CS 267
Source of Parallelism

Kathy Yelick

http://www.cs.berkeley.edu/~yelick/cs267
Parallelism and Locality in Simulation

• 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.

• Scientific models may introduce more parallelism:
  • When a continuous problem is discretized, temporal domain dependencies are generally limited to adjacent time steps.
  • Far-field effects may be ignored or approximated in many cases.

• Many problems exhibit parallelism at multiple levels
  • Example: circuits can be simulated at many levels, and within each there may be parallelism within and between subcircuits.
# 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>
Basic Kinds of Simulation

• Discrete event systems:
  • Examples: “Game of Life,” logic level circuit simulation.

• Particle systems:
  • Examples: billiard balls, semiconductor device simulation, galaxies.

• Lumped variables depending on continuous parameters:
  • ODEs, e.g., circuit simulation (Spice), structural mechanics, chemical kinetics.

• Continuous variables depending on continuous parameters:
  • PDEs, e.g., heat, elasticity, electrostatics.

• A given phenomenon can be modeled at multiple levels.
• Many simulations combine more than one of these techniques.
Outline

• Discrete event systems
  • Time and space are discrete

• Particle systems
  • Important special case of lumped systems
  • Previous lecture

• Ordinary Differential Equations (ODEs)
  • Lumped systems
  • Location/entities are discrete, time is continuous

• Partial Different Equations (PDEs)
  • Time and space are continuous
  • Next lecture
Review on Particle Systems

• In particle systems there are
  • External forces are trivial to parallelize
  • Near-field forces can be done with limited communication
  • Far-field are hardest (require a lot of communication)
    • $O(n^2)$ algorithms require that all particles “talk to” all others
    • Expensive in computation on a serial machine
    • Also expensive in communication on a parallel one
• Clever algorithms and data structures can help
  • Particle mesh methods
  • Tree-based methods
Discrete Event Systems
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:”
  • Also known as Sharks and Fish #3:
    http://www.cs.berkeley.edu/~yelick/cs267/SharksAndFish
  • Space divided into cells, rules govern cell contents at each step
Parallelism in Sharks and Fish (Recap)

• 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

\[
\begin{array}{ccc}
P1 & P2 & P3 \\
P4 & P5 & P6 \\
P7 & P8 & P9 \\
\end{array}
\]

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.
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 (NP-complete)
  - Determines parallelism and locality.
  - Attempts to evenly distribute subgraphs to nodes (load balance).
  - Attempts to minimize edge crossing (minimize communication).

![Synchronous Circuit Simulation Diagram]

edge crossings = 6

edge crossings = 10
Asynchronous Simulation

• Synchronous simulations may waste time:
  • Simulate even when the inputs do not change.

• Asynchronous 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
  • In MPI, 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.
  • May need deadlock detection if there are cycles in graph, or else “null messages”.
  • 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.
  • 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.
Summary of Discrete Even Simulations

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

• Approach
  • 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
      – Speculative scheduling – assume no inputs, roll back if necessary
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
Circuit Example

- State of the system is represented by
  - $v_n(t)$ node voltages
  - $i_b(t)$ branch currents
  - $v_b(t)$ branch voltages

  All at time $t$

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

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

- Write as single large system of ODEs (possibly with constraints).
Structural Analysis Example

- 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 is looking at this section of 880, among others
Solving ODEs

• 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.

• 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
Solving ODEs: Explicit Methods

• Assume ODE is \( x'(t) = f(x) = A^*x \), where \( A \) is a sparse matrix
  
  • Compute \( x(i*dt) = x[i] \)
    
    at \( i=0,1,2,... \)
  
  • Approximate \( x'(i*dt) \)
    
    \( x[i+1] = x[i] + dt \times \text{slope} \)

  Use slope at \( x[i] \)

• Explicit methods, e.g., (Forward) Euler’s method.
  
  • Approximate \( x'(t) = A^*x \) by \( \frac{x[i+1] - x[i]}{dt} = A^*x[i] \).
  
  • \( x[i+1] = x[i] + dt \times A^*x[i] \), i.e. sparse matrix-vector multiplication.

• Tradeoffs:
  
  • Simple algorithm: sparse matrix vector multiply.
  
  • Stability problems: May need to take very small time steps, especially if system is stiff (i.e. can change rapidly).
Solving ODEs: Implicit Methods

• Assume ODE is \( x'(t) = f(x) = A \cdot x \), where A is a sparse matrix
  • Compute \( x(i \cdot dt) = x[i] \)
    at \( i = 0, 1, 2, \ldots \)
  • Approximate \( x'(i \cdot dt) \)
    \[ x[i+1] = x[i] + dt \cdot \text{slope} \]

Use slope at \( x[i+1] \)

• Implicit method, e.g., Backward Euler solve:
  • Approximate \( x'(t) = A \cdot x \) by \( (x[i+1] - x[i]) / dt = A \cdot x[i+1] \).
  • \( (I - dt \cdot A) \cdot x[i+1] = x[i] \), i.e. we need to solve a sparse linear system of equations.

• Trade-offs:
  • Larger timestep possible: especially for stiff problems
  • More difficult algorithm: need to do a sparse solve at each step
Solving ODEs: Eigensolvers

• Computing modes of vibration: finding eigenvalues and eigenvectors.
  • Seek solution of $x''(t) = A\cdot x$ of form $x(t) = \sin(f\cdot t)\cdot x_0$, where $x_0$ is a constant vector.
  • Plug in to get $-f^2 \cdot x_0 = A\cdot x_0$, so that $-f^2$ is an eigenvalue and $x_0$ is an eigenvector of $A$.
  • Solution schemes reduce either to sparse-matrix multiplication, or solving sparse linear systems.
ODEs and Sparse Matrices

• All these reduce to sparse matrix problems
  • Explicit: sparse matrix-vector multiplication.
  • Implicit: solve a sparse linear system
    • direct solvers (Gaussian elimination).
    • iterative solvers (use sparse matrix-vector multiplication).
  • Eigenvalue/vector algorithms may also be explicit or implicit.
Parallel Sparse Matrix-vector multiplication

- \( y = A \times x \), where \( A \) is a sparse \( n \times n \) matrix

Questions

- which processors store \( y[i], x[i], \) and \( A[i,j] \)
- which processors compute \( y[i] = \sum_{j=1}^{n} A[i,j] \times x[j] \)
  \( = \text{(row i of } A) \times x \) \hspace{1cm} \cdots \text{a sparse dot product}

Partitioning

- Partition index set \( \{1, \ldots, n\} = N_1 + N_2 + \ldots + N_p \).
- For all \( i \) in \( N_k \), Processor \( k \) stores \( y[i], x[i], \) and row \( i \) of \( A \)
- For all \( i \) in \( N_k \), Processor \( k \) computes \( y[i] = (\text{row } i \text{ of } A) \times x \)
  \( \text{“owner computes” rule: Processor } k \text{ compute the } y[i]s \text{ it owns.} \)
Matrix Reordering via Graph Partitioning

• “Ideal” matrix structure for parallelism: block diagonal
  • $p$ (number of processors) blocks, can all be computed locally.
  • few non-zeros outside these blocks, which require communication.

• Can we reorder the rows/columns to achieve this?
Goals of Reordering

• Performance goals
  • balance load (how is load measured?).
  • balance storage (how much does each processor store?).
  • minimize communication (how much is communicated?).

• Some algorithms reorder for other reasons
  • Reduce # nonzeros in answer (fill)
  • Improve numerical properties
Graph Partitioning and Sparse Matrices

• Relationship between matrix and graph

- A “good” partition of the graph has
  - equal (weighted) number of nodes in each part (load and storage balance).
  - minimum number of edges crossing between (minimize communication).
- Reorder the rows/columns by putting all nodes in one partition together.
Implicit Methods and Eigenproblems

- Direct methods (Gaussian elimination)
  - Called LU Decomposition, because we factor $A = L^*U$.
  - Future lectures will consider both dense and sparse cases.
  - More complicated than sparse-matrix vector multiplication.

- Iterative solvers
  - Will discuss several of these in future.
    - Jacobi, Successive over-relaxation (SOR), Conjugate Gradient (CG), Multigrid,
  - Most have sparse-matrix-vector multiplication in kernel.

- Eigenproblems
  - Future lectures will discuss dense and sparse cases.
  - Also depend on sparse-matrix-vector multiplication, direct methods.
Partial Differential Equations
PDEs
Continuous Variables, Continuous Parameters

Examples of such systems include

- **Parabolic (time-dependent) problems:**
  - Heat flow: Temperature(position, time)
  - Diffusion: Concentration(position, time)

- **Elliptic (steady state) problems:**
  - Electrostatic or Gravitational Potential: Potential(position)

- **Hyperbolic problems (waves):**
  - Quantum mechanics: Wave-function(position,time)

Many problems combine features of above

- **Fluid flow:** Velocity, Pressure, Density(position, time)
- **Elasticity:** Stress, Strain(position, time)
Terminology

• Term hyperbolic, parabolic, elliptic, come from special cases of the general form of a second order linear PDE

\[ a \frac{d^2u}{dx} + b \frac{d^2u}{dxdy} + c \frac{d^2u}{dy^2} + d \frac{du}{dx} + e \frac{du}{dy} + f = 0 \]

where y is time

• Analog to solutions of general quadratic equation

\[ a x^2 + b xy + c y^2 + d x + e y + f \]
Example: Deriving the Heat Equation

Consider a simple problem
- A bar of uniform material, insulated except at ends
- Let $u(x,t)$ be the temperature at position $x$ at time $t$
- Heat travels from $x-h$ to $x+h$ at rate proportional to:

$$\frac{d u(x,t)}{dt} = C \cdot \frac{\left(u(x-h,t)-u(x,t)\right)/h - \left(u(x,t)- u(x+h,t)\right)/h}{h}$$

- As $h \rightarrow 0$, we get the heat equation:

$$\frac{d u(x,t)}{dt} = C \cdot \frac{d^2 u(x,t)}{dx^2}$$
Details of the Explicit Method for Heat

- From experimentation (physical observation) we have:
  \[ \frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2} \quad \text{(assume } C = 1 \text{ for simplicity)} \]

- Discretize time and space and use explicit approach (as described for ODEs) to approximate derivative:
  \[ \frac{u(x,t+1) - u(x,t)}{dt} = \frac{u(x-h,t) - 2u(x,t) + u(x+h,t)}{h^2} \]
  \[ u(x,t+1) = u(x,t) + \frac{dt}{h^2} \left( u(x-h,t) - 2u(x,t) + u(x+h,t) \right) \]

- Let \( z = \frac{dt}{h^2} \)
  \[ u(x,t+1) = z \cdot u(x-h,t) + (1-2z) \cdot u(x,t) + z \cdot u(x+h,t) \]

- By changing variables (\( x \) to \( j \) and \( y \) to \( i \)):
  \[ u[j,i+1] = z \cdot u[j-1,i] + (1-2z) \cdot u[j,i] + z \cdot u[j+1,i] \]
Explicit Solution of the Heat Equation

- Use finite differences with $u[j,i]$ as the heat at
  - time $t = i \times dt$ ($i = 0, 1, 2, \ldots$) and position $x = j \times h$ ($j = 0, 1, \ldots, N = 1/h$)
  - initial conditions on $u[j,0]$
  - boundary conditions on $u[0,i]$ and $u[N,i]$

- At each timestep $i = 0, 1, 2, \ldots$

For $j = 0$ to $N$

$$u[j,i+1] = z \times u[j-1,i] + (1-2z) \times u[j,i] + z \times u[j+1,i]$$

where $z = dt/h^2$

- This corresponds to
  - matrix vector multiply
  - nearest neighbors on grid
Matrix View of Explicit Method for Heat

- Multiplying by a tridiagonal matrix at each step

\[
T = \begin{pmatrix}
1-2z & z \\
z & 1-2z & z \\
z & 1-2z & z \\
z & 1-2z
\end{pmatrix}
\]

Graph and “3 point stencil”

- For a 2D mesh (5 point stencil) the matrix is pentadiagonal
  - More on the matrix/grid views later
Parallelism in Explicit Method for PDEs

- Partitioning the space \((x)\) into \(p\) largest chunks
  - good load balance (assuming large number of points relative to \(p\))
  - minimized communication (only \(p\) chunks)

- Generalizes to
  - multiple dimensions.
  - arbitrary graphs (= arbitrary sparse matrices).

- Explicit approach often used for hyperbolic equations
- Problem with explicit approach for heat (parabolic):
  - numerical instability.
  - solution blows up eventually if \(z = dt/h^2 > .5\)
  - need to make the time steps very small when \(h\) is small: \(dt < .5*h^2\)
Instability in Solving the Heat Equation Explicitly

Explicit Solution of Heat equation, $z=0.42$

Explicit Solution of Heat equation, $z=0.58$
Implicit Solution of the Heat Equation

• From experimentation (physical observation) we have:
  \[ \frac{\delta u(x,t)}{\delta t} = \delta^2 u(x,t)/\delta x \]  
  (assume C = 1 for simplicity)

• Discretize time and space and use implicit approach (backward Euler) to approximate derivative:
  \[ \frac{(u(x,t+1) - u(x,t))}{dt} = \frac{(u(x-h,t+1) - 2*u(x,t+1) + u(x+h,t+1))}{h^2} \]
  \[ u(x,t) = u(x,t+1) + \frac{dt}{h^2} * (u(x-h,t+1) - 2*u(x,t+1) + u(x+h,t+1)) \]

• Let \( z = \frac{dt}{h^2} \) and change variables (t to j and x to i)
  \[ u(:,i) = (I - z *L)* u(:, i+1) \]

• Where I is identity and
  \[ L = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \]
Implicit Solution of the Heat Equation

• The previous slide used Backwards Euler, but using the trapezoidal rule gives better numerical properties.
• This turns into solving the following equation:
  \[(I + (z/2)*L) \cdot u[:,i+1] = (I - (z/2)*L) \cdot u[:,i]\]
• Again \(I\) is the identity matrix and \(L\) is:

\[
L = \begin{pmatrix}
    2 & -1 & & & \\
    -1 & 2 & -1 & & \\
    & -1 & 2 & -1 & \\
    & & -1 & 2 & -1 \\
    & & & -1 & 2
\end{pmatrix}
\]

• This is essentially solving Poisson’s equation in 1D
2D Implicit Method

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

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

• Multiplying by this matrix (as in the explicit case) is simply nearest neighbor computation on 2D grid.

• To solve this system, there are several techniques.

Graph and “5 point stencil”

3D case is analogous (7 point stencil)
Relation of Poisson to Gravity, Electrostatics

- Poisson equation arises in many problems
- E.g., force on particle at \((x,y,z)\) due to particle at 0 is
  \[-(x,y,z)/r^3, \text{ where } r = \sqrt{x^2 + y^2 + z^2}\]
- Force is also gradient of potential \(V = -1/r\)
  \[-(d/dx V, d/dy V, d/dz V) = -\nabla V\]
- \(V\) satisfies Poisson’s equation (try working this out!)

![Diagram](image.png)
### Algorithms for 2D Poisson Equation (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</td>
</tr>
<tr>
<td>Conj.Grad.</td>
<td>N^(3/2)</td>
<td>N^(1/2)*log N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>RB 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*log N</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>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>

PRAM is an idealized parallel model with zero cost communication
Overview of Algorithms

- Sorted in two orders (roughly):
  - from slowest to fastest on sequential machines.
  - from most general (works on any matrix) to most specialized (works on matrices “like” T).
- Dense LU: Gaussian elimination; works on any N-by-N matrix.
- Band LU: Exploits the fact that T is nonzero only on sqrt(N) diagonals nearest main diagonal.
- Jacobi: Essentially does matrix-vector multiply by T in inner loop of iterative algorithm.
- Explicit Inverse: Assume we want to solve many systems with T, so we can precompute and store inv(T) “for free”, and just multiply by it (but still expensive).
- Conjugate Gradient: Uses matrix-vector multiplication, like Jacobi, but exploits mathematical properties of T that Jacobi does not.
- Red-Black SOR (successive over-relaxation): Variation of Jacobi that exploits yet different mathematical properties of T. Used in multigrid schemes.
- LU: Gaussian elimination exploiting particular zero structure of T.
- FFT (fast Fourier transform): Works only on matrices very like T.
- Multigrid: Also works on matrices like T, that come from elliptic PDEs.
- Lower Bound: Serial (time to print answer); parallel (time to combine N inputs).
- Details in class notes and www.cs.berkeley.edu/~demmel/ma221.
Mflop/s Versus Run Time in Practice

- Problem: Iterative solver for a convection-diffusion problem; run on a 1024-CPU NCUBE-2.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Flops</th>
<th>CPU Time</th>
<th>Mflop/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi</td>
<td>$3.82 \times 10^{12}$</td>
<td>2124</td>
<td>1800</td>
</tr>
<tr>
<td>Gauss-Seidel</td>
<td>$1.21 \times 10^{12}$</td>
<td>885</td>
<td>1365</td>
</tr>
<tr>
<td>Least Squares</td>
<td>$2.59 \times 10^{11}$</td>
<td>185</td>
<td>1400</td>
</tr>
<tr>
<td>Multigrid</td>
<td>$2.13 \times 10^9$</td>
<td>7</td>
<td>318</td>
</tr>
</tbody>
</table>

- Which solver would you select?
Summary of Approaches to Solving PDEs

• As with ODEs, either explicit or implicit approaches are possible
  • Explicit, sparse matrix-vector multiplication
  • Implicit, sparse matrix solve at each step
    • Direct solvers are hard (more on this later)
    • Iterative solves turn into sparse matrix-vector multiplication

• Grid and sparse matrix correspondence:
  • Sparse matrix-vector multiplication is nearest neighbor “averaging” on the underlying mesh

• Not all nearest neighbor computations have the same efficiency
  • Factors are the mesh structure (nonzero structure) and the number of Flops per point.
Comments on practical meshes

• Regular 1D, 2D, 3D meshes
  • Important as building blocks for more complicated meshes
• Practical meshes are often irregular
  • Composite meshes, consisting of multiple “bent” regular meshes joined at edges
  • Unstructured meshes, with arbitrary mesh points and connectivities
  • Adaptive meshes, which change resolution during solution process to put computational effort where needed
Parallelism in Regular meshes

• Computing a Stencil on a regular mesh
  • need to communicate mesh points near boundary to neighboring processors.
    • Often done with ghost regions
  • Surface-to-volume ratio keeps communication down, but
    • Still may be problematic in practice

Implemented using “ghost” regions.
Adds memory overhead
Adaptive Mesh Refinement (AMR)

- Adaptive mesh around an explosion
  - Refinement done by calculating errors
- Parallelism
  - Mostly between “patches,” dealt to processors for load balance
  - May exploit some within a patch (SMP)
- Projects:
  - Titanium ([http://www.cs.berkeley.edu/projects/titanium](http://www.cs.berkeley.edu/projects/titanium))
  - Chombo (P. Colella, LBL), KeLP (S. Baden, UCSD), J. Bell, LBL
Adaptive Mesh

Shock waves in a gas dynamics using AMR (Adaptive Mesh Refinement)
See: http://www.llnl.gov/CASC/SAMRAI/

3/1/2004 CS267 Lecture 10
Composite Mesh from a Mechanical Structure
Converting the Mesh to a Matrix
Effects of Reordering on Gaussian Elimination
Irregular mesh: NASA Airfoil in 2D
Irregular mesh: Tapered Tube (Multigrid)

Example of Prometheus meshes

Figure 6. Sample input grid and coarse grids
Challenges of Irregular Meshes

• How to generate them in the first place
  • Triangle, a 2D mesh partitioner by Jonathan Shewchuk
  • 3D harder!
• How to partition them
  • ParMetis, a parallel graph partitioner
• How to design iterative solvers
  • PETSc, a Portable Extensible Toolkit for Scientific Computing
  • Prometheus, a multigrid solver for finite element problems on irregular meshes
• How to design direct solvers
  • SuperLU, parallel sparse Gaussian elimination

• These are challenges to do sequentially, more so in parallel
CS267 Final Projects

• Project proposal
  • Teams of 3 students, typically across departments
  • Interesting parallel application or system
  • Conference-quality paper
  • High performance is key:
    • Understanding performance, tuning, scaling, etc.
    • More important the difficulty of problem

• Leverage
  • Projects in other classes (but discuss with me first)
  • Research projects
Project Ideas

• Applications
  • Implement existing sequential or shared memory program on distributed memory
  • Investigate SMP trade-offs (using only MPI versus MPI and thread based parallelism)

• Tools and Systems
  • Effects of reordering on sparse matrix factoring and solves

• Numerical algorithms
  • Improved solver for immersed boundary method
  • Use of multiple vectors (blocked algorithms) in iterative solvers
Project Ideas

• Novel computational platforms
  • Exploiting hierarchy of SMP-clusters in benchmarks
  • Computing aggregate operations on ad hoc networks (Culler)
  • Push/explore limits of computing on “the grid”
  • Performance under failures
• Detailed benchmarking and performance analysis, including identification of optimization opportunities
  • Titanium
  • UPC
  • IBM SP (Blue Horizon)