PARLab Parallel Boot Camp

Sources of Parallelism and Locality in Simulation

Jim Demmel
EECS and Mathematics
University of California, Berkeley
Parallelism and Locality in Simulation

- Parallelism and data locality both critical to performance
  - Moving data most expensive operation
- Real world problems have parallelism and locality:
  - Many objects operate independently of others.
  - Objects often depend much more on nearby than distant objects.
  - Dependence on distant objects can often be simplified.
    » Example of all three: particles moving under gravity
- Scientific models may introduce more parallelism:
  - When a continuous problem is discretized, time dependencies are generally limited to adjacent time steps.
    » Helps limit dependence to nearby objects (eg collisions)
  - Far-field effects may be ignored or approximated in many cases.
- Many problems exhibit parallelism at multiple levels
Basic Kinds of Simulation

• Discrete Event Systems
  - “Game of Life”, Manufacturing Systems, Finance, Circuits, Pacman …

• Particle Systems
  - Billiard balls, Galaxies, Atoms, Circuits, Pinball …

• Lumped Systems (Ordinary Differential Eqns - ODEs)
  - Structural Mechanics, Chemical kinetics, Circuits, Star Wars: The Force Unleashed

• Continuous Systems (Partial Differential Eqns - PDEs)
  - Heat, Elasticity, Electrostatics, Finance, Circuits, Medical Image Analysis, Terminator 3: Rise of the Machines

• A given phenomenon can be modeled at multiple levels

• Many simulations combine multiple techniques

• For more on simulation in games, see
  • [www.cs.berkeley.edu/b-cam/Papers/Parker-2009-RTD](http://www.cs.berkeley.edu/b-cam/Papers/Parker-2009-RTD)
### Example: Circuit Simulation

- Circuits are simulated at many different levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Primitives</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction level</td>
<td>Instructions</td>
<td>SimOS, SPIM</td>
</tr>
<tr>
<td>Cycle level</td>
<td>Functional units</td>
<td>VIRAM-p</td>
</tr>
<tr>
<td>Register Transfer Level (RTL)</td>
<td>Register, counter, MUX</td>
<td>VHDL</td>
</tr>
<tr>
<td>Gate Level</td>
<td>Gate, flip-flop, memory cell</td>
<td>Thor</td>
</tr>
<tr>
<td>Switch level</td>
<td>Ideal transistor</td>
<td>Cosmos</td>
</tr>
<tr>
<td>Circuit level</td>
<td>Resistors, capacitors, etc.</td>
<td>Spice</td>
</tr>
<tr>
<td>Device level</td>
<td>Electrons, silicon</td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Discrete event systems
  - Time and space are discrete

• Particle systems
  - Important special case of lumped systems

• Lumped systems (ODEs)
  - Location/entities are discrete, time is continuous

• Continuous systems (PDEs)
  - Time and space are continuous

• Identify common problems and solutions
Model Problem: Sharks and Fish

• Illustrates parallelization of these simulations
• Basic idea: sharks and fish living in an ocean
  - rules for movement (discrete and continuous)
  - breeding, eating, and death
  - forces in the ocean
  - forces between sea creatures
• 6 different versions
  - Different sets of rules, to illustrate different simulations
• Available in many languages
  - Matlab, pThreads, MPI, OpenMP, Split-C, Titanium, CMF, ...
  - See bottom of www.cs.berkeley.edu/~demmel/cs267_Spr09/
• One (or a few) will be used as lab assignments
  - www.cs.berkeley.edu/~volkov/cs267.sp09/hw2/
  - Rest available for your own classes!
"7 Dwarfs" of High Performance Computing

- Phil Colella (LBL) identified 7 kernels of which most simulation and data-analysis programs are composed:

1. **Dense Linear Algebra**
   - Ex: Solve $Ax=b$ or $Ax = \lambda x$ where $A$ is a dense matrix

2. **Sparse Linear Algebra**
   - Ex: Solve $Ax=b$ or $Ax = \lambda x$ where $A$ is a sparse matrix (mostly zero)

3. **Operations on Structured Grids**
   - Ex: $A_{\text{new}}(i,j) = 4\times A(i,j) - A(i-1,j) - A(i+1,j) - A(i,j-1) - A(i,j+1)$

4. **Operations on Unstructured Grids**
   - Ex: Similar, but list of neighbors varies from entry to entry

5. **Spectral Methods**
   - Ex: Fast Fourier Transform (FFT)

6. **Particle Methods**
   - Ex: Compute electrostatic forces on $n$ particles

7. **Monte Carlo**
   - Ex: Many independent simulations using different inputs

8/20/09  Jim Demmel  Sources: 7
DISCRETE EVENT SYSTEMS
Systems are represented as:
- finite set of variables.
- the set of all variable values at a given time is called the state.
- each variable is updated by computing a transition function depending on the other variables.

System may be:
- synchronous: at each discrete timestep evaluate all transition functions; also called a state machine.
- asynchronous: transition functions are evaluated only if the inputs change, based on an “event” from another part of the system; also called event driven simulation.

Example: The “game of life:”
- Space divided into cells, rules govern cell contents at each step
- Also available as Sharks and Fish #3 (S&F 3)
• The simulation is synchronous
  - use two copies of the grid (old and new).
  - the value of each new grid cell depends only on 9 cells (itself plus 8 neighbors) in old grid.
  - simulation proceeds in timesteps-- each cell is updated at every step.
• Easy to parallelize by dividing physical domain: Domain Decomposition

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4</td>
<td>P5</td>
<td>P6</td>
</tr>
<tr>
<td>P7</td>
<td>P8</td>
<td>P9</td>
</tr>
</tbody>
</table>

Repeat
  - compute locally to update local system
  - barrier()
  - exchange state info with neighbors
  - until done simulating

• Locality is achieved by using large patches of the ocean
  - Only boundary values from neighboring patches are needed.
• How to pick shapes of domains?
Regular Meshes

- Suppose graph is nxn mesh with connection NSEW neighbors
  - Which partition has less communication? (n=18, p=9)
- Minimizing communication on mesh ≡ minimizing “surface to volume ratio” of partition

\[ n^*(p-1) \] edge crossings

\[ 2n^*(p^{1/2} - 1) \] edge crossings
Synchronous Circuit Simulation

- Circuit is a **graph** made up of subcircuits connected by wires
  - Component simulations need to interact if they share a wire.
  - Data structure is (irregular) graph of subcircuits.
  - Parallel algorithm is timing-driven or **synchronous**:
    » Evaluate all components at every timestep (determined by known circuit delay)

- **Graph partitioning** assigns subgraphs to processors
  - Determines parallelism and locality.
  - Goal 1 is to evenly distribute subgraphs to nodes (load balance).
  - Goal 2 is to minimize edge crossings (minimize communication).
  - Easy for meshes, NP-hard in general, so we will approximate (tools available!)

\[
\begin{align*}
\text{better} & \quad \text{edge crossings} = 6 & \text{edge crossings} = 10
\end{align*}
\]
Sharks & Fish in Loosely Connected Ponds

- Parallelization: each processor gets a set of ponds with roughly equal total area
  - work is proportional to area, not number of creatures
- One pond can affect another (through streams) but infrequently
Asynchronous Simulations

- Synchronous simulations may waste time:
  - Simulates even when the inputs do not change.

- Asynchronous (event-driven) simulations update only when an event arrives from another component:
  - No global time steps, but individual events contain time stamp.
  - Example: Game of life in loosely connected ponds (don’t simulate empty ponds).
  - Example: Circuit simulation with delays (events are gates changing).
  - Example: Traffic simulation (events are cars changing lanes, etc.).

- Asynchronous is more efficient, but harder to parallelize
  - With message passing, events are naturally implemented as messages, but how do you know when to execute a “receive”?
Scheduling Asynchronous Circuit Simulation

- **Conservative:**
  - Only simulate up to (and including) the minimum time stamp of inputs.
  - Need deadlock detection if there are cycles in graph
    » Example on next slide
  - Example: Pthor circuit simulator in Splash1 from Stanford.

- **Speculative (or Optimistic):**
  - Assume no new inputs will arrive and keep simulating.
  - May need to backup if assumption wrong, using timestamps
  - Example: Timewarp [D. Jefferson], Parswec [Wen,Yelick].

- **Optimizing load balance and locality is difficult:**
  - Locality means putting tightly coupled subcircuit on one processor.
  - Since “active” part of circuit likely to be in a tightly coupled subcircuit, this may be bad for load balance.
Deadlock in Conservative Asynchronous Circuit Simulation

- Example: Sharks & Fish 3, with 3 processors simulating 3 ponds connected by streams along which fish can move

- Suppose all ponds simulated up to time $t_0$, but no fish move, so no messages sent from one proc to another
  - So no processor can simulate past time $t_0$
- Fix: After waiting for an incoming message for a while, send out an “Are you stuck too?” message
  - If you ever receive such a message, pass it on
  - If you receive such a message that you sent, you have a deadlock cycle, so just take a step with latest input
- Can be a serial bottleneck
Summary of Discrete Event Simulations

- Model of the world is discrete
  - Both time and space

- Approaches
  - Decompose domain, i.e., set of objects
  - Run each component ahead using
    » Synchronous: communicate at end of each timestep
    » Asynchronous: communicate on-demand
    • Conservative scheduling – wait for inputs
      – need deadlock detection
    • Speculative scheduling – assume no inputs
      – roll back if necessary
PARTICLE SYSTEMS
Particle Systems

- A particle system has
  - a finite number of particles
  - moving in space according to Newton’s Laws (i.e. \( F = ma \))
  - time is continuous

- Examples
  - stars in space with laws of gravity
  - electron beam in semiconductor manufacturing
  - atoms in a molecule with electrostatic forces
  - neutrons in a fission reactor
  - cars on a freeway with Newton’s laws plus model of driver and engine
  - balls in a pinball game

- Reminder: many simulations combine techniques such as particle simulations with some discrete events (Ex Sharks and Fish)
Forces in Particle Systems

• Force on each particle can be subdivided

\[
\text{force} = \text{external\_force} + \text{nearby\_force} + \text{far\_field\_force}
\]

• External force
  • ocean current to sharks and fish world (S&F 1)
  • externally imposed electric field in electron beam

• Nearby force
  • sharks attracted to eat nearby fish (S&F 5)
  • balls on a billiard table bounce off of each other
  • Van der Waals forces in fluid \((1/r^6)\) ... how Gecko feet work?

• Far-field force
  • fish attract other fish by gravity-like \((1/r^2)\) force (S&F 2)
  • gravity, electrostatics, radiosity in graphics
  • forces governed by elliptic PDE
Example S&F 1: Fish in an External Current

% fishp = array of initial fish positions (stored as complex numbers)
% fishv = array of initial fish velocities (stored as complex numbers)
% fishm = array of masses of fish
% tfinal = final time for simulation (0 = initial time)
% Algorithm: update position [velocity] using velocity [acceleration]
% at each time step
% Initialize time step, iteration count, and array of times
   dt = .01;   t = 0;
% loop over time steps
   while t < tfinal,
      t = t + dt;
      fishp = fishp + dt*fishv;
      accel = current(fishp)./fishm;   % current depends on position
      fishv = fishv + dt*accel;
% update time step (small enough to be accurate, but not too small)
      dt = min(.1*max(abs(fishv))/max(abs(accel)),1);
   end

8/20/09  Jim Demmel
Parallelism in External Forces

- These are the simplest
- The force on each particle is independent
- Called "embarrassingly parallel"
  - Corresponds to "map reduce" pattern

- Evenly distribute particles on processors
  - Any distribution works
  - Locality is not an issue, no communication

- For each particle on processor, apply the external force
  - May need to "reduce" (e.g., compute maximum) to compute time step, other data
Parallelism in Nearby Forces

- Nearby forces require interaction and therefore communication.
- Force may depend on other nearby particles:
  - Example: collisions.
  - Simplest algorithm is $O(n^2)$: look at all pairs to see if they collide.
- Usual parallel model is **domain decomposition** of physical region in which particles are located
  - $O(n/p)$ particles per processor if evenly distributed.
Parallelism in Nearby Forces

- **Challenge 1:** interactions of particles near processor boundary:
  - need to communicate particles near boundary to neighboring processors.
  - **Low surface to volume ratio** means low communication.
    » Use squares, not slabs

Communicate particles in boundary region to neighbors

Need to check for collisions between regions
Parallelism in Nearby Forces

• **Challenge 2: load imbalance, if particles cluster:**
  - galaxies, electrons hitting a device wall.

• **To reduce load imbalance, divide space unevenly.**
  - Each region contains roughly equal number of particles.
  - Quad-tree in 2D, oct-tree in 3D.

Example: each square contains at most 3 particles
Parallelism in Far-Field Forces

- Far-field forces involve all-to-all interaction and therefore communication.
- Force depends on all other particles:
  - Examples: gravity, protein folding
  - Simplest algorithm is $O(n^2)$ as in S&F 2, 4, 5.
  - Just decomposing space does not help since every particle needs to “visit” every other particle.

Implement by rotating particle sets.

- Keeps processors busy
- All processor eventually see all particles

- Use more clever algorithms to beat $O(n^2)$.
Far-field Forces: $O(n \log n)$ or $O(n)$, not $O(n^2)$

• Based on approximation:
  - Settle for the answer to just 3 digits, or just 15 digits ...

• Two approaches
  - “Particle-Mesh”
    » Approximate by particles on a regular mesh
    » Exploit structure of mesh to solve for forces fast (FFT)
  - “Tree codes” (Barnes-Hut, Fast-Multipole-Method)
    » Approximate clusters of nearby particles by single “metaparticles”
    » Only need to sum over (many fewer) metaparticles

: Particle-Mesh

Tree code:
LUMPED SYSTEMS - ODES
System of Lumped Variables

• Many systems are approximated by
  - System of “lumped” variables.
  - Each depends on continuous parameter (usually time).

• Example -- circuit:
  - approximate as graph.
    » wires are edges.
    » nodes are connections between 2 or more wires.
    » each edge has resistor, capacitor, inductor or voltage source.
  - system is “lumped” because we are not computing the voltage/current at every point in space along a wire, just endpoints.
  - Variables related by Ohm’s Law, Kirchoff’s Laws, etc.

• Forms a system of ordinary differential equations (ODEs)
  - Differentiated with respect to time
  - Variant: ODEs with some constraints
    » Also called DAEs, Differential Algebraic Equations
Circuit Example

- State of the system is represented by
  - $v_n(t)$ node voltages
  - $i_b(t)$ branch currents
  - $v_b(t)$ branch voltages
  \[ \text{all at time } t \]

- Equations include
  - Kirchoff’s current
  - Kirchoff’s voltage
  - Ohm’s law
  - Capacitance
  - Inductance

\[ \begin{pmatrix}
  0 & A & 0 \\
  A^T & 0 & -I \\
  0 & R & -I \\
  0 & -I & C \frac{d}{dt} \\
  0 & L \frac{d}{dt} & I
\end{pmatrix} \begin{pmatrix}
  v_n \\
  i_b \\
  v_b
\end{pmatrix} = \begin{pmatrix}
  0 \\
  S \\
  0 \\
  0 \\
  0
\end{pmatrix} \]

- $A$ is sparse matrix, representing connections in circuit
  - One column per branch (edge), one row per node (vertex) with $+1$ and $-1$ in each column at rows indicating end points

- Write as single large system of ODEs or DAEs
Another example is structural analysis in civil engineering:

- Variables are displacement of points in a building.
- Newton’s and Hook’s (spring) laws apply.
- Static modeling: exert force and determine displacement.
- Dynamic modeling: apply continuous force (earthquake).
- Eigenvalue problem: do the resonant modes of the building match an earthquake.

OpenSees project in CE at Berkeley looks at this section of 880, among others.
Star Wars - The Force Unleashed...
• In these examples, and most others, the matrices are sparse:
  - i.e., most array elements are 0.
  - neither store nor compute on these 0’s.
  - Sparse because each component only depends on a few others

• Given a set of ODEs, two kinds of questions are:
  - Compute the values of the variables at some time t
    » Explicit methods
    » Implicit methods
  - Compute modes of vibration
    » Eigenvalue problems

8/20/09
Jim Demmel
Solving ODEs

- Suppose ODE is \( x'(t) = A \cdot x(t) \), where \( A \) is a sparse matrix
  - Discretize: only compute \( x(i \cdot dt) = x[i] \) at \( i = 0, 1, 2, ... \)
  - ODE gives \( x'(t) = \text{slope at } t \), and so \( x[i+1] \approx x[i] + dt \cdot \text{slope} \)

- **Explicit methods (ex: Forward Euler)**
  - Use slope at \( t = i \cdot dt \), so slope = \( A \cdot x[i] \).
  - \( x[i+1] = x[i] + dt \cdot A \cdot x[i] \), i.e. **sparse matrix-vector multiplication**.

- **Implicit methods (ex: Backward Euler)**
  - Use slope at \( t = (i+1) \cdot dt \), so slope = \( A \cdot x[i+1] \).
  - Solve \( x[i+1] = x[i] + dt \cdot A \cdot x[i+1] \) for \( x[i+1] = (I - dt \cdot A)^{-1} \cdot x[i] \), i.e. **solve a sparse linear system of equations** for \( x[i+1] \)

- **Tradeoffs:**
  - Explicit: simple algorithm but may need tiny time steps \( dt \) for stability
  - Implicit: more expensive algorithm, but can take larger time steps \( dt \)

- **Modes of vibration – eigenvalues of \( A \)**
  - Algorithms also either multiply \( A \cdot x \) or solve \( y = (I - d \cdot A) \cdot x \) for \( x \)
CONTINUOUS SYSTEMS - PDES
Continuous Systems - PDEs

Examples of such systems include

- Elliptic problems (steady state, global space dependence)
  - Electrostatic or Gravitational Potential: Potential(position)

- Hyperbolic problems (time dependent, local space dependence):
  - Sound waves: Pressure(position,time)

- Parabolic problems (time dependent, global space dependence)
  - Heat flow: Temperature(position, time)
  - Diffusion: Concentration(position, time)

Global vs Local Dependence

- Global means either a lot of communication, or tiny time steps
- Local arises from finite wave speeds: limits communication

Many problems combine features of above

- Fluid flow: Velocity, Pressure, Density(position, time)
- Elasticity: Stress, Strain(position, time)
Implicit Solution of the 1D Heat Equation

\[
\frac{d}{dt} u(x,t) = C \cdot \frac{d^2 u(x,t)}{dx^2}
\]

- Discretize time and space using **implicit** approach (Backward Euler) to approximate time derivative:
  \[
  \frac{u(x, t + \delta) - u(x, t)}{\delta} = C \cdot \frac{(u(x-h, t + \delta) - 2 \cdot u(x, t + \delta) + u(x+h, t + \delta))}{h^2}
  \]

- Let \( z = C \cdot \delta / h^2 \) and discretize variable \( x \) to \( j \cdot h \), \( t \) to \( i \cdot \delta \), and \( u(x, t) \) to \( u[j,i] \); solve for \( u \) at next time step:
  \[
  (I + z \cdot L) \cdot u[:, i+1] = u[:, i]
  \]

- \( I \) is identity and \( L \) is Laplacian

- Solve sparse linear system again
2D Implicit Method

- Similar to the 1D case, but the matrix $L$ is now

\[
L = \begin{pmatrix}
4 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 4 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 4 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 4 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 4 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & 4 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & 4 \\
\end{pmatrix}
\]

Graph and “5 point stencil”

- Multiplying by this matrix (as in the explicit case) is simply nearest neighbor computation on 2D mesh.
- To solve this system, there are several techniques.
## Algorithms for Solving $Ax=b$ (N vars)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Serial</th>
<th>PRAM</th>
<th>Memory</th>
<th>#Procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense LU</td>
<td>$N^3$</td>
<td>$N$</td>
<td>$N^2$</td>
<td>$N^2$</td>
</tr>
<tr>
<td>Band LU</td>
<td>$N^2$</td>
<td>$N$</td>
<td>$N^{3/2}$</td>
<td>$N$</td>
</tr>
<tr>
<td>Jacobi</td>
<td>$N^2$</td>
<td>$N$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>Explicit Inv.</td>
<td>$N^2$</td>
<td>$\log N$</td>
<td>$N^2$</td>
<td>$N^2$</td>
</tr>
<tr>
<td>Conj.Gradients</td>
<td>$N^{3/2}$</td>
<td>$N^{1/2} \times \log N$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>Red/Black SOR</td>
<td>$N^{3/2}$</td>
<td>$N^{1/2}$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>Sparse LU</td>
<td>$N^{3/2}$</td>
<td>$N^{1/2}$</td>
<td>$N \times \log N$</td>
<td>$N$</td>
</tr>
<tr>
<td>FFT</td>
<td>$N \times \log N$</td>
<td>$\log N$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>Multigrid</td>
<td>$N \log N$</td>
<td>$\log^2 N$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>Lower bound</td>
<td>$N$</td>
<td>$\log N$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
</tbody>
</table>

All entries in “Big-Oh” sense (constants omitted)

PRAM is an idealized parallel model with zero cost communication

## Algorithms for 2D (3D) Poisson Equation (N = n² (n³) vars)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Serial</th>
<th>PRAM</th>
<th>Memory</th>
<th>#Procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense LU</td>
<td>N³</td>
<td>N</td>
<td>N²</td>
<td>N²</td>
</tr>
<tr>
<td>Band LU</td>
<td>N² (N⁷/₃)</td>
<td>N</td>
<td>N³/₂ (N⁵/₃)</td>
<td>N(N⁴/₃)</td>
</tr>
<tr>
<td>Jacobi</td>
<td>N² (N⁵/₃)</td>
<td>N (N²/₃)</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Explicit Inv.</td>
<td>N²</td>
<td>log N</td>
<td>N²</td>
<td>N²</td>
</tr>
<tr>
<td>Conj. Gradients</td>
<td>N³/₂ (N⁴/₃)</td>
<td>N¹/₂ (1/₃) *log N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Red/Black SOR</td>
<td>N³/₂ (N⁴/₃)</td>
<td>N¹/₂ (N¹/₃)</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Sparse LU</td>
<td>N³/₂ (N²)</td>
<td>N¹/₂</td>
<td>N*log N(N⁴/₃)</td>
<td>N</td>
</tr>
<tr>
<td>FFT</td>
<td>N*log N</td>
<td>log N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Multigrid</td>
<td>N</td>
<td>log² N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lower bound</td>
<td>N</td>
<td>log N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

PRAM is an idealized parallel model with ∞ procs, zero cost communication.


For more information: take Ma221 this semester!
## Algorithms and Motifs

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Motifs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense LU</td>
<td>Dense linear algebra</td>
</tr>
<tr>
<td>Band LU</td>
<td>Dense linear algebra</td>
</tr>
<tr>
<td>Jacobi</td>
<td>(Un)structured meshes, Sparse Linear Algebra</td>
</tr>
<tr>
<td>Explicit Inv.</td>
<td>Dense linear algebra</td>
</tr>
<tr>
<td>Conj.Gradients</td>
<td>(Un)structured meshes, Sparse Linear Algebra</td>
</tr>
<tr>
<td>Red/Black SOR</td>
<td>(Un)structured meshes, Sparse Linear Algebra</td>
</tr>
<tr>
<td>Sparse LU</td>
<td>Sparse Linear Algebra</td>
</tr>
<tr>
<td>FFT</td>
<td>Spectral</td>
</tr>
<tr>
<td>Multigrid</td>
<td>(Un)structured meshes, Sparse Linear Algebra</td>
</tr>
</tbody>
</table>
Irregular mesh: NASA Airfoil in 2D

Mesh of airfoil

Pattern of sparse matrix $A$

Pattern of $A$ after LU
Source of Irregular Mesh:
Finite Element Model of Vertebra

Study failure modes of trabecular Bone under stress

Source: M. Adams, H. Bayraktar, T. Keaveny, P. Papadopoulos, A. Gupta
Methods: \( \mu \)FE modeling (Gordon Bell Prize, 2004)

Mechanical Testing

\[ E, \varepsilon_{\text{yield}}, \sigma_{\text{ult}}, \text{etc.} \]

Source: Mark Adams, PPPL

Micro-Computed Tomography

\( \mu \)CT @ 22 \( \mu \)m resolution

3D image

\( \mu \)FE mesh

2.5 mm cube

44 \( \mu \)m elements

Up to 537M unknowns

8/20/09

Jim Demmel

Sources: 44
Adaptive Mesh Refinement (AMR)

- Adaptive mesh around an explosion
  - Refinement done by estimating errors; refine mesh if too large
- Parallelism
  - Mostly between “patches,” assigned to processors for load balance
  - May exploit parallelism within a patch
- Projects:
  - Titanium (http://www.cs.berkeley.edu/projects/titanium)
  - Chombo (P. Colella, LBL), KeLP (S. Baden, UCSD), J. Bell, LBL
Summary: Some Common Problems

• Load Balancing
  - Dynamically - if load changes significantly during job
  - Statically - Graph partitioning
    » Discrete systems
    » Sparse matrix vector multiplication

• Linear algebra
  - Solving linear systems (sparse and dense)
  - Eigenvalue problems will use similar techniques

• Fast Particle Methods
  - $O(n \log n)$ instead of $O(n^2)$