

CS 267 Source of Parallelism

Kathy Yelick

<http://www.cs.berkeley.edu/~yelick/cs267>

3/1/2004

CS267 Lecture 10

1

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.

3/1/2004

CS267 Lecture 10

2

Example: Circuit Simulation

- Circuits are simulated at many different levels

Level	Primitives	Examples
Instruction level	Instructions	SimOS, SPIM
Cycle level	Functional units	↑ ↓ VIRAM-p
Register Transfer Level (RTL)	Register, counter, MUX	VHDL
Gate Level	Gate, flip-flop, memory cell	↓ Thor
Switch level	Ideal transistor	Cosmos
Circuit level	Resistors, capacitors, etc.	Spice
Device level	Electrons, silicon	

3/1/2004

CS267 Lecture 10

3

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.

3/1/2004

CS267 Lecture 10

4

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



3/1/2004

CS267 Lecture 10

5

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

3/1/2004

CS267 Lecture 10

6

Discrete Event Systems

3/1/2004

CS267 Lecture 10

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:"
 - 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

3/1/2004

CS267 Lecture 10

8

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

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.

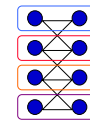
3/1/2004

CS267 Lecture 10

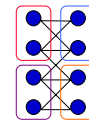
9

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).



edge crossings = 6



edge crossings = 10

3/1/2004

CS267 Lecture 10

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"?

3/1/2004

CS267 Lecture 10

11

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.

3/1/2004

CS267 Lecture 10

12

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

3/1/2004

CS267 Lecture 10

13

Lumped Systems ODEs

3/1/2004

CS267 Lecture 10

14

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

3/1/2004

CS267 Lecture 10

15

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

$$\begin{pmatrix} 0 & A & 0 \\ A^T & 0 & -I \\ 0 & R & -I \\ 0 & -I & C^*d/dt \\ 0 & L^*d/dt & I \end{pmatrix} * \begin{pmatrix} v_n \\ i_b \\ v_b \end{pmatrix} = \begin{pmatrix} 0 \\ S \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

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

3/1/2004

CS267 Lecture 10

16

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 this section of 880, among others

3/1/2004

CS267 Lecture 10

17

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

3/1/2004

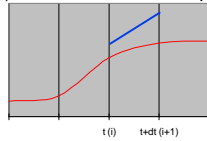
CS267 Lecture 10

18

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 $i=0,1,2,\dots$
- Approximate $x'(i*dt)$
 $x[i+1]=x[i] + dt*slope$



Use slope at $x[i]$

- Explicit methods, e.g., (Forward) Euler's method.
 - Approximate $x'(t)=A^*x$ by $(x[i+1] - x[i]) / dt = A^*x[i]$.
 - $x[i+1] = x[i] + dt * A^*x[i]$, 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).

3/1/2004

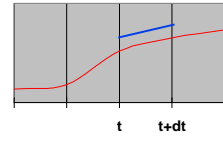
CS267 Lecture 10

19

Solving ODEs: Implicit Methods

- Assume ODE is $x'(t) = f(x) = A^*x$, where A is a sparse matrix

- Compute $x(i*dt) = x[i]$ at $i=0,1,2,\dots$
- Approximate $x'(i*dt)$
 $x[i+1]=x[i] + dt*slope$



Use slope at $x[i+1]$

- Implicit method, e.g., Backward Euler solve:
 - Approximate $x'(t)=A^*x$ by $(x[i+1] - x[i]) / dt = A^*x[i+1]$.
 - $(I - dt^*A)^*x[i+1] = x[i]$, 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

3/1/2004

CS267 Lecture 10

20

Solving ODEs: Eigensolvers

- Computing modes of vibration: finding eigenvalues and eigenvectors.
 - Seek solution of $x''(t) = A^*x$ of form $x(t) = \sin(f^*t)^*x_0$, where x_0 is a constant vector.
 - Plug in to get $-f^2 * x_0 = A^*x_0$, 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.

3/1/2004

CS267 Lecture 10

21

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.

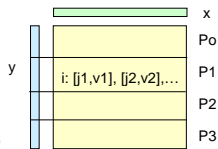
3/1/2004

CS267 Lecture 10

22

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
 - $y[i] = \text{sum (from 1 to n) } A[i,j] * x[j]$
 - $= (\text{row } i \text{ of } A) * x$... a sparse dot product

- Partitioning

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

Most problematic

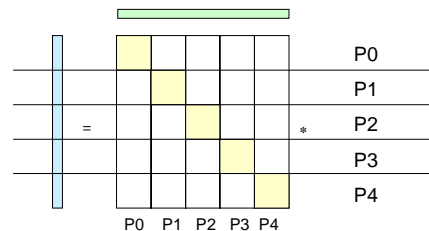
3/1/2004

CS267 Lecture 10

23

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?



3/1/2004

CS267 Lecture 10

24

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

3/1/2004

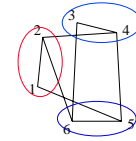
CS267 Lecture 10

25

Graph Partitioning and Sparse Matrices

- Relationship between matrix and graph

	1	2	3	4	5	6
1	1	1			1	
2	1	1		1		1
3			1	1		1
4			1	1	1	1
5	1				1	1
6					1	1



- 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.

3/1/2004

CS267 Lecture 10

26

Implicit Methods and Eigenproblems

- Direct methods (Gaussian elimination)
 - Called LU Decomposition, because we factor $A = L \cdot U$.
 - 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.

3/1/2004

CS267 Lecture 10

27

Partial Differential Equations PDEs

3/1/2004

CS267 Lecture 10

28

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)

3/1/2004

CS267 Lecture 10

29

Terminology

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

$$a \cdot d^2u/dx^2 + b \cdot d^2u/dxdy + c \cdot d^2u/dy^2 + d \cdot du/dx + e \cdot du/dy + f = 0$$

where y is time

- Analog to solutions of general quadratic equation

$$a \cdot x^2 + b \cdot xy + c \cdot y^2 + d \cdot x + e \cdot y + f$$

Backup slide: currently hidden.

3/1/2004

CS267 Lecture 10

30

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

$$\frac{d u(x,t)}{dt} = C * \frac{d^2 u(x,t)}{dx^2}$$

3/1/2004

CS267 Lecture 10

31

Details of the Explicit Method for Heat

- From experimentation (physical observation) we have:

$$\delta u(x,t) / \delta t = \delta^2 u(x,t) / \delta x^2 \quad (\text{assume } C = 1 \text{ for simplicity})$$

- Discretize time and space and use explicit approach (as described for ODEs) to approximate derivative:

$$(u(x,t+1) - u(x,t)) / dt = (u(x-h,t) - 2 * u(x,t) + u(x+h,t)) / h^2$$

$$u(x,t+1) - u(x,t) = dt/h^2 * (u(x-h,t) - 2 * u(x,t) + u(x+h,t))$$

$$u(x,t+1) = u(x,t) + dt/h^2 * (u(x-h,t) - 2 * u(x,t) + u(x+h,t))$$

- Let $z = dt/h^2$

$$u(x,t+1) = z * u(x-h,t) + (1-2z) * u(x,t) + z * u(x+h,t)$$

- By changing variables (x to j and y to i):

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

3/1/2004

CS267 Lecture 10

32

Explicit Solution of the Heat Equation

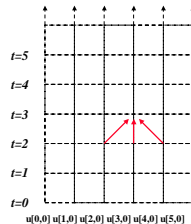
- Use finite differences with $u[j,i]$ as the heat at
 - time $t = i * dt$ ($i = 0, 1, 2, \dots$) and position $x = j * h$ ($j = 0, 1, \dots, N = 1/h$)
 - initial conditions on $u[j,0]$
 - boundary conditions on $u[0,i]$ and $u[N,i]$
- At each timestep $i = 0, 1, 2, \dots$

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 $z = dt/h^2$

- This corresponds to
 - matrix vector multiply
 - nearest neighbors on grid



3/1/2004

CS267 Lecture 10

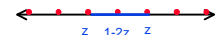
33

Matrix View of Explicit Method for Heat

- Multiplying by a tridiagonal matrix at each step

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

Graph and "3 point stencil"



- For a 2D mesh (5 point stencil) the matrix is pentadiagonal
 - More on the matrix/grid views later

3/1/2004

CS267 Lecture 10

34

Parallelism in Explicit Method for PDEs

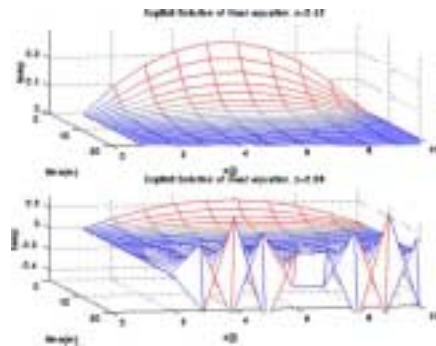
- Partitioning the space (x) into p largest chunks
 - good load balance (assuming large number of points relative to p)
 - minimized communication (only p chunks)
- Generalizes to
 - multiple dimensions.
 - arbitrary graphs (= arbitrary sparse matrices).
- Explicit approach often used for hyperbolic 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: $dt < .5 * h^2$

3/1/2004

CS267 Lecture 10

35

Instability in Solving the Heat Equation Explicitly



3/1/2004

CS267 Lecture 10

36

Implicit Solution of the Heat Equation

- From experimentation (physical observation) we have:
 $\delta u(x,t) / \delta t = \delta^2 u(x,t) / \delta x^2$ (assume $C = 1$ for simplicity)

- Discretize time and space and use implicit approach (backward Euler) to approximate derivative:
 $(u(x,t+1) - u(x,t)) / dt = (u(x-h,t+1) - 2u(x,t+1) + u(x+h,t+1)) / h^2$
 $u(x,t) = u(x,t+1) + dt/h^2 * (u(x-h,t+1) - 2u(x,t+1) + u(x+h,t+1))$

- Let $z = dt/h^2$ and change variables (t to j and x to i)
 $u(:,i) = (I - z * L) * u(:, i+1)$

- Where I is identity and L is Laplacian

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

3/1/2004

CS267 Lecture 10

37

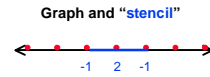
Implicit Solution of the Heat Equation

- The previous slide used Backwards Euler, but using the trapezoidal rule gives better numerical properties.

- This turns into solving the following equation:
 $(I + (z/2)*L) * u[:,i+1] = (I - (z/2)*L) * u[:,i]$

- 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

3/1/2004

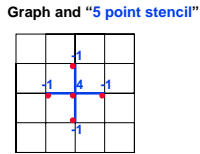
CS267 Lecture 10

38

2D Implicit Method

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

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



3D case is analogous (7 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.

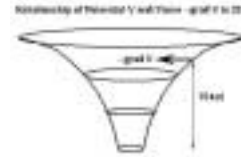
3/1/2004

CS267 Lecture 10

39

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 $-(x,y,z)/r^3$, 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!)



3/1/2004

CS267 Lecture 10

40

Algorithms for 2D Poisson Equation (N vars)

Algorithm	Serial	PRAM	Memory	#Procs
Dense LU	N^3	N	N^2	N^2
Band LU	N^2	N	$N^{3/2}$	N
Jacobi	N^2	N	N	N
Explicit Inv.	N^2	$\log N$	N^2	N^2
Conj.Grad.	$N^{3/2}$	$N^{1/2} * \log N$	N	N
RB SOR	$N^{3/2}$	$N^{1/2}$	N	N
Sparse LU	$N^{3/2}$	$N^{1/2}$	$N * \log N$	N
FFT	$N * \log N$	$\log N$	N	N
Multigrid	N	$\log^2 N$	N	N
• Lower bound	N	$\log N$	N	

PRAM is an idealized parallel model with zero cost communication
 Reference: James Demmel, Applied Numerical Linear Algebra, SIAM, 1997.

3/1/2004

CS267 Lecture 10

41

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 $\text{inv}(T)$ "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.
- 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.

3/1/2004

CS267 Lecture 10

42

Mflop/s Versus Run Time in Practice

- Problem: Iterative solver for a convection-diffusion problem; run on a 1024-CPU NCUBE-2.
- Reference: Shadid and Tuminaro, SIAM Parallel Processing Conference, March 1991.

Solver	Flops	CPU Time	Mflop/s
Jacobi	3.82×10^{12}	2124	1800
Gauss-Seidel	1.21×10^{12}	885	1365
Least Squares	2.59×10^{11}	185	1400
Multigrid	2.13×10^9	7	318

- Which solver would you select?

3/1/2004

CS267 Lecture 10

43

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.

3/1/2004

CS267 Lecture 10

44

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

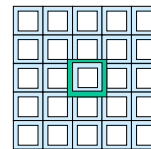
3/1/2004

CS267 Lecture 10

45

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



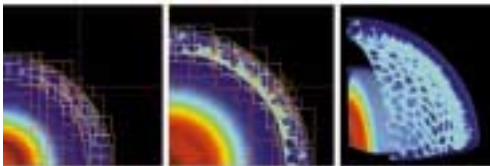
Implemented using "ghost" regions. Adds memory overhead

3/1/2004

CS267 Lecture 10

46

Adaptive Mesh Refinement (AMR)



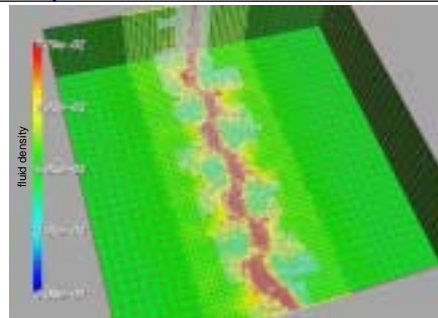
- Adaptive mesh around an explosion
 - Refinement done by calculating errors
- Parallelism
 - Mostly between "patches," dealt to processors for load balance
 - May exploit some within a patch (SMP)
- Projects:
 - Titanium (<http://www.cs.berkeley.edu/projects/titanium>)
 - Chombo (P. Colella, LBL), KeLP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

3/1/2004

CS267 Lecture 10

47

Adaptive Mesh



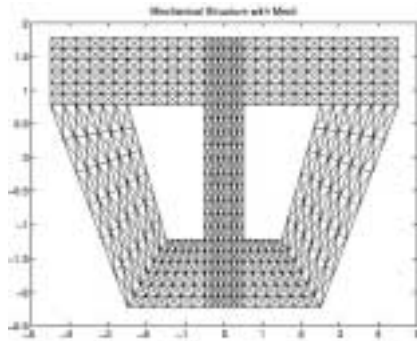
Shock waves in a gas dynamics using AMR (Adaptive Mesh Refinement)
See: <http://www.llnl.gov/CASC/SAMRAI/>

3/1/2004

CS267 Lecture 10

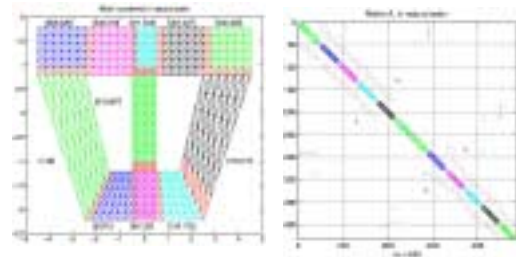
48

Composite Mesh from a Mechanical Structure



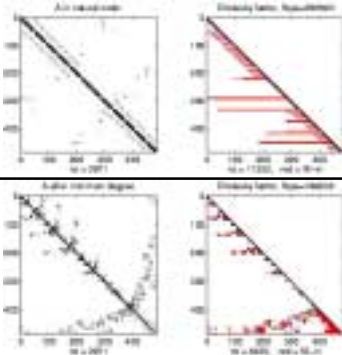
3/1/2004 CS267 Lecture 10 49

Converting the Mesh to a Matrix



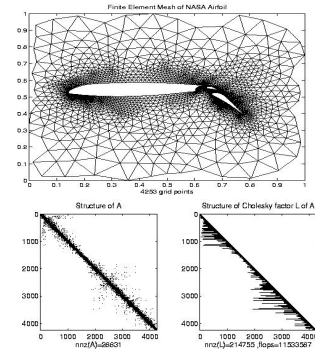
3/1/2004 CS267 Lecture 10 50

Effects of Reordering on Gaussian Elimination



3/1/2004 CS267 Lecture 10 51

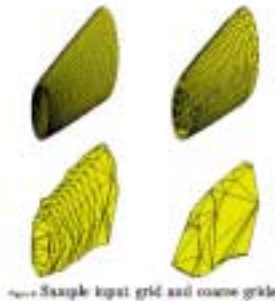
Irregular mesh: NASA Airfoil in 2D



3/1/2004 CS267 Lecture 10 52

Irregular mesh: Tapered Tube (Multigrid)

Example of Prometheus meshes



Sample input grid and coarse grids

3/1/2004 53

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

3/1/2004 CS267 Lecture 10 54

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

3/1/2004

CS267 Lecture 10

55

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

3/1/2004

CS267 Lecture 10

56

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)

3/1/2004

CS267 Lecture 10

57