CS 498
Hot Topics in High Performance Computing

Networks and Fault Tolerance

2. Introduction to Parallel Computer Architecture (II)

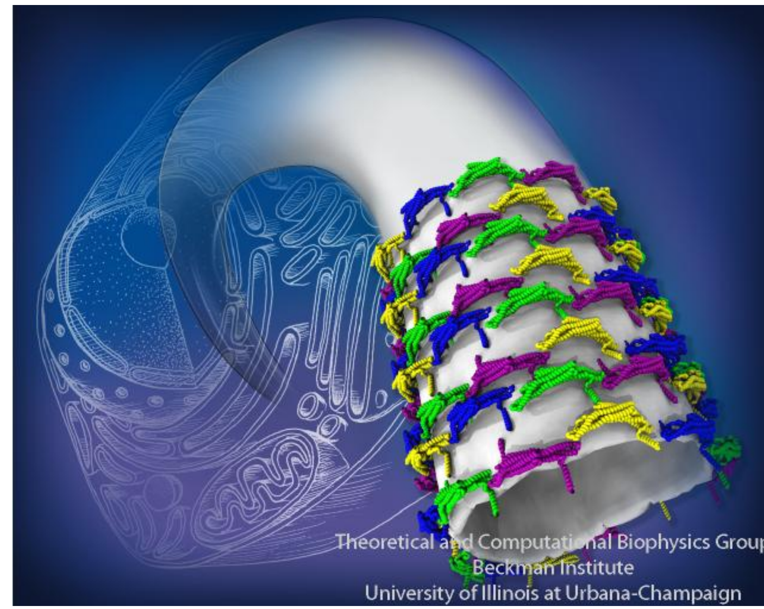
Intro

- What did we learn in the last lecture
  - HPC as “Formula 1” of computing
  - Parallelism will be inevitable
  - Networks will grow
- What will we learn today
  - 101 Parallel Architectures and Programming
  - A first simple network model
  - Multicast/Broadcast in a simple model

Computational Microscope

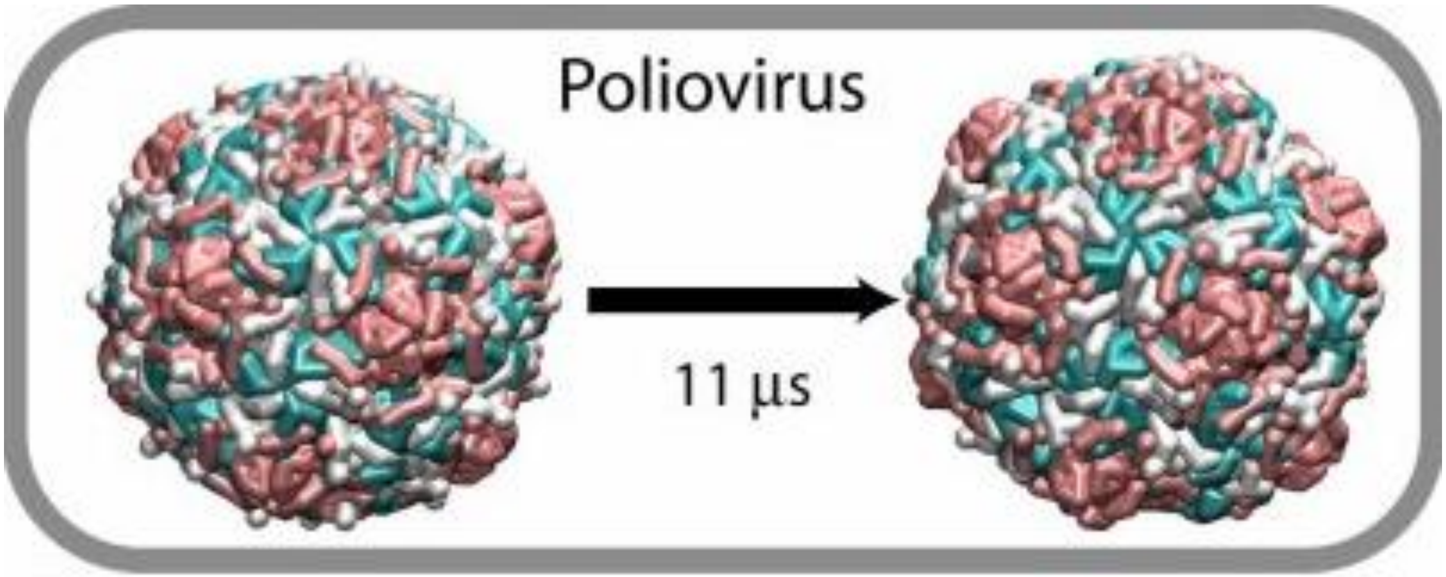
Klaus Schulten (Illinois)

- Discipline & science goals
  - Classical molecular dynamics
  - Simulate large (10s of millions of atoms) molecular structures for sufficiently long time scales
  - Capture time-dependent behavior
    - Difficult to discern, even by X-ray crystallography or cryo-electron microscopy
- Application: NAMD
  - Active development for many years
  - Large user base



BAR domains sculpting membranes into tubes. Yin, et al., *Structure*, 17:882-892, 2009.

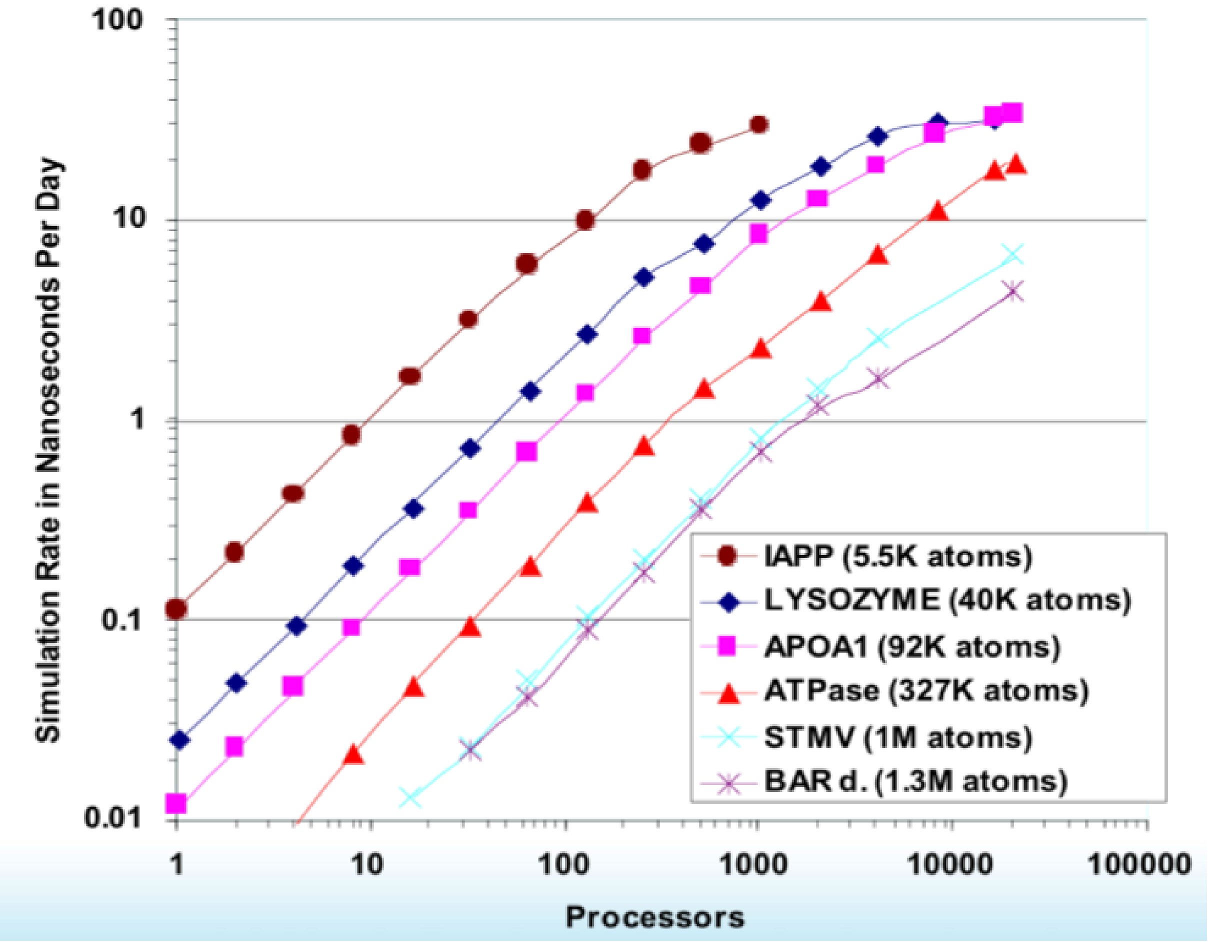
Computational Microscope (cont'd)



- Structural transitions in poliovirus entry (10M atoms + 5M coarse-grain beads
  - Virus binds to cell surface receptor CD155 & externalizes buried capsid protein domains
  - Capsid interacts with host cell to form membrane pore through which viral RNA enters

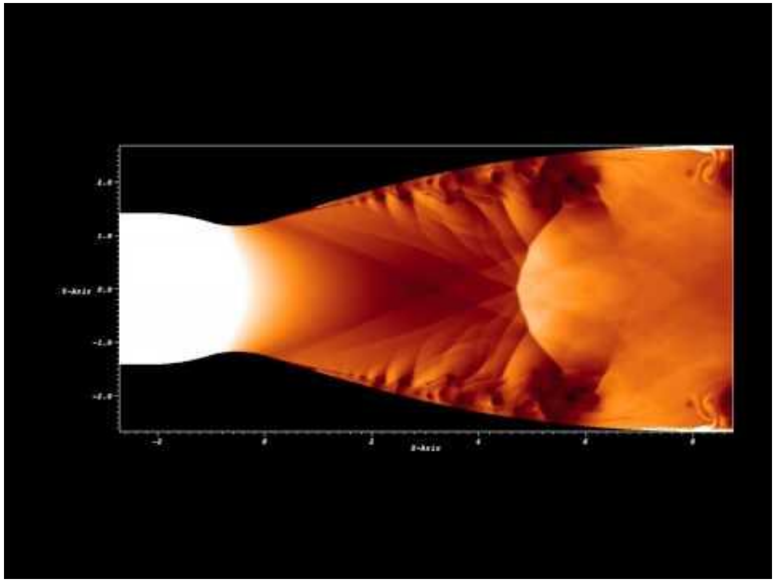
Computational Microscope (cont'd)

- Computational challenges
  - ~1B time steps
  - Scalability limited by communication latency
  - Load balance
- Solutions
  - Assign 1 patch per SMP to reduce use of interconnect
  - Hierarchical load balancing scheme



Petascale Computations for Complex Turbulent Flows

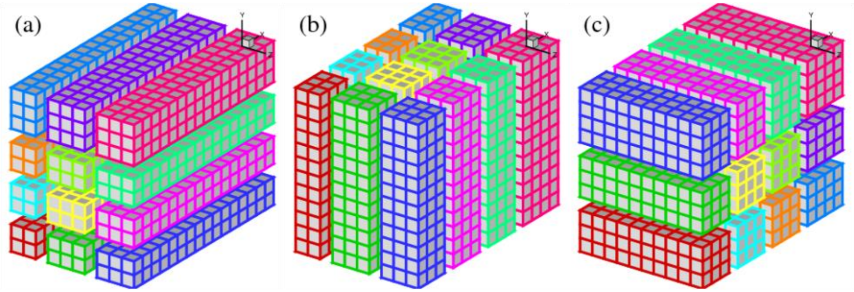
P. K. Yeung (Georgia Tech)



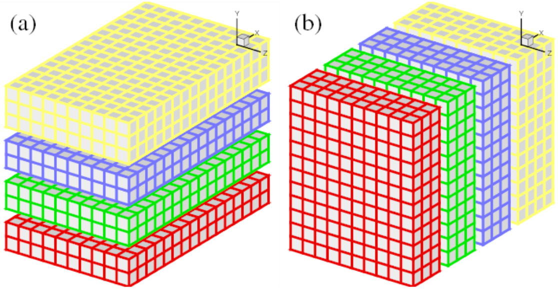
- Discipline & science goals
  - Computational Fluid Dynamics
  - High Reynolds number flows
  - Study effects of stratification, chemical reactions, & wall boundaries on flow structure, mixing & dispersion
  - Need adequate resolution of wide range of length and time scales
- Code names
  - PSDNS – fluid code
  - P3DFFT – 3D FFT package

PSDNS, Yeung (cont'd)

- Pseudo-spectral method
  - Fixed, structured grid in 3D
  - Fourier transform most terms of equations of motion to wave-number space & back
    - 3D FFTs composed of 1D FFTs that span entire domain
    - Must transpose globally distributed 3D arrays
- Simulations for Blue Waters
  - Simulations with chemical reactions and density varying with wall distance
  - 8192^3 grid using ~200K cores



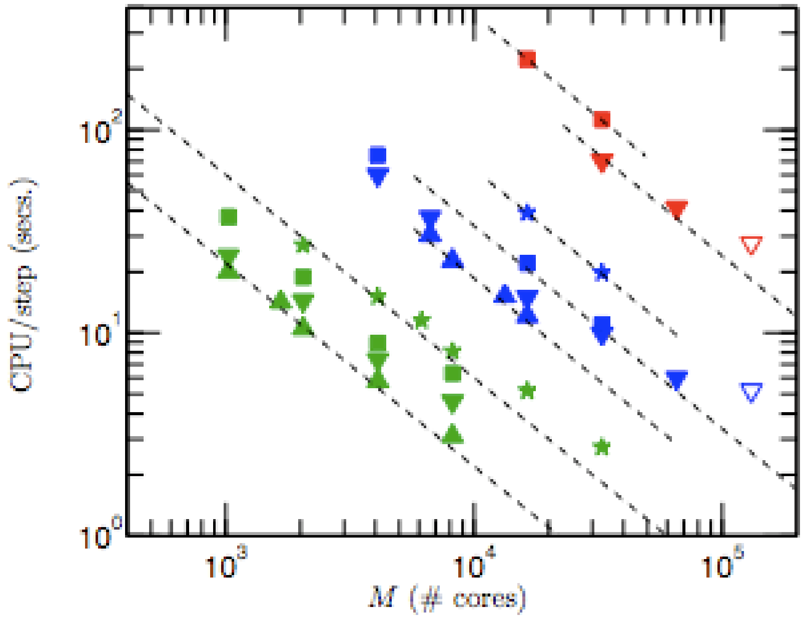
Pencil decomposition



Slab decomposition

PSDNS, Yeung (cont'd)

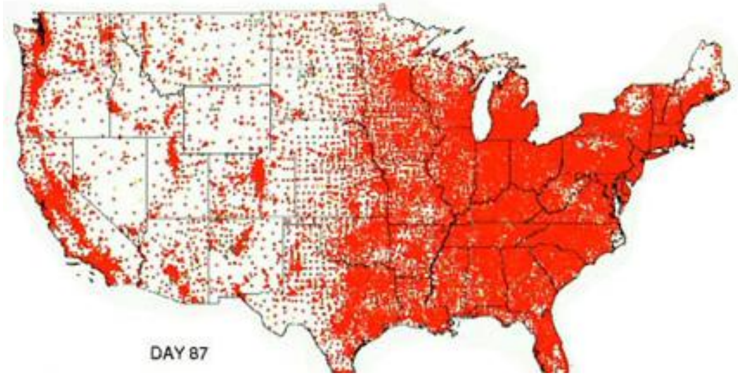
- Computational challenges
  - 3D array transposes require transferring a large amount of data
  - Run times bounded by interconnect bandwidth
  - Communication time dominates wall clock time



CPU time per step at grid resolution 2048 (green), 4096(blue), 8192(red). On IBM BG/L, Sun and Cray clusters, Yeung *et. al*, 2009.

Simulation of Contagion on Very Large Social Networks

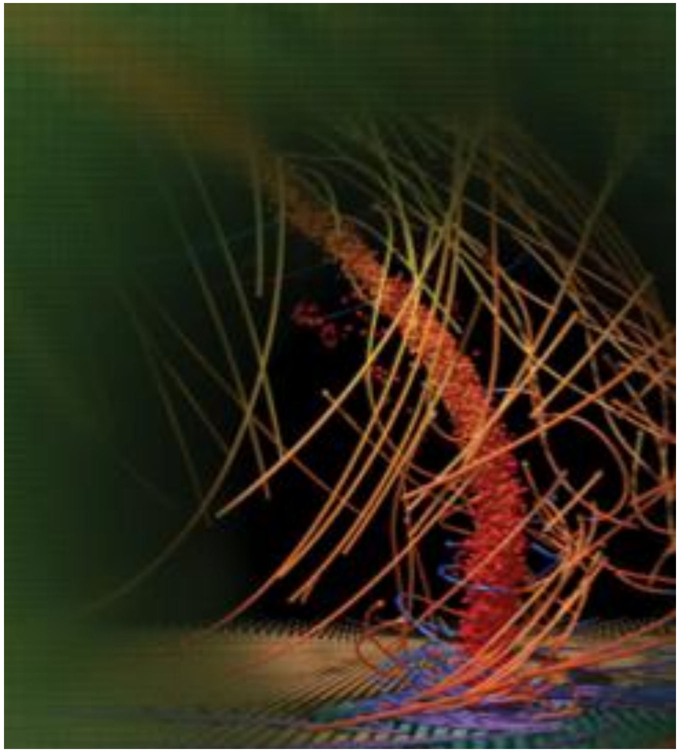
Keith Bisset, Shawn Brown, Douglas Roberts



- Goal
  - Develop the Scalable Petascale Contagion Environment Simulator (SPACES), which uses agent-based models to evaluate mitigation strategies for contagion on extremely large social contact networks
- Approach
  - Build on EpiSims code (pure MPI)
  - Design and implement an environment for executing large, semi-adaptive experimental designs to support realistic case studies on national and global-scale social networks

Understanding Tornadoes and Their Parent Supercells Through Ultra-High Resolution Simulation/Analysis

Bob Wilhelmson (UIUC), et al

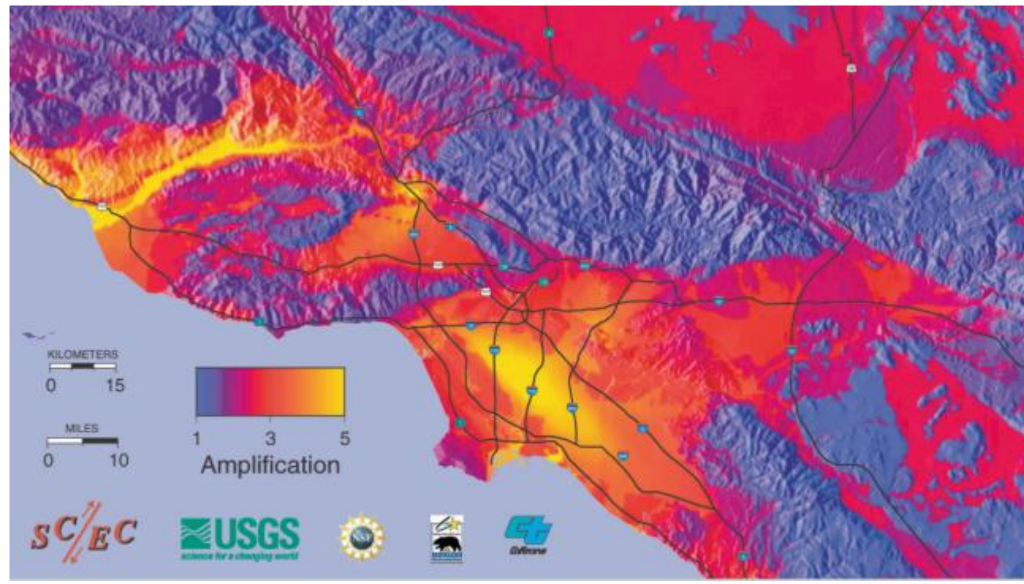


- Goal
  - Simulate development, structure, & demise of large tornadoes in supercells
  - Resolution sufficient to capture low-level tornado inflow, thin precipitation shaft that forms “hook echo” adjacent to tornado, & other smaller scale structures
- Approach
  - Use finite differences to solve equations of motion for air and water substances (droplets, rain, ice, …)

Earthquake System Science

Tom Jordan, Phil Maechling

Ground-motion amplification factors are higher in areas of softer rock and thicker sediments

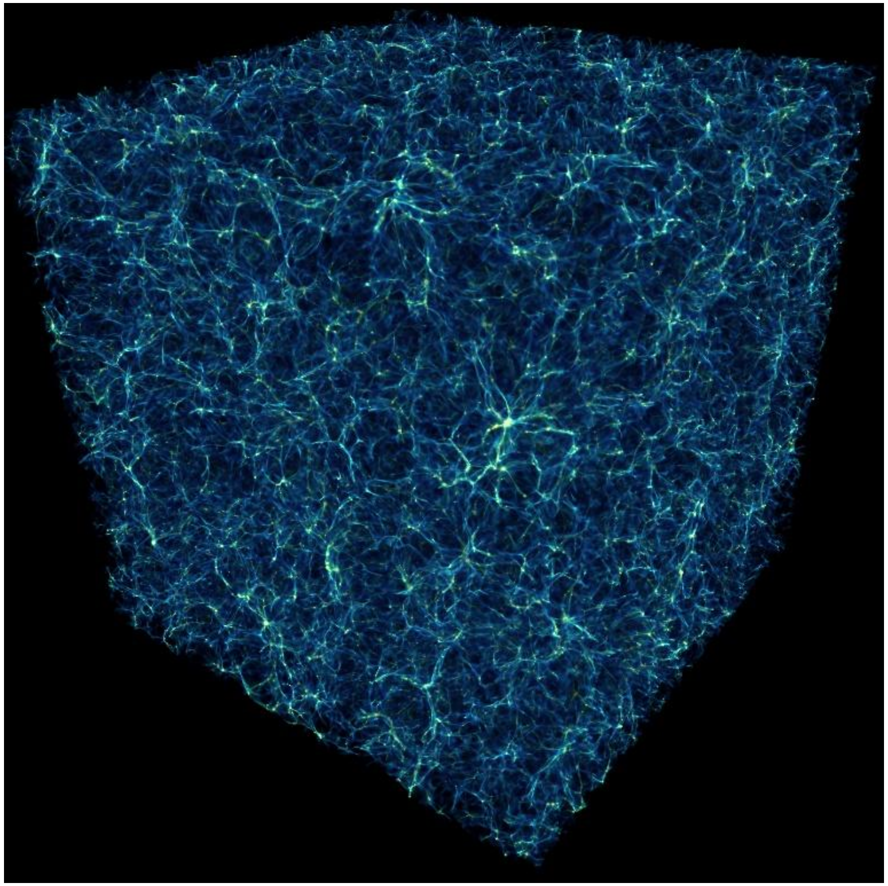


- Prepare 3 seismic & engineering codes for Blue Waters
  1. AWP-Olsen: finite difference dynamic rupture & wave propagation
  2. Hercules: finite element wave propagation
  3. OpenSees: structure modeling for earthquake engineering

Formation of the first galaxies

Brian O'Shea (MSU) & Michael Norman (UCSD)

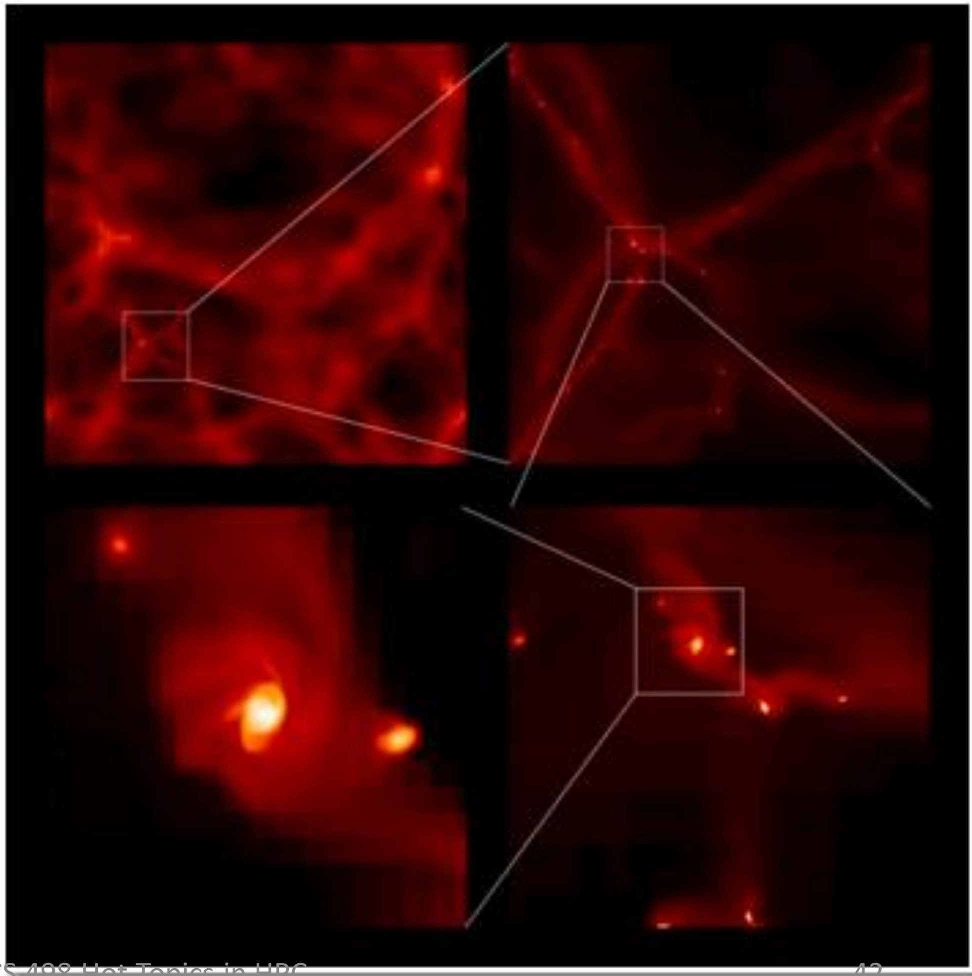
- New science
  - Simulation of galaxies with self-consistent radiation transport & magnetic fields
  - Predictions for upcoming James Webb Space Telescope and 30-meter telescope
- Application Code
  - ENZO (C++) with F77 kernels, optional UPC or Global Arrays
  - Cello – object-oriented redesign of ENZO using Charm++
  - Adaptive (nested) Mesh Refinement (up to 35 levels)



Galaxy formation and virtual astronomy

Nagamine (UNLV), Bryan (Columbia), Ostriker & Cen (Princeton)

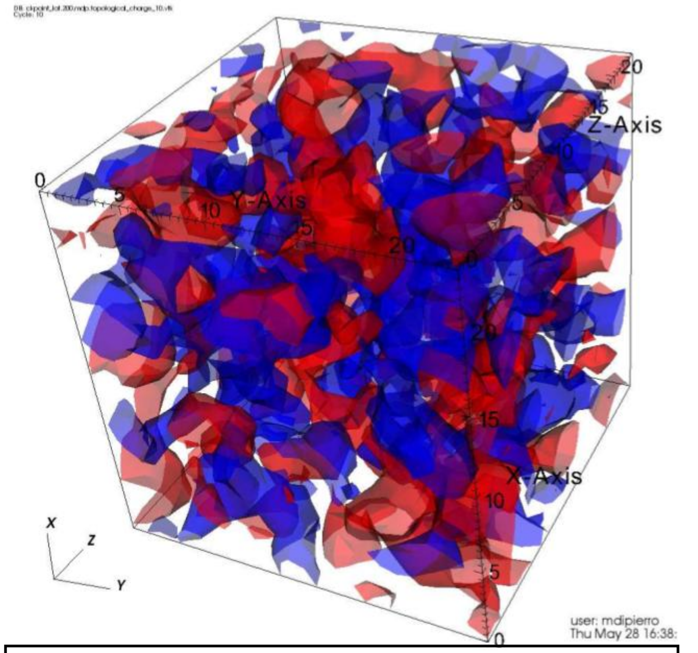
- Overall questions addressed
  - When and how did galaxies form?
  - How did galaxies evolve?
  - Do predictions of the ΛCDM cosmological model agree with observations?
- Quantitative predictions
  - Star formation and mass assembly history
  - Luminosity functions and colors in various bands at different epochs
  - Light-cone output and galaxy number counts
  - Galactic clustering and evolution as functions of luminosity, color, and environment
- Numerical method
  - Compare Enzo and new hydro tree particle mesh code (HTPM)



Lattice Quantum Chromodynamics

PI: Robert Sugar (UCSB) with LQCD community

- Discipline & science goals
  - Determine predictions of lattice field theories (QCD & Beyond Standard Model)
  - Compute Feynman path integrals for a given theory using importance-sampling techniques to generate field configurations which are used to evaluate a large range of physical properties
    - Masses, internal structures, particle interactions



3D slice of topological charge density iso-surface.

Addendum to Grading

- Only for the 4cr students (mostly grads):
  - 25% Midterm
  - 25% Final
  - 25% Presentation
  - 25% Group project
    - Groups of 2-3 students work on a class project
    - Will be applied (involve coding and running)

Section II: Parallel Architectures

- What is a parallel platform?
  – Parallel Computer Hardware
  – + Operating System (+Middleware)
  – (+ Programming Model)

- Historically, architectures and programming models were coupled tightly
  – Architecture designed for PM (or vice versa)

Some (Historical) Examples

- Systolic Array
  - Data-stream driven (data counters)
  - Multiple streams for parallelism
  - Specialized for applications (reconfigurable)
- Dataflow Architectures
  - No program counter, execute instructions when all input arguments are available
  - Fine-grained, high overheads
    - Example: compute f = (a+b) * (c+d)

More Recent Examples

- Von Neumann Architecture (program counter)
- Flynn's Taxonomy:
  - SISD – standard serial computer (nearly extinct)
  - SIMD – vector machines or additions (MMX, SSE)
  - MISD – fault tolerant computing (planes etc.)
  - MIMD – multiple autonomous PEs
- Typical supercomputers use a combination of techniques to achieve highest performance!

Parallel Architectures 101

- Two general platform types:
  - Shared Memory Machines (SMM)
    - Shared address space
    - Hardware for cache-coherent remote memory access
    - Cache-coherent Non Uniform Memory Access (cc NUMA)
  - Distributed Memory Machines (DMM)
    - Either pure distributed memory or ncc-NUMA
    - ncc-NUMA may support global address space (GAS)

Programming Model Basics

- The PM reflect machine concepts/model for the programmer to use
  - How to make elements work together
  - Performance is key!
  - E.g., communication and synchronization
- Four major classes
  - Multiprogramming (multiple applications)
  - Shared address space (cf. SMM, bulletin board)
  - Message passing (cf. postal system)
  - Data parallel (cf. factory, coordinated actions on data)

Shared Memory Machines

- Two historical architectures:
  - "Mainframe" – all-to-all connection between memory, I/O and PEs
    - Often used if PE is the most expensive part
    - Bandwidth scales with P!
    - PE Cost scales with P, Question: what about network cost?
    - Cost can be cut with multistage connections (butterfly)
  - "Minicomputer" – bus-based connection
    - All traditional SMP systems
    - High latency, low bandwidth (cache becomes important)
    - Tricky to achieve highest performance (contention)
    - Low cost, extensible

SMM Architecture

- Today's architectures blur together
  - E.g., Hypertransport, Advanced Switching Interconnect, or Quick Path Interconnect
  - Switch-based networks
  - Use "traditional" topologies
  - Similar issues as we will discuss but at smaller scale
- Often basic building blocks in Supercomputers, i.e., networks are hierarchical!

SMM Capabilities

- Any PE can access all memory
- Any I/O can access all memory (maybe limited)
- OS (resource management) can run on any PE
  - Can run multiple threads in shared memory
- Communication through shared memory
  - Load/store commands to memory controller
- Coordination through shared memory

SMM Programming Model

- Threads (e.g., POSIX threads) or processes
- Communication through memory
- Synchronization through memory or OS objects
  - Lock/mutex (protect critical region)
  - Semaphore (generalization of mutex (binary sem.))
  - Barrier (synchronize a group of activities)
  - Atomic Operations (CAS, Fetch-and-add)
  - Transactional Memory (execute regions atomically)

An Example: Compute Pi

- Using Gregory-Leibnitz Series:
  - $4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}$
  - Iterations of sum can be computed in parallel
  - Needs to sum all contributions at the end

Pthreads Compute Pi Example

```
int main( int argc, char *argv[] )
{
   // definitions …
   thread_arr = (pthread_t*)malloc(nthreads
      * sizeof(pthread_t));
   resultarr= (double*)malloc(nthreads *
      sizeof(double));

   for(i=0; i<nthreads; ++i) {
     int ret = pthread_create( &thread_arr[i],
      NULL, compute_pi, (void*) i);
   }
   for(i=0; i<nthreads; ++i) {
     pthread_join( thread_arr[i], NULL);
   }
   pi = 0;
   for(i=0; i<nthreads; ++i) pi += resultarr[i];

   printf("pi is approximately %.16f, Error is
      %.16f\n", pi, fabs(pi - PI25DT));
}
```

```
int n=10000;
double *resultarr;
int nthreads;

void *compute_pi(void *data) {
  int i, j;
  int myid = (int)(long)data;
  double mypi, h, x, sum;

  for (j=0; j<n; ++j) {
    h   = 1.0 / (double) n;
    sum = 0.0;
    for (i = myid + 1; i <= n; i += nthreads) {
      x = h * ((double)i - 0.5);
      sum += (4.0 / (1.0 + x*x));
    }
    mypi = h * sum;
  }
  resultarr[myid] = mypi;
}
```

Some More Comments

- OpenMP would allow to implement this example much simpler (but has other issues)
- Transparent shared memory has some issues in practice:
  - False sharing (e.g., resultarr[])
  - Race conditions (complex mutual exclusion protocols)
  - Little tool support (debuggers need some work)
- Achieving performance is harder than it seems!

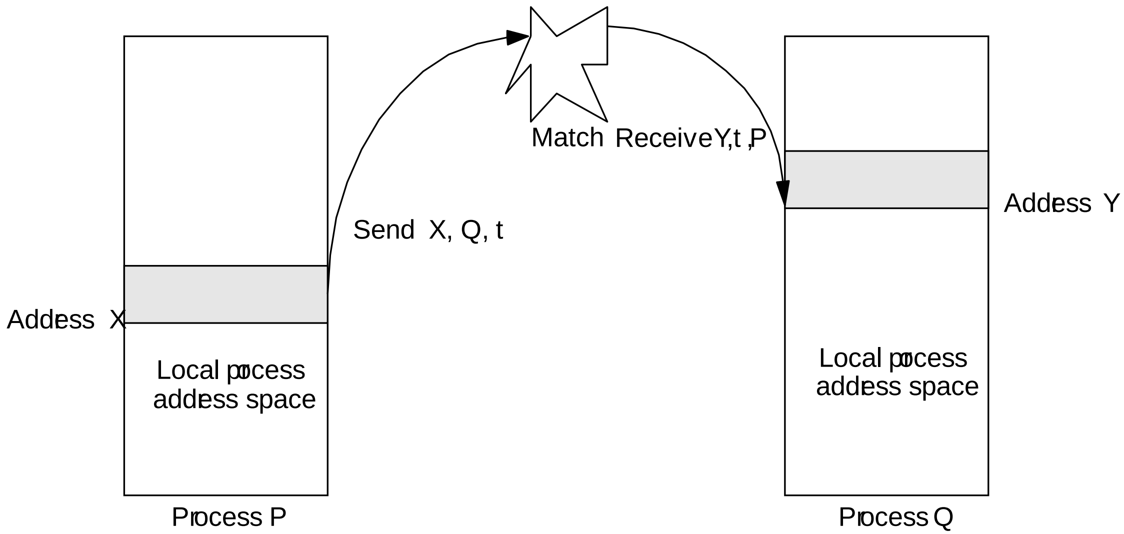
DMM Capabilities

- Explicit communication between PEs
  - Message passing or channels
- Only local memory access, no direct access to remote memory
  - No shared resources
- Communication through packets or channels
- Synchronization through packets or hardware

DMM Programming Model

- Typically Message Passing (MPI, PVM)
- Communication through messages or group operations (broadcast, reduce, etc.)
- Synchronization through messages (sometimes unwanted side effect) or group operations (barrier)
- Typically supports message matching and communication contexts

Message Passing Example



- Send specifies buffer to be transmitted
- Recv specifies buffer to receive into
- Implies copy operation between named PEs
- Optional tag matching
- Pair-wise synchronization (cf. happens before)
- Explicit buffer copy is an overhead

MPI Compute Pi Example

```
int main( int argc, char *argv[] )
{
   // definitions
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);

   double t = -MPI_Wtime();
   for (j=0; j<n; ++j) {
     h   = 1.0 / (double) n;
     sum = 0.0;
     for (i = myid + 1; i <= n; i += numprocs) { x = h * ((double)i - 0.5);  sum += (4.0 / (1.0 + x*x)); }
     mypi = h * sum;
     MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
   }
   t+=MPI_Wtime();

   if(!myid) {
     printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));
     printf("time: %f\n", t);
   }

   MPI_Finalize();
}
```

DMM - PGAS

- Partitioned Global Address Space
  - Shared memory emulation for DMM
  - "Distributed Shared Memory"
- Simplifies shared access to distributed data
  - Has similar problems as SMM programming
  - Sometimes lacks performance transparency
    - Local vs. remote accesses
- Examples:
  - UPC, CAF, Titanium, X10, …

How to Tame the Beast?

- How to program large machines?
- No single approach, PMs are not converging yet
  - MPI, PGAS, OpenMP, Hybrid, …
- Architectures converge
  - General purpose nodes connected by general purpose or specialized networks
  - Small scale often uses commodity networks
  - Specialized networks become necessary at scale

Performance Matters

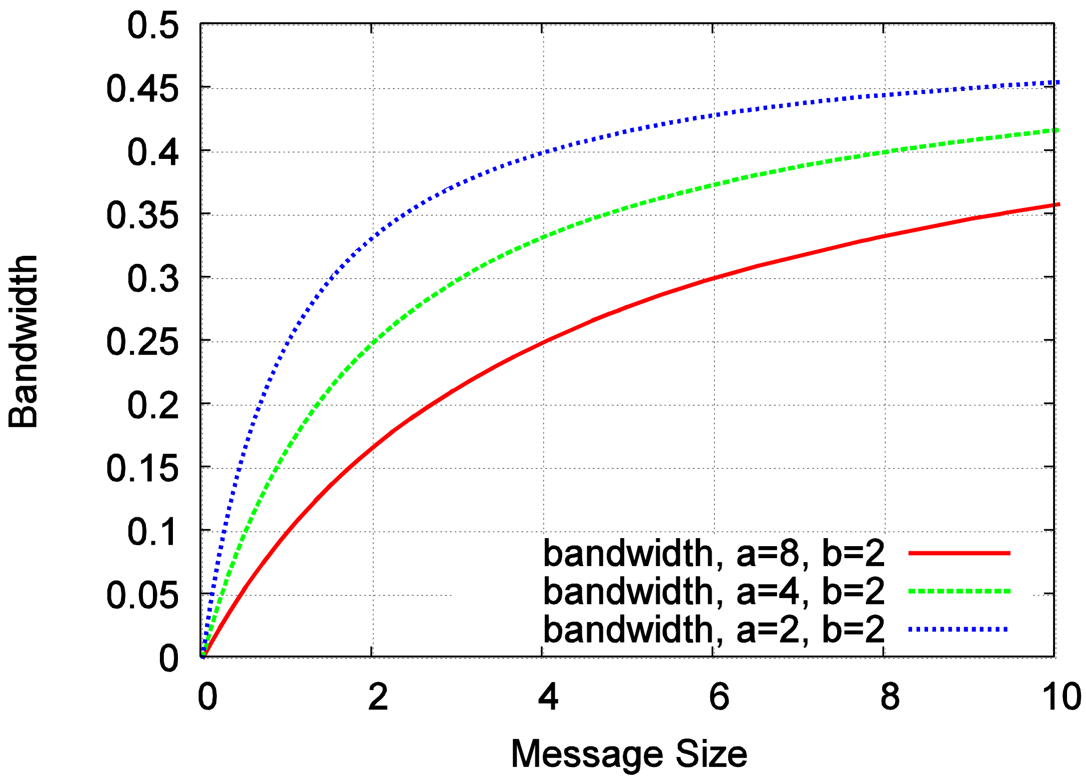
- Developers need to understand expected performance
  - Serial performance is not topic of this class
  - Focus on communication performance
- Simplest metric for networks: bandwidth
  - Same for Internet connection and HPC networks
  - Example: 10 Gbit/s = 1.25 GB/s
    - 1 MiB transfers in 800 microseconds
- Class Question: What other network metric do you know? And how is it different?

A Simple Model for Communication

- Transfer time $T(s) = \alpha+\beta s$
  - α = startup time (latency)
  - β = cost per byte (bandwidth=$1/\beta$)
- As s increases, bandwidth approaches $1/\beta$ asymptotically
- Convergence rate depends on α
- $s_{1/2} = \alpha/\beta$
- Often assuming no pipelining (new messages can only be issued from a process after all arrived)

Bandwidth vs. Latency

- $s_{1/2} = \alpha/\beta$ often used to distinguish bandwidth- and latency-bound messages



Quick Example

- Simplest linear broadcast
  - One process has a data item to be distributed to all processes
- Sending s bytes to P processes:
  - T(s) = P * (α+βs) = $\mathcal{O}(P)$
- Class question: Do you know a faster method to accomplish the same?

k-ary Tree Broadcast

- Origin process is the root of the tree, passes messages to k neighbors which pass them on
  - k=2 -> binary tree
- Class Question: What is the broadcast time in the simple latency/bandwidth model?