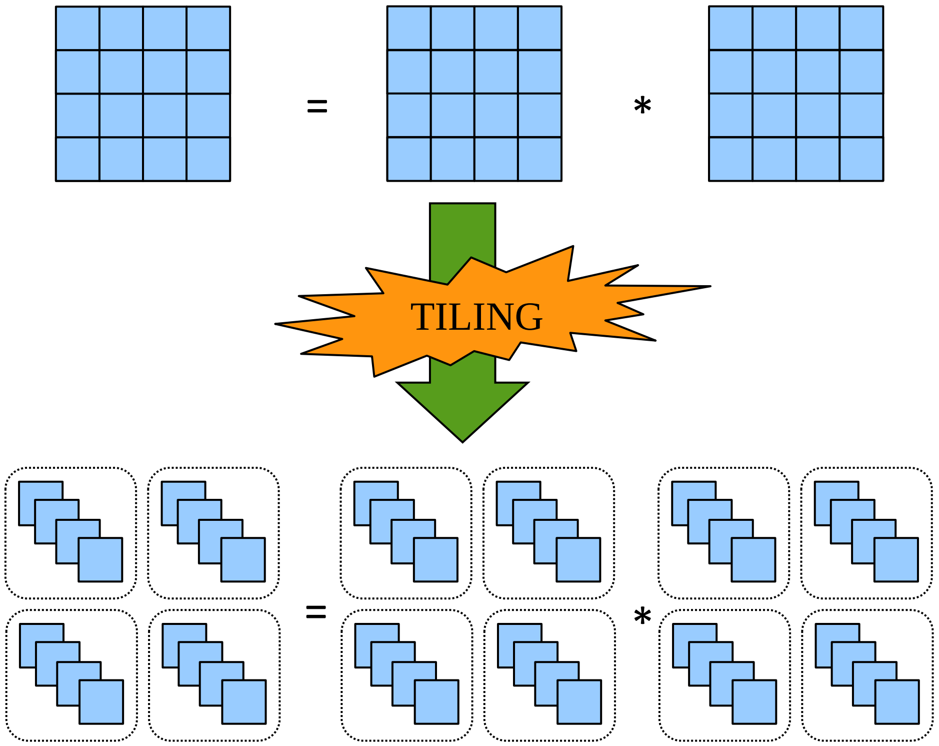
Future quantum chemistry studies expose both short- and long-range interactions:



Note that the figures are phenomenological. Quantum chemistry methods treat correlation using a variety of approaches and have different short/long-range cutoffs.

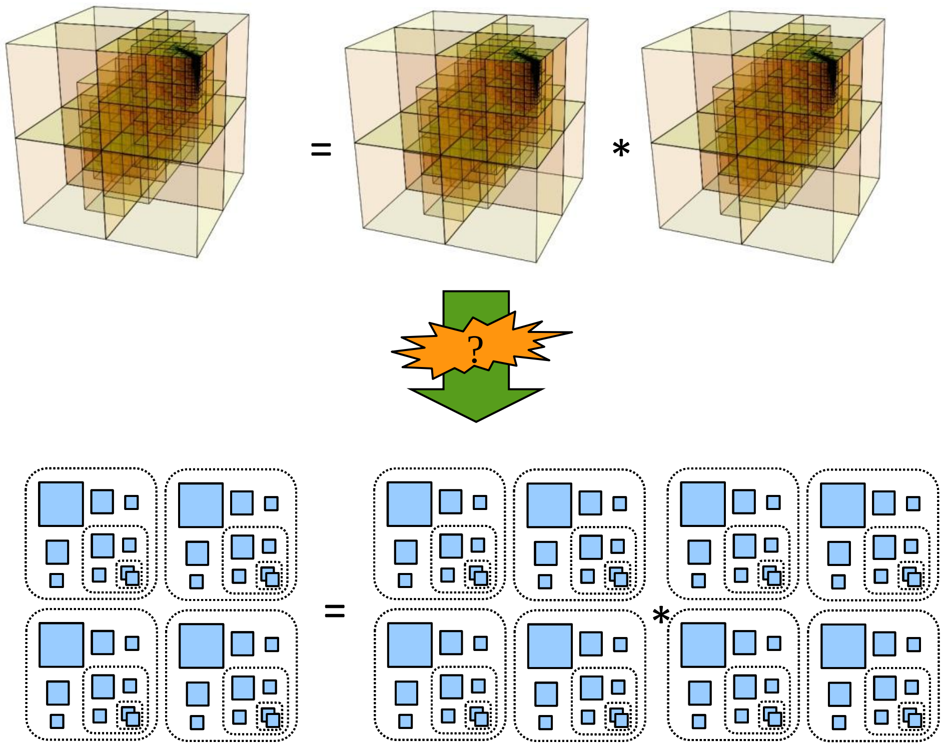
Courtesy Jeff Hammond, Argonne National Laboratory

Current: Regular Dense Computation



- Traditional models such as MPI or GA alone have been sufficient for this model of computation
  - Fetch data locally and compute

Exascale: Irregular Dense/Sparse Computation



- Traditional models "individually" are no longer sufficient
  - MPI or GA like model is good for dense parts of the data (fetch data locally and compute)
  - Charm++, ADLB or Scioto like model is good for the sparse parts

Another Motivating Example: GFMC

- Green's Function Monte Carlo -- the "gold standard" for *ab initio* calculations in nuclear physics at Argonne
- A non-trivial master/slave algorithm, with assorted work types and priorities; multiple processes create work; large work units
- Scaled to 2000 processors on BG/L a little over two years ago, then hit scalability wall
- Need to get to 10's of thousands of processors at least, in order to carry out calculations on ^{12}C, an explicit goal of the UNEDF SciDAC project
- The algorithm has had to become even more complex, with more types and dependencies among work units, together with smaller work units
- Want to maintain master/slave structure of physics code
- This situation called for the invention of a new library -- ADLB, the Asynchronous Dynamic Load Balancing Library (written in MPI)
  - Attacking general problem: how to devise a programming model that makes things simpler and more scalable at the same time

Memory Scalability of GFMC

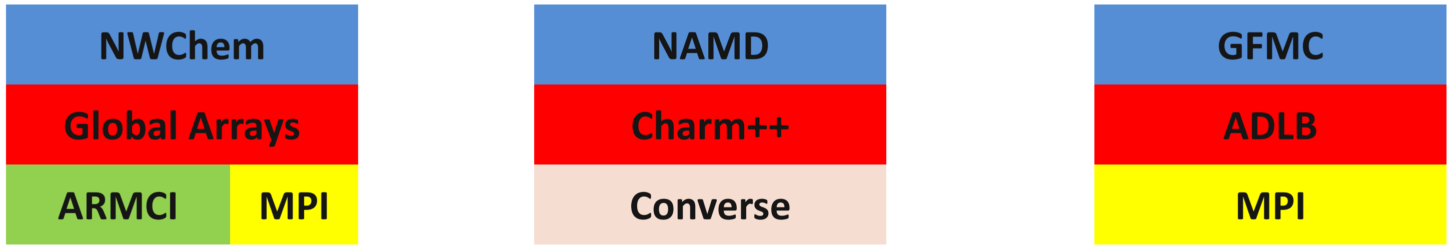
- GFMC's view of ADLB is that of a "generalized master-worker"
  - Each worker provides tasks to the "master" (physically distributed set of servers), and other workers can steal this work
  - Issues related to task dependencies/load-balancing are handled within ADLB (GFMC gives hints, but doesn't explicitly handle it)
- As GFMC moved to larger elements, the memory available to each task was no longer sufficient (factorial of atomic weight)
- First solution was MPI + OpenMP: allowed GFMC to scale to C-12
- Next steps forward are C-14 and O-16, and a simple task-based model such as ADLB is no longer sufficient
  - We need to investigate using ADLB in conjunction with GA or UPC, …
  - MPI to move data within an address space, but GA or UPC to expand the address space available to each process (global space)

Presentation Layout

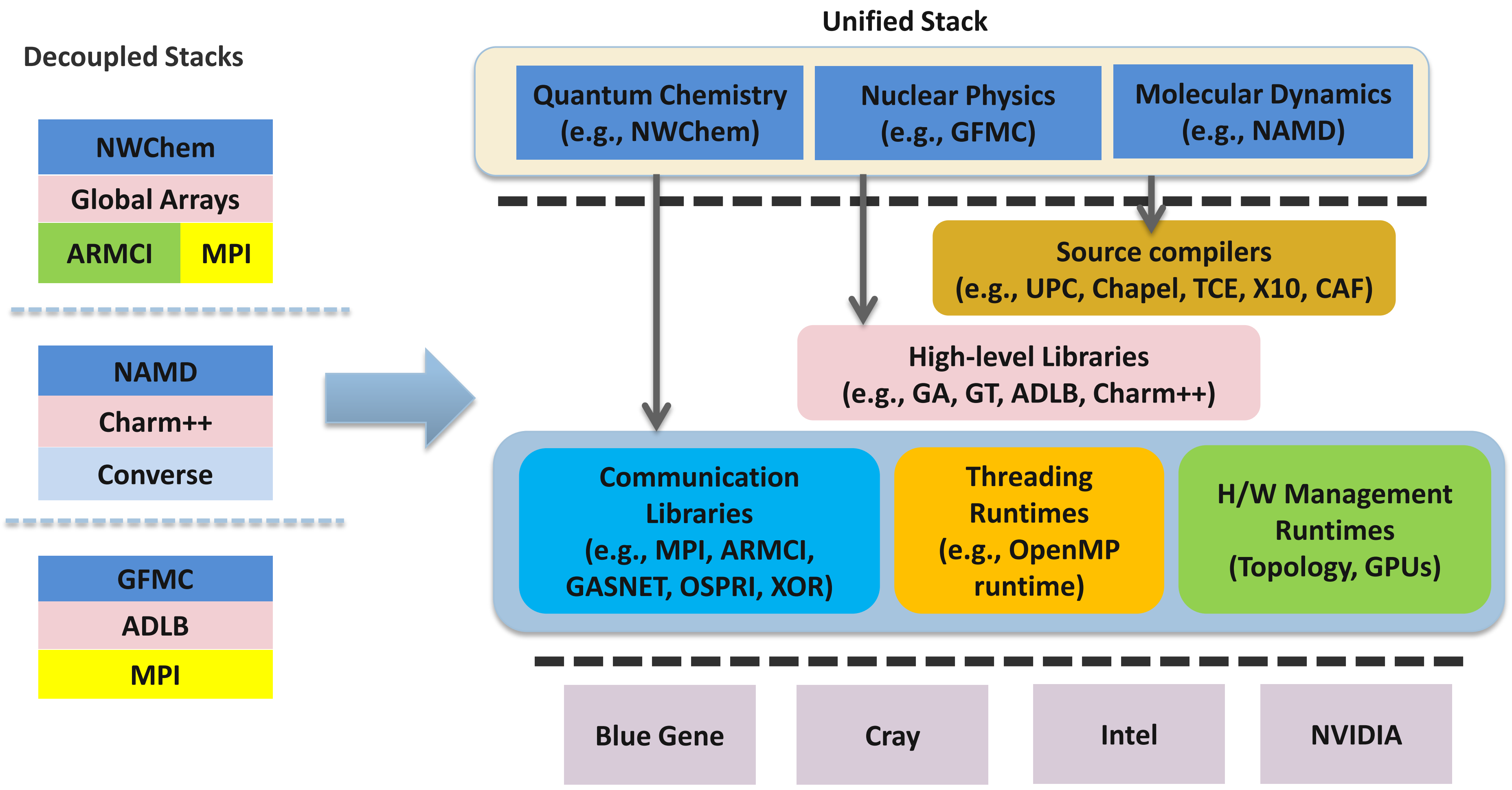
- State of Programming Models
- Application Requirements for the Exascale Era
- **The Vision for a Unified Programming Infrastructure**
- Technical Challenges
- Concluding Remarks

A Separate Runtime System for each Application

- Each application packaged with its own high-level programming library (GA, Charm++, ADLB, MADNESS runtime) on top of a different low-level runtime (MPI, ARMCI, GASNET)
- This model is fundamentally not sustainable at Exascale
  - *Interoperability between application models is difficult* – underlying runtime infrastructure has to be either interoperable or integrated
  - *Research optimizations are either redundant or not interoperable*
    - GA, GT, Data Spaces, etc., mostly do the same optimizations
    - For what's not repeated (e.g., if GA does something DS doesn't), they are not interoperable
  - *Commercial support impractical* – vendors will not support five runtime libraries – hard enough to get support for MPI + <anything else>



Vision for a Unified Programming Infrastructure



Decoupled Stacks
NWChem
Global Arrays
ARMCI
MPI
NAMD
Charm++
Converse
GFMC
ADLB
MPI
Unified Stack
Quantum Chemistry (e.g., NWChem)
Nuclear Physics (e.g., GFMC)
Molecular Dynamics (e.g., NAMD)
Source compilers (e.g., UPC, Chapel, TCE, X10, CAF)
High-level Libraries (e.g., GA, GT, ADLB, Charm++)
Communication Libraries (e.g., MPI, ARMCI, GASNET, OSPRI, XOR)
Threading Runtimes (e.g., OpenMP runtime)
H/W Management Runtimes (Topology, GPUs)
Blue Gene
Cray
Intel
NVIDIA

The key is to provide a unified architectures with multiple levels of capabilities and ALLOW APPLICATIONS TO BREAK THE LAYERING → transition path for applications!

Fundamental Concept of the Unified Stack

- Integrated runtime infrastructure
  - One single, unified stack that can provide multiple interfaces: MPI, ARMCI, GASNET, OSPRI, threading models
  - Extensions for capabilities not directly available in any model: hardware topology information, accelerator-specific extensions
- High-level libraries utilize the unified runtime infrastructure making interoperability simpler
  - Global Arrays can interoperate with ADLB, or Charm++ can interoperate with Data spaces
- High-level languages (such as UPC, CAF) can utilize either high-level libraries or the unified runtime infrastructure as their target runtime infrastructure

Community-wide Collaborative Effort

- DOE Laboratories
  - Argonne National Laboratory
  - Oak Ridge National Laboratory
  - Lawrence Berkeley National Laboratory
  - Lawrence Livermore National Laboratory
  - Pacific Northwest National Laboratory
  - Sandia National Laboratory
- Universities
  - University of Illinois at Urbana-Champaign
  - Rice University
  - Ohio State University
  - University of Houston
- Industry
  - IBM
  - Cray

Planning Steps

- Collaboration between different laboratories, universities and industry labs
  - Effort includes programming model leads (co-producers of the unified stack) and co-design centers (consumers of the unified stack)
  - Everyone believes that is the right step forward, and we as a community need to make it happen
- We are having a few workshops to plan how this unified programming infrastructure will shape up
- IBM graciously hosted the first workshop in October
  - Initial discussions on what the goals of the unified stack would be
- We are working on a white paper that details what applications should expect from this effort

Presentation Layout

- State of Programming Models
- Application Requirements for the Exascale Era
- The Vision for a Unified Programming Infrastructure
- **Technical Challenges**
- Concluding Remarks

Technical Challenges and Key Functionality

- Memory Consistency
- Computation Management and Load Balancing
- Unified Communication Runtime and Progress Model
- Coordinated Management of Shared Resources
- Hybridization and Interoperability
- Heterogeneous/Accelerator Computing
- Memory Hierarchy and Threading
- Fault Tolerance
- Interaction with Performance and Debugger Tools

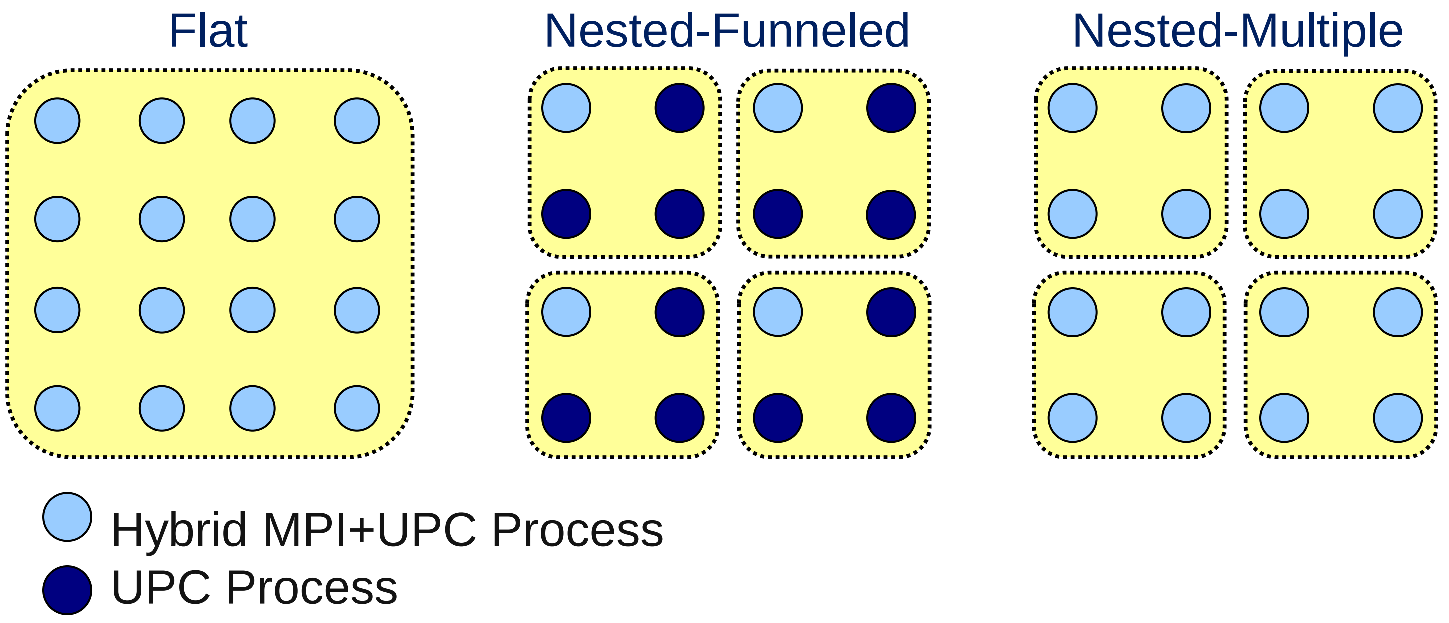
Challenges for the Unified Programming Infrastructure

- Unified semantics
  - What does it mean to have a non-blocking UPC collective followed by an MPI collective? How is it expected to behave?
  - Does an **ARMCI_FENCE** call guarantee completion of GASNET operations?
  - How are operations ordered between MPI and CAF?
- Interoperability of Data Objects
  - I should be able to do MPI operations on GASNET allocated memory regions – how will this work?
- Integrated Resource Management
  - Progress threads, buffer allocations, memory registration

Ad-hoc interactions being studied currently

- Some form of: MPI + threads, MPI + UPC, ADLB + MPI
- But we need a more truly unified (drag-and-drop) model
  - Migration path for applications to start using other models (any other model!) in conjunction with what they are already doing

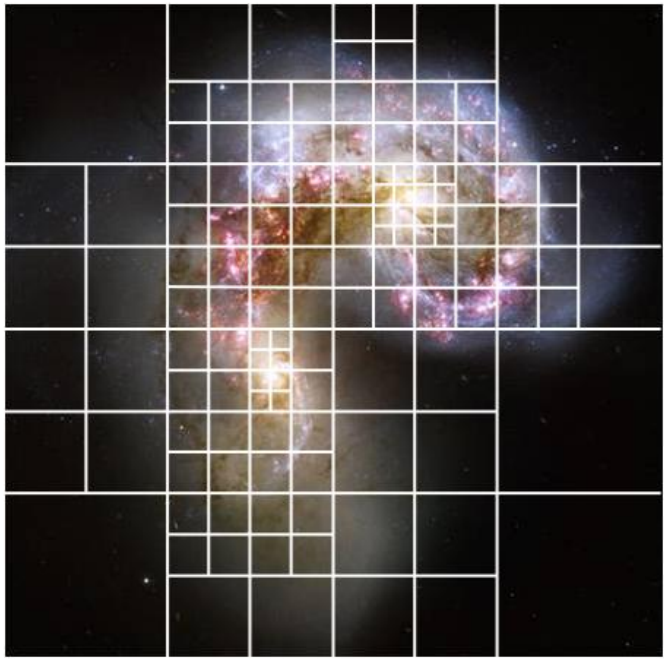
Case Study: Hybrid MPI+UPC Programming



- Many possible ways to combine MPI
  - Flat: One global address space
  - Nested: Multiple global address spaces (UPC groups)

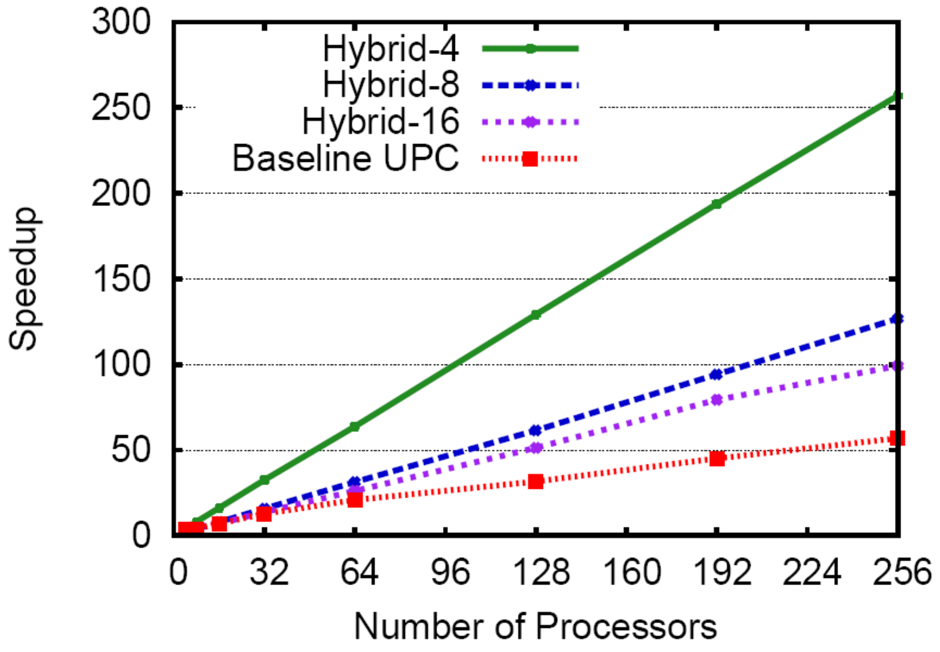
Barnes-Hut n-Body Simulation

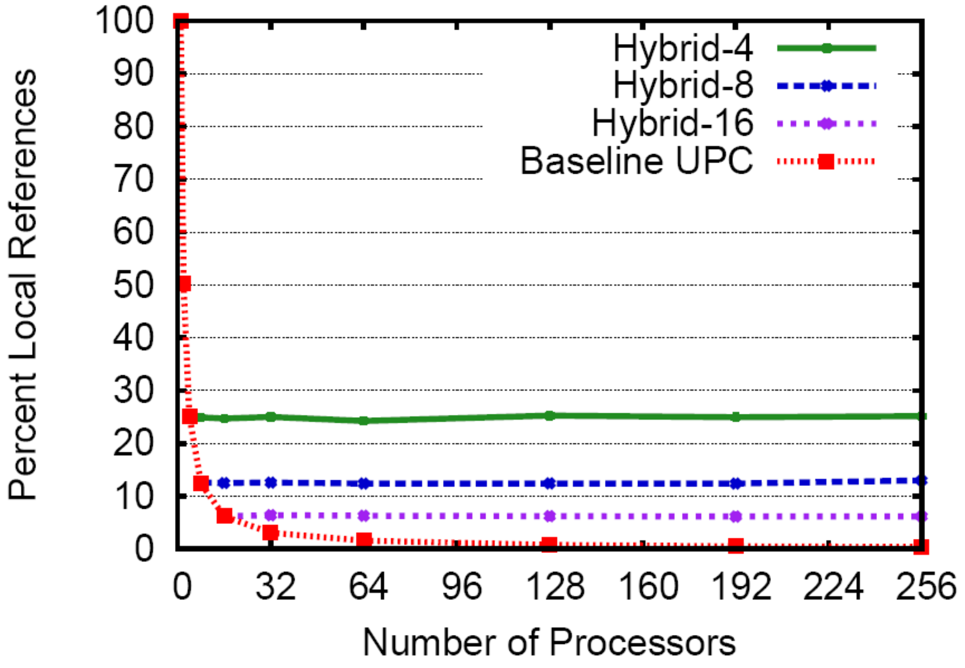
- Simulate motion and gravitational interactions of *n* astronomical bodies over time
- Represents 3-d space using an oct-tree
  - Space is sparse
- Summarize distant interactions using center of mass

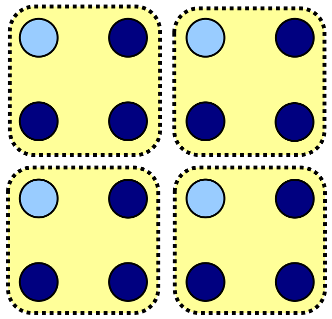


Colliding Antennae Galaxies (Hubble Space Telescope)

Hybrid MPI+UPC Barnes-Hut







- Nested-funneled model
  - Tree is replicated across UPC groups
- 51 new lines of code (2% increase)
  - Distribute work and collect results

Case Study: MPI + ARMCI (collaboration with ORNL and PNNL)

- Understanding what parts of ARMCI can sit on top of MPI-2 RMA and what parts need to be natively implemented for each platform
- Guide what is needed in MPI-3 RMA
- MPI-3 RMA functionality might encompass a lot of the features required by ARMCI (and even GASNET)

Presentation Layout

- State of Programming Models
- Application Requirements for the Exascale Era
- The Vision for a Unified Programming Infrastructure
- Technical Challenges
- **Concluding Remarks**

Concluding Remarks

- Several programming models out there, but applications might need more than what each of the provides
- It might be time for us to be able to allow applications to use multiples of these models together
  - While there has been some work that performs ad-hoc interactions between select model, we need a focused effort in combining the capabilities of many (or all) of these models
  - Applications should be able to pick and choose what they want to use based on application characteristics and requirements
- The unified programming infrastructure is a community-wide effort to bring together the capabilities of virtually all of the existing programming models available today

Web Pointers

Email: balaji@mcs.anl.gov

Web: http://www.mcs.anl.gov/~balaji

Argonne Programming Models Group

MPICH2: http://www.mcs.anl.gov/mpich2

Radix Systems Software: http://www.mcs.anl.gov/research/radix