
CS 267

Sources of

Parallelism and Locality

in Simulation – Part 2

James Demmel

www.cs.berkeley.edu/~demmel

Recap of Last Lecture

- 4 kinds of simulations
 - Discrete Event Systems
 - Particle Systems
 - Ordinary Differential Equations (ODEs)
 - Partial Differential Equations (PDEs) (today)
- Common problems:
 - Load balancing
 - May be due to lack of parallelism or poor work distribution
 - Statically, divide grid (or graph) into blocks
 - Dynamically, if load changes significantly during run
 - Locality
 - Partition into large chunks with low surface-to-volume ratio
 - To minimize communication
 - Distributed particles according to location, but use irregular spatial decomposition (e.g., quad tree) for load balance
 - Constant tension between these two
 - Particle-Mesh method: can't balance particles (moving), balance mesh (fixed) and keep particles near mesh points without communication

Partial Differential Equations

PDEs

Continuous Variables, Continuous Parameters

Examples of such systems include

- Elliptic problems (steady state, global space dependence)
 - Electrostatic or Gravitational Potential: Potential(position)
- Hyperbolic problems (time dependent, local space dependence):
 - Sound waves: Pressure(position,time)
- Parabolic problems (time dependent, global space dependence)
 - Heat flow: Temperature(position, time)
 - Diffusion: Concentration(position, time)

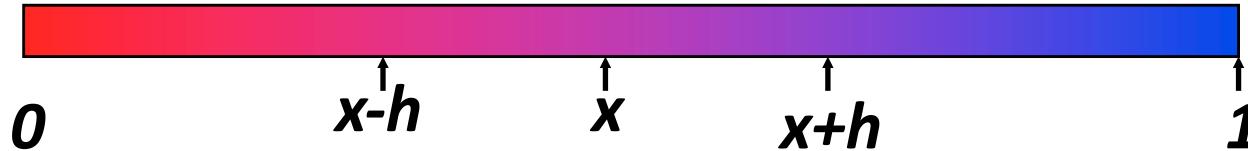
Global vs Local Dependence

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

Many problems combine features of above

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

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:

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

Details of the Explicit Method for Heat

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

- Discretize time and space using explicit approach (forward Euler) to approximate time derivative:

$$\begin{aligned}(u(x,t+\delta) - u(x,t))/\delta &= C [(u(x-h,t)-u(x,t))/h - (u(x,t)-u(x+h,t))/h] / h \\ &= C [u(x-h,t) - 2*u(x,t) + u(x+h,t)]/h^2\end{aligned}$$

Solve for $u(x,t+\delta)$:

$$u(x,t+\delta) = u(x,t) + C * \delta/h^2 * (u(x-h,t) - 2*u(x,t) + u(x+h,t))$$

- Let $z = C * \delta / h^2$, simplify:

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

- Change variable x to $j * h$, t to $i * \delta$, and $u(x,t)$ to $u[j,i]$

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

Explicit Solution of the Heat Equation

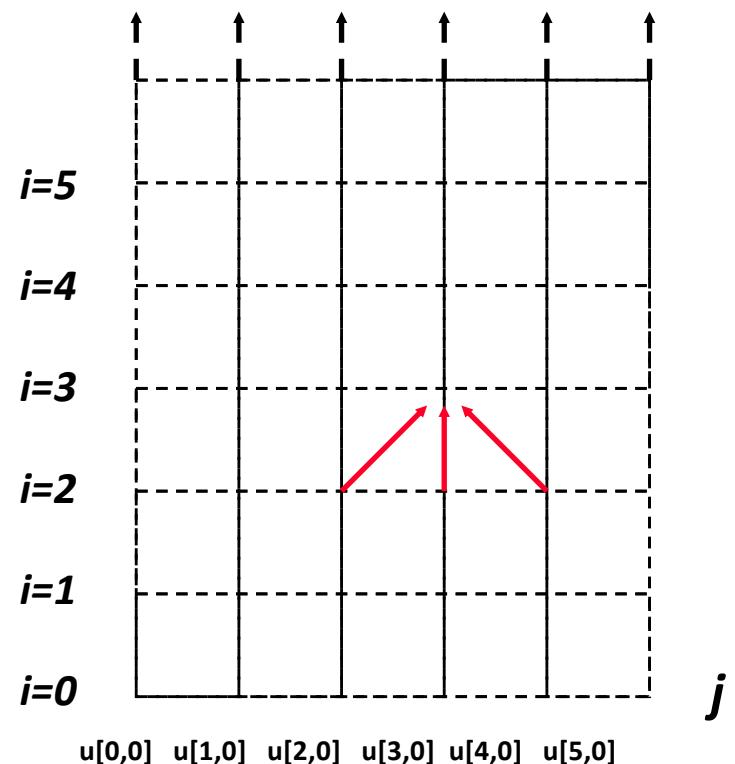
- Use “finite differences” with $u[j,i]$ as the temperature at
 - time $t = i * \delta$ ($i = 0, 1, 2, \dots$) and position $x = j * h$ ($j=0,1,\dots,N=1/h$)
 - initial conditions on $u[j,0]$
 - boundary conditions on $u[0,i]$ and $u[N,i]$
- At each timestep $i = 0, 1, 2, \dots$

For $j=1$ to $N-1$

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

where $z = C * \delta / h^2$

- This corresponds to
 - Matrix-vector-multiply by T (next slide)
 - Combine nearest neighbors on grid



Matrix View of Explicit Method for Heat

- $u[j,i+1] = z^*u[j-1,i] + (1-2^*z)^*u[j,i] + z^*u[j+1,i]$, same as:
- $u[:, i+1] = T * u[:, i]$ where T is tridiagonal:

$$T = \begin{pmatrix} 1-2z & z & & & \\ z & 1-2z & z & & \\ & z & 1-2z & z & \\ & & z & 1-2z & z \\ & & & z & 1-2z \end{pmatrix} = I - z^*L, \quad L = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$$

Graph and “3 point stencil”

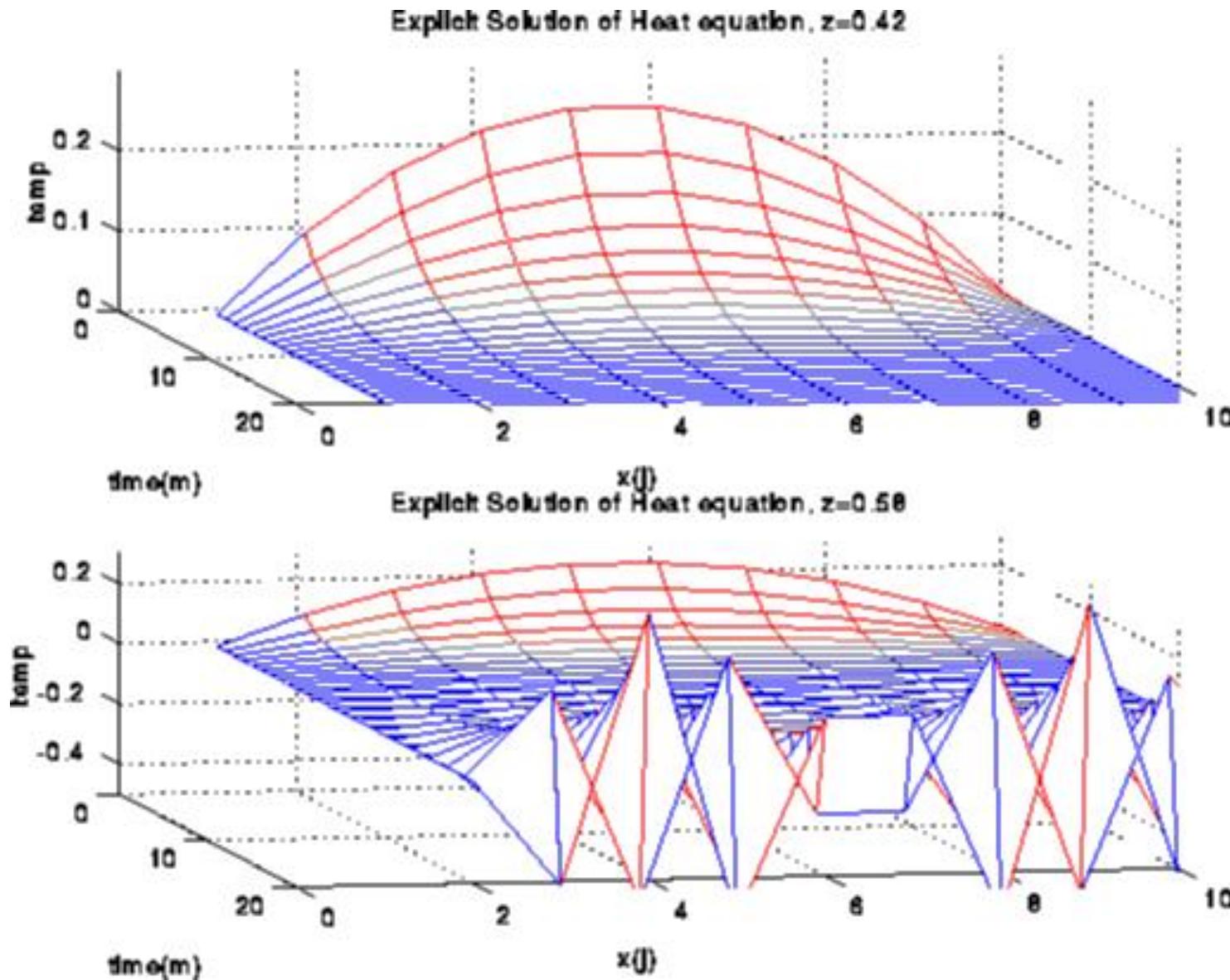


- L called Laplacian (in 1D)
- For a 2D mesh (5 point stencil) the Laplacian is pentadiagonal
 - More on the matrix/grid views later

Parallelism in Explicit Method for PDEs

- Sparse matrix vector multiply, via Graph Partitioning
- Partitioning the space (x) into p chunks
 - good load balance (assuming large number of points relative to p)
 - minimize communication (least dependence on data outside chunk)
- Generalizes to
 - multiple dimensions.
 - arbitrary graphs (= arbitrary sparse matrices).
- Explicit approach often used for hyperbolic equations
 - Finite wave speed, so only depend on nearest chunks
- Problem with explicit approach for heat (parabolic):
 - numerical instability.
 - solution blows up eventually if $z = C\delta/h^2 > .5$
 - need to make the time step δ very small when h is small: $\delta < .5*h^2 / C$

Instability in Solving the Heat Equation Explicitly



Implicit Solution of the Heat Equation

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

- Discretize time and space using **implicit** approach (**Backward Euler**) to approximate time derivative:

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

- Let $z = C*\delta/h^2$ and change variable t to $i*\delta$, x to $j*h$ and $u(x,t)$ to $u[j,i]$

$$(I + z * L) * u[:, i+1] = u[:, i]$$

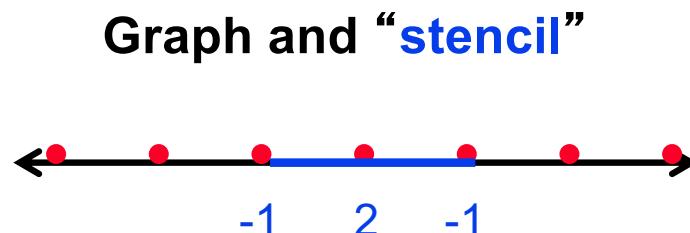
- Where I is identity and L is Laplacian as before

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

Implicit Solution of the Heat Equation

- The previous slide derived Backward Euler
 - $(I + z * L)^* u[:, i+1] = u[:, i]$
- But the Trapezoidal Rule has better numerical properties:
$$(I + (z/2)*L) * u[:, i+1] = (I - (z/2)*L) * u[:, i]$$
- Again I is the identity matrix and L is:

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

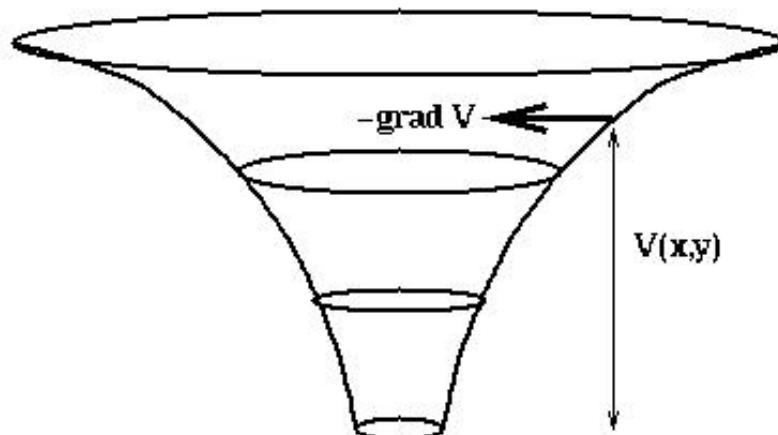


- Other problems (elliptic instead of parabolic) yield Poisson's equation ($Lx = b$ in 1D)

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 $\nabla V = -1/r$
 $= -(d/dx V, d/dy V, d/dz V) = -\nabla V$
- V satisfies Poisson's equation (try working this out!)

Relationship of Potential V and Force $-\nabla V$ in 2D



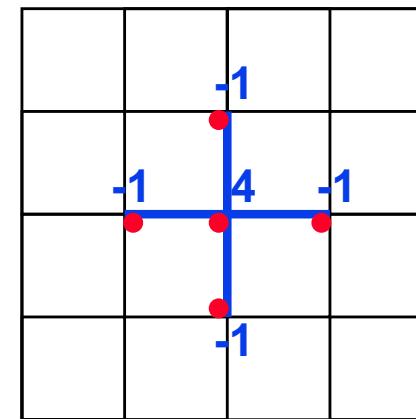
$$\frac{d^2V}{dx^2} + \frac{d^2V}{dy^2} + \frac{d^2V}{dz^2} = 0$$

2D Implicit Method

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

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

Graph and “5 point stencil”



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.

Algorithms for 2D (3D) Poisson Equation (N vars)

Algorithm	Serial	PRAM	Memory	#Procs
• Dense LU	N^3	N	N^2	N^2
• Band LU	$N^2 (N^{7/3})$	N	$N^{3/2} (N^{5/3})$	$N(N^{4/3})$
• Jacobi	$N^2 (N^{5/3})$	$N (N^{2/3})$	N	N
• Explicit Inv.	N^2	$\log N$	N^2	N^2
• Conj.Gradients	$N^{3/2} (N^{4/3})$	$N^{1/2} (1/3) * \log N$	N	N
• Red/Black SOR	$N^{3/2} (N^{4/3})$	$N^{1/2} (N^{4/3})$	N	N
• Sparse LU	$N^{3/2} (N^2)$	$N^{1/2} (N^{2/3})$	$N * \log N (N^{4/3})$	$N(N^{4/3})$
• FFT	$N * \log N$	$\log N$	N	N
• Multigrid	N	$\log^2 N$	N	N
• Lower bound	N	$\log N$	N	

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

PRAM is an idealized parallel model with zero cost communication

References: James Demmel, Applied Numerical Linear Algebra, SIAM, 1997.

Decision tree to help choose algorithms:

www.netlib.org/linalg/html_templates/Templates.html

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 $\text{sqrt}(N)$ 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. Used in multigrid schemes.
- **Sparse 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.
- Reference: Shadid and Tuminaro, SIAM Parallel Processing Conference, March 1991.

Solver	Flops	CPU Time(s)	Mflop/s
Jacobi	3.82×10^{12}	2124	1800
Gauss-Seidel	1.21×10^{12}	885	1365
Multigrid	2.13×10^9	7	318

- Which solver would you select?

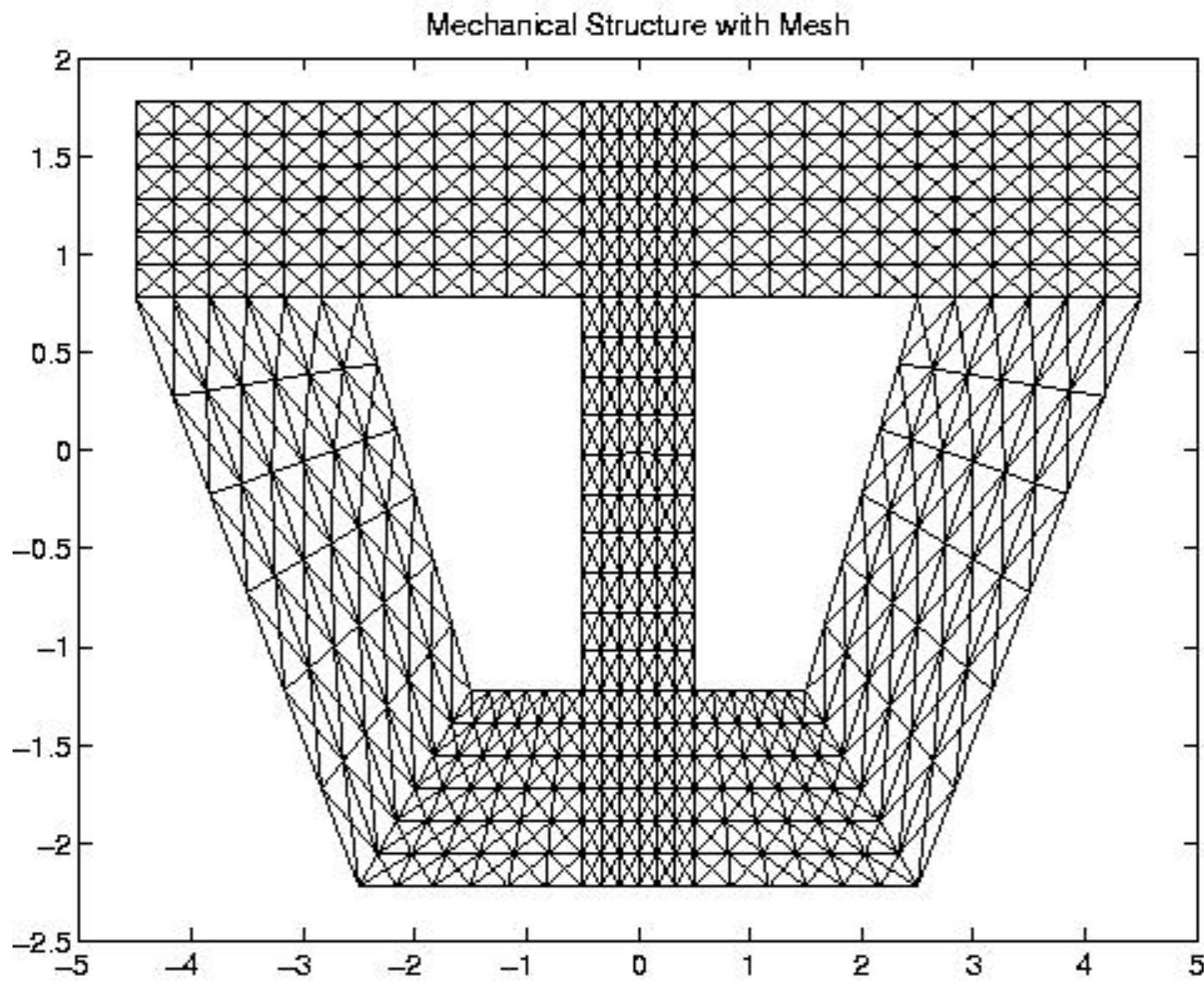
Summary of Approaches to Solving PDEs

- 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
 - Graph partitioning
- Graph 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
 - Depends on the mesh structure (nonzero structure) and the number of Flops per point.

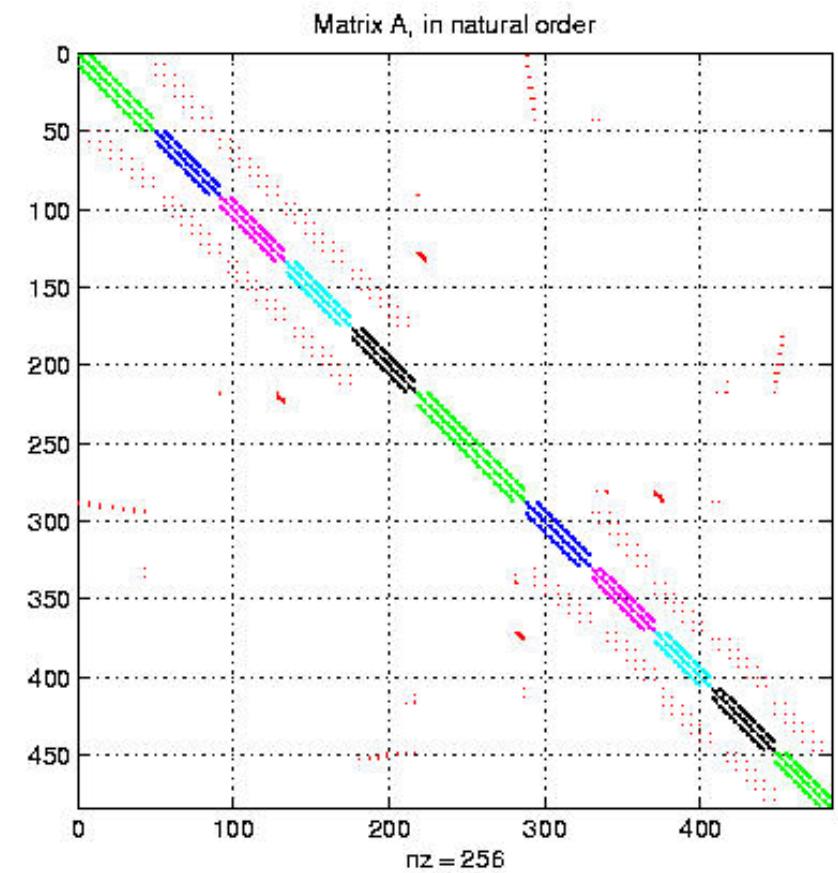
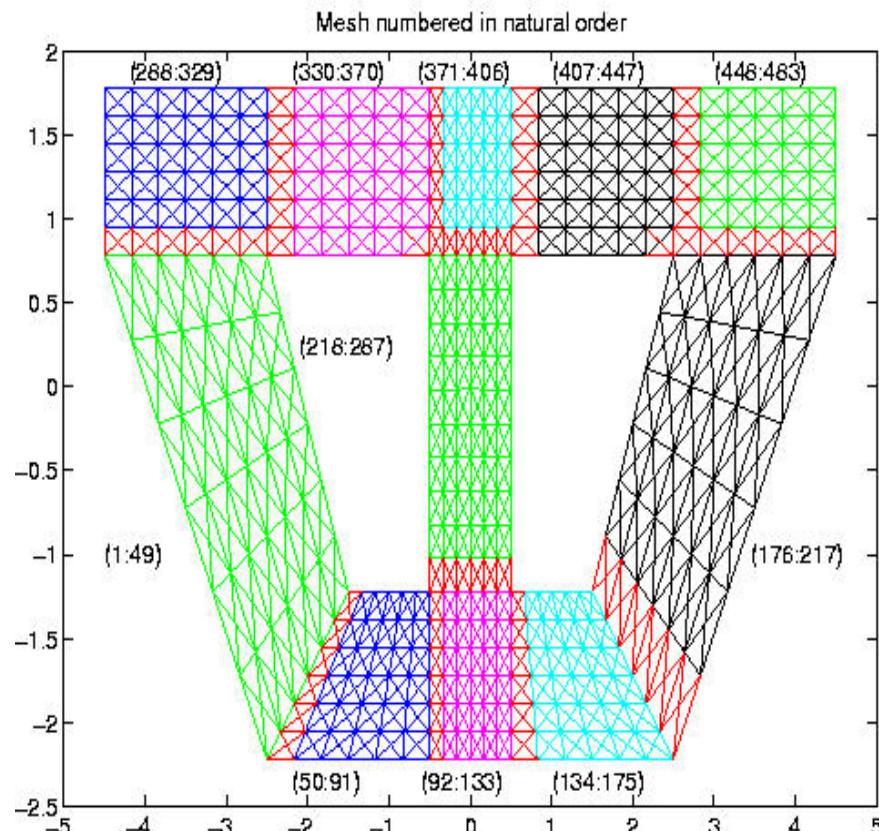
Comments on practical meshes

- 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

Composite mesh from a mechanical structure

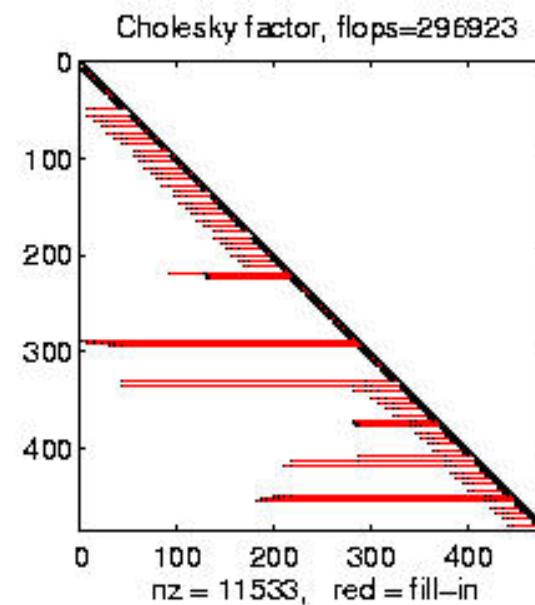
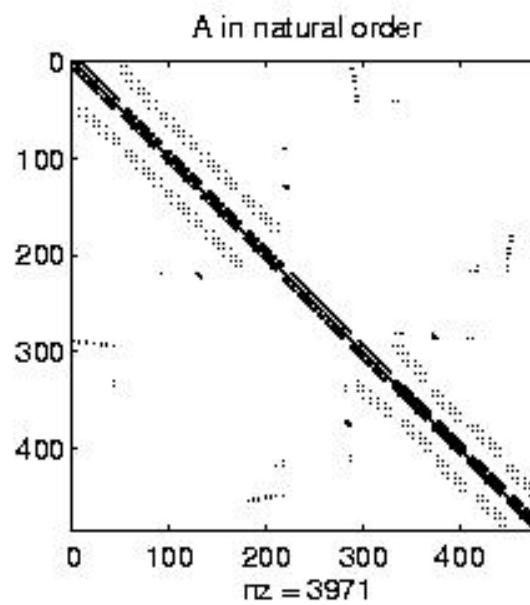


Converting the mesh to a matrix

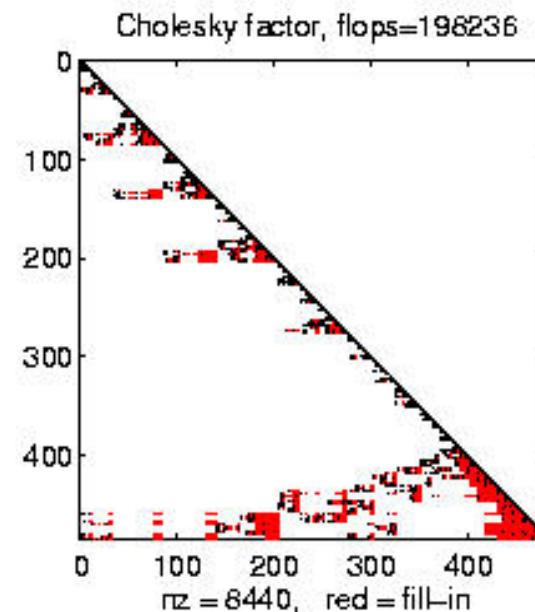
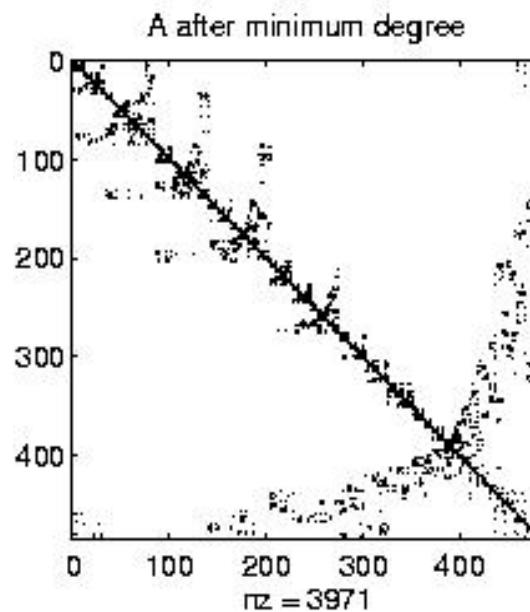


Example of Matrix Reordering Application

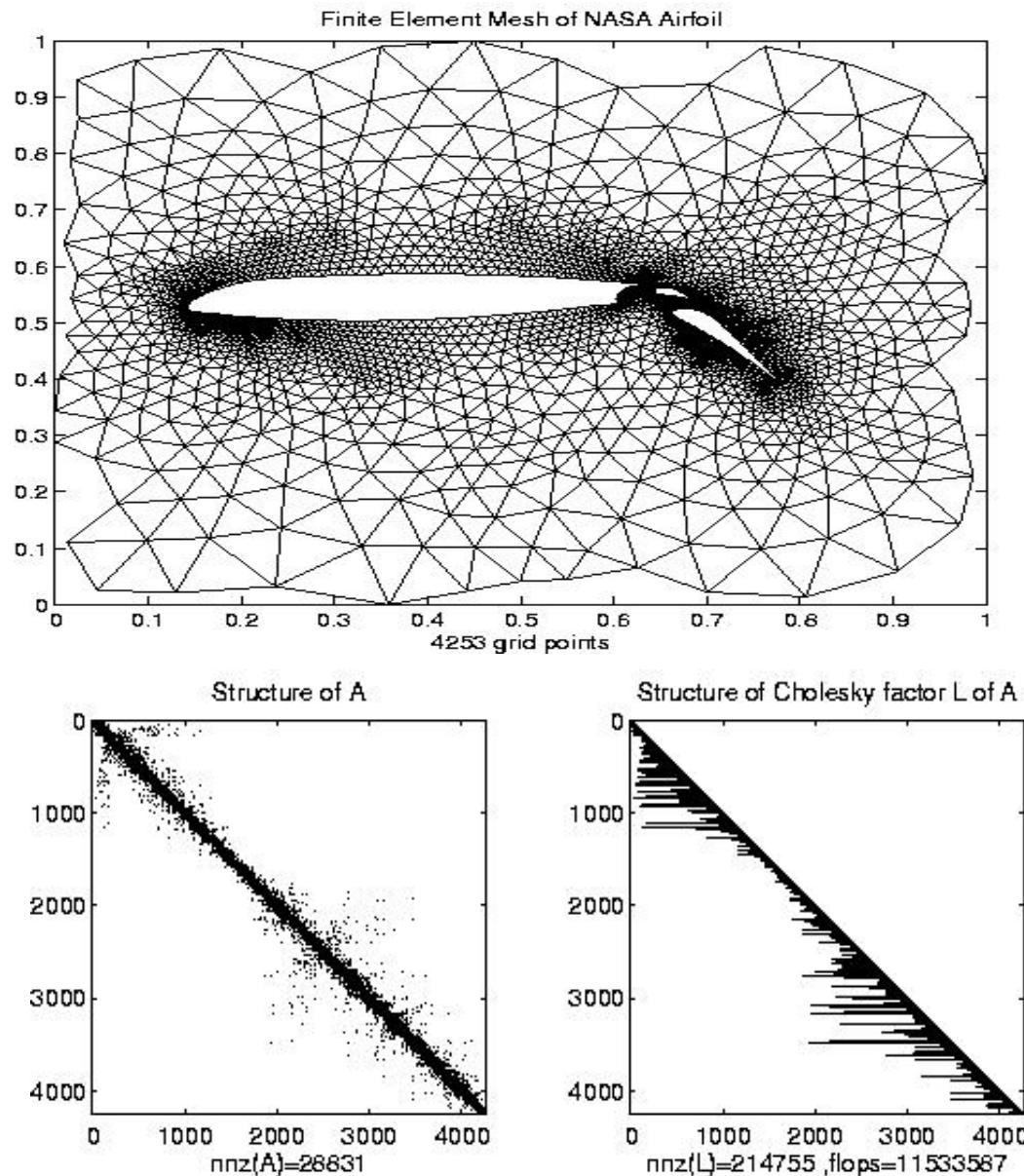
When performing Gaussian Elimination
Zeros can be filled ☹



Matrix can be reordered
to reduce this fill
But it's not the same
ordering as for
parallelism



Irregular mesh: NASA Airfoil in 2D (direct solution)



Irregular mesh: Tapered Tube (multigrid)

Example of Prometheus meshes

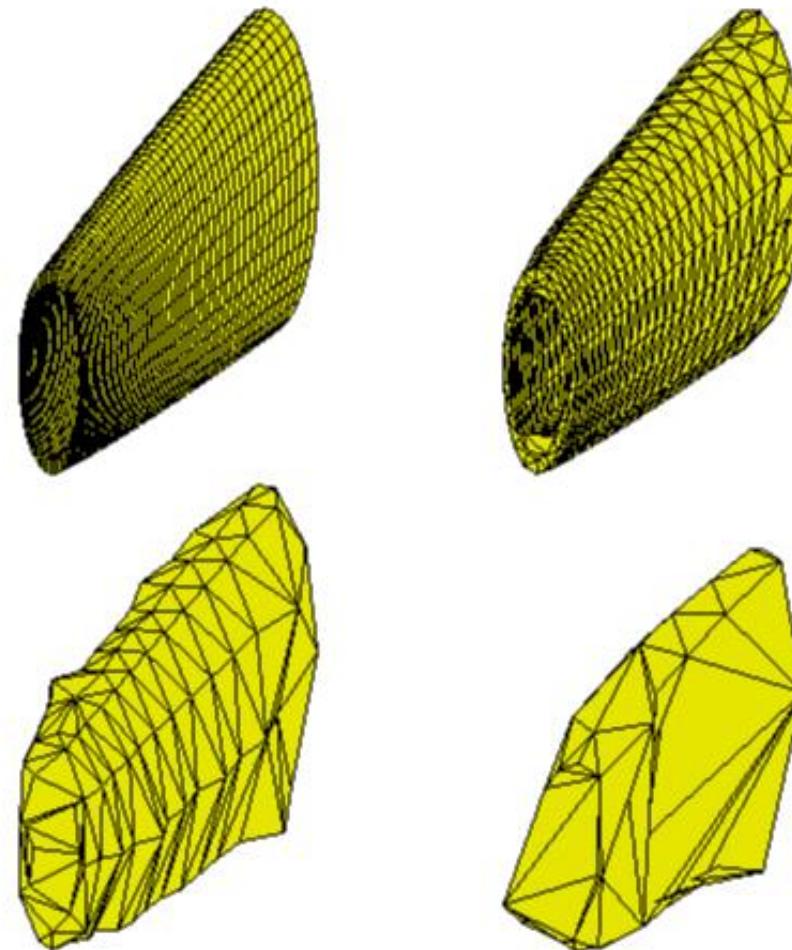
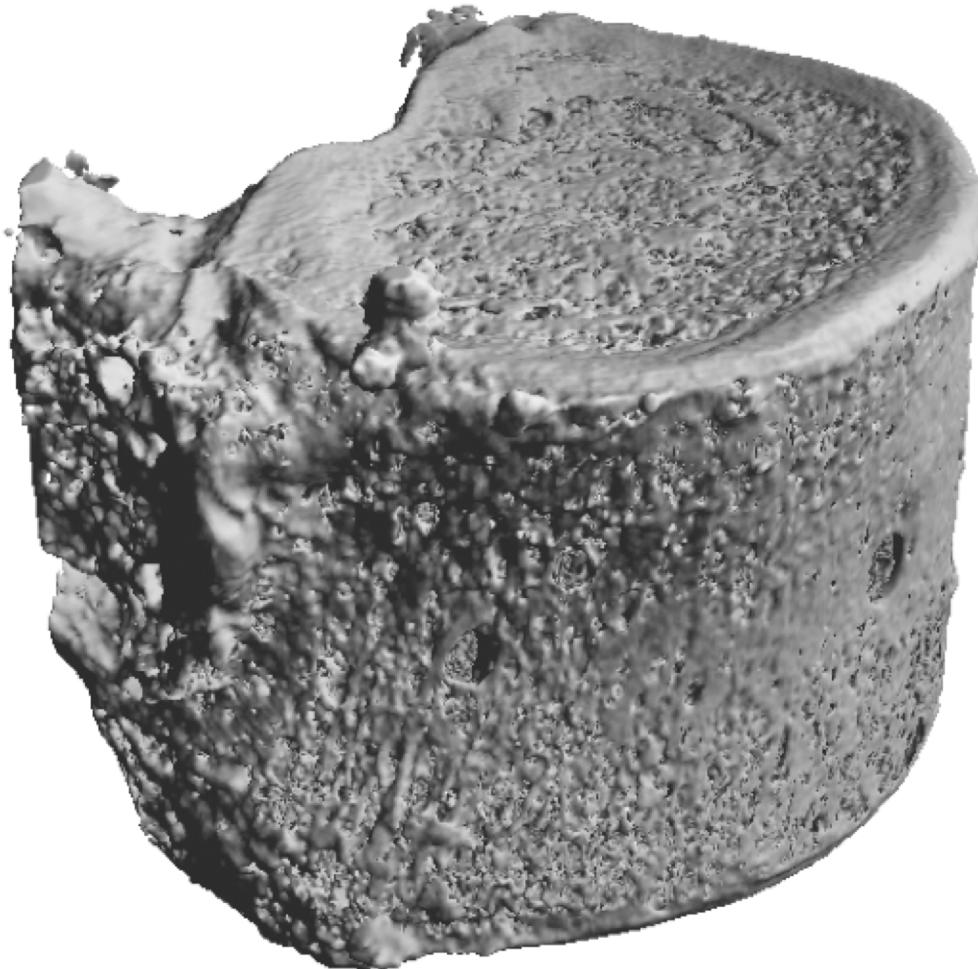


Figure 6: Sample input grid and coarse grids

Source of Unstructured Finite Element Mesh: Vertebra

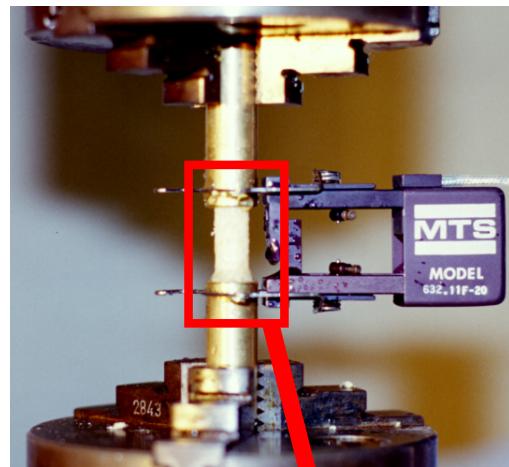
Study failure modes of trabecular Bone under stress



Source: M. Adams, H. Bayraktar, T. Keaveny, P. Papadopoulos, A. Gupta

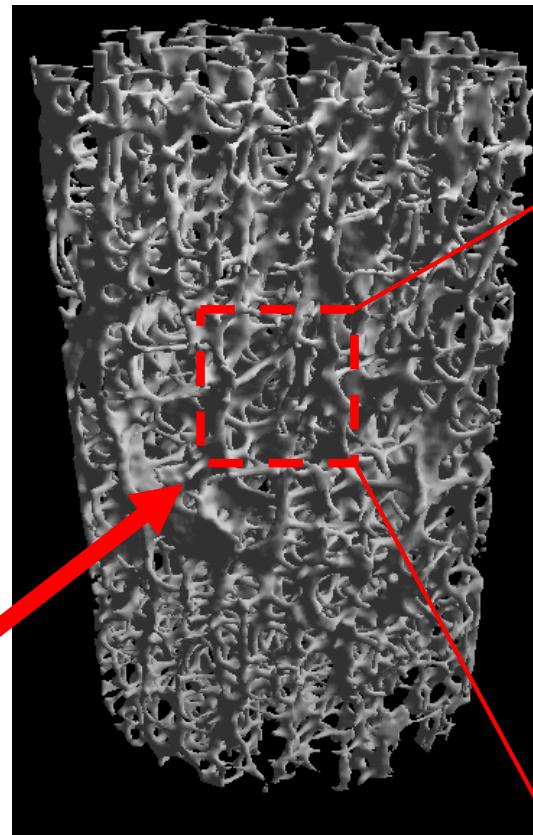
Methods: μ FE modeling (Gordon Bell Prize, 2004)

Mechanical Testing
 E , ε_{yield} , σ_{ult} , etc.

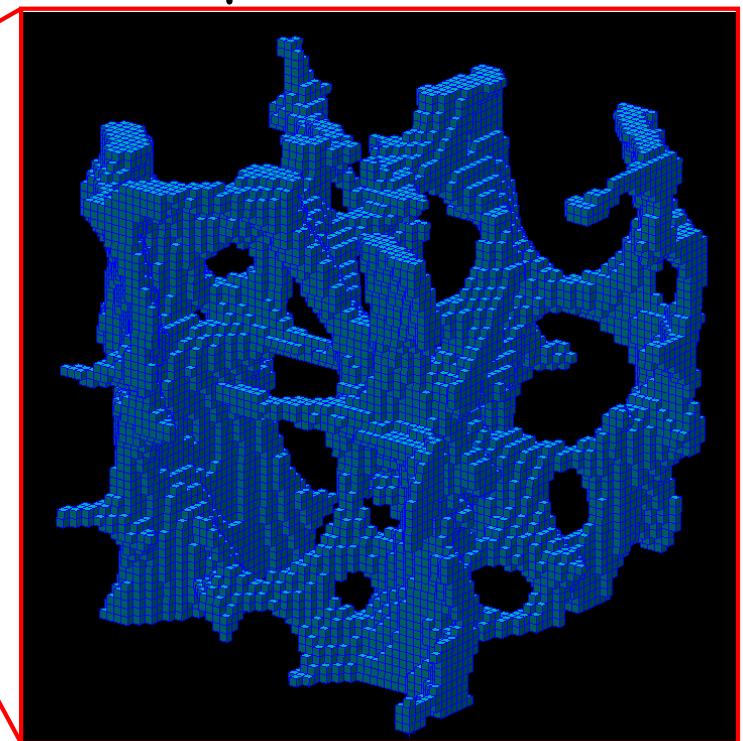


Source: Mark Adams, PPPL

3D image



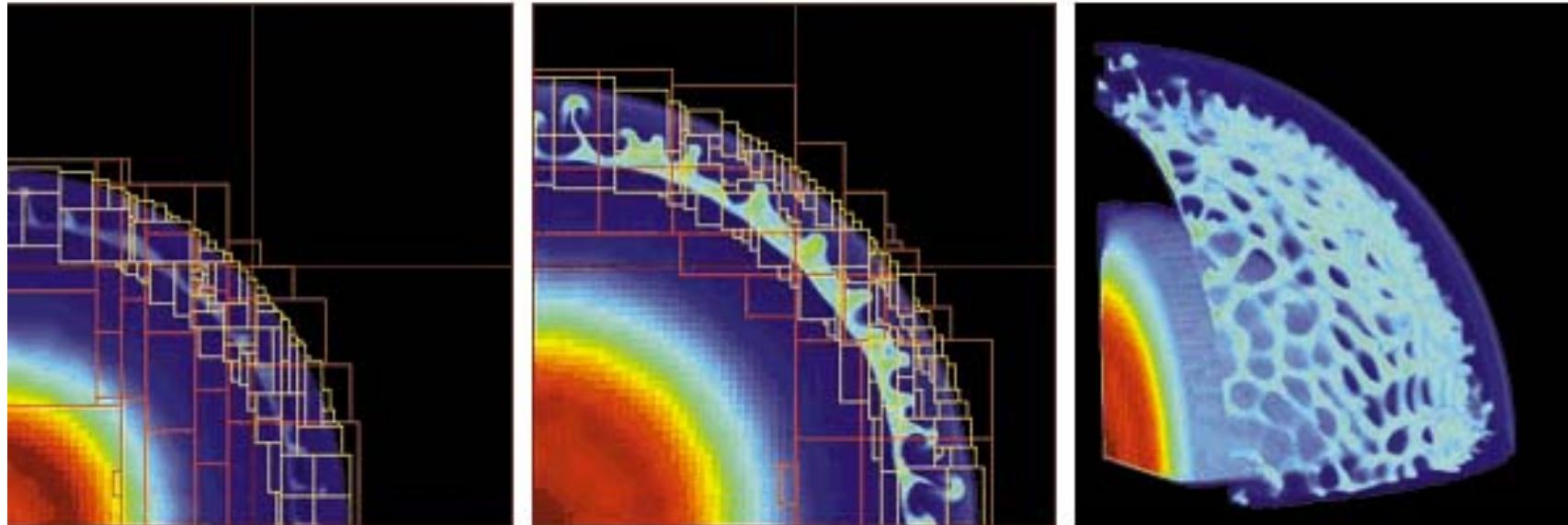
μ FE mesh
2.5 mm cube
44 μ m elements



Micro-Computed Tomography
 μ CT @ 22 μ m resolution

Up to 537M unknowns

Adaptive Mesh Refinement (AMR)



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

Challenges of Irregular Meshes

- How to generate them in the first place
 - Start from geometric description of object
 - 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

Summary – sources of parallelism and locality

- Current attempts to categorize main “kernels” dominating simulation codes
- “Seven Dwarfs” (P. Colella)
 - Structured grids
 - including locally structured grids, as in AMR
 - Unstructured grids
 - Spectral methods (Fast Fourier Transform)
 - Dense Linear Algebra
 - Sparse Linear Algebra
 - Both explicit (SpMV) and implicit (solving)
 - Particle Methods
 - Monte Carlo/Embarrassing Parallelism/Map Reduce (easy!)

What do commercial and CSE applications have in common?

Motif/Dwarf: Common Computational Methods (Red Hot → Blue Cool)

