CS 267
Sources of Parallelism and Locality in Simulation

Kathy Yelick

www.cs.berkeley.edu/~yelick/cs267_sp07
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, time 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.
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.
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
• Ordinary Differential Equations (ODEs)
  • Lumped systems
  • Location/entities are discrete, time is continuous
• Partial Differential Equations (PDEs)
  • Time and space are continuous
  • Next lecture

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

• Illustration of parallel programming
  • Original version (discrete event only) proposed by Geoffrey Fox
  • Called WATOR
• 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 problems (S&F1 - S&F6)
  • Different sets of rules, to illustrate different phenomena
• Available in many languages (see class web page)
  • Matlab, pThreads, MPI, OpenMP, Split-C, Titanium, CMF, CMMD, pSather (not all problems in all languages)
  • See http://www.cs.berkeley.edu/~demmel/cs267/Sharks_and_Fish
Sharks and Fish

• **S&F 1.** Fish alone move continuously subject to an external current and Newton's laws.
• **S&F 2.** Fish alone move continuously subject to gravitational attraction and Newton's laws.
• **S&F 3.** Fish alone play the "Game of Life" on a square grid.
• **S&F 4.** Fish alone move randomly on a square grid, with at most one fish per grid point.
• **S&F 5.** Sharks and Fish both move randomly on a square grid, with at most one fish or shark per grid point, including rules for fish attracting sharks, eating, breeding and dying.
• **S&F 6.** Like Sharks and Fish 5, but continuous, subject to Newton's laws.
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:
  - Space divided into cells, rules govern cell contents at each step
Parallelism in Game of Life (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

```
P1  P2  P3
P4  P5  P6
P7  P8  P9
```

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 (eg Game of Life)

• Suppose graph is nxn mesh with connection NSEW neighbors
• Which partition has less communication?
• 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.
  - Attempts to evenly distribute subgraphs to nodes (load balance).
  - Attempts to minimize edge crossing (minimize communication).
  - Easy for meshes, NP-hard in general

edge crossings = 6

edge crossings = 10
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 Simulation

- 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
  - 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.
  • Need deadlock detection if there are cycles in graph
  • 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.
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
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

• 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 Wals forces in fluid \((1/r^6)\)

• Far-field force
  • fish attract other fish by gravity-like \((1/r^2)\) force (S&F 2)
  • gravity, electrostatics, radiosity
  • 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: integrate using Euler's method with varying step size

Initialize time step, iteration count, and array of times
dt = .01; i = 0; time=[]; t = 0;

loop over time steps
while t < tfinal,
  t = t + dt;
  i = i + 1;
  fishp = fishp + dt*fishv;
  accel = current(fishp)./fishm;  % current function is circular force
  fishv = fishv + dt*accel;

update time step
  dt = min(.1*max(abs(fishv))/max(abs(accel)),1);
end
Parallelism in External Forces

• These are the simplest
• The force on each particle is independent
• Called “embarrassingly parallel”

• Evenly distribute particles on processors
  • Any distribution works
  • Locality is not an issue, no communication
• For each particle on processor, apply the external force
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 decomposition of physical domain:
  - $O(n/p)$ 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
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: Particle-Mesh Methods

- Based on approximation:
  - Superimpose a regular mesh.
  - "Move" particles to nearest grid point.
- Exploit fact that the far-field force satisfies a PDE that is easy to solve on a regular mesh:
  - FFT, multigrid (described in future lectures)
- Accuracy depends on the fineness of the grid is and the uniformity of the particle distribution.

1) Particles are moved to mesh (scatter)
2) Solve mesh problem
3) Forces are interpolated at particles (gather)
Far-field forces: Tree Decomposition

- Based on approximation.
  - Forces from group of far-away particles “simplified” -- resembles a single large particle.
  - Use tree; each node contains an approximation of descendants.

- $O(n \log n)$ or $O(n)$ instead of $O(n^2)$.

- Several Algorithms
  - Barnes-Hut.
  - Fast multipole method (FMM) of Greengard/Rohklin.
  - Anderson’s method.

- Discussed in later lecture.
Summary of Particle Methods

- Model contains discrete entities, namely, particles
- Time is continuous – is discretized to solve

- Simulation follows particles through timesteps
  - All-pairs algorithm is simple, but inefficient, $O(n^2)$
  - Particle-mesh methods approximates by moving particles
  - Tree-based algorithms approximate by treating set of particles as a group, when far away

- May think of this as a special case of a “lumped” system
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 all at time \( t \)
  - \( v_b(t) \) branch voltages

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

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

- \( A \) is sparse matrix, representing connections in circuit
- 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 looks 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.
  • 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
Solving ODEs: Explicit Methods

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

• Explicit methods, e.g., (Forward) Euler’s method.
  • Approximate \( x'(t)=A*x(t) \) by \( (x[i+1] - x[i]) / \Delta t = A*x[i] \).
  • \( x[i+1] = x[i] + \Delta t A x[i] \), 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. \( A \) has some large entries, so \( x \) can change rapidly).
Solving ODEs: Implicit Methods

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

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

• Implicit method, e.g., Backward Euler solve:
  • Approximate \( x'(t)=A\times x(t) \) by \( (x[i+1] - x[i]) / dt = A\times x[i+1] \).
    \( (I - dt\times A)\times x[i+1] = x[i], \) 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 $x''(t) = A\times x(t)$ of form $x(t) = \sin(\omega t) \times x_0$, where $x_0$ is a constant vector.
  • Plug in to get $-\omega^2 \times x_0 = A\times x_0$, so that $-\omega^2$ 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 problems reduce to sparse matrix problems
  - Explicit: sparse matrix-vector multiplication (SpMV).
  - 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.
Key facts about SpMV

• y = y + A*x for sparse matrix A

• Choice of data structure for A
  • Compressed sparse row (CSR): popular for general A on cache-based microprocessors
    • Automatic tuning a good idea (bebop.cs.berkeley.edu)
  • CSR: 3 arrays:
    • col_index: Column index of each nonzero value
    • values: Nonzero values
    • row_ptr: For each row, the index into the col_index/values array

for i = 1:m
    start = row_ptr(i);
    end   = row_ptr(i + 1);
    for j = start : end - 1
        y(i) = y(i) + values(j) * x(col_index(j));
Parallel Sparse Matrix-vector multiplication

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

**Questions**

- which processors store
  - \( y[i] \), \( x[i] \), and \( A[i,j] \)

- which processors compute
  - \( y[i] = \text{sum (from 1 to n) } A[i,j] \times x[j] \)
  - \( = \text{ (row i of } A \text{) } \times x \) … a sparse dot product

**Partitioning**

- Partition index set \( \{1, \ldots, n\} = N_1 \cup N_2 \cup \ldots \cup N_p \).
- 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 \( y[i] = \text{(row } i \text{ of } A \text{) } \times x \)
  - "owner computes" rule: Processor \( k \) compute the \( y[i] \)s it owns.

May require communication
Matrix Reordering via Graph Partitioning

- “Ideal” matrix structure for parallelism: block diagonal
  - $p$ (number of processors) blocks, can all be computed locally.
  - If no non-zeros outside these blocks, no communication needed

- Can we reorder the rows/columns to get close to this?
  - Most nonzeros in diagonal blocks, few outside

\[
\begin{array}{c|c|c|c|c}
\hline
P0 & P1 & P2 & P3 & P4 \\
\hline
P0 & & & & \\
P1 & & & & \\
P2 & & & & \\
P3 & & & & \\
\hline
\end{array}
\]
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?).

• Other algorithms reorder for other reasons
  • Reduce # nonzeros in matrix after Gaussian elimination
  • Improve numerical stability
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; Eigenproblems

• Implicit methods for ODEs solve linear systems
• 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.
Summary: 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 $O(n^2)$
Extra Slides
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

\[ a*d^2u/dx + b*d^2u/dxdy + c*d^2u/dy^2 + d*du/dx + e*du/dy + f = 0 \]

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:

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

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

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

- 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 explicit approach (as described for ODEs) to approximate derivative:
  \[ \frac{(u(x,t+1) - u(x,t))}{dt} = \frac{(u(x-h,t) - 2*u(x,t) + u(x+h,t))}{h^2} \]
  \[ u(x,t+1) = u(x,t) + \frac{dt}{h^2} * (u(x-h,t) - 2*u(x,t) + u(x+h,t)) \]

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

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

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

  where $z = \frac{dt}{h^2}$

- 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

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

- 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
Implicit Solution of the Heat Equation

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

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

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

• Where I is identity and
  \[ L = \begin{pmatrix} 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix} \]
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:

\[(I + (\frac{z}{2})L) \ast u[:,i+1] = (I - (\frac{z}{2})L) \ast u[:,i]\]

• 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

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

Graph and “5 point stencil”

- Multiplying by this matrix (as in the explicit case) is simply nearest neighbor computation on 2D grid.
- To solve this system, there are several techniques.

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 \(V = -1/r\)
  \[-=(d/dx V, d/dy V, d/dz V) = -\text{grad } V\]
- \(V\) satisfies Poisson’s equation (try working this out!)
## 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>$N^{2/3}$</td>
<td>log $N$</td>
<td>$N^2$</td>
<td>$N^2$</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)
  - 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/
Composite Mesh from a Mechanical Structure
Converting the Mesh to a Matrix

Mesh numbered in natural order

Matrix $A$, in natural order

nz = 256
Effects of Reordering on Gaussian Elimination

- A in natural order
- Cholesky factor, flops=296923

- A after minimum degree
- Cholesky factor, flops=196236
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
Extra Slides
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)
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.
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
Regular Meshes (eg Sharks and Fish)

• Suppose graph is 2D mesh with connection NSEW nbrs
• Which partition is better?