Multigrid

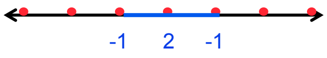
James Demmel

www.cs.berkeley.edu/~demmel/ma221_Spr16

Poisson's equation in 1D: $\partial^2 u/\partial x^2 = f(x)$

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

Graph and "stencil"

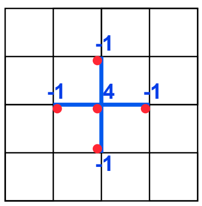


2D Poisson’s equation

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

$$T = \left(\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
& -1 & 4 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
& -1 & & -1 & 4 & -1 & & -1 & \\
& & -1 & & -1 & 4 & & & -1 \\
\hline
& & & -1 & & & 4 & -1 & \\
& & & & -1 & & -1 & 4 & -1 \\
& & & & & -1 & & -1 & 4
\end{array}\right)$$

Graph and “stencil”



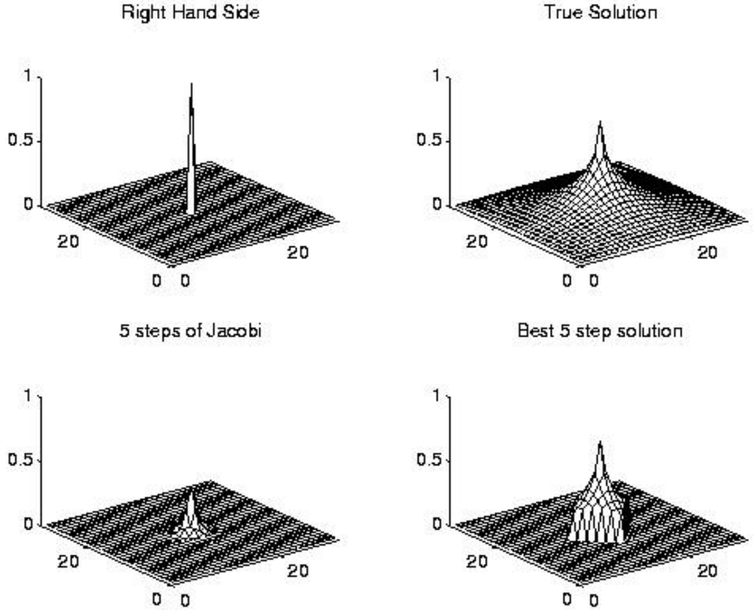
- 3D is analogous

Algorithms for 2D (3D) Poisson Equation (N = n^2 (n^3) 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	$N^{3/2}$ ($N^{4/3}$)	$N^{1/2}$ ($N^{1/3}$)	N	N
° Sparse LU	$N^{3/2}$ (N^2)	$N^{1/2}$	N*log N ($N^{4/3}$)	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

Multigrid Motivation



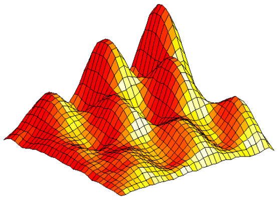
Multigrid Motivation

- Recall that Jacobi, SOR, CG, or any other sparse-matrix-vector-multiply-based algorithm can only move information one grid cell at a time
  - Take at least n steps to move information across n x n grid
- Can show that decreasing error by fixed factor c<1 takes $\Omega(\log n)$ steps
  - Convergence to fixed error < 1 takes $\Omega(\log n)$ steps
- Therefore, converging in O(1) steps requires moving information across grid faster than to one neighboring grid cell per step
  - One step can't just do sparse-matrix-vector-multiply

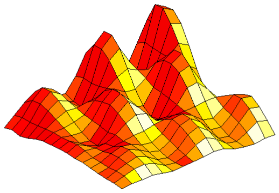
Big Idea used in multigrid and elsewhere

- If you are far away, problem looks simpler
  - For gravity: approximate earth, distant galaxies, … by point masses
- Can solve such a coarse approximation to get an approximate solution, iterating if necessary
  - Solve coarse approximation problem by using an even coarser approximation of it, and so on recursively
- Ex: Multigrid for solving PDE in O(n) time
  - Use coarser mesh to get approximate solution of Poisson's Eq.
- Ex: Fast Multipole Method, Barnes-Hut for computing gravitational forces on n particles in O(n log n) time:
  - Approximate particles in box by total mass, center of gravity
  - Good enough for distant particles; for close ones, divide box recursively
- Ex: Graph Partitioning (used to parallelize SpMV)
  - Replace graph to be partitioned by a coarser graph (CS267 for details)

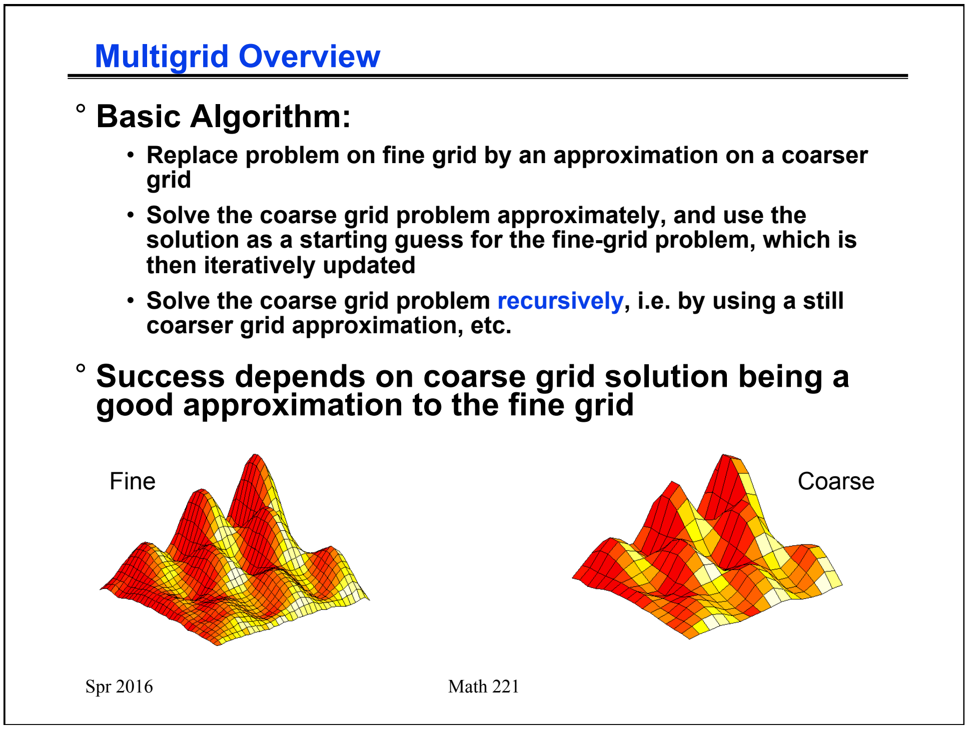
Fine and Coarse Approximations



Fine



Coarse



Multigrid Overview

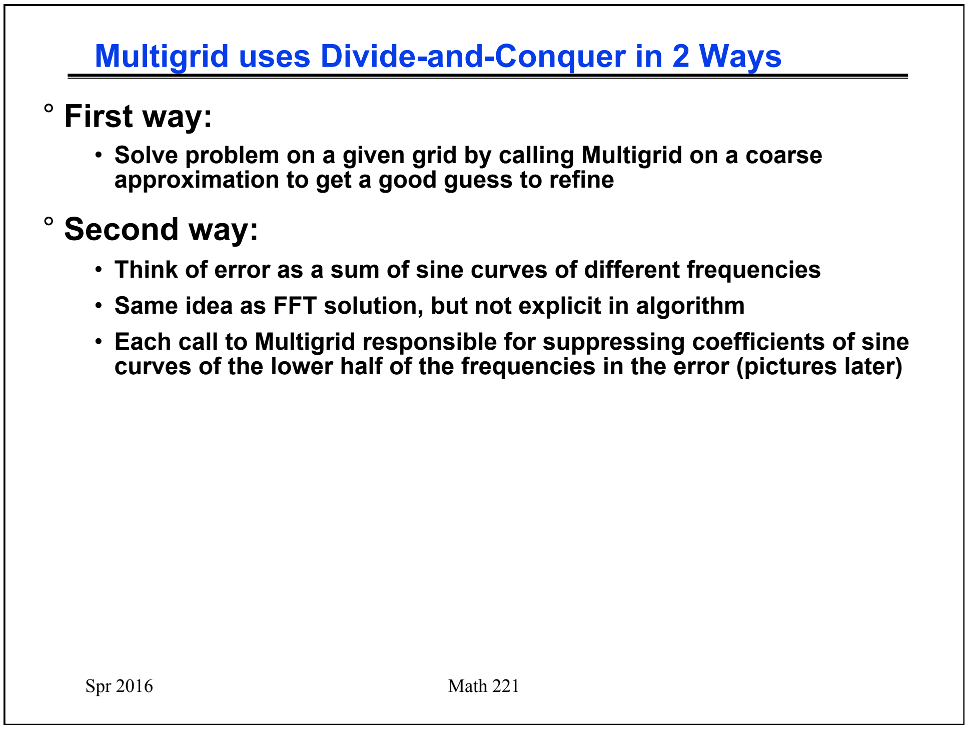
° **Basic Algorithm:**

- **Replace problem on fine grid by an approximation on a coarser grid**
- **Solve the coarse grid problem approximately, and use the solution as a starting guess for the fine-grid problem, which is then iteratively updated**
- **Solve the coarse grid problem recursively, i.e. by using a still coarser grid approximation, etc.**

° **Success depends on coarse grid solution being a good approximation to the fine grid**

Fine

Coarse



Multigrid uses Divide-and-Conquer in 2 Ways

° **First way:**

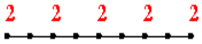
- **Solve problem on a given grid by calling Multigrid on a coarse approximation to get a good guess to refine**

° **Second way:**

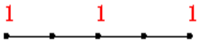
- **Think of error as a sum of sine curves of different frequencies**
- **Same idea as FFT solution, but not explicit in algorithm**
- **Each call to Multigrid responsible for suppressing coefficients of sine curves of the lower half of the frequencies in the error (pictures later)**

Multigrid Sketch in 1D

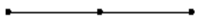
- Consider a 2^m+1 grid in 1D for simplicity
- Let $P^{(i)}$ be the problem of solving the discrete Poisson equation on a 2^i+1 grid in 1D. Write linear system as T(i) * x(i) = b(i)
- $P^{(m)}$, $P^{(m-1)}$, … , $P^{(1)}$ is sequence of problems from finest to coarsest



$P^{(3)}$: 1D grid of 9 points
7 unknowns
Points labeled 2 are
part of next coarser grid



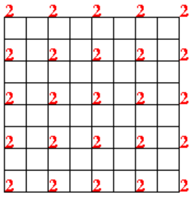
$P^{(2)}$: 1D grid of 5 points
3 unknowns
Points labeled 1 are
part of next coarser grid



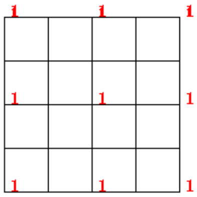
$P^{(1)}$: 1D grid of 3 points
1 unknown

Multigrid Sketch (1D and 2D)

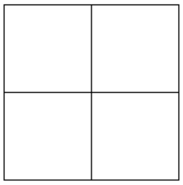
- Consider a 2^m+1 grid in 1D (2^m+1 by 2^m+1 grid in 2D) for simplicity
- Let $P^{(i)}$ be the problem of solving the discrete Poisson equation on a 2^i+1 grid in 1D (2^i+1 by 2^i+1 grid in 2D)
  - Write linear system as T(i) * x(i) = b(i)
- $P^{(m)}$, $P^{(m-1)}$, … , $P^{(1)}$ is sequence of problems from finest to coarsest



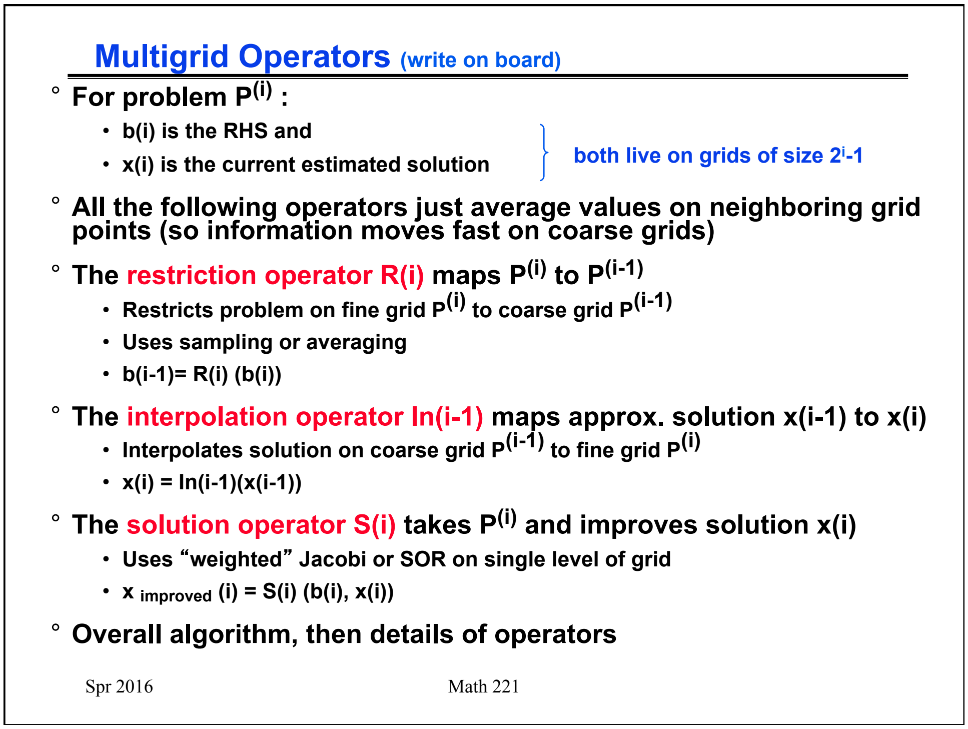
$P^{(3)}$: 9 by 9 grid of points
7 by 7 grid of unknowns
Points labeled 2 are
part of next coarser grid



$P^{(2)}$: 5 by 5 grid of points
3 by 3 grid of unknowns
Points labeled 1 are
part of next coarser grid



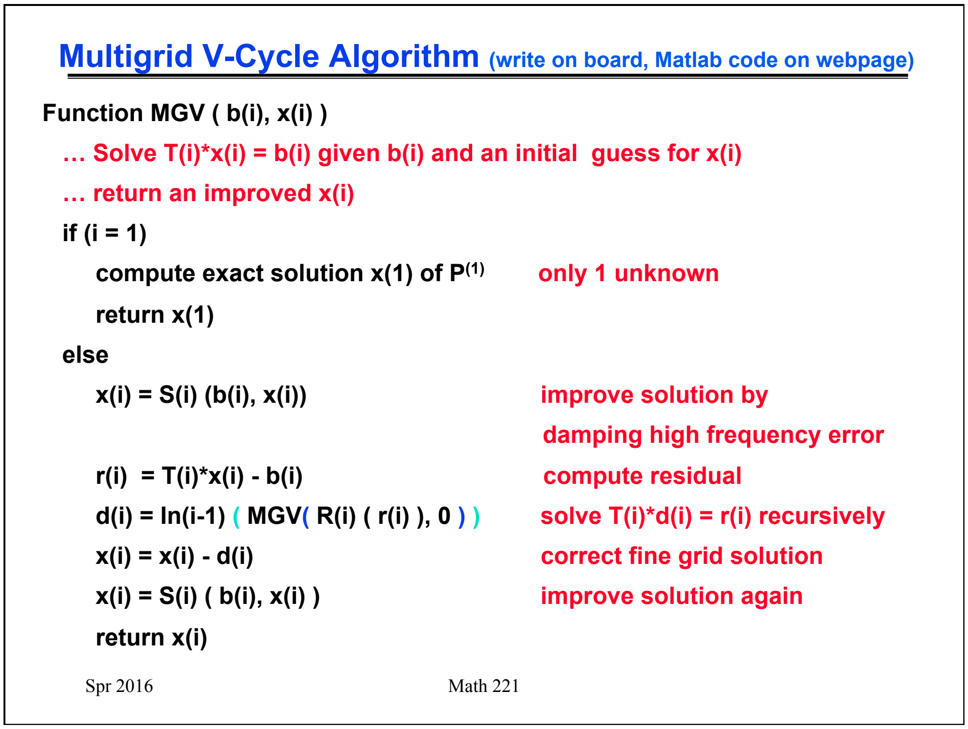
$P^{(1)}$: 3 by 3 grid of points
1 by 1 grid of unknowns



Multigrid Operators (write on board)

- For problem $P^{(i)}$:
  - b(i) is the RHS and
  - x(i) is the current estimated solution

  both live on grids of size 2^i-1
- All the following operators just average values on neighboring grid points (so information moves fast on coarse grids)
- The restriction operator R(i) maps $P^{(i)}$ to $P^{(i-1)}$
  - Restricts problem on fine grid $P^{(i)}$ to coarse grid $P^{(i-1)}$
  - Uses sampling or averaging
  - b(i-1)= R(i) (b(i))
- The interpolation operator In(i-1) maps approx. solution x(i-1) to x(i)
  - Interpolates solution on coarse grid $P^{(i-1)}$ to fine grid $P^{(i)}$
  - x(i) = In(i-1)(x(i-1))
- The solution operator S(i) takes $P^{(i)}$ and improves solution x(i)
  - Uses “weighted” Jacobi or SOR on single level of grid
  - x_{improved} (i) = S(i) (b(i), x(i))
- Overall algorithm, then details of operators

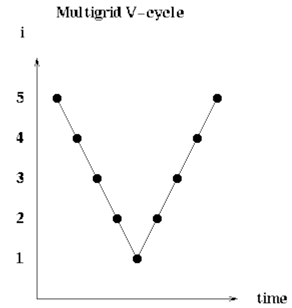


Multigrid V-Cycle Algorithm (write on board, Matlab code on webpage)

```
Function MGV ( b(i), x(i) )
  … Solve T(i)*x(i) = b(i) given b(i) and an initial  guess for x(i)
  … return an improved x(i)
  if (i = 1)
      compute exact solution x(1) of $P^{(1)}$      only 1 unknown
      return x(1)
  else
      x(i) = S(i) (b(i), x(i))                     improve solution by
                                                   damping high frequency error
      r(i)  = T(i)*x(i) - b(i)                     compute residual
      d(i) = In(i-1) ( MGV( R(i) ( r(i) ), 0 ) )   solve T(i)*d(i) = r(i) recursively
      x(i) = x(i) - d(i)                           correct fine grid solution
      x(i) = S(i) ( b(i), x(i) )                   improve solution again
      return x(i)
```

Why is this called a V-Cycle?

- Just a picture of the call graph
- In time a V-cycle looks like the following



Cost (#flops) of a V-Cycle for 2D Poisson

- Constant work per mesh point (average with neighbors)
- Work at each level in a V-cycle is O(the number of unknowns)
- Cost of Level i is $O((2^i-1)^2) = O(4^i)$
- If finest grid level is m, total time is:

$$\Sigma_{i=1}^{m} O(4^i) = O(4^m) = O(\#\text{ unknowns})$$

Full Multigrid (FMG)

° **Intuition:**

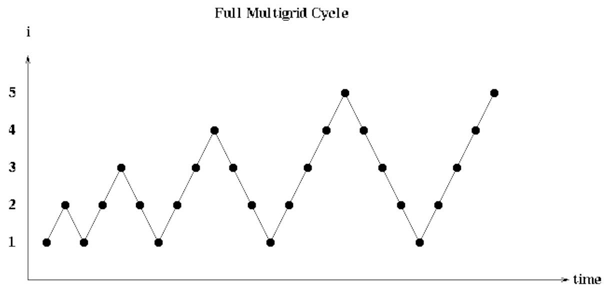
- improve solution by doing multiple V-cycles
- avoid expensive fine-grid (high frequency) cycles
- analysis of why this works is beyond the scope of this class

```
Function FMG (b(m), x(m))
    … return improved x(m) given initial guess
    compute the exact solution x(1) of P(1)
    for i=2 to m
        x(i) = MGV ( b(i), In (i-1) (x(i-1) ) )
```

° **In other words:**

- Solve the problem with 1 unknown
- Given a solution to the coarser problem, $P^{(i-1)}$, map it to starting guess for $P^{(i)}$
- Solve the finer problem using the Multigrid V-cycle

Full Multigrid Cost Analysis



° **One V-cycle for each call to FMG**

- people also use “W cycles” and other compositions

° **#Flops:** $\Sigma_{i=1}^{m} O(4^{i}) = O(4^{m}) = O(\# \text{unknowns})$

Complexity of Solving Poisson's Equation

- Theorem: error after one FMG call ≤ c · error before, where c < 1/2, independent of # unknowns

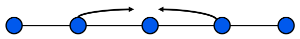
- Corollary: We can make the error < any fixed tolerance in a fixed number of steps, independent of size of finest grid

- This is the most important convergence property of MG, distinguishing it from all other methods, which converge more slowly for large grids

- Total complexity just proportional to cost of one FMG call

The Solution Operator S(i) – Details (on board)

- The solution operator, S(i), is a weighted Jacobi
- Consider the 1D problem



- At level i, pure Jacobi replaces:

  x(j) := 1/2 (x(j-1) + x(j+1) + b(j))

  in notation from lecture: R_J = I – T/2

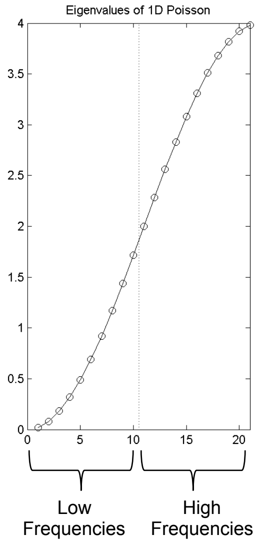
- Weighted Jacobi uses:

  x(j) := 1/3 (x(j-1) + x(j) + x(j+1) + b(j))

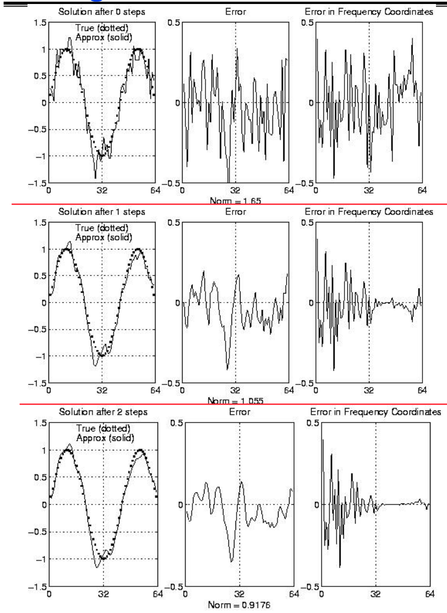
  R_wJ = I – T/3

- In 2D, similar average of nearest neighbors
  - Chosen so that "high frequency" eigenvector components of error get decreased by as much as possible (1/3)

Eigenvalues of Solution Operator S(i) (R_wJ = I – T/3)



Weighted Jacobi chosen to damp high frequency error



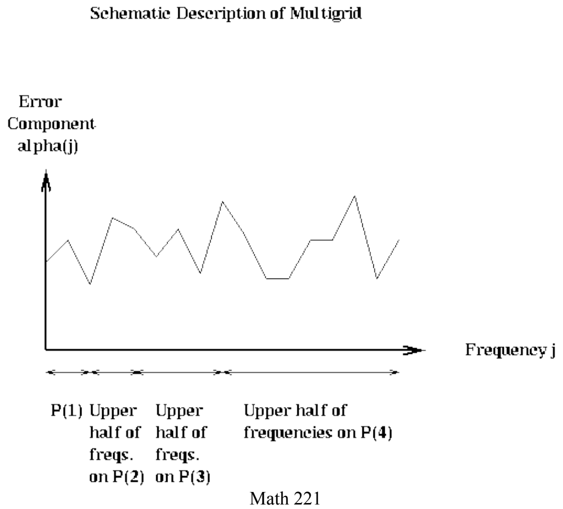
Initial error
“Rough”
Lots of high frequency components
Norm = 1.65

Error after 1 weighted Jacobi step
“Smoother”
Less high frequency component
Norm = 1.06

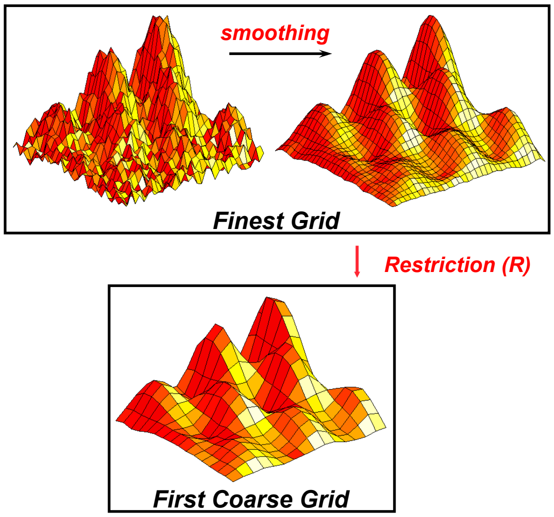
Error after 2 weighted Jacobi steps
“Smooth”
Little high frequency component
Norm = .92,
won’t decrease much more

Multigrid as Divide and Conquer Algorithm

- Each level in a V-Cycle reduces the error in one part of the frequency domain



Error on fine and coarse grids



The Restriction Operator R(i) - Details

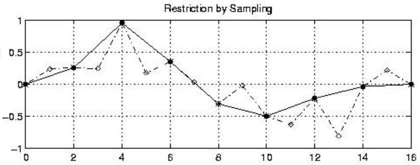
° The restriction operator, R(i), takes

- a problem $P^{(i)}$ with Right-Hand-Side (RHS) b_{fine} and
- maps it to a coarser problem $P^{(i-1)}$ with RHS $b_{coarse} = R(i)(b_{fine})$

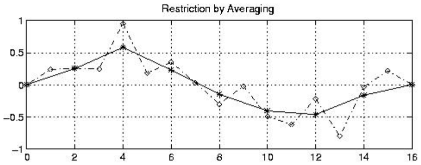
° In 1D, average values of neighbors

- Simplest: Sampling: $b_{coarse}(k) = b_{fine}(k)$
- Better: Averaging: $b_{coarse}(k) = 1/4 * b_{fine}(k-1) + 1/2 * b_{fine}(k) + 1/4 * b_{fine}(k+1)$

Simplest: Sampling



Better: Averaging



° In 2D, average with all 8 neighbors (N,S,E,W,NE,NW,SE,SW)

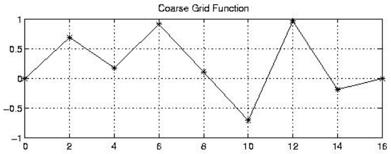
Interpolation Operator In(i-1): details

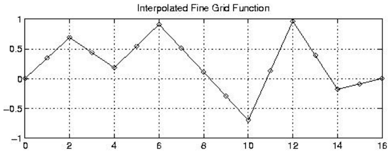
° The interpolation operator In(i-1), takes a function x_{coarse} on a coarse grid $P^{(i-1)}$, and produces a function x_{fine} on a fine grid $P^{(i)}$:

° $x_{fine} = In(i-1)(x_{coarse})$

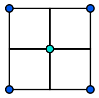
° In 1D, linearly interpolate nearest coarse neighbors

- $x_{fine}(k) = x_{coarse}(k)$ if the fine grid point k is also a coarse one, else
- $x_{fine}(k) = 1/2 * x_{coarse}(\text{left of } k) + 1/2 * x_{coarse}(\text{right of } k)$

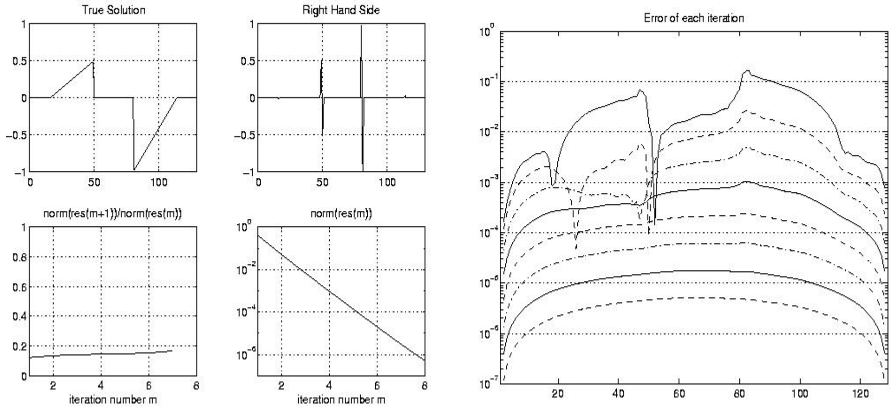




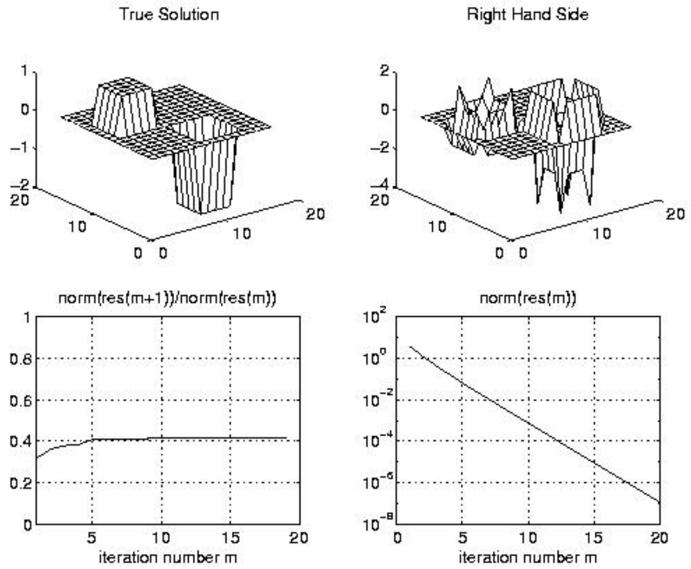
° In 2D, interpolation requires averaging with 4 nearest neighbors (NW,SW,NE,SE)

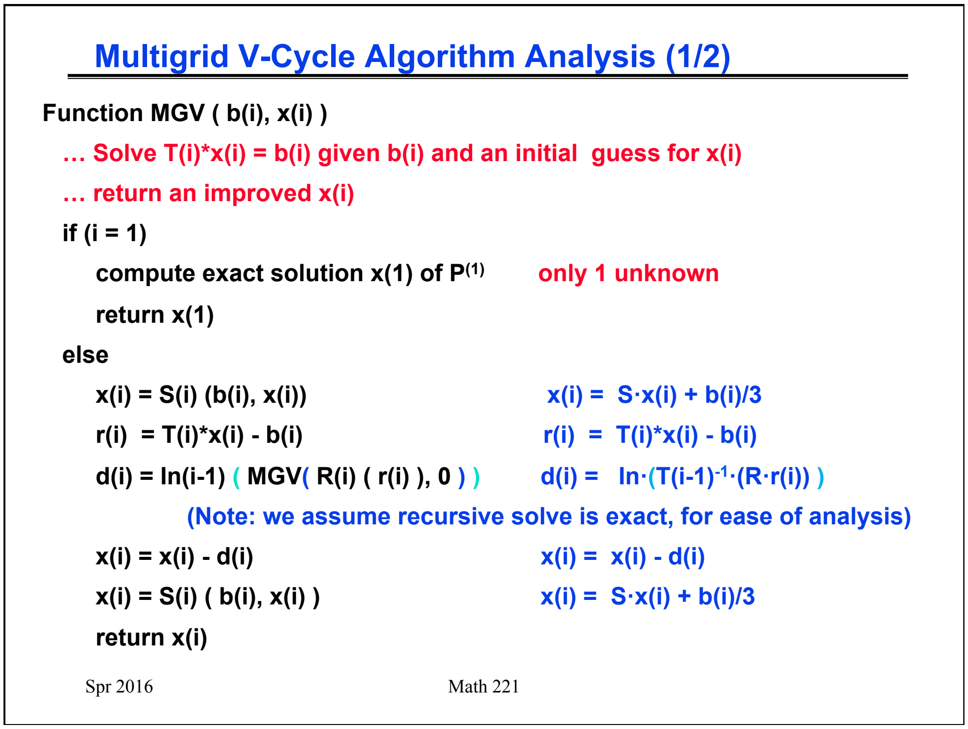


Convergence Picture of Multigrid in 1D



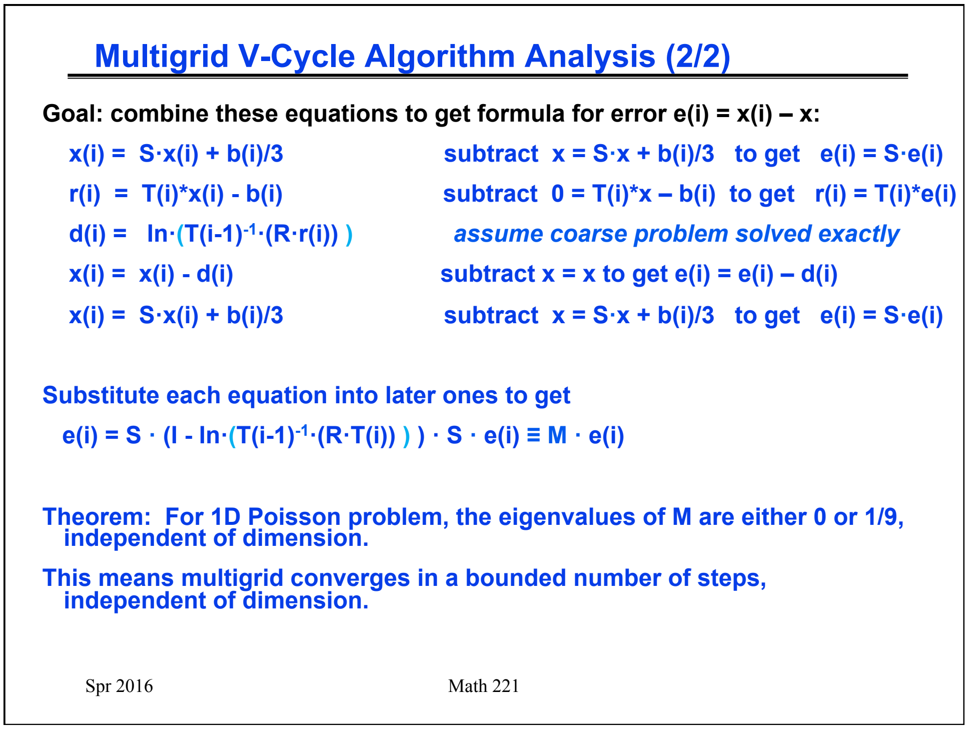
Convergence Picture of Multigrid in 2D





Multigrid V-Cycle Algorithm Analysis (1/2)

```
Function MGV ( b(i), x(i) )
   … Solve T(i)*x(i) = b(i) given b(i) and an initial  guess for x(i)
   … return an improved x(i)
   if (i = 1)
        compute exact solution x(1) of $P^{(1)}$        only 1 unknown
        return x(1)
   else
        x(i) = S(i) (b(i), x(i))                        x(i) =  S·x(i) + b(i)/3
        r(i)  = T(i)*x(i) - b(i)                        r(i)  =  T(i)*x(i) - b(i)
        d(i) = In(i-1) ( MGV( R(i) ( r(i) ), 0 ) )      d(i) =   In·(T(i-1)$^{-1}$·(R·r(i)) )
                     (Note: we assume recursive solve is exact, for ease of analysis)
        x(i) = x(i) - d(i)                              x(i) =  x(i) - d(i)
        x(i) = S(i) ( b(i), x(i) )                      x(i) =  S·x(i) + b(i)/3
        return x(i)
```



Multigrid V-Cycle Algorithm Analysis (2/2)

Goal: combine these equations to get formula for error e(i) = x(i) – x:

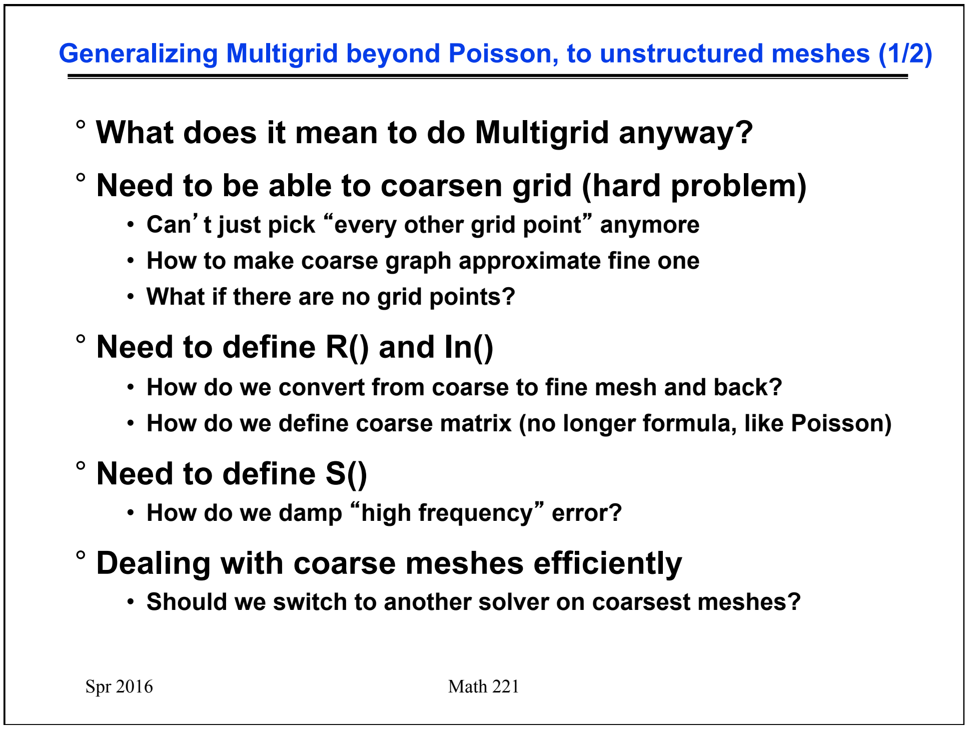
x(i) = S·x(i) + b(i)/3	subtract x = S·x + b(i)/3 to get e(i) = S·e(i)
r(i) = T(i)*x(i) - b(i)	subtract 0 = T(i)*x – b(i) to get r(i) = T(i)*e(i)
d(i) = In·(T(i-1)$^{-1}$·(R·r(i)))	*assume coarse problem solved exactly*
x(i) = x(i) - d(i)	subtract x = x to get e(i) = e(i) – d(i)
x(i) = S·x(i) + b(i)/3	subtract x = S·x + b(i)/3 to get e(i) = S·e(i)

Substitute each equation into later ones to get

e(i) = S · (I - In·(T(i-1)$^{-1}$·(R·T(i)))) · S · e(i) ≡ M · e(i)

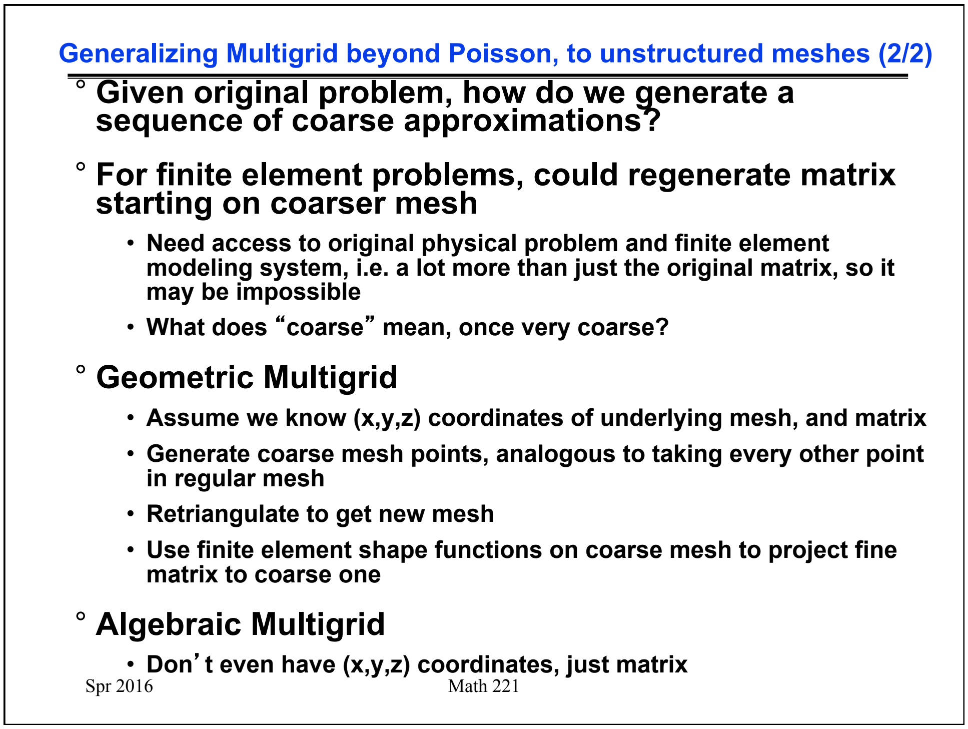
Theorem: For 1D Poisson problem, the eigenvalues of M are either 0 or 1/9, independent of dimension.

This means multigrid converges in a bounded number of steps, independent of dimension.



Generalizing Multigrid beyond Poisson, to unstructured meshes (1/2)

- What does it mean to do Multigrid anyway?
- Need to be able to coarsen grid (hard problem)
  - Can’t just pick “every other grid point” anymore
  - How to make coarse graph approximate fine one
  - What if there are no grid points?
- Need to define R() and In()
  - How do we convert from coarse to fine mesh and back?
  - How do we define coarse matrix (no longer formula, like Poisson)
- Need to define S()
  - How do we damp “high frequency” error?
- Dealing with coarse meshes efficiently
  - Should we switch to another solver on coarsest meshes?



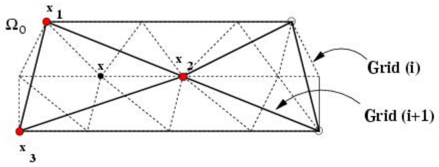
Generalizing Multigrid beyond Poisson, to unstructured meshes (2/2)

- Given original problem, how do we generate a sequence of coarse approximations?
- For finite element problems, could regenerate matrix starting on coarser mesh
  - Need access to original physical problem and finite element modeling system, i.e. a lot more than just the original matrix, so it may be impossible
  - What does “coarse” mean, once very coarse?
- Geometric Multigrid
  - Assume we know (x,y,z) coordinates of underlying mesh, and matrix
  - Generate coarse mesh points, analogous to taking every other point in regular mesh
  - Retriangulate to get new mesh
  - Use finite element shape functions on coarse mesh to project fine matrix to coarse one
- Algebraic Multigrid
  - Don’t even have (x,y,z) coordinates, just matrix

Geometric Multigrid

- **Need matrix, (x,y,z) coordinates of mesh points**
  - **Not minimum information (just matrix), but a little more**
  - **Based on work of Guillard, Chan, Smith**
- **Finite element intuition**
  - **Goal is to compute function, represented by values at points**
  - **Think of approximation by piecewise linear function connecting points**
    - **Easy in 1D, need triangulated mesh in 2D, 3D uses tetrahedra**
- **Geometric coarsening**
  - **Pick a subset of coarse points "evenly spaced" among fine points**
    - **Use Maximal Independent Sets**
    - **Try to keep important points, like corners, edges of object**
  - **Retriangulate coarse points**
    - **Try to approximate answer by piecewise linear function on new triangles**
  - **Let columns of P ("prolongator") be values at fine grid points given values at coarse ones**
    - **Generalizes Interpolation operator "In" from before**
  - **$A_{coarse} = P^T A_{fine} P$ – Galerkin method**
  - **For Poisson: $P = In$, $P^T = 2*R$, $T_{coarse} = 2 * P^T * T_{fine} * P$**

Example of Geometric Coarsening



Simple Greedy Algorithm:

```
repeat
    pick unmarked vertex
    mark it and its neighbors
until no unmarked vertices
```

Examples of meshes from geometric coarsening

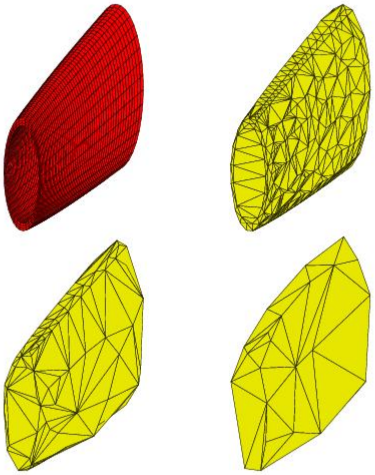
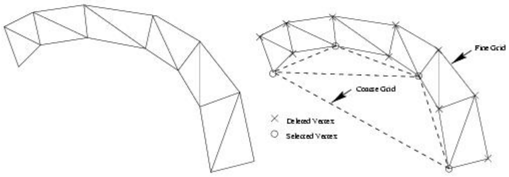


Figure 6: Sample input grid and coarse grids

What can go wrong

- Care needed so coarse grid preserves geometric features of fine grid
  - Label fine grid points as corner, edge, face, interior
  - Delete edges between same-labeled points in different features
  - Ex: delete edges between points on different faces
  - Keeps feature represented on coarse meshes

Pathological example:



How to coarsen carefully

Example - classify vertices - modify graph

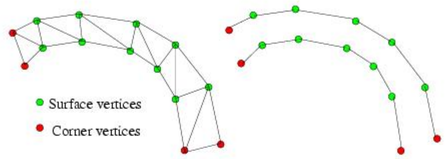


Figure 1: Modify graph

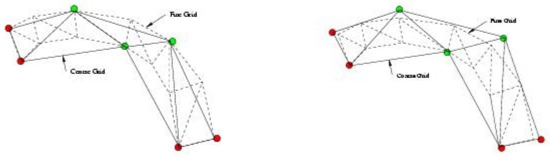


Figure 2: New mesh - fixed mesh

Algebraic Multigrid

- No information beyond matrix needed
- Galerkin still used to get $A_{coarse} = P^T A_{fine} P$
- Prolongator P defined purely algebraically
  - Cluster fine grid points into nearby groups
    - Can use Maximal Independent Sets or Graph Partitioning
    - Use magnitude of entries of A_{fine} to cluster
  - Associate one coarse grid node to each group
  - To interpolate coarse grid values to associated fine grid point, can use properties of PDE, eg elasticity:
    - Rigid body modes of coarse grid point
    - Let coarse grid point have 6 dof (3 translation, 3 rotation)
    - Can be gotten from QR factorization of submatrix of A_{fine}
  - Can also apply smoother to resulting columns of P
  - “Smoothed Aggregation”
- Based on work of Vanek, Mandel, Brezina, Farhat, Roux, Bulgakov, Kuhn …

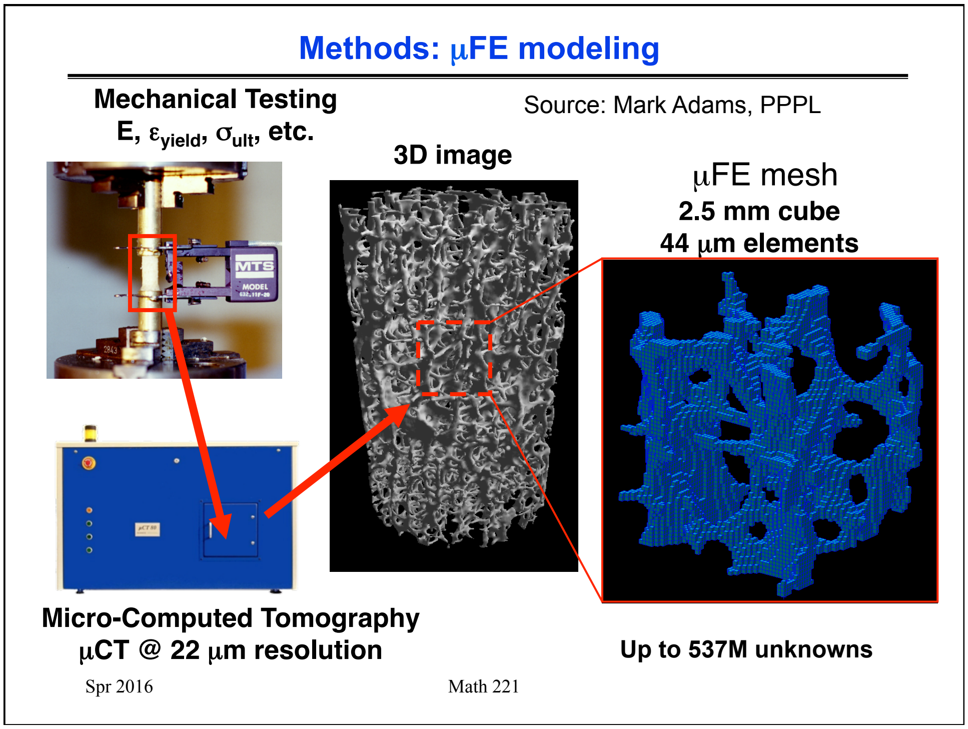
Parallel Smoothers for Unstructured Multigrid

- **Weighted Jacobi**
  - **Easy to implement, hard to choose weight**
- **Gauss-Seidel**
  - **Works well, harder to parallelize because of triangular solve**
- **Polynomial Smoothers**
  - **Chebyshev polynomial $p(A_{fine})$**
  - **Easy to implement (just SpMVs with A_{fine})**
  - **Chebyshev chooses $p(y)$ such that**
    - **$|1 - p(y)\, y| = min$ over interval $[\lambda^*, \lambda_{max}]$ estimated to contain eigenvalues of A_{fine}**

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: μFE modeling

Source: Mark Adams, PPPL

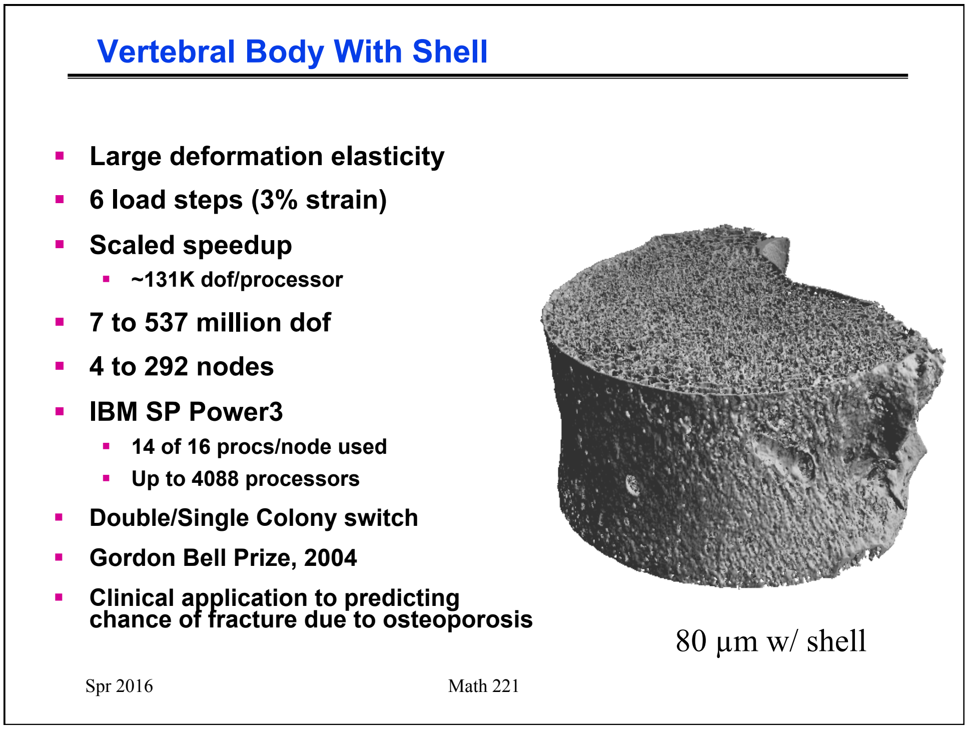
Mechanical Testing
E, ε_{yield}, σ_{ult}, etc.

3D image

μFE mesh
2.5 mm cube
44 μm elements

Micro-Computed Tomography
μCT @ 22 μm resolution

Up to 537M unknowns

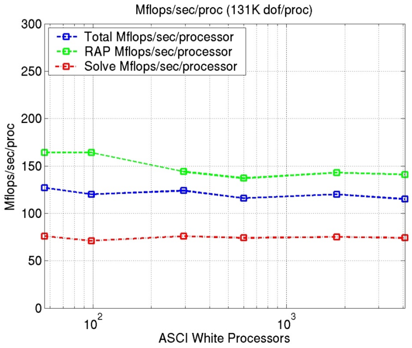


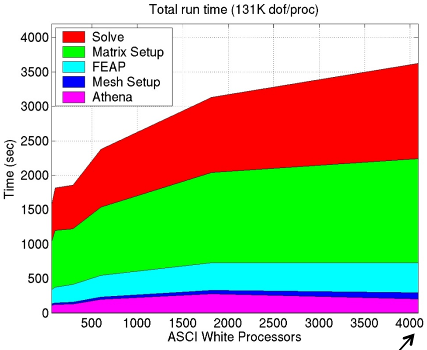
Vertebral Body With Shell

- **Large deformation elasticity**
- **6 load steps (3% strain)**
- **Scaled speedup**
  - **~131K dof/processor**
- **7 to 537 million dof**
- **4 to 292 nodes**
- **IBM SP Power3**
  - **14 of 16 procs/node used**
  - **Up to 4088 processors**
- **Double/Single Colony switch**
- **Gordon Bell Prize, 2004**
- **Clinical application to predicting chance of fracture due to osteoporosis**

80 μm w/ shell

131K dof / proc (weak scaling) - Flops/sec/proc
.47 Teraflops - 4088 processors





537M dof !

Conclusions

° **Multigrid can be very fast**

- **Provably "optimal" (does O(N) flops to compute N unknowns) for many problems in which one can show that using a coarse grid gives a good approximation**
- **Can be parallelized effectively**

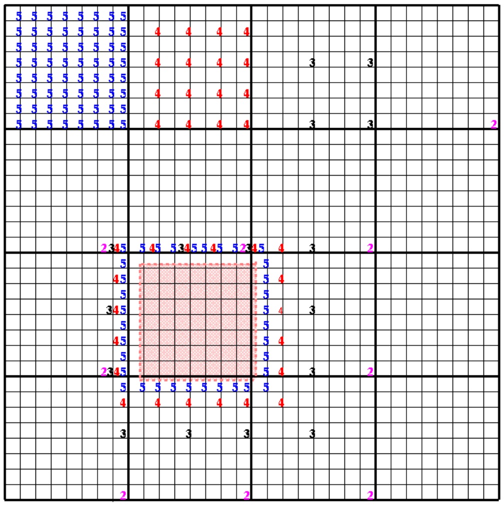
° **Multigrid can be complicated to implement**

- **Lots of software available (see web page for pointers)**
  - **PETSc (includes many iterative solvers, interfaces to other packages, Python interface, runs in parallel)**
  - **ACTS (repository for PETSc and other packages)**
    - Offers periodic short courses on using these packages
  - **MGNET**
- **Sample Matlab implementation for 1D and 2D Poisson**
  - **See class web page under "Matlab Programs for Homework Assignments"**

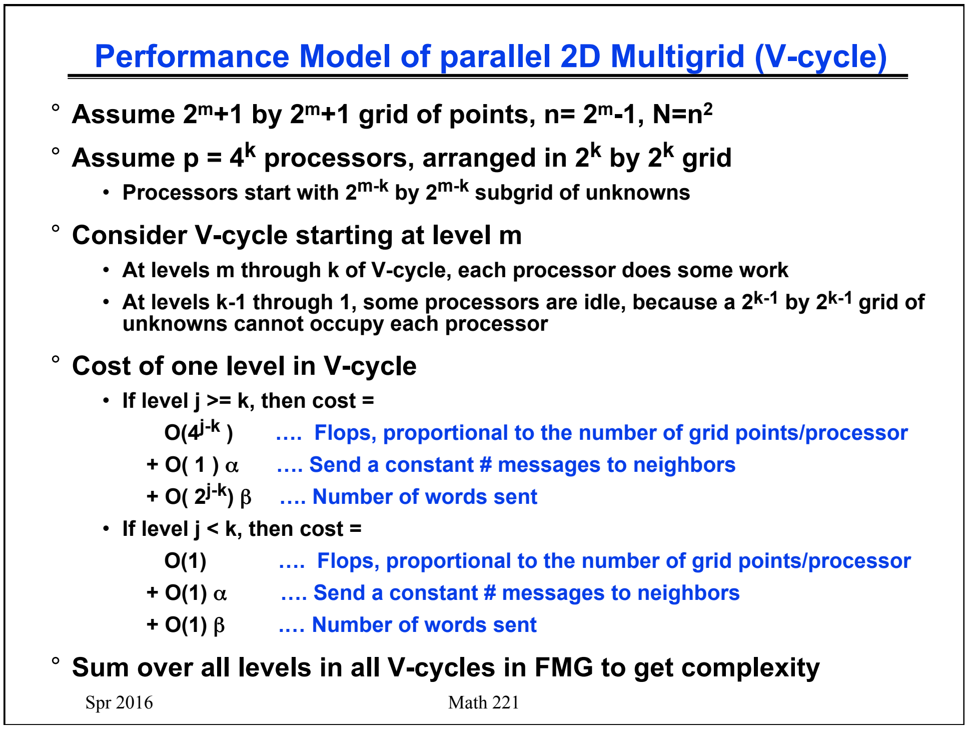
Extra slides

Parallel 2D Multigrid

- Multigrid on 2D requires nearest neighbor (up to 8) computation at each level of the grid
- Start with n=2^m+1 by 2^m+1 grid (here m=5)
- Use an s by s processor grid (here s=4)



Communication pattern for Multigrid on 33 by 33 mesh with 4 by 4 processor grid
In top processor row, grid points labeled m are updated in problem P(m) of multigrid
Pink processor owns grid points inside pink box
In lower half of graph, grid points labeled m need to be communicated to pink processor
in problem P(m) of multigrid

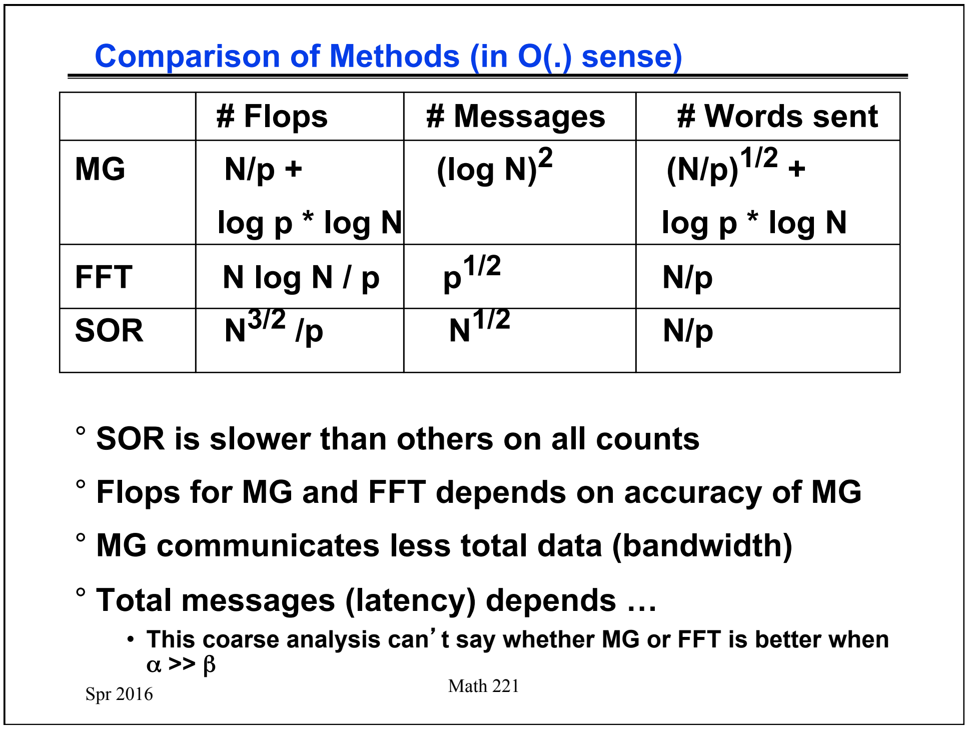


Performance Model of parallel 2D Multigrid (V-cycle)

- Assume 2^m+1 by 2^m+1 grid of points, $n= 2^m-1$, $N=n^2$
- Assume $p = 4^k$ processors, arranged in 2^k by 2^k grid
  - Processors start with 2^{m-k} by 2^{m-k} subgrid of unknowns
- Consider V-cycle starting at level m
  - At levels m through k of V-cycle, each processor does some work
  - At levels k-1 through 1, some processors are idle, because a 2^{k-1} by 2^{k-1} grid of unknowns cannot occupy each processor
- Cost of one level in V-cycle
  - If level j >= k, then cost =

    $O(4^{j-k})$ …. Flops, proportional to the number of grid points/processor
    $+ O(1)\,\alpha$ …. Send a constant # messages to neighbors
    $+ O(2^{j-k})\,\beta$ …. Number of words sent
  - If level j < k, then cost =

    $O(1)$ …. Flops, proportional to the number of grid points/processor
    $+ O(1)\,\alpha$ …. Send a constant # messages to neighbors
    $+ O(1)\,\beta$ …. Number of words sent
- Sum over all levels in all V-cycles in FMG to get complexity



Comparison of Methods (in O(.) sense)

	# Flops	# Messages	# Words sent
MG	N/p + $\log p * \log N$	$(\log N)^2$	$(N/p)^{1/2}$ + $\log p * \log N$
FFT	$N \log N / p$	$p^{1/2}$	N/p
SOR	$N^{3/2}/p$	$N^{1/2}$	N/p

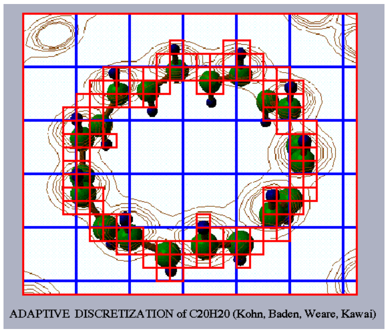
- SOR is slower than others on all counts
- Flops for MG and FFT depends on accuracy of MG
- MG communicates less total data (bandwidth)
- Total messages (latency) depends …
  - This coarse analysis can't say whether MG or FFT is better when $\alpha >> \beta$

Practicalities

- In practice, we don’t go all the way to $P^{(1)}$
- In sequential code, the coarsest grids are negligibly cheap, but on a parallel machine they are not.
  - Consider 1000 points per processor
  - In 2D, the surface to communicate is 4xsqrt(1000) ~= 128, or 13%
  - In 3D, the surface is 1000-8^3 ~= 500, or 50%
- See Tuminaro and Womble, SIAM J. Sci. Comp., v14, n5, 1993 for analysis of MG on 1024 nCUBE2
  - on 64x64 grid of unknowns, only 4 per processor
    - efficiency of 1 V-cycle was .02, and on FMG .008
  - on 1024x1024 grid
    - efficiencies were .7 (MG Vcycle) and .42 (FMG)
    - although worse parallel efficiency, FMG is 2.6 times faster that V-cycles alone
  - nCUBE had fast communication, slow processors

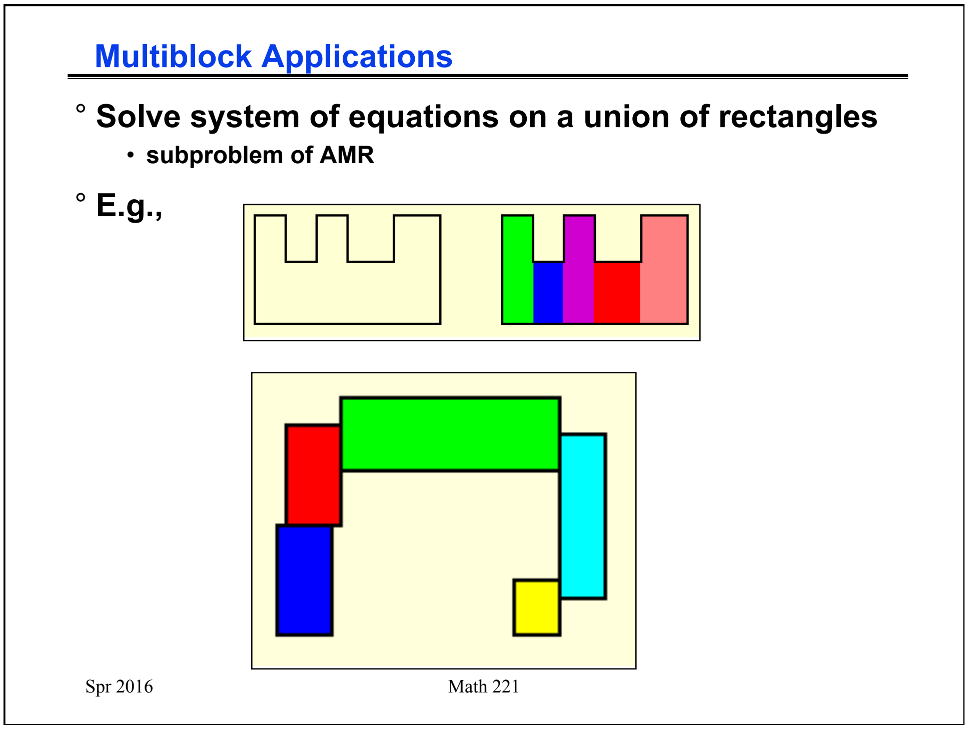
Multigrid on an Adaptive Mesh

- For problems with very large dynamic range, another level of refinement is needed
- Build adaptive mesh and solve multigrid (typically) at each level



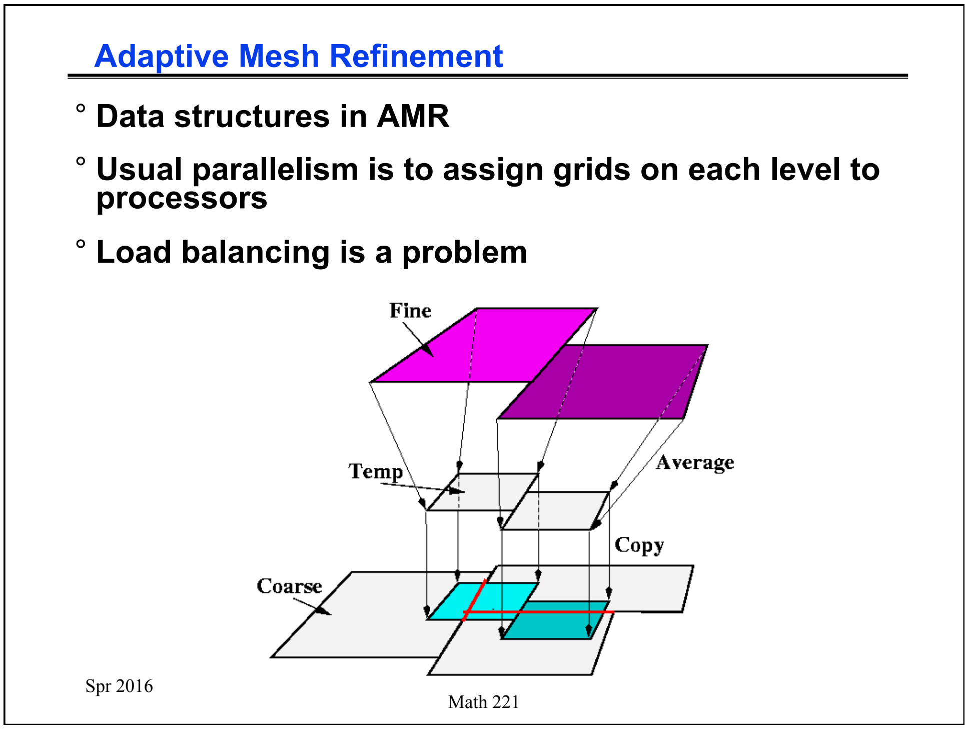
ADAPTIVE DISCRETIZATION of C20H20 (Kohn, Baden, Weare, Kawai)

- Can’t afford to use finest mesh everywhere



Multiblock Applications

- Solve system of equations on a union of rectangles
  - subproblem of AMR
- E.g.,

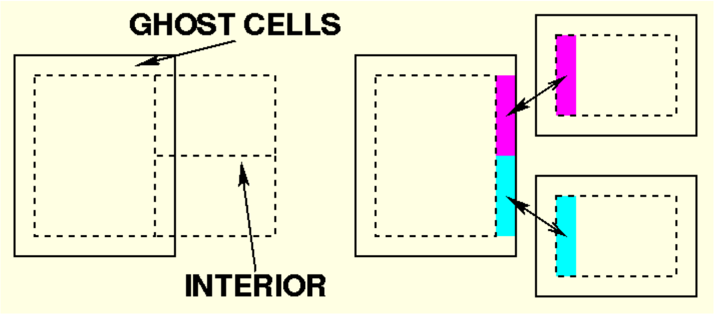


Adaptive Mesh Refinement

- Data structures in AMR
- Usual parallelism is to assign grids on each level to processors
- Load balancing is a problem

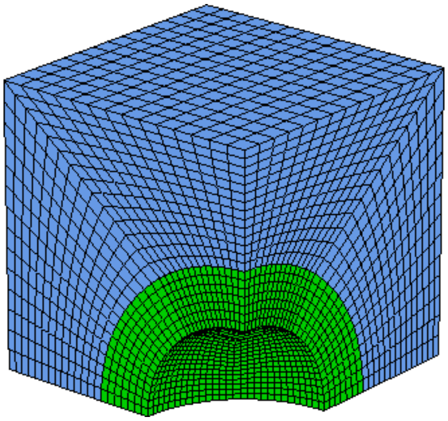
Support for AMR

- Domains in Titanium designed for this problem
- Kelp, Boxlib, and AMR++ are libraries for this
- Primitives to help with boundary value updates, etc.



Multigrid on an Unstructured Mesh

- Another approach to variable activity is to use an unstructured mesh that is more refined in areas of interest
- Adaptive rectangular or unstructured?
  - Numerics easier on rectangular
  - Supposedly easier to implement (arrays without indirection) but boundary cases tend to dominate code



Up to 39M unknowns on 960 processors,
With 50% efficiency (Source: M. Adams)