Component Frameworks:

Laxmikant (Sanjay) Kale

Parallel Programming Laboratory

Department of Computer Science

University of Illinois at Urbana-Champaign

http://charm.cs.uiuc.edu

Group Mission and Approach

- To enhance *Performance and Productivity* in programming complex parallel applications
  - Performance: scalable to thousands of processors
  - Productivity: of human programmers
  - complex: irregular structure, dynamic variations
- Approach: Application Oriented yet CS centered research
  - Develop enabling technology, for a wide collection of apps.
  - Develop, use and test it in the context of real applications
  - Optimal division of labor between "system" and programmer:
    - Decomposition done by programmer, everything else automated
    - Develop standard library of reusable parallel components

Motivation

- Parallel Computing in Science and Engineering
  - Competitive advantage
  - Pain in the neck
  - Necessary evil
- It is not so difficult
  - But tedious, and error-prone
  - New issues: race conditions, load imbalances, modularity in presence of concurrency,..
  - Just have to bite the bullet, right?

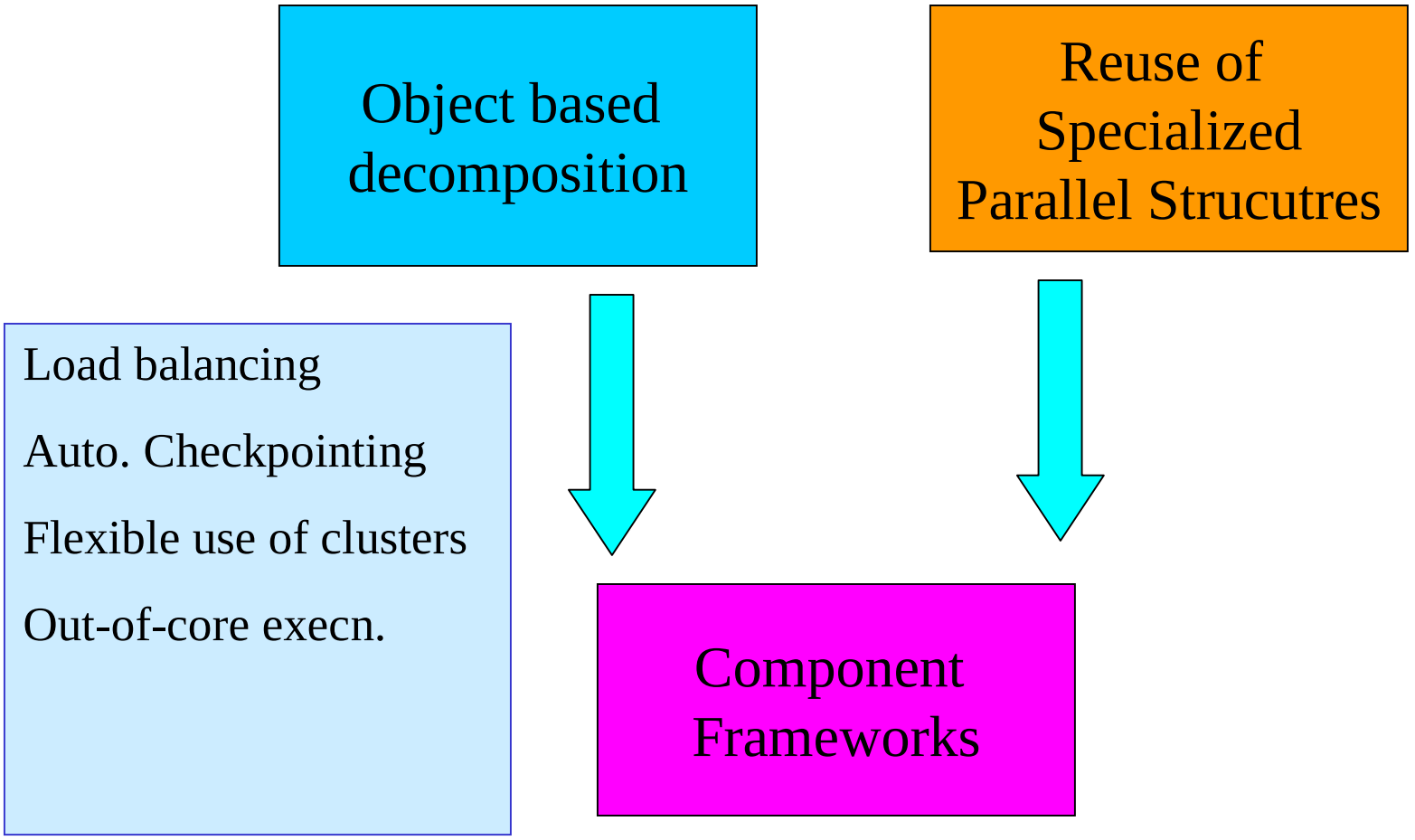
But wait…

- Parallel computation structures
  - The set of the parallel applications is diverse and complex
  - Yet, the underlying parallel data structures and communication structures are small in number
    - Structured and unstructured grids, trees (AMR,..), particles, interactions between these, space-time
- One should be able to *reuse* those
  - Avoid doing the same parallel programming again and again

A second idea

- Many problems require dynamic load balancing
  - We should be able to reuse load rebalancing strategies
    - It should be possible to separate load balancing code from application code
- This strategy is embodied in Charm++
  - Express the program as a collection of interacting entities (objects).
  - Let the system control mapping to processors

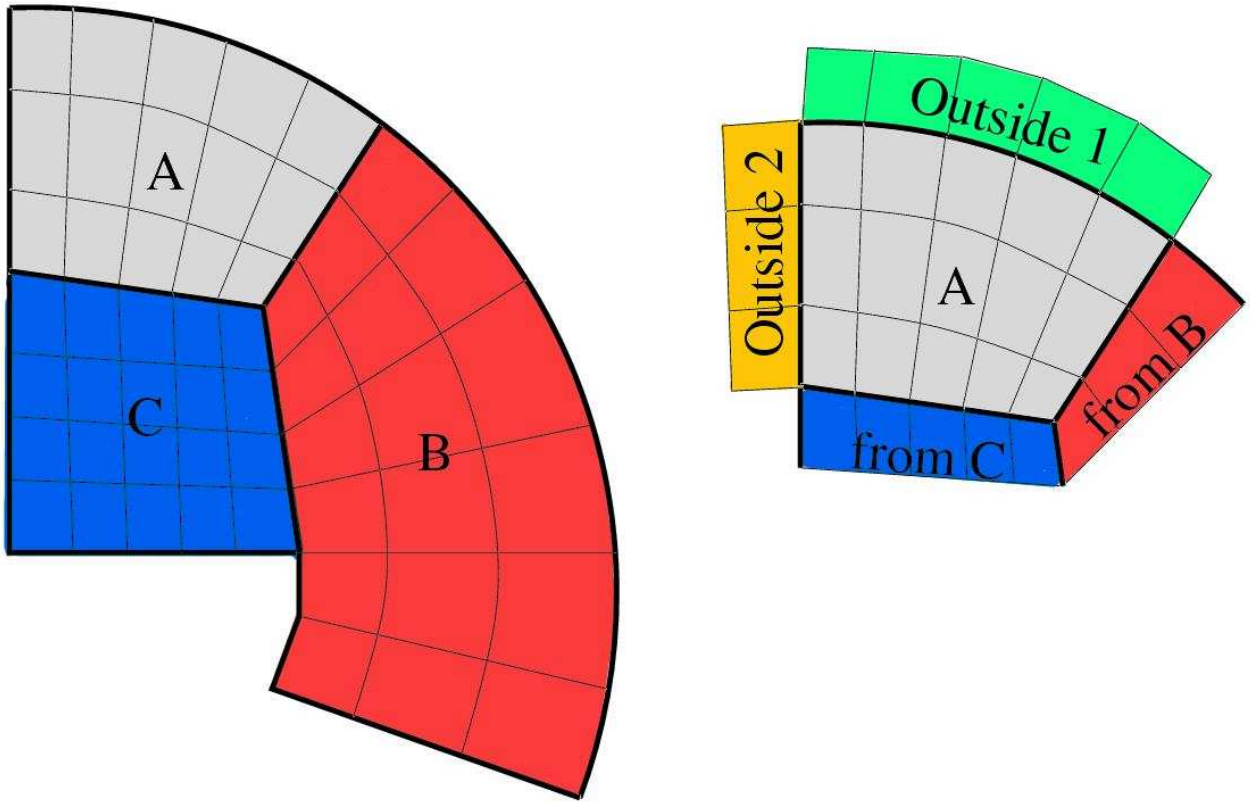
Charm Component Frameworks



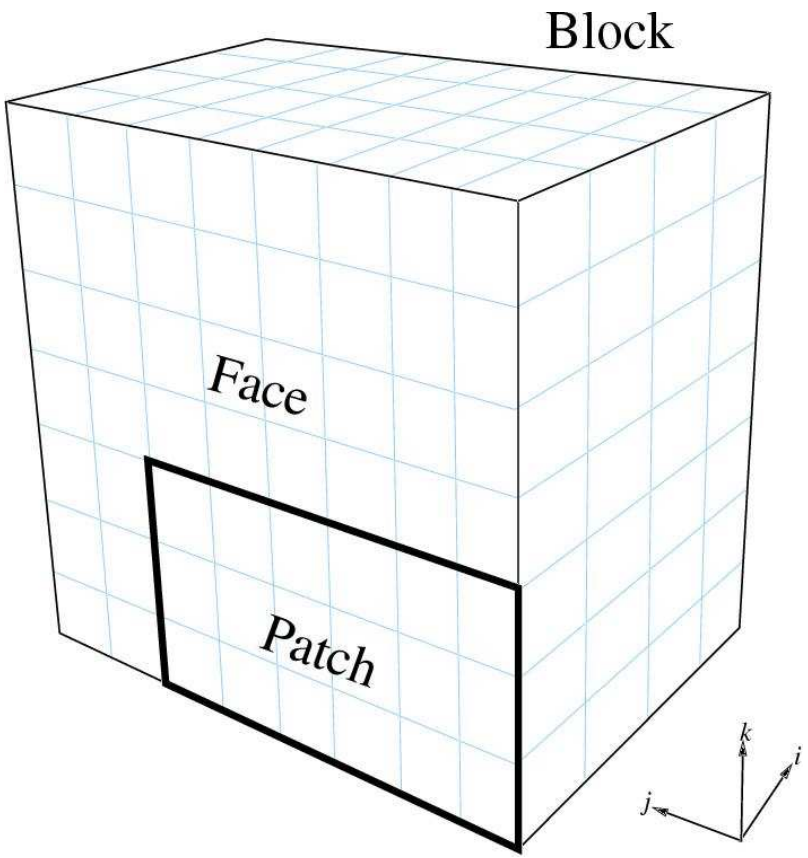
Current Set of Component Frameworks

- FEM / unstructured meshes:
  - "Mature", with several applications already
- Multiblock: multiple structured grids
  - New, but very promising
- AMR:
  - oct and quad trees

Multiblock Constituents



Terminology

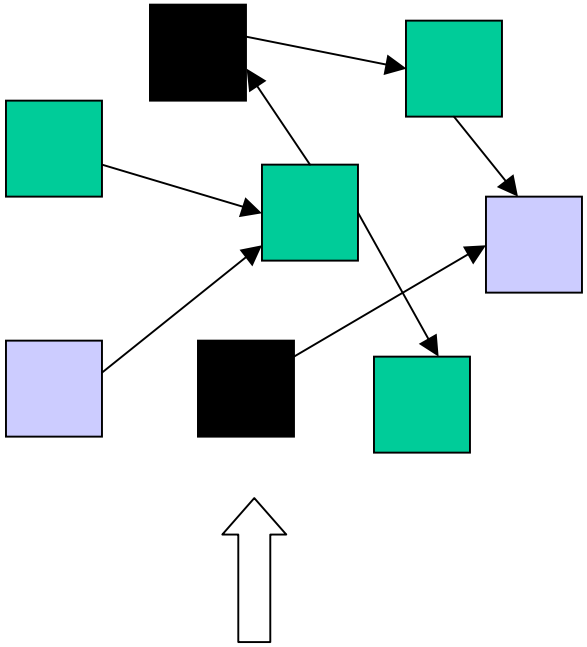


Multi-partition decomposition

- Idea: divide the computation into a large number of pieces
  - Independent of number of processors
  - typically larger than number of processors
  - Let the system map entities to processors

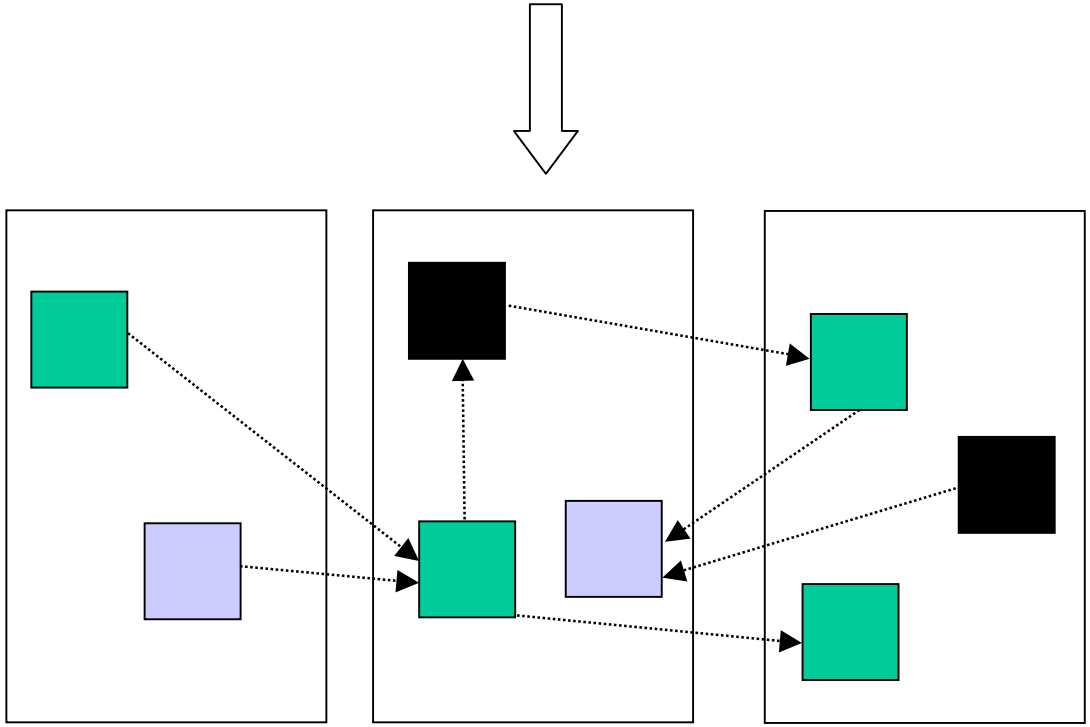
Object-based Parallelization

User is only concerned with interaction between objects



User View

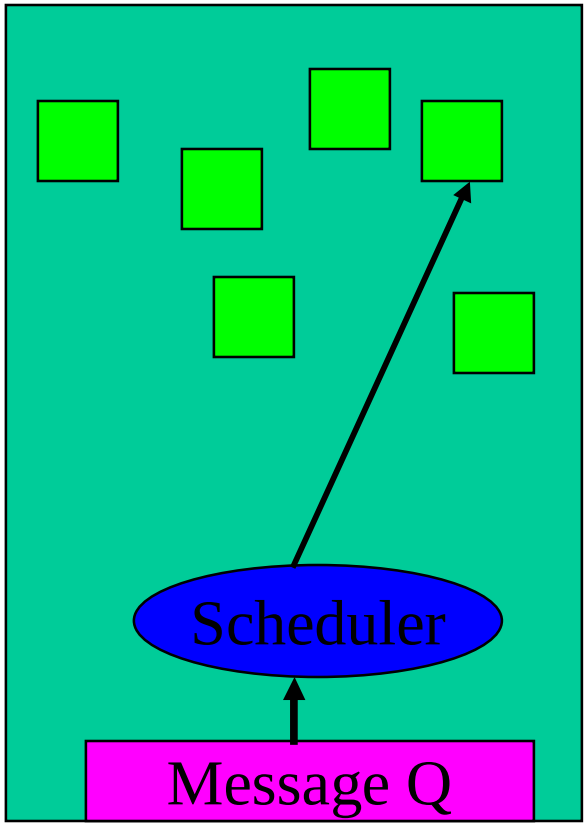
System implementation

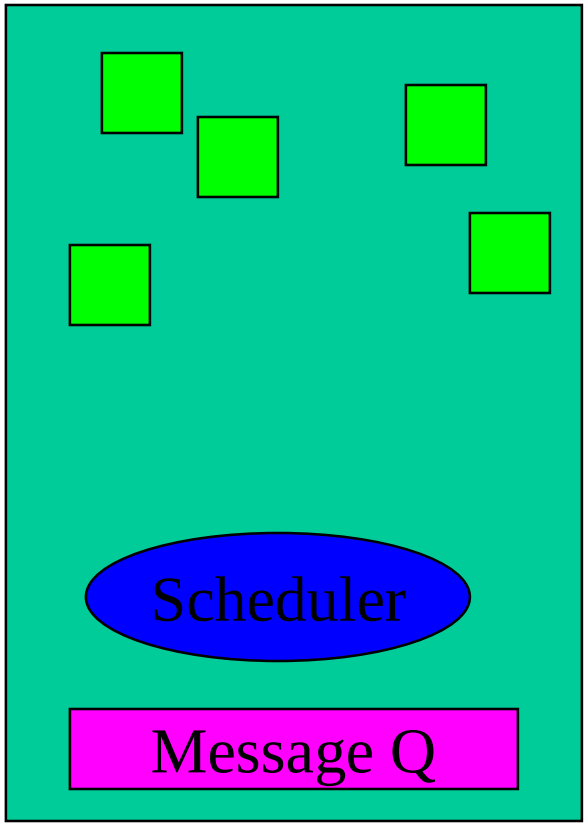


Charm++

- Parallel C++ with *Data Driven* Objects
- Object Arrays/ Object Collections
- Object Groups:
  - Global object with a “representative” on each PE
- Asynchronous method invocation
- Prioritized scheduling
- Mature, robust, portable
- http://charm.cs.uiuc.edu

Data driven execution

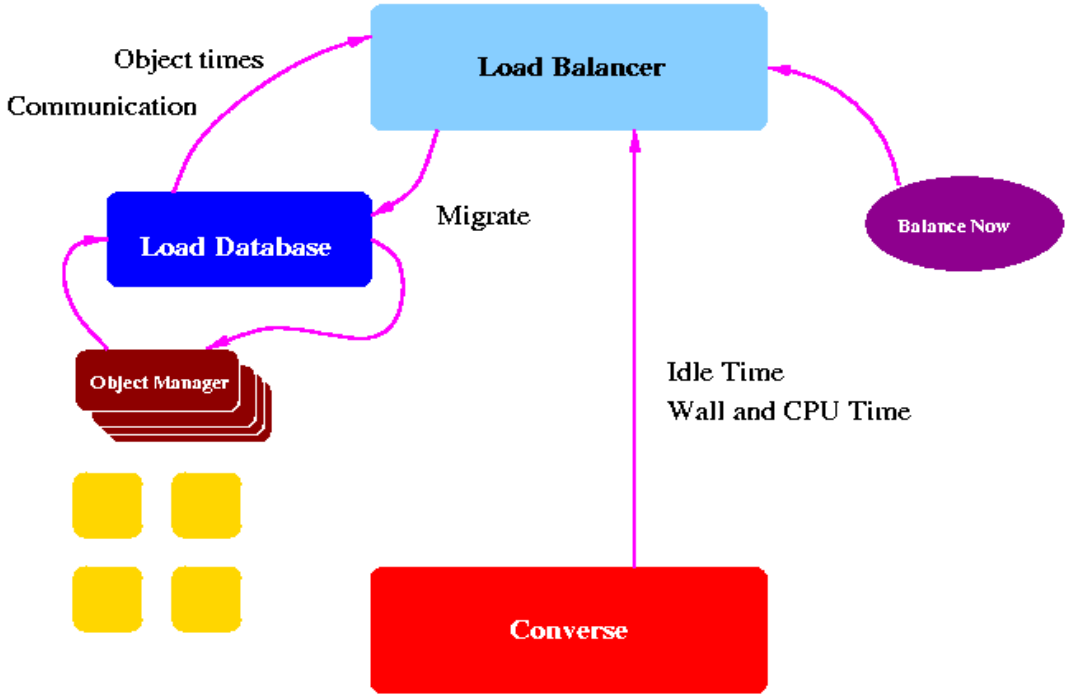




Load Balancing Framework

- Based on object migration and measurement of load information
- Partition problem more finely than the number of available processors
- Partitions implemented as objects (or threads) and mapped to available processors by LB framework
- Runtime system measures actual computation times of every partition, as well as communication patterns
- Variety of “plug-in” LB strategies available

Load Balancing Framework

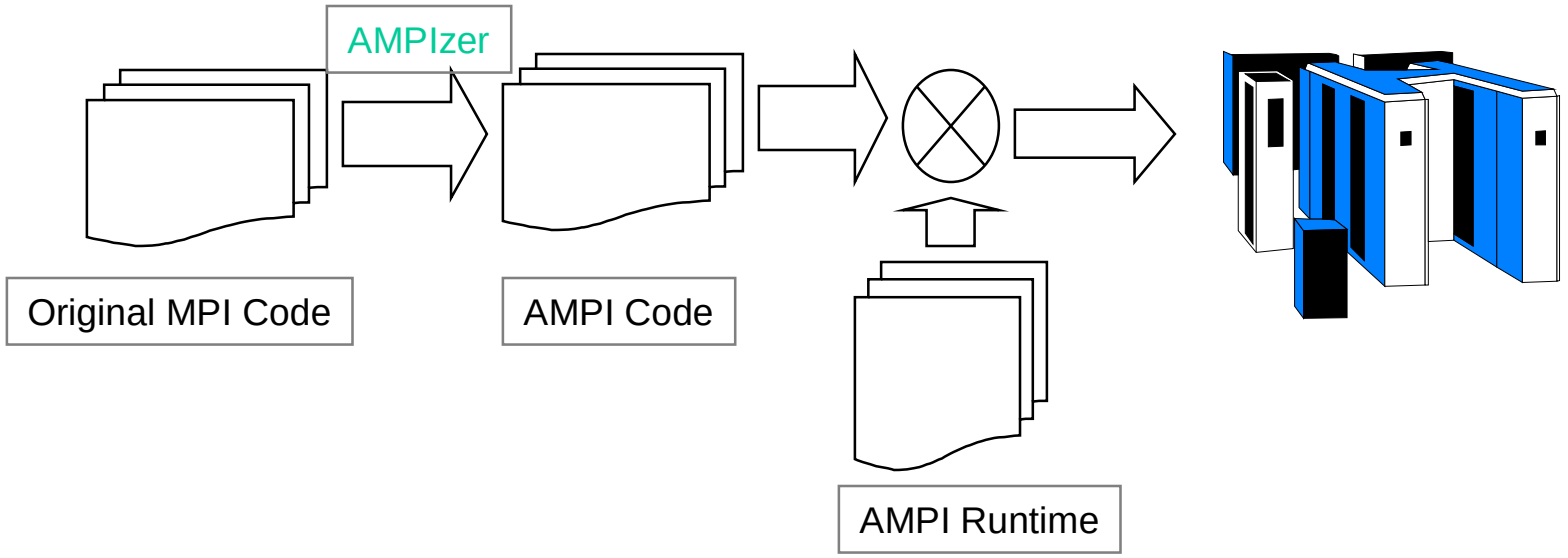


Building on Object-based Parallelism

- Application induced load imbalances
- Environment induced performance issues:
  - Dealing with extraneous loads on shared m/cs
  - Vacating workstations
  - Automatic checkpointing
  - Automatic prefetching for out-of-core execution
  - Heterogeneous clusters
- Reuse: object based components
- But: Must use Charm++!

AMPI: Goals

- Runtime adaptivity for MPI programs
  - Based on multi-domain decomposition and dynamic load balancing features of Charm++
  - Minimal changes to the original MPI code
  - Full MPI 1.1 standard compliance
  - Additional support for coupled codes
  - Automatic conversion of existing MPI programs



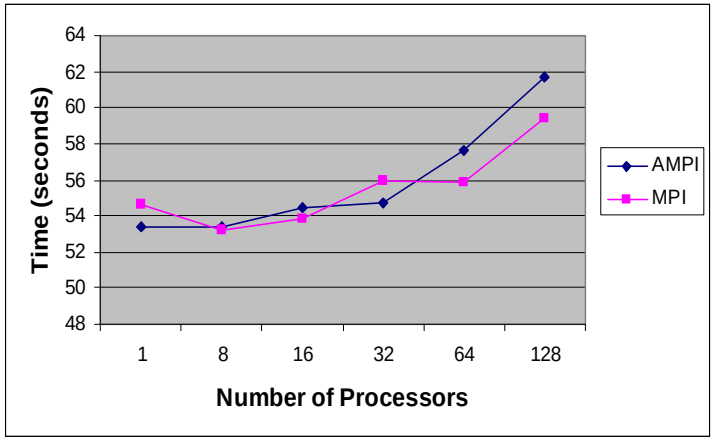
Adaptive MPI

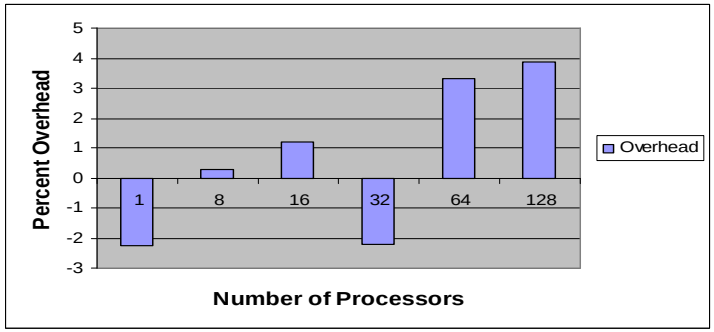
- A bridge between legacy MPI codes and dynamic load balancing capabilities of Charm++
- AMPI = MPI + dynamic load balancing
- Based on Charm++ object arrays and Converse's migratable threads
- Minimal modification needed to convert existing MPI programs (to be automated in future)
- Bindings for C, C++, and Fortran90
- Currently supports most of the MPI 1.1 standard

AMPI Features

- Over 70+ common MPI routines
  - C, C++, and Fortran 90 bindings
  - Tested on IBM SP, SGI Origin 2000, Linux clusters
- Automatic conversion: AMPIzer
  - Based on Polaris front-end
  - Source-to-source translator for converting MPI programs to AMPI
  - Generates supporting code for migration

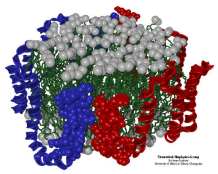
Very low "overhead" compared with native MPI





AMPI Extensions

- Integration of multiple MPI-based modules
  - Example: integrated rocket simulation
    - ROCFLO, ROCSOLID, ROCBURN, ROCFACE
- Each module gets its own MPI_COMM_WORLD
  - All COMM_WORLDs form MPI_COMM_UNIVERSE
- Point-to-point communication among different MPI_COMM_WORLDs using the same AMPI functions
- Communication across modules also considered for balancing load
- Automatic checkpoint-and-restart
  - On different number of processors
  - Number of virtual processors remain the same, but can be mapped to different number of physical processors



Molecular Dynamics

Faucets

Other Programming Pardigms

Load balancing

Obj. Allocation

Remapping

Charm++

Converse

Instrumentation

Mini Languages & libraries

Performance Visualization

Web based interface

Standard Libraries for parallel programming

Component Frameworks

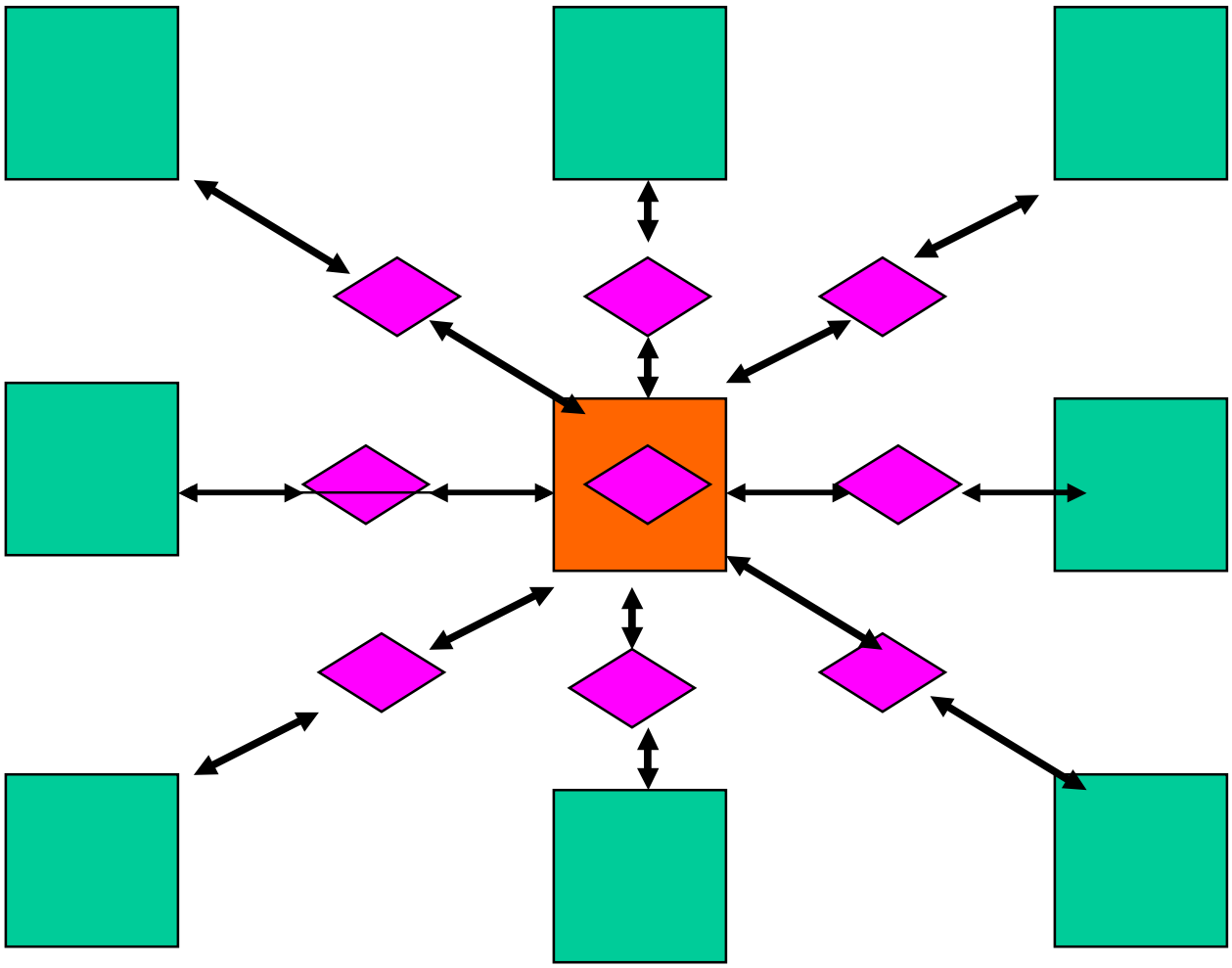
Rocket Simulation & other App's.

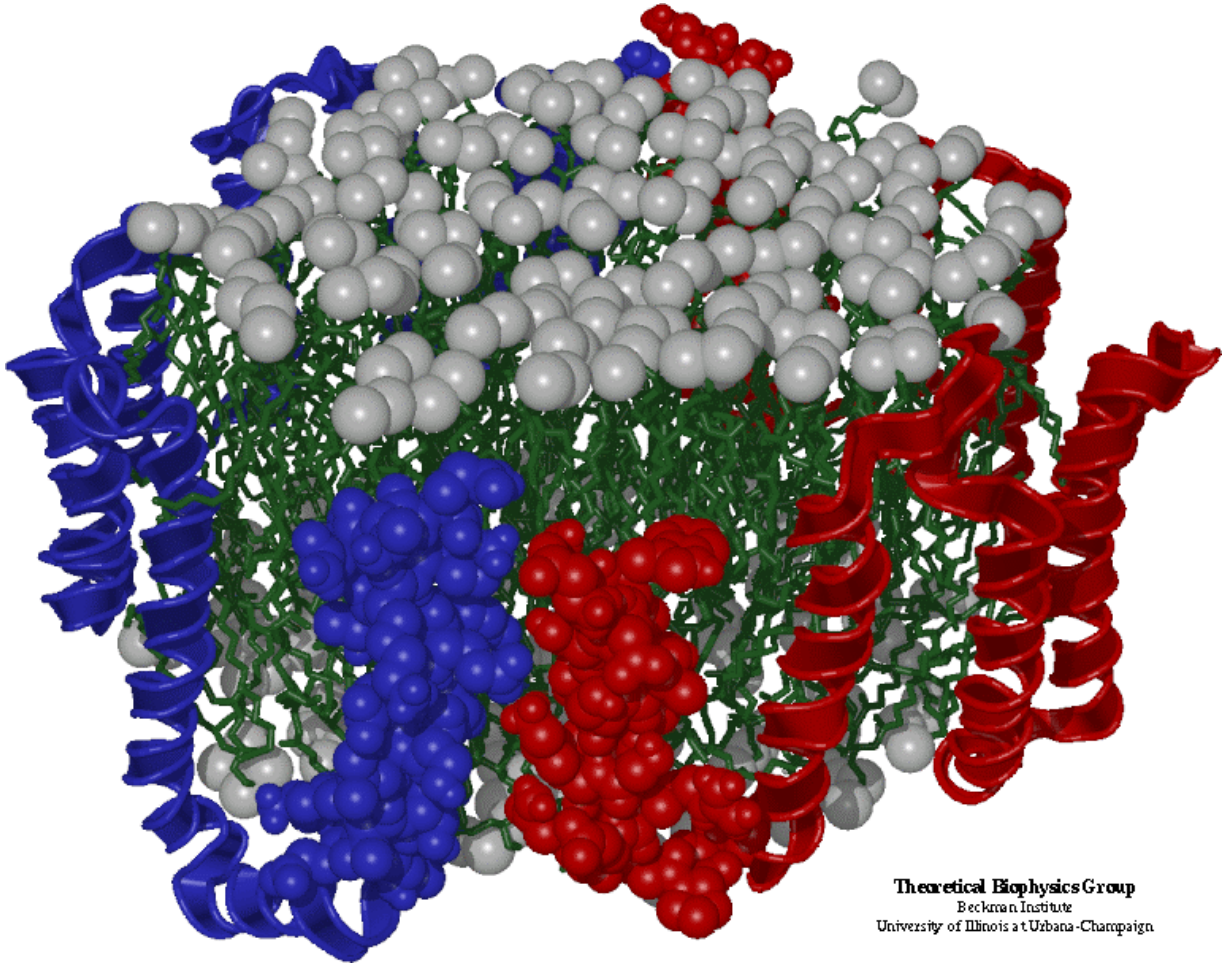
Application Areas and Collaborations

- Molecular Dynamics:
  - Simulation of biomolecules
  - Material properties and electronic structures
- CSE applications:
  - Rocket Simulation
  - Industrial process simulation
  - Cosmology visualizer
- Combinatorial Search:
  - State space search, game tree search, optimization

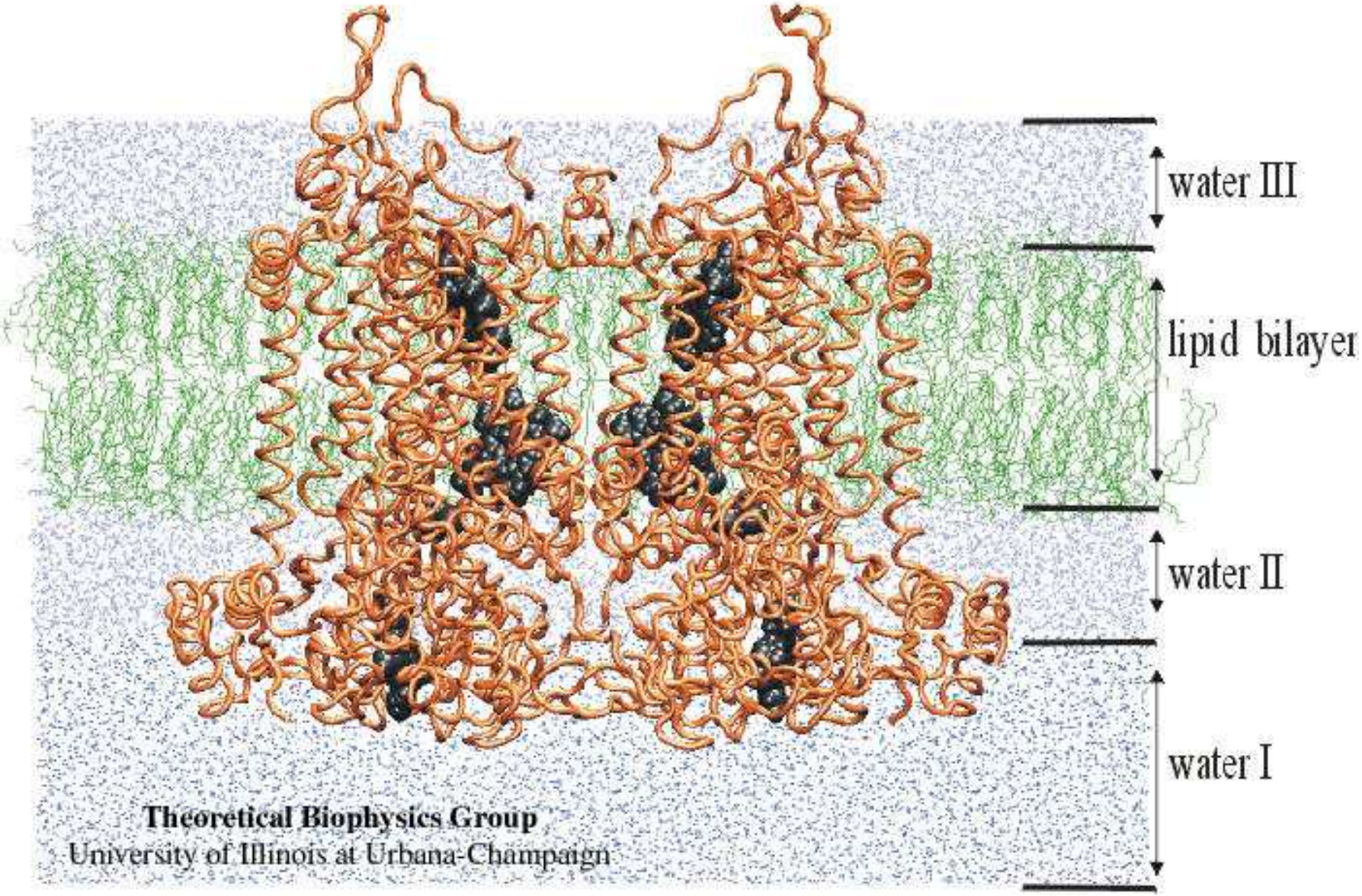
Molecular Dynamics

- Collection of [charged] atoms, with bonds
- Newtonian mechanics
- At each time-step
  - Calculate forces on each atom
    - Bonds:
    - Non-bonded: electrostatic and van der Waal's
  - Calculate velocities and advance positions
- 1 femtosecond time-step, millions needed!
- Thousands of atoms (1,000 - 100,000)

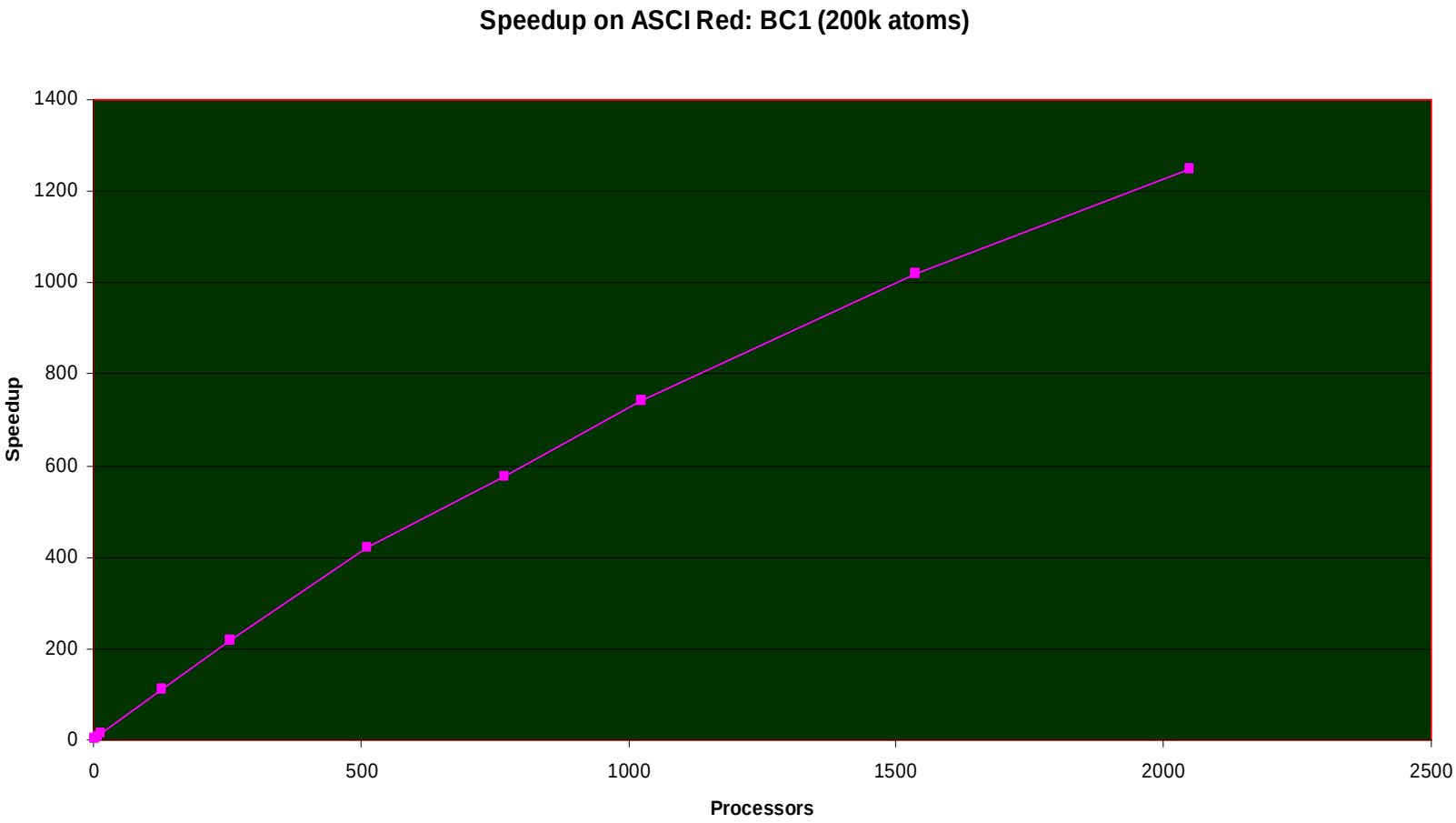



Theoretical Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champaign

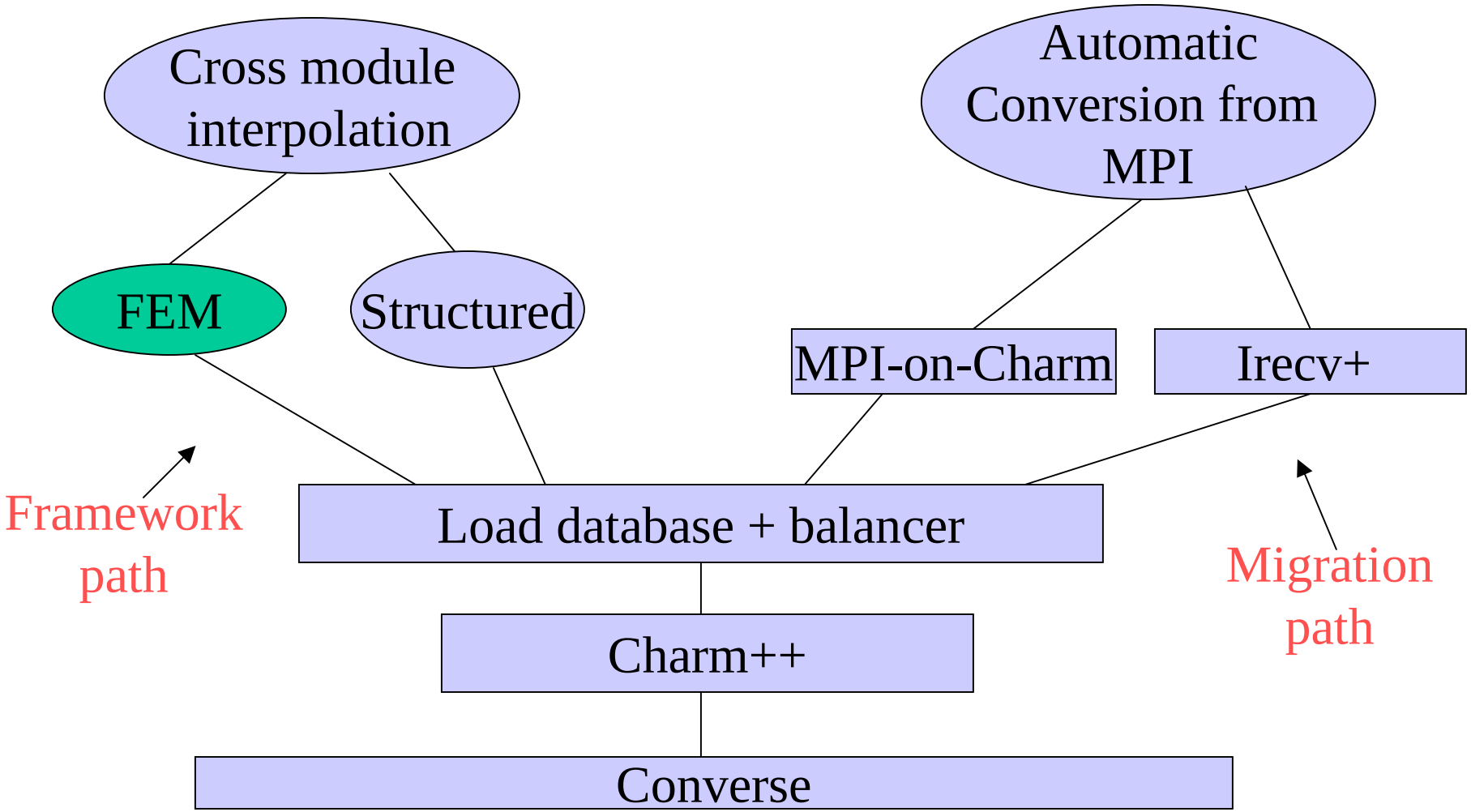
BC1 complex: 200k atoms



Performance Data: SC2000



Component Frameworks: Using the Load Balancing Framework



Finite Element Framework Goals

- Hide parallel implementation in the runtime system
- Allow adaptive parallel computation and dynamic automatic load balancing
- Leave physics and numerics to user
- Present clean, “almost serial” interface:

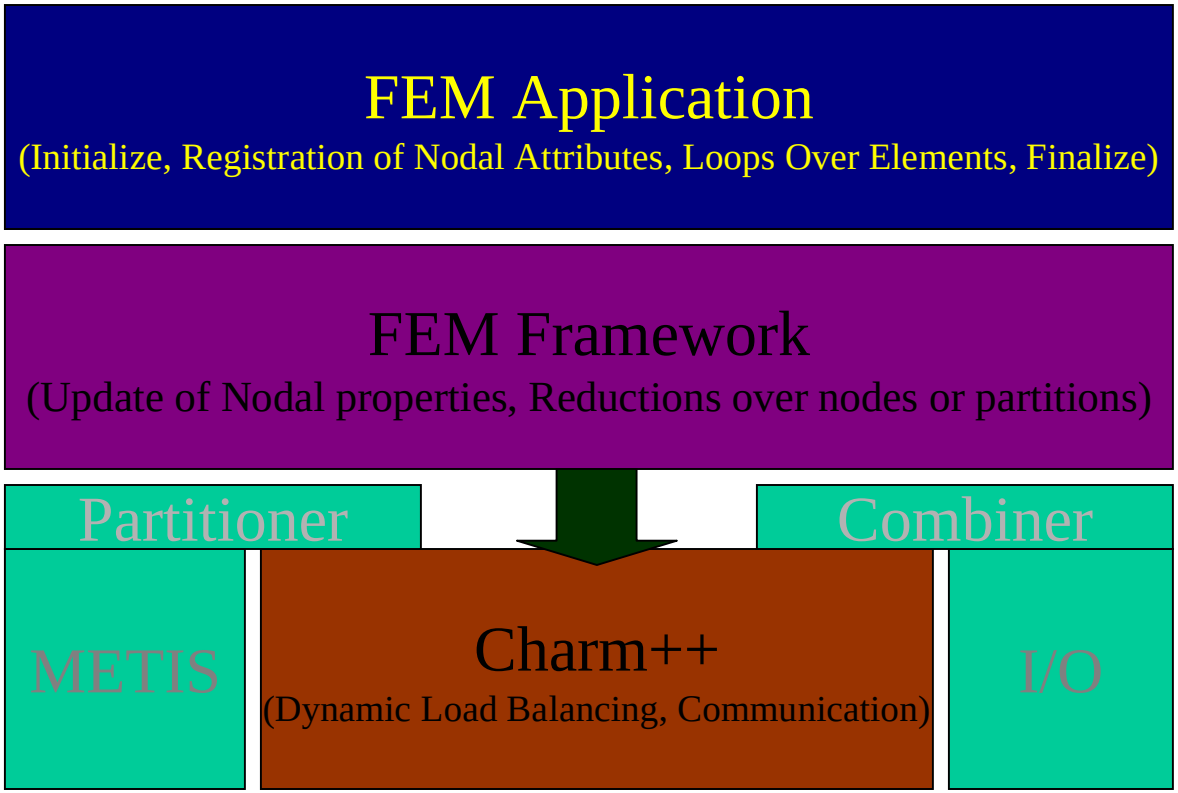
```
begin time loop
  compute forces
  update node positions
end time loop
```

Serial Code for entire mesh

```
begin time loop
  compute forces
  communicate shared nodes
  update node positions
end time loop
```

Framework Code for mesh partition

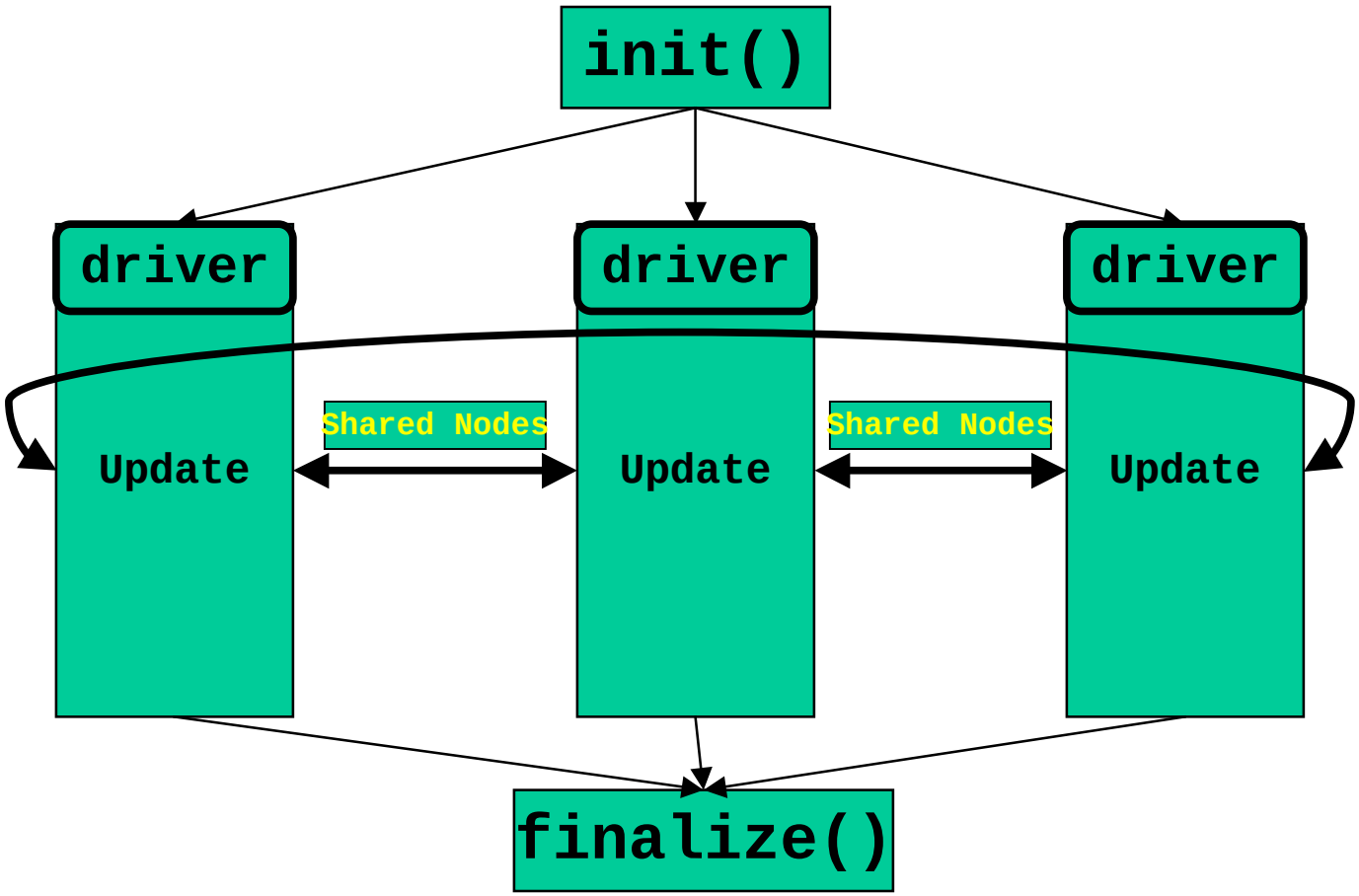
FEM Framework: Responsibilities



Structure of an FEM Program

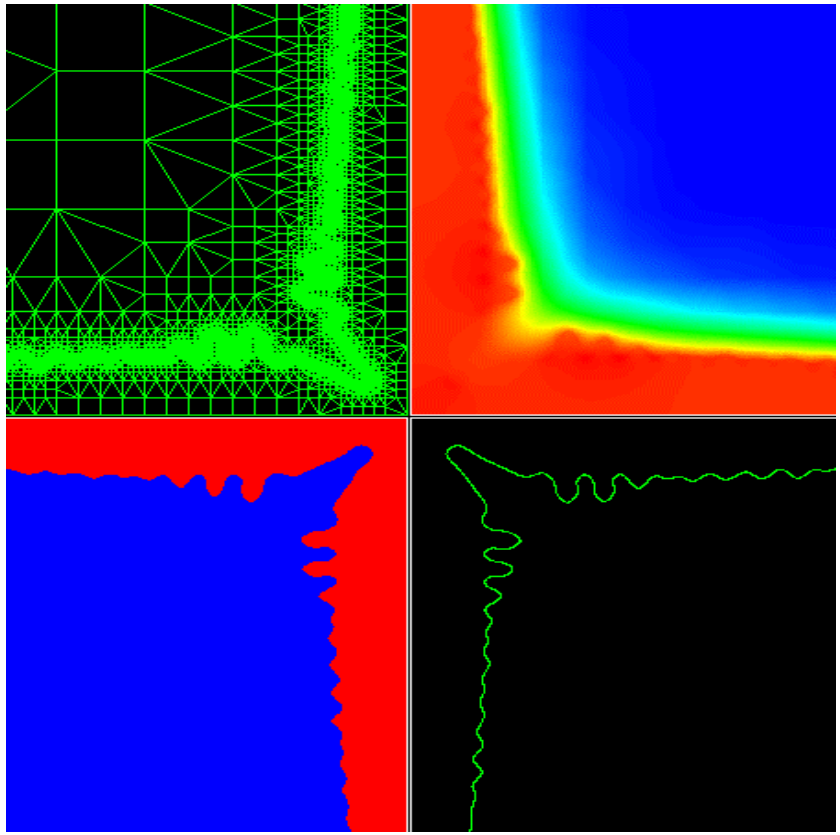
- Serial **`init()`** and **`finalize()`** subroutines
  - Do serial I/O, read serial mesh and call **`FEM_Set_Mesh`**
- Parallel **`driver()`** main routine:
  - One driver per partitioned mesh chunk
  - Runs in a thread: time-loop looks like serial version
  - Does computation and call **`FEM_Update_Field`**
- Framework handles partitioning, parallelization, and communication

Structure of an FEM Application

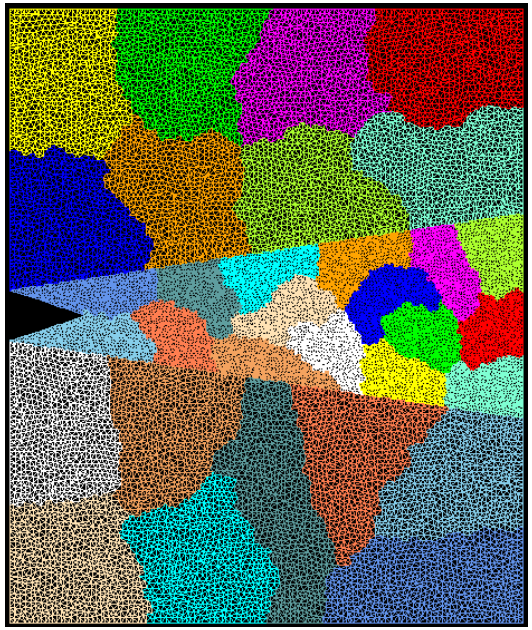


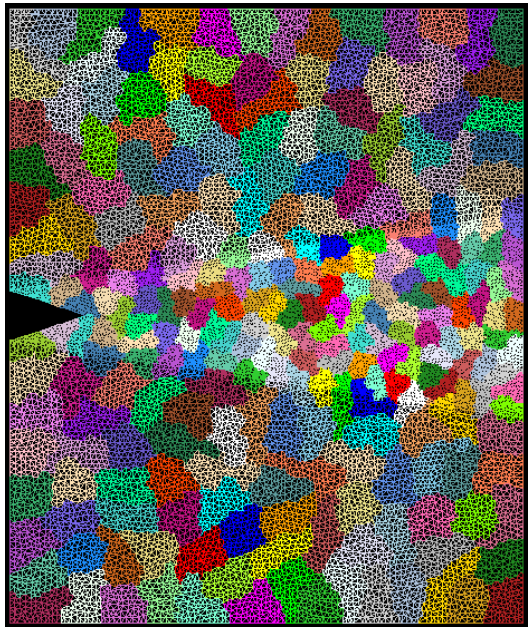
Dendritic Growth

- Studies evolution of solidification microstructures using a phase-field model computed on an adaptive finite element grid
- Adaptive refinement and coarsening of grid involves re-partitioning



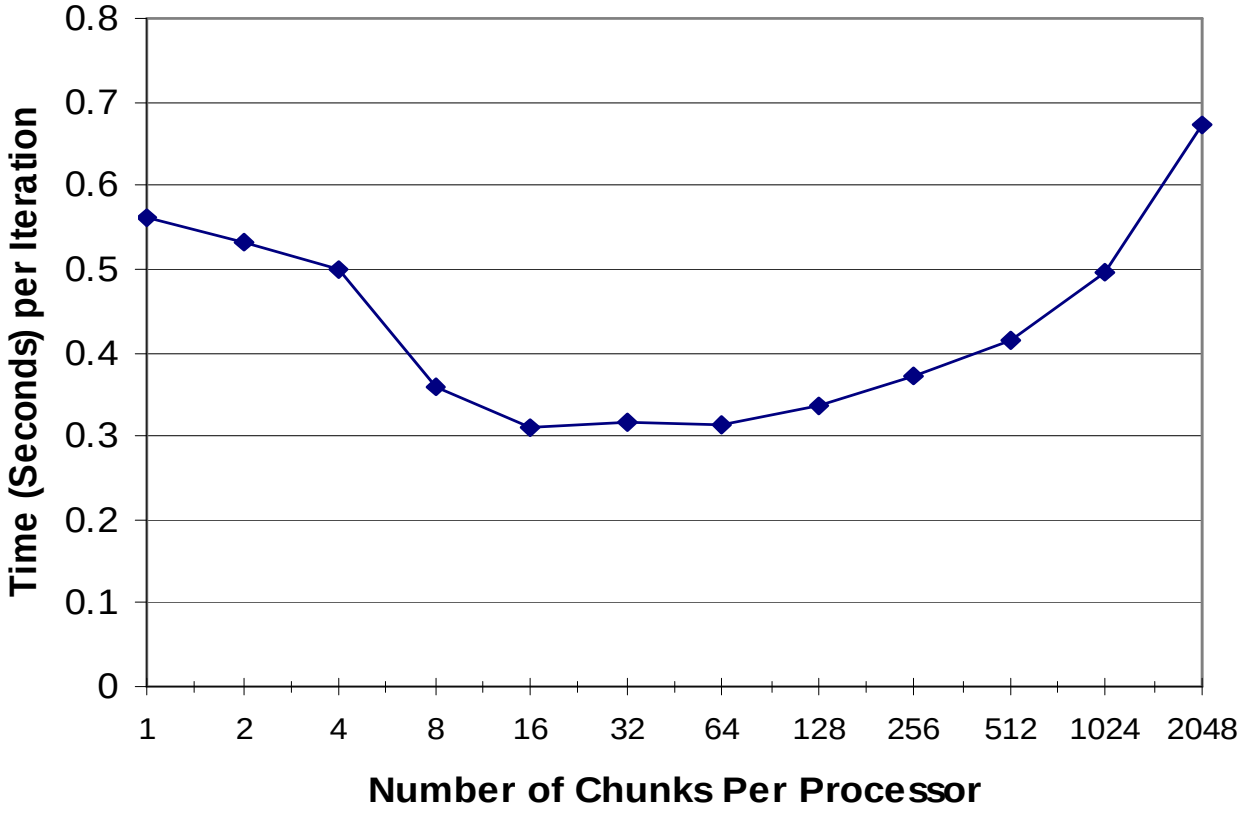
Crack Propagation



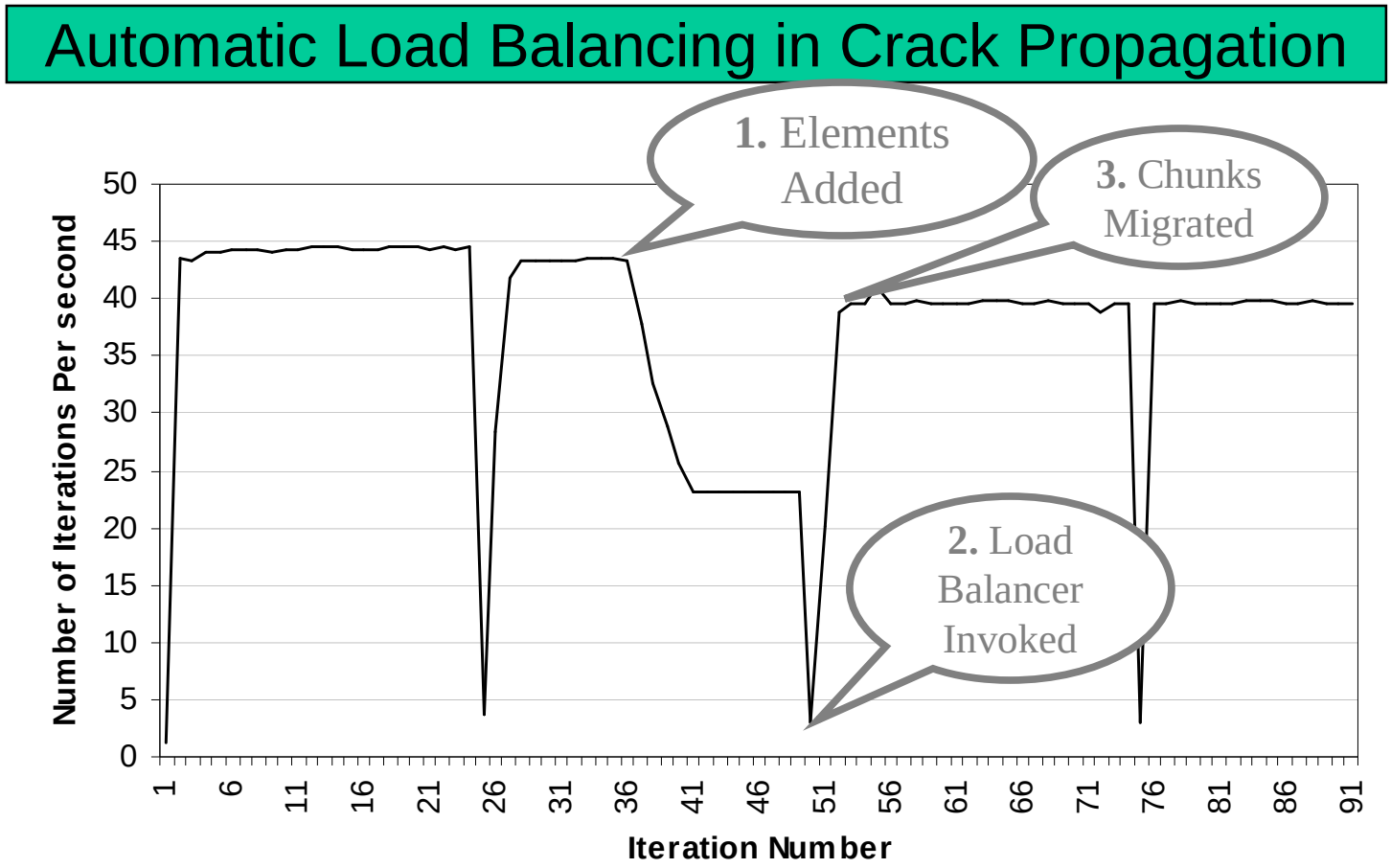


Decomposition into 16 chunks (left) and 128 chunks, 8 for each PE (right). The middle area contains cohesive elements. Both decompositions obtained using Metis. Pictures: S. Breitenfeld, and P. Geubelle

“Overhead” of Multipartitioning

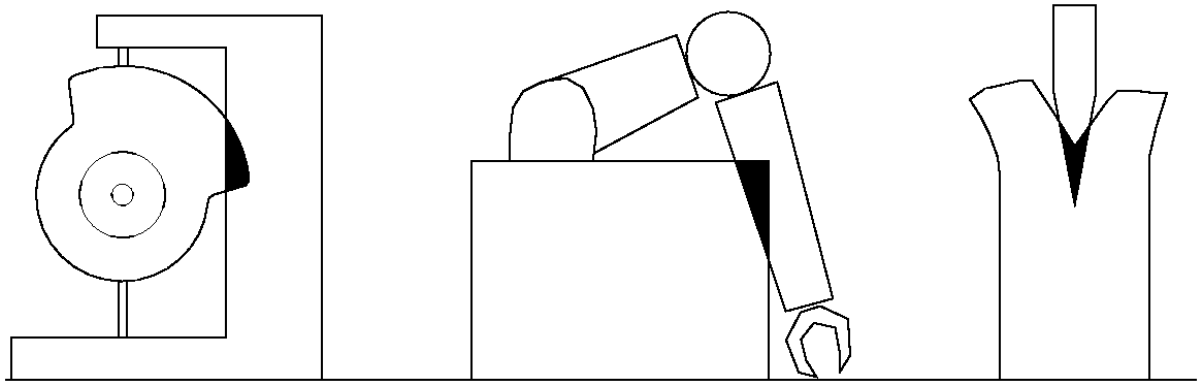


Load balancer in action



Parallel Collision Detection

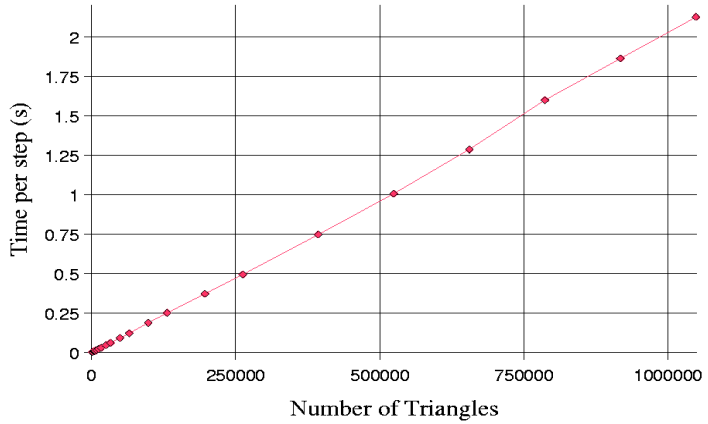
- Detect collisions (intersections) between objects scattered across



- **Approach, based on Charm++ Arrays**
  - **Overlay regular, sparse 3D grid of voxels (boxes)**
  - **Send objects to all voxels they touch**
  - **Collide voxels independently and collect results**
- **Leave collision response to user code**

Collision Detection Speed

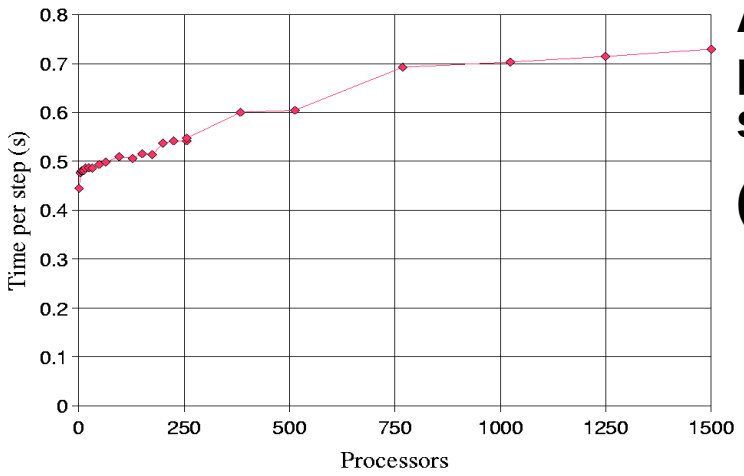
- O(n) serial performance



Single Linux PC

2us per polygon serial performance

- **Good speedups to 1000s of processors**

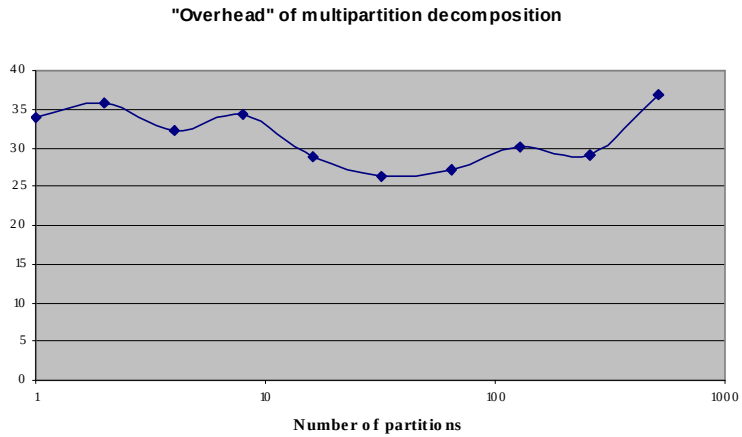


ASCI Red, 65,000 polygons per processor scaling problem

(to 100 million polygons)

Rocket Simulation

- Our Approach:
  - Multi-partition decomposition
  - Data-driven objects (Charm++)
  - Automatic load balancing framework
- AMPI: Migration path for existing MPI+Fortran90 codes
  - ROCFLO, ROCSOLID, and ROCFACE



Center for Simulation of Advanced Rockets

University of Illinois at Urbana-Champaign

Timeshared parallel machines

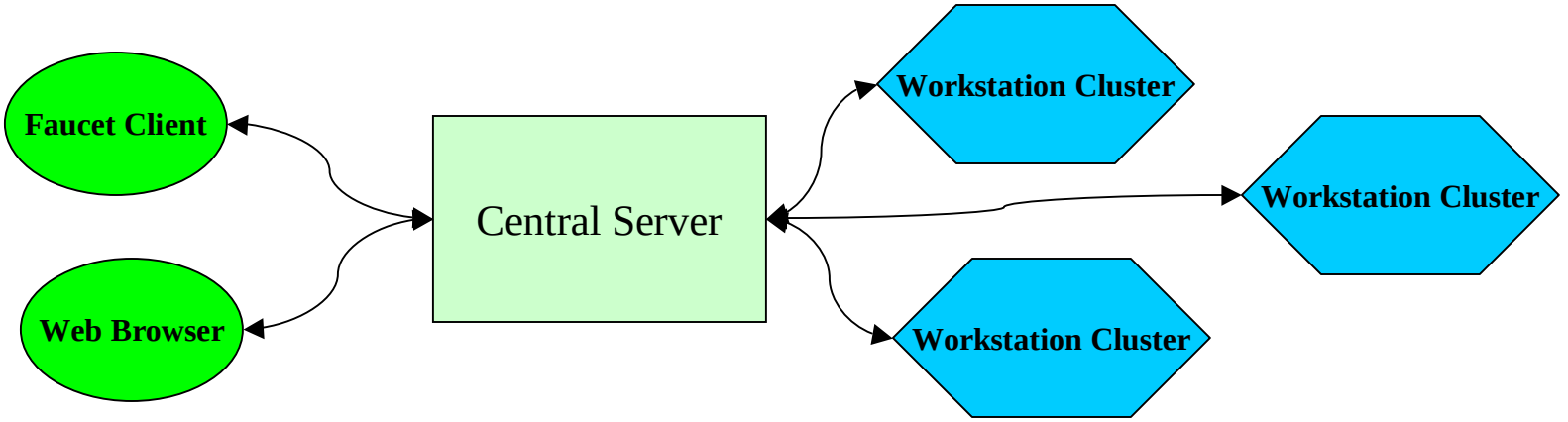
- How to use parallel machines effectively?
- Need resource management
  - Shrink and expand individual jobs to available sets of processors
  - Example: Machine with 100 processors
    - Job1 arrives, can use 20-150 processors
    - Assign 100 processors to it
    - Job2 arrives, can use 30-70 processors,
      - and will pay more if we meet its deadline
- We can do this with migratable objects!

Faucets: Multiple Parallel Machines

- Faucet submits a request, with a QoS contract:
  - CPU seconds, min-max cpus, deadline, interacive?
- Parallel machines submit bids:
  - A job for 100 cpu hours may get a lower price bid if:
    - It has less tight deadline,
    - more flexible PE range
  - A job that requires 15 cpu minutes and a deadline of 1 minute
    - Will generate a variety of bids
    - A machine with idle time on its hand: low bid

Faucets QoS and Architecture

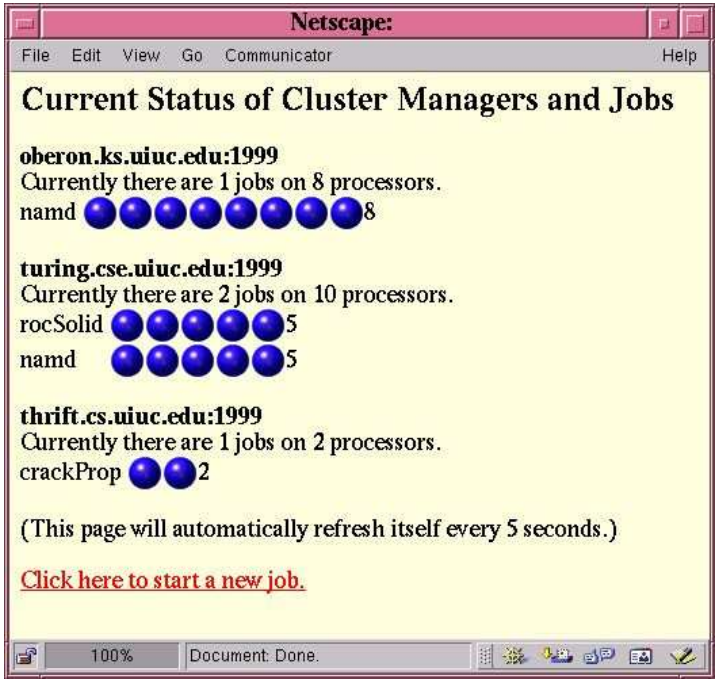
- User specifies desired job parameters such as:
  - min PE, max PE, estimated CPU-seconds, priority, etc.
- User does not specify machine. .
- Planned: Integration with Globus

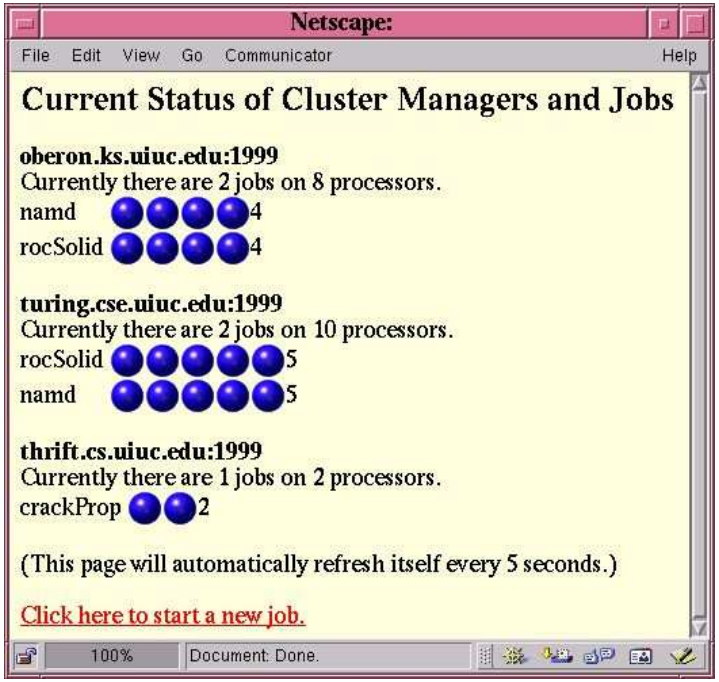


How to make all of this work?

- The key: fine-grained resource management model
  - Work units are objects and threads
    - rather than processes
  - Data units are object data, thread stacks, ..
    - Rather than pages
  - Work/Data units can be migrated automatically
    - *during* a run

Time-Shared Parallel Machines



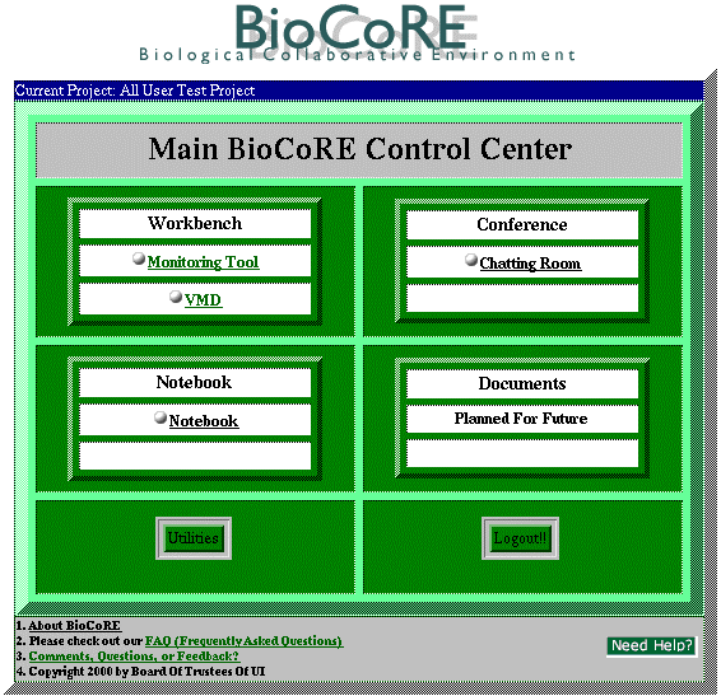


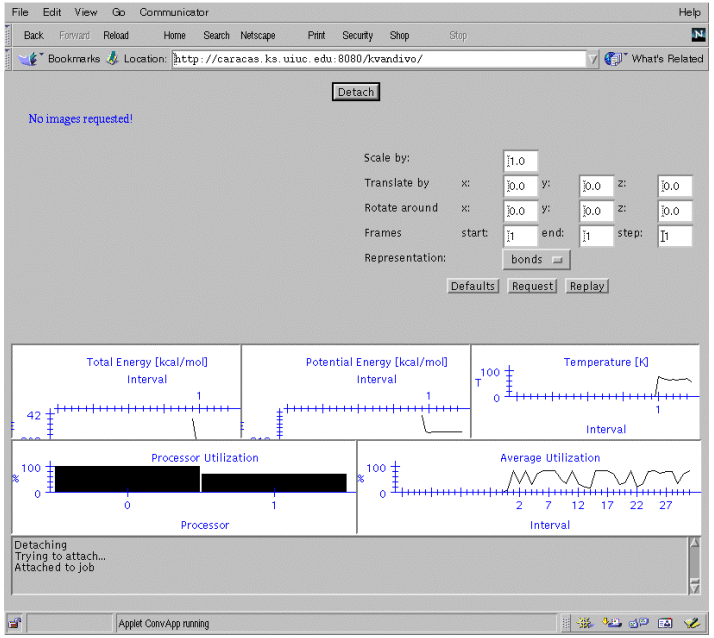
Appspector: Web-based Monitoring and Steering of Parallel Programs

- Parallel Jobs submitted via a server
  - Server maintains database of running programs
  - Charm++ client-server interface
    - Allows one to inject messages into a running application
- From any web browser:
  - You can attach to a job (if authenticated)
  - Monitor performance
  - Monitor behavior
  - Interact and steer job (send commands)

BioCoRE

Goal: Provide a web-based way to virtually bring scientists together.





- Project Based
- Workbench for Modeling
- Conferences/Chat Rooms
- Lab Notebook
- Joint Document Preparation

Some New Projects

- Load Balancing for really large machines:
  - 30k-128k processors
- Million-processor Petaflops class machines
  - Emulation for software development
  - Simulation for Performance Prediction
- Operations Research
  - Combinatorial optiization
- Parallel Discrete Event Simulation

Summary

- Exciting times for parallel computing ahead
- We are preparing an object based infrastructure
  - To exploit future apps on future machines
- Charm++, AMPI, automatic load balancing
- Application-oriented research that produces enabling CS technology
- Rich set of collaborations
- More information: http://charm.cs.uiuc.edu