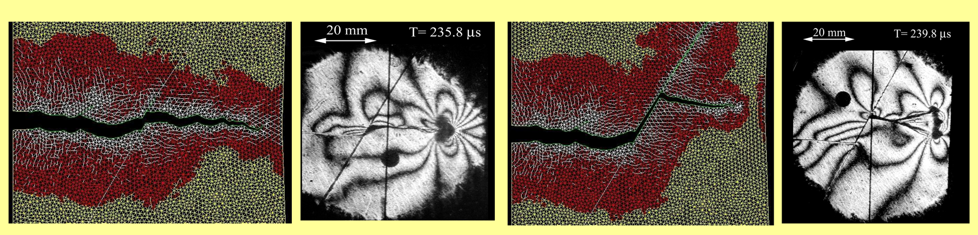
SPEEDING UP PARALLEL SIMULATION WITH AUTOMATIC LOAD BALANCING

Hari Govind, Gengbin Zheng, Laxmikant Kale, Michael Breitenfeld, Philippe Geubelle
University of Illinois at Urbana-Champaign

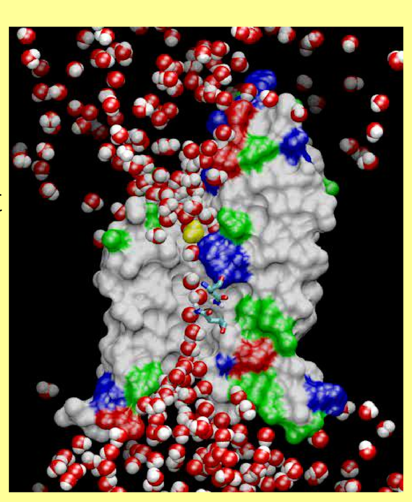
Motivations

- Parallel machines abound
  - Capabilities enhanced as machines get more powerful
    - PSC Lemieux, ASCI White, Earth Simulator, BG/L
  - Clusters becoming ubiquitous
  - Desktops and Games consoles go parallel:
    - Cell processor, multi-core chips,
- Applications get more ambitious and complex
  - Adaptive algorithms
  - Irregular or dynamic behavior
  - Multi-component and multi-physics
  - MPI based code limitations
    - No adaptive load balancing



- Versatile, automatic load balancers are desired
  - Application independent
  - No/little user effort is needed to balance load
  - Addresses the load balancing needs of many different types of applications

- Applications
  - CSE applications
    - Crack propagation
    - Adaptive mesh refinement
  - Molecular dynamics
    - NAMD
    - CPAIMD
  - Cosmology simulation
  - Fault tolerance



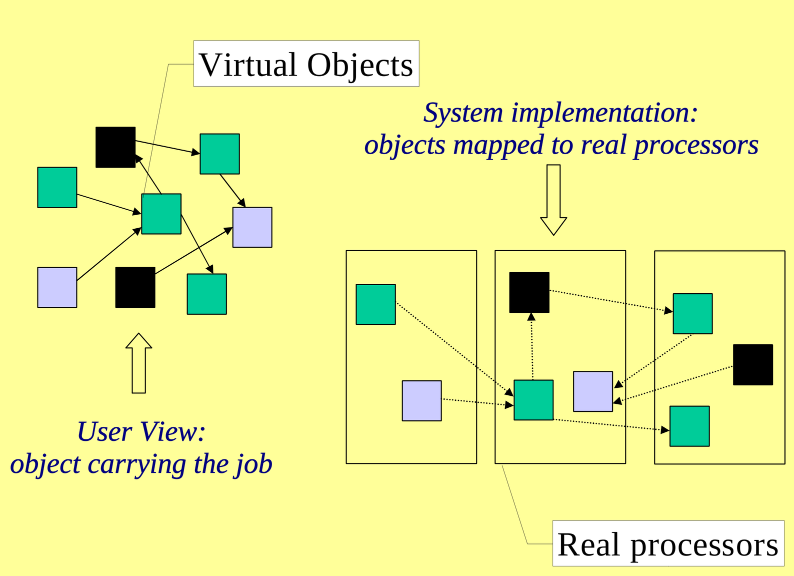
Processor Virtualization

Programmer: **[Over] decomposition into virtual processors (VP)**

Runtime: **Assigns VPs to processors**

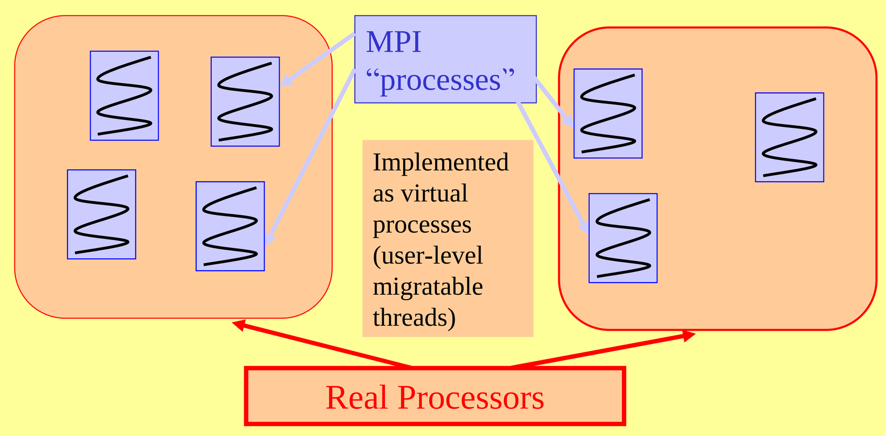
Enables *adaptive runtime strategies*

Charm++ Architecture



AMPI: Adaptive MPI

- Each virtual process implemented as a *user-level thread* embedded in a Charm++ object

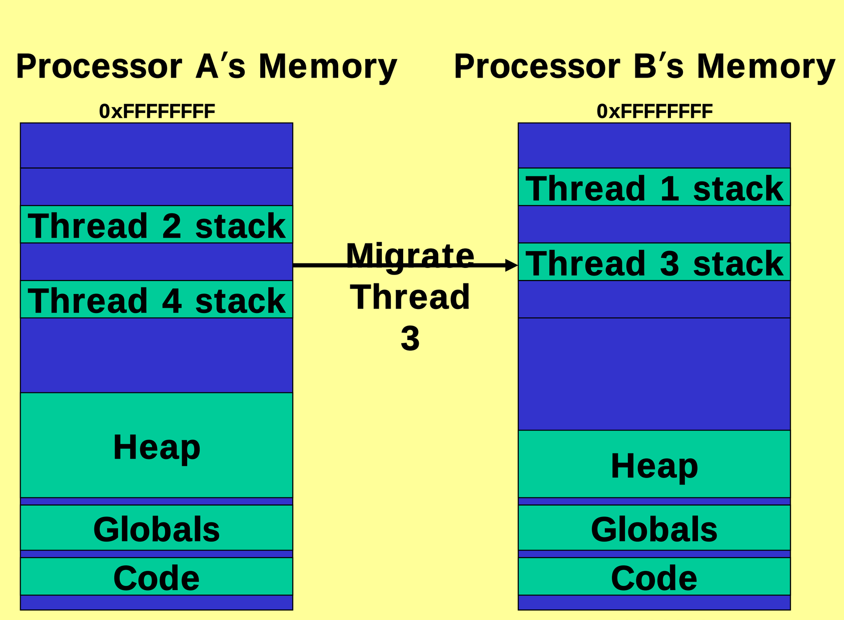


Benefits

- Software engineering
  - Number of virtual processors can be independently controlled
  - Separate VPs for different modules
- Message driven execution
  - Computation performed upon receipt of a message
  - Adaptive overlap of communication
  - Predictability :
    - Automatic out-of-core execution
  - Asynchronous reductions
- Dynamic mapping
  - Heterogeneous clusters
    - Vacate, adjust to speed, share
  - Automatic checkpointing/restarting
  - Automatic dynamic load balancing
  - Change set of processors used
  - Communication optimization

How to Migrate Objects

- Objects
  - Packing/unpacking functions
- User-level Threads
  - Global variables:
    - ELF object format: switch GoT pointer
    - Alternative: compiler/pre-processor support
  - Migration of stack
    - Isomalloc (from PM2 in France):
      - Reserve virtual space on all processors for each thread
      - Mmap it when you migrate there
  - Migration of Heap data:
    - Isomalloc heaps
    - User-supplied or compiler generated pack function



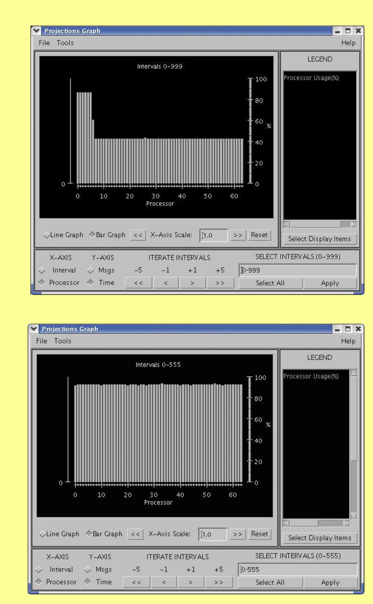
Thread migration with isomalloc

Charm++ Load Balancing Framework

- Load balancing task in Charm++
  - Given a collection of migratable objects and a set of computers connected in a certain topology
  - Find a mapping of objects to processors
    - Almost same amount of computation on each processor
    - Communication between processors is minimum
- Dynamic mapping of objects to processors
- Two major approaches
  - No predictability of load patterns
    - Fully dynamic
    - Early work on State Space Search, Branch&Bound, ...
    - Seed load balancers
  - With certain predictability
    - CSE, molecular dynamics simulation
    - Measurement-based load balancing strategy

Seed Load Balancing

- Tasks are initially represented by object creation messages, or ``seeds".
- Seed load balancing involves the movement of seeds, to balance work across processors
- Low responsiveness
  - Load balancing request blocked by long entries
- Neighborhood averaging with work-stealing when Idle using immediate messages
  - Interruption-based message
  - Fast response to the request
  - Work-stealing at idle time



80000 objects, 10% heavy objects

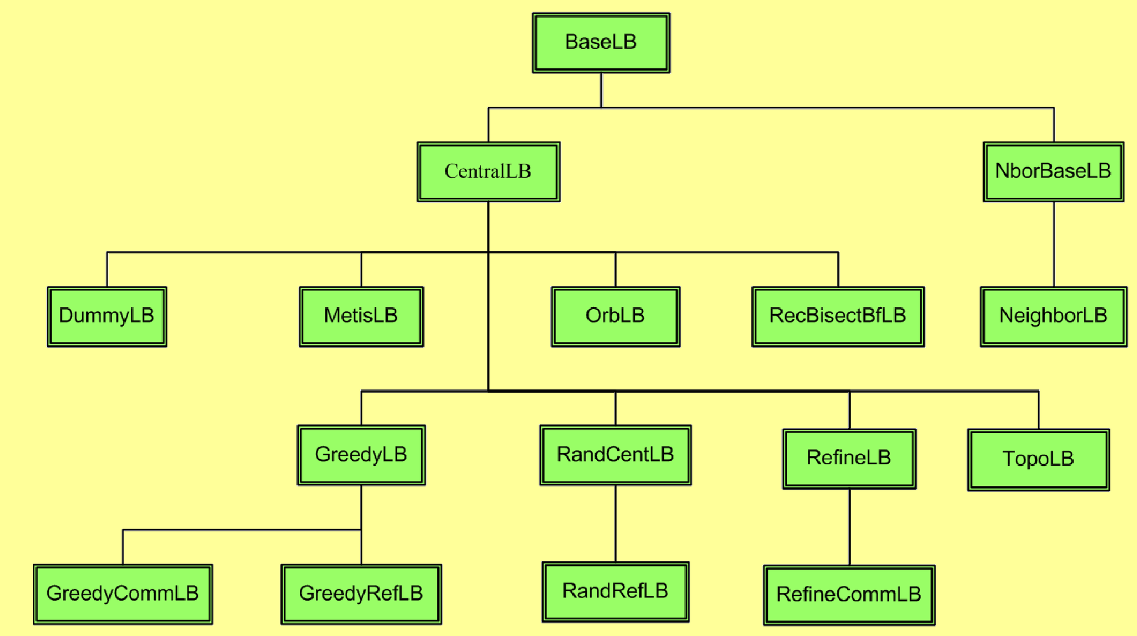
Measurement Based Load Balancing

- Based on Principle of persistence
- Runtime instrumentation
  - Measures communication volume and computation time
- Measurement based load balancers
  - Use the instrumented data-base periodically to make new decisions
  - Many alternative strategies can use the database
    - Centralized vs. distributed
    - Greedy improvements vs. complete reassignments
    - Taking communication into account
    - Taking dependences into account (More complex)
    - Topology-aware

Principle of Persistence

- Once an application is expressed in terms of interacting objects, object communication patterns and computational loads tend to persist over time
  - In spite of dynamic behavior
    - Abrupt and large,but infrequent changes (eg:AMR)
    - Slow and small changes (eg: particle migration)
- Parallel analog of principle of locality
  - Heuristics, that holds for most CSE applications

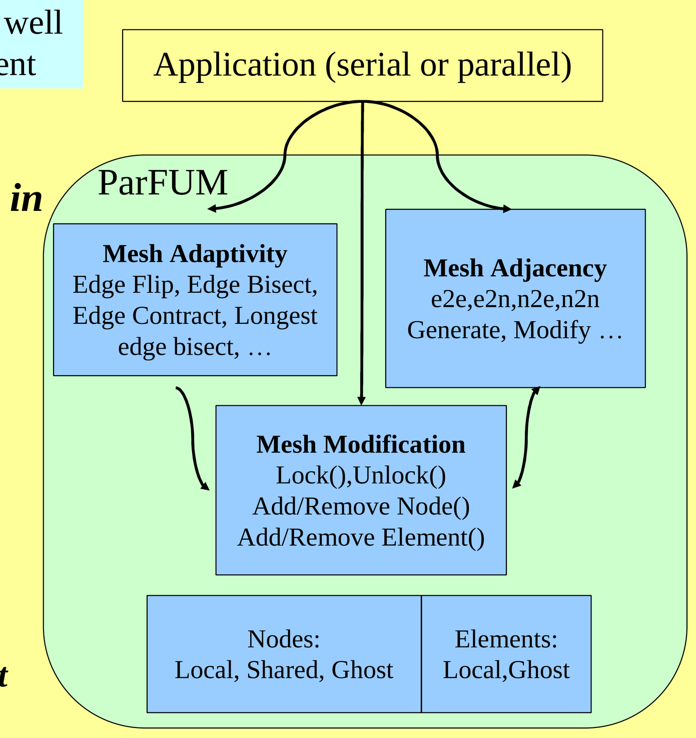
Load Balancing Strategies



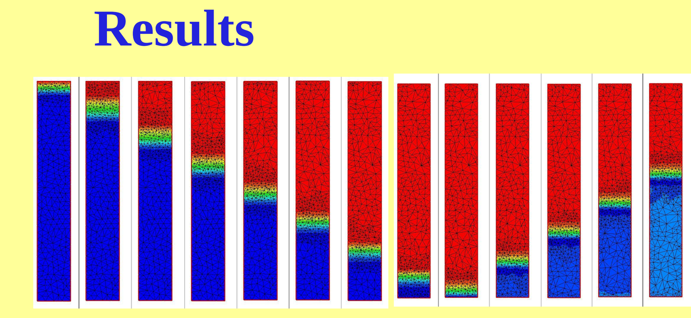
Parallel Framework for Unstructured Meshes (ParFUM)

Integrated: Geometry as well as ghost layer management

- ***Independent mesh adaptivity operations in parallel***
- ***Locking individual nodes***
- ***No global synchronization***
- ***Adjacency data structures:***
  - ***Node-to-node***
  - ***Element-to-element***
  - ***Node-to-element***

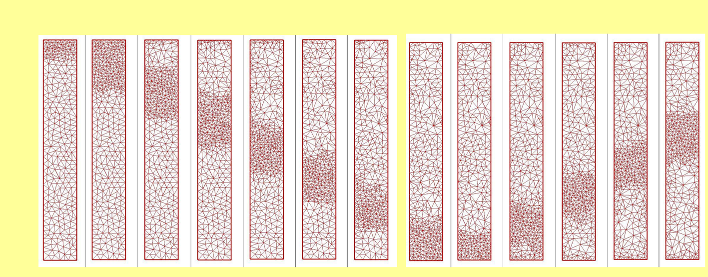


Sequential Refinement and Coarsening Results

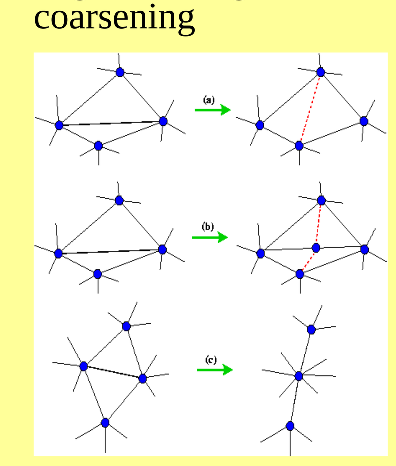


Shock propagation and reflection down the length of the bar

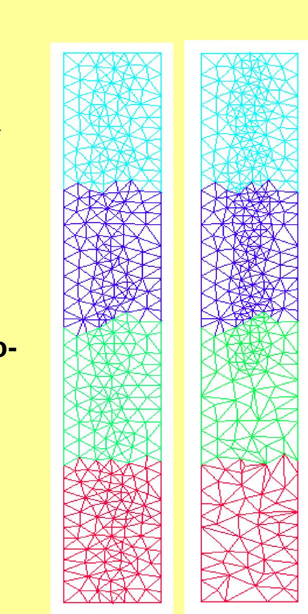
Adaptive mesh modification to capture the shock propagation



- Adaptivity code integrated with ParFUM
- High-level algorithms for refinement and coarsening



- **Built on low-level parallel mesh modification primitives**
- **Maintain up-to-date parallel mesh state including adjacencies, ghost layers and user data**



PARALLEL PROGRAMMING LABORATORY

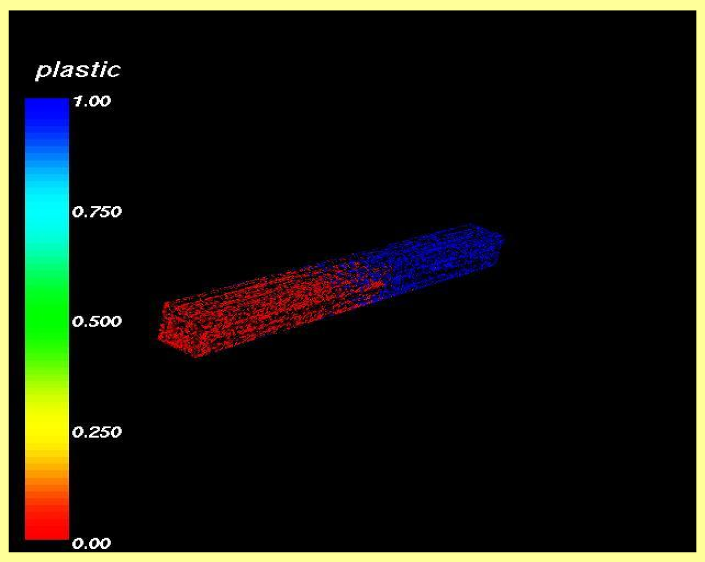
http://charm.cs.uiuc.edu

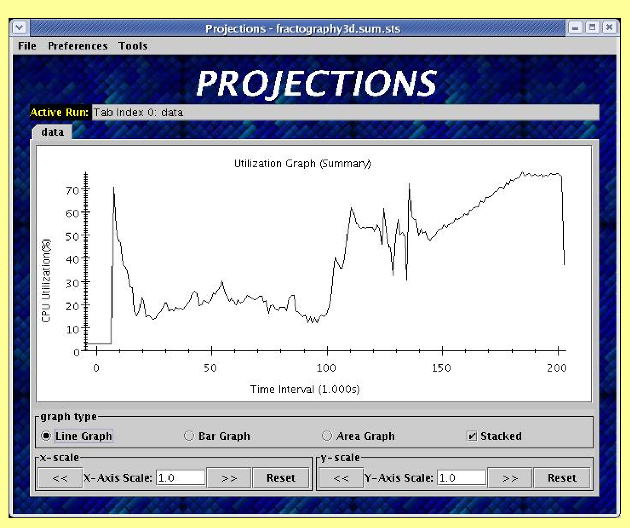
SPEEDING UP PARALLEL SIMULATION WITH AUTOMATIC LOAD BALANCING

Hari Govind, Gengbin Zheng, Laxmikant Kale, Michael Breitenfeld, Philippe Geubelle
University of Illinois at Urbana-Champaign

Crack Propagation Simulation

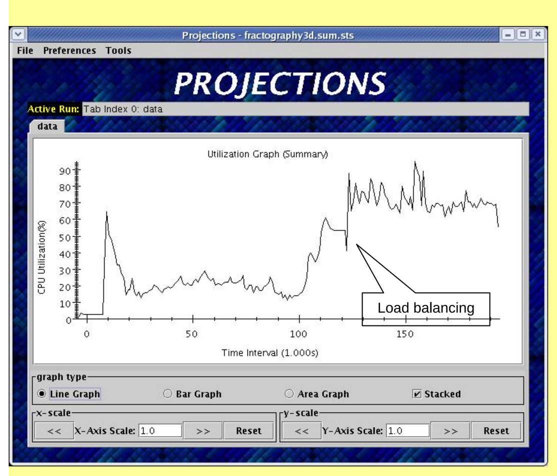
- 1-D elastic-plastic wave propagation
  - Bar is dynamically loaded resulting in an elastic wave propagating down bar, upon reflection from the fixed end the material becomes plastic
- Written in AMPI





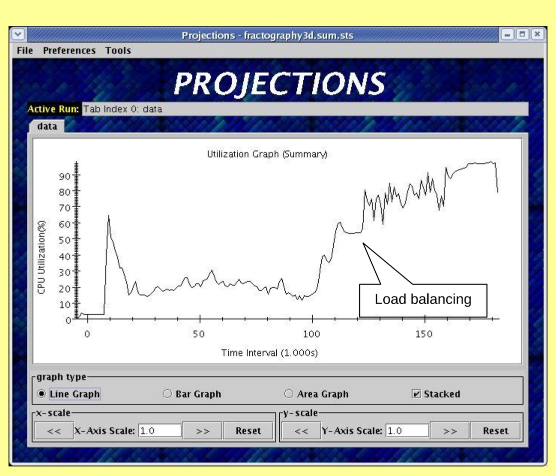
- •Dynamic 3D crack propagation simulation
- •400,000 linear strain tetrahedral elements
- •SGi Altix (NCSA)
- •32 processors
- •160 AMPI threads
- •Simulating an elastic bar
- •Total run time: 207 seconds

With "stop and go" load balancing scheme



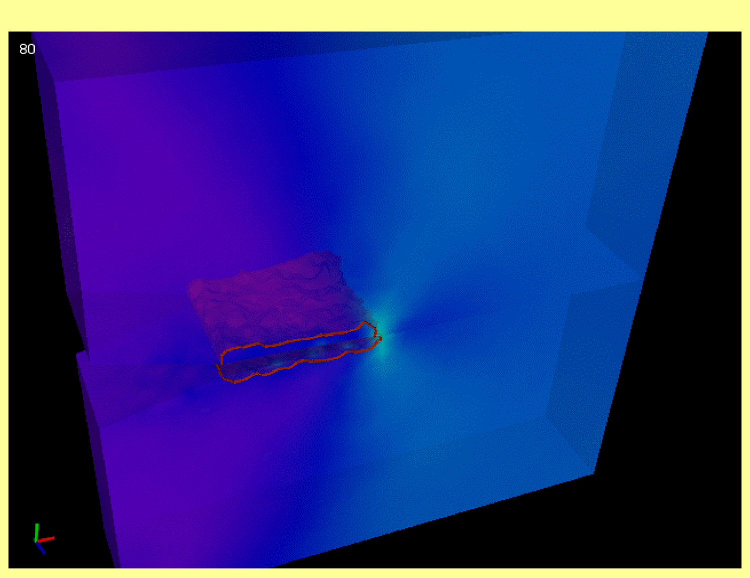
Total execution time: 198 seconds

With agile load balancing scheme



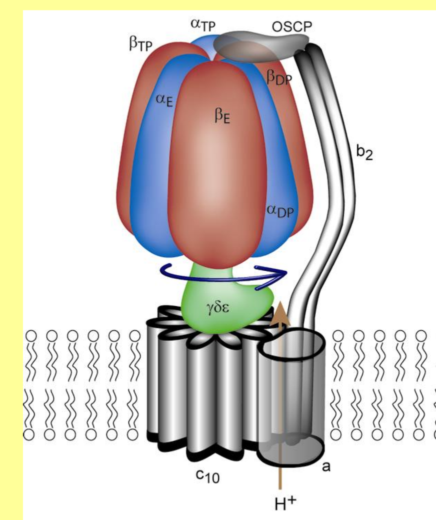
187 seconds

- 3-D dynamic elastic-plastic fracture
  - 3D Plastic Fracture
  - A single edge notched specimen pulled at both ends with a ramping magnitude of 1 m/s over .01 seconds
  - Isosurface is the extent of the plastic zone
  - Load imbalance occurs at the onset of an element turning from elastic to plastic, zone of plasticity forms over a limited number of processors as the crack propagates

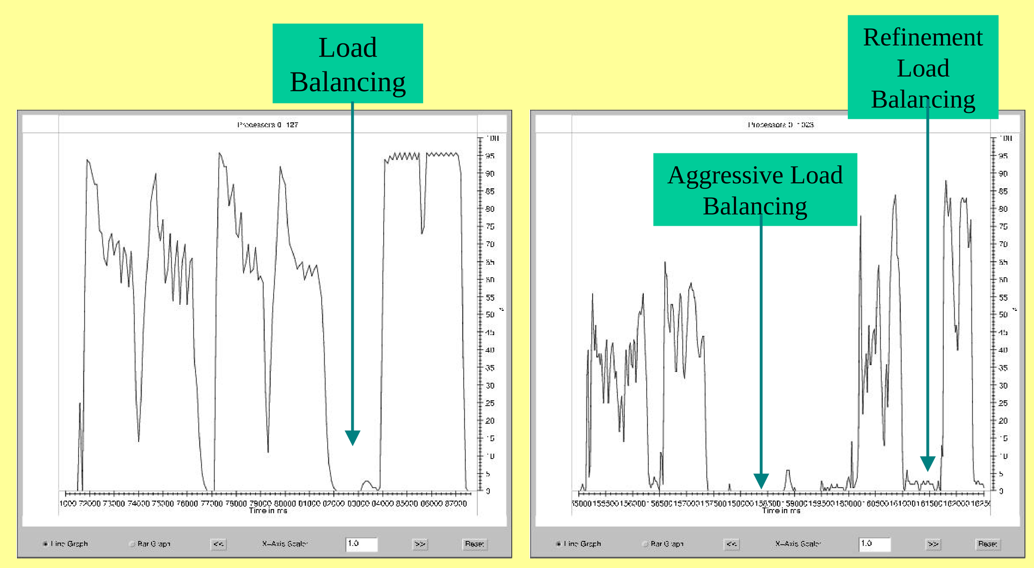


Molecular Dynamics Simulation

- Molecular dynamics and related algorithms
  - e.g., minimization, steering, locally enhanced sampling, alchemical and conformational free energy perturbation
- Efficient algorithms for full electrostatics
- Effective on affordable commodity hardware
- Building a complete modeling environment
- Written in Charm++



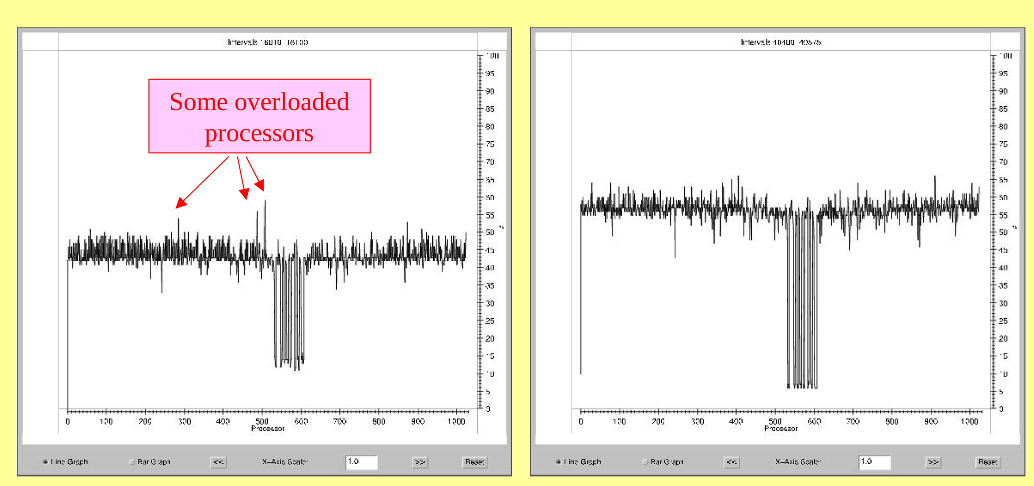
ATP-Synthase



Processor Utilization against Time on (a) 128 (b) 1024 processors

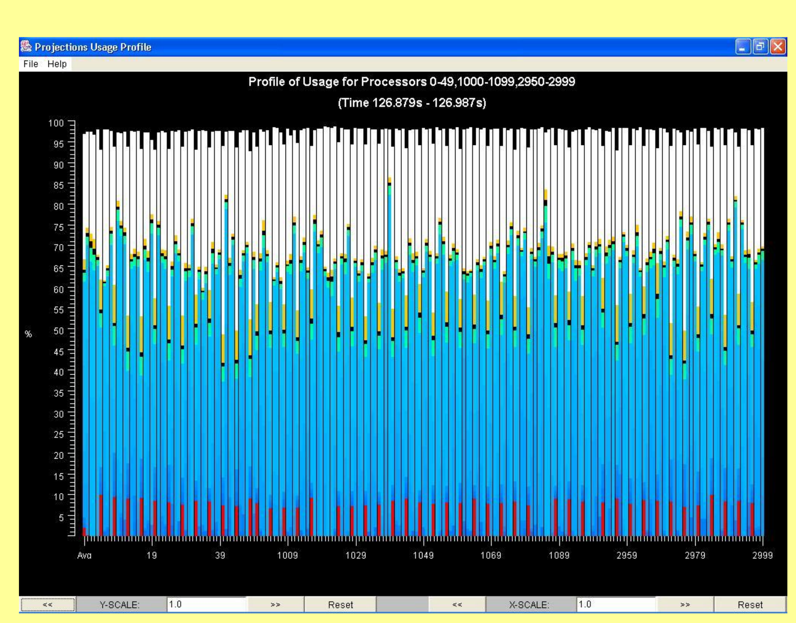
On 128 processor, a single load balancing step suffices, but

On 1024 processors, we need a "refinement" step.



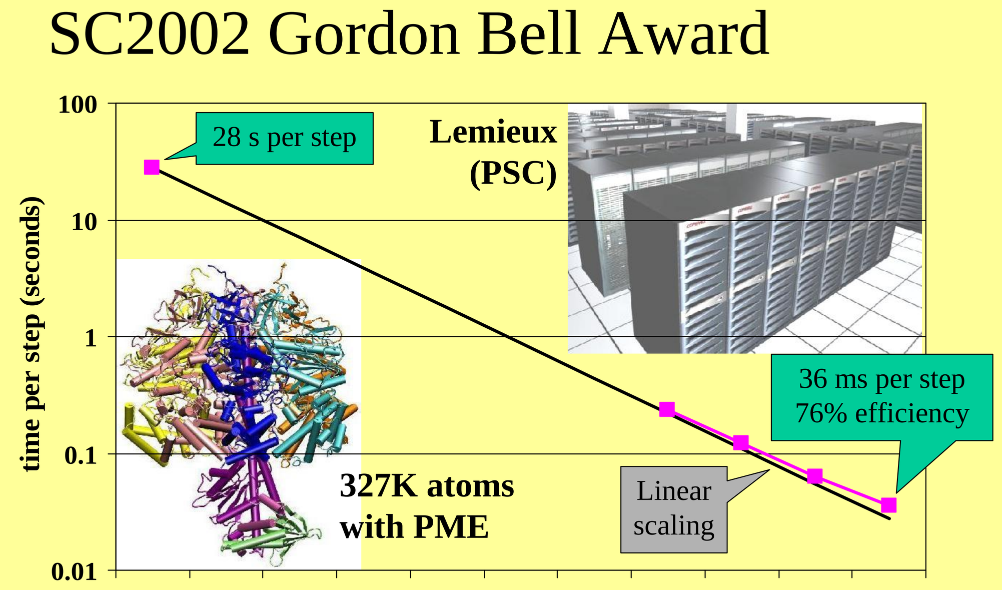
Processor Utilization across processors after (a) greedy load balancing and (b) refining

Note that the underloaded processors are left underloaded (as they don't impact performamnce); *refinement* deals only with the overloaded ones



Profile view of a 3000 processor run of NAMD (White shows idle time)

SC2002 Gordon Bell Award

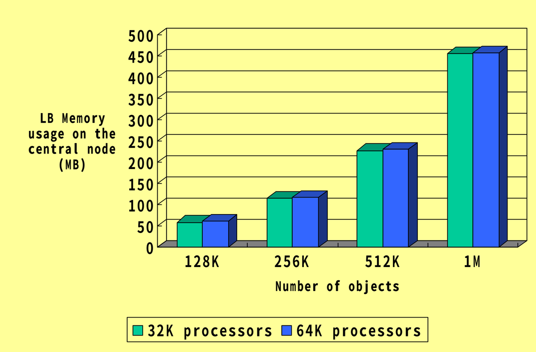


PARALLEL PROGRAMMING LABORATORY

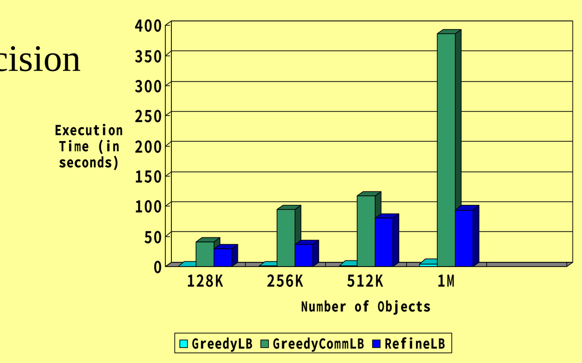
http://charm.cs.uiuc.edu

Load Balancing on Very Large Machines

- Scalability limits
  - Consider an application with 1M objects on 64K processors
- Metrics for a multi-dimensional optimization
  - Memory usage on any one processor
  - Decision-making time
  - Quality of load balancing decision

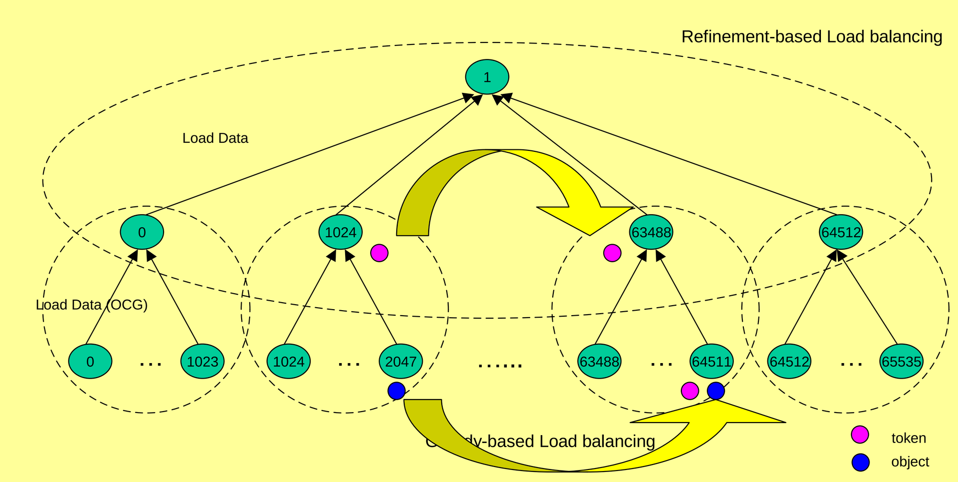


Simulation Results : using BigSim



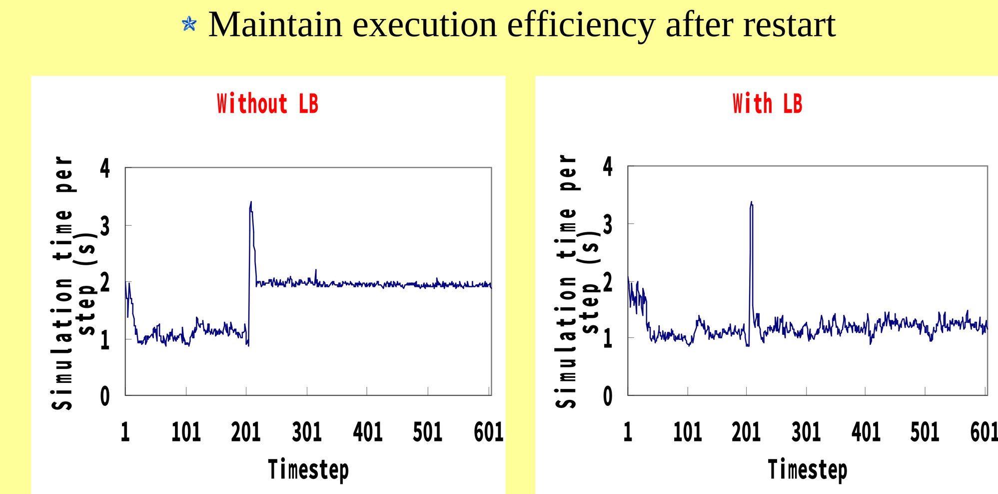
benchmark creates a specified number of communicating objects in 2D-mesh.
Run on Lemieux 64 processors, using BigSim

Hierarchical Load Balancing

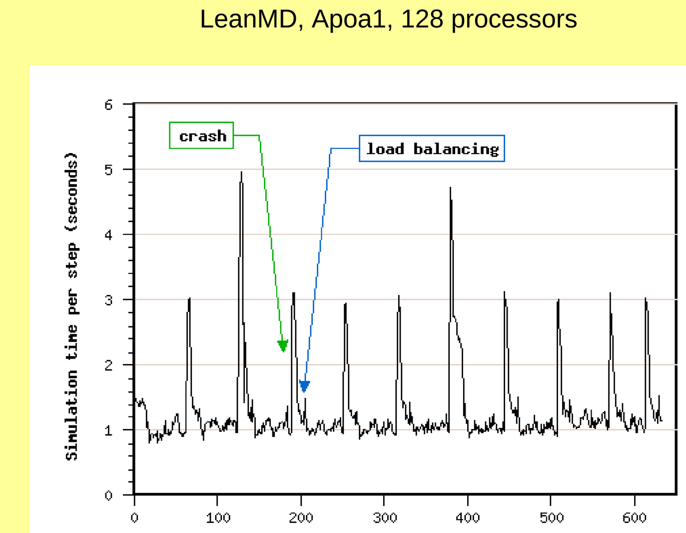


Load Balancing in Fault Tolerance

- Double in-memory checkpoint/restart
  - Does not rely on extra processors
  - Maintain execution efficiency after restart



LeanMD, Apoa1, 128 processors



- LeanMD application
- 10 crashes
- 128 processors
- Checkpoint every 10 time steps

Future work

- Apply adaptive load balancing framework for increasingly complex simulations
  - Adaptive insertion/activation of cohesive elements for dynamic fracture simulations
  - Adaptive mesh adaptation
- Conduct experiments using the load balancing framework on very large parallel machines such as Blue Gene/L
  - Requires mesh to be partitioned into very large number of chunks
  - Experiment with the hierarchical load balancing strategy