CS 267: Multigrid

Kathy Yelick

www.cs.berkeley.edu/~yelick/cs267_s07
### Algorithms for 2D (3D) Poisson Equation (N = n^2 (n^3) 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 (N^{7/3})</td>
<td>N</td>
<td>N^{3/2} (N^{5/3})</td>
<td>N (N^{4/3})</td>
</tr>
<tr>
<td>Jacobi</td>
<td>N^2 (N^{5/3})</td>
<td>N (N^{2/3})</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Explicit Inv.</td>
<td>N^2</td>
<td>log N</td>
<td>N^2</td>
<td>N^2</td>
</tr>
<tr>
<td>Conj.Gradients</td>
<td>N^{3/2} (N^{4/3})</td>
<td>N^{1/2(1/3)} *log N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Red/Black SOR</td>
<td>N^{3/2} (N^{4/3})</td>
<td>N^{1/2} (N^{1/3})</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Sparse LU</td>
<td>N^{3/2} (N^2)</td>
<td>N^{1/2}</td>
<td>N*log N (N^{4/3})</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

## Algorithms for 2D (3D) Poisson Equation ($N = n^2 (n^3)$ 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>$N^2 (N^{7/3})$</td>
<td>$N$</td>
<td>$N^{3/2} (N^{5/3})$</td>
<td>$N (N^{4/3})$</td>
</tr>
<tr>
<td>° Jacobi</td>
<td>$N^2 (N^{5/3})$</td>
<td>$N (N^{2/3})$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>° Explicit Inv.</td>
<td>$N^2$</td>
<td>$\log N$</td>
<td>$N^2$</td>
<td>$N^2$</td>
</tr>
<tr>
<td>° Conj.Gradients</td>
<td>$N^{3/2} (N^{4/3})$</td>
<td>$N^{1/2} (1/3) \log N$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>° Red/Black SOR</td>
<td>$N^{3/2} (N^{4/3})$</td>
<td>$N^{1/2} (N^{1/3})$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>° Sparse LU</td>
<td>$N^{3/2} (N^2)$</td>
<td>$N^{1/2}$</td>
<td>$N \log N (N^{4/3})$</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.

Review of Previous Lectures

° Review Poisson equation
° Overview of Methods for Poisson Equation
° Jacobi’s method
° Red-Black SOR method
° Conjugate Gradients
° FFT

\{ Reduce to sparse-matrix-vector multiply
Need them to understand Multigrid \}

° Multigrid
° Comparison of methods
Poisson’s equation in 1D: \( \frac{\partial^2 u}{\partial x^2} = f(x) \)

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

Graph and “stencil”
2D Poisson’s equation

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

\[ T = \begin{pmatrix}
4 & -1 & -1 & -1 \\
-1 & 4 & -1 & -1 \\
-1 & -1 & 4 & -1 \\
-1 & -1 & -1 & 4 \\
\end{pmatrix} \]

° 3D is analogous

Graph and “stencil”
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 \( n \times n \) 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
Multigrid Methods

- We studied several iterative methods
  - Jacobi, SOR, Guass-Seidel, Red-Black variations, Conjugate Gradients (CG)
  - All use sparse matrix-vector multiply (nearest neighbor communication on grid)

- Key problem with iterative methods is that:
  - detail (short wavelength) is correct
  - convergence controlled by coarse (long wavelength) structure

- In simple methods one needs of order $N^2$ iterations to get good results
  - Ironically, one goes to large $N$ (fine mesh) to get detail
  - If all you wanted was coarse structure, a smaller mesh would be fine

- Basic idea in multigrid is key in many areas of science
  - Solve a problem at multiple scales

- We get coarse structure from small $N$ and fine detail from large $N$
  - Good qualitative idea but how do we implement?
Gauss Seidel is Slow I

- Take Laplace’s equation in the Unit Square with initial guess as $\phi = 0$ and boundary conditions that are zero except on one side.
- For $N=31 \times 31$ Grid it takes around 1000 ($N^2$) iterations to get a reasonable answer.

Boundary Conditions

Exact Solution
Gauss Seidel is Slow II

1 Iteration

10 Iterations

100 Iterations

1000 Iterations

Slide source: Geoffrey Fox and (indirectly) Ulrich Ruede
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
Same Big Idea used elsewhere

° Replace fine problem by coarse approximation, recursively

° Multilevel Graph Partitioning (METIS):
  • Replace graph to be partitioned by a coarser graph, obtained via Maximal Independent Set
  • Given partitioning of coarse grid, refine using Kernighan-Lin

° Barnes-Hut (and Fast Multipole Method) 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

° All examples depend on coarse approximation being accurate enough (at least if we are far enough away)
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 $T(i) \cdot x(i) = b(i)$
• $P^{(m)} , P^{(m-1)} , \ldots , P^{(1)}$ is sequence of problems from finest to coarsest

<table>
<thead>
<tr>
<th>$2$</th>
<th>$2$</th>
<th>$2$</th>
<th>$2$</th>
<th>$2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P^{(3)}$: 1D grid of 9 points</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 unknowns</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Points labeled 2 are part of next coarser grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$1$</th>
<th>$1$</th>
<th>$1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P^{(2)}$: 1D grid of 5 points</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 unknowns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Points labeled 1 are part of next coarser grid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| $P^{(1)}$: 1D grid of 3 points |
| 1 unknown |
Multigrid Sketch in 2D

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

![Grids](image)

$P^{(3)}$: 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 Hierarchy

- Relax
- Restrict
- Interpolate
- Relax
- Relax
- Relax
- Relax
- Relax

3x3
5x5
9x9
17x17
35x35

Slide source: Geoffrey Fox
Basic Multigrid Ideas

- In picture, relax is application of standard iteration scheme
  - “solve” short wavelength solution at a given level
  - i.e. use Jacobi, Gauss-Seidel, Conjugate Gradient

- Interpolation is taking a solution at a coarser grid and interpolating to find a solution at half the grid size

- Restriction is taking solution at given grid and averaging to find solution at coarser grid
Multigrid Operators

- For problem $P^{(i)}$ at varying coarsening levels ($i$, grid size grows with $i$):
  - $b(i)$ is the Right Hand Side (RHS) and
  - $x(i)$ is the current estimated solution

- 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 $I_n(i-1)$ maps approx. solution $x(i-1)$ to $x(i)$
  - Interpolates solution on coarse grid $P^{(i-1)}$ to fine grid $P^{(i)}$
  - $x(i) = I_n(i-1)(x(i-1))$

- The solution operator $S(i)$ takes $P^{(i)}$ and improves solution $x(i)$
  - Uses “weighted” Jacobi or SOR on single level of grid
  - $x_{\text{improved}}(i) = S(i) (b(i), x(i))$

- Overall algorithm, then details of operators

both live on grids of size $2^i-1$
Multigrid Operators

• For problem $P^{(i)}$ at varying coarsening levels ($i$, grid size grows with $i$):
  • $b(i)$ is the Right Hand Side (RHS) and
  • $x(i)$ is the current estimated solution

• 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 Operators

- For problem $P^{(i)}$ at varying coarsening levels ($i$, grid size grows with $i$):
  - $b(i)$ is the Right Hand Side (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
  - $x_{\text{improved}} (i) = S(i)(b(i), x(i))$

- Overall algorithm, then details of operators
Multigrid Operators

- For problem $P^{(i)}$ at varying coarsening levels ($i$, grid size grows with $i)$:
  - $b(i)$ is the Right Hand Side (RHS) and
  - $x(i)$ is the current estimated solution

\[
\text{both live on grids of size } 2^i-1
\]

- 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
    - $x_{\text{improved}} (i) = S(i) (b(i), x(i))$

- Overall algorithm, then details of operators

03/23/07 CS267 Lecture 16
Multigrid V-Cycle Algorithm

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
    solve recursively
    $x(i) = S(i) (b(i), x(i))$
    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 $x(i)$
This is called a V-Cycle

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

° On a serial machine

• Work at each point in a V-cycle is $O(\text{the number of unknowns})$
• Cost of Level $i$ is $(2^i-1)^2 = O(4^i)$ for a 2D grid
• If finest grid level is $m$, total time is:
  $\sum_{i=1}^{m} O(4^i) = O(4^m)$ for a 2D grid
  $= O(\# \text{unknowns})$ in general

° On a parallel machine (PRAM)

• with one processor per grid point and free communication, each step in the V-cycle takes constant time
• Total V-cycle time is $O(m) = O(\log \#\text{unknowns})$
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 for each call to FMG
- people also use Ws and other compositions

Serial time: \( \sum_{i=1}^{m} O(4^i) = O(4^m) = O(\text{# unknowns}) \)

PRAM time: \( \sum_{i=1}^{m} O(i) = O(m^2) = O(\log^2 \text{# unknowns}) \)
Complexity of Solving Poisson’s Equation

- Theorem: error after one FMG call:
  - error_after \( \leq (0.5 \times \text{error}_{\text{before}}) \)
  - independent of \# unknowns
  - At least 1 bit each time

- 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

° The solution operator, $S(i)$, is a weighted Jacobi

° Consider the 1D problem

\[ x(j) := \frac{1}{2} \left( x(j-1) + x(j+1) + b(j) \right) \]

° At level $i$, pure Jacobi replaces:

\[ x(j) := \frac{1}{3} \left( x(j-1) + x(j) + x(j+1) + b(j) \right) \]

° Weighted Jacobi uses:

\[ x(j) := \frac{1}{3} \left( x(j-1) + x(j) + x(j+1) + b(j) \right) \]

° In 2D, similar average of nearest neighbors
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
Each level in a V-Cycle reduces the error in one part of the frequency domain.

**Schematic Description of Multigrid**

Error Component \( \alpha(j) \)

P(1) Upper half of frequencies on P(4)
P(2) Upper half of frequencies on P(3)

Frequency j
The Restriction Operator R(i) - Details

- The restriction operator, R(i), takes
  - a problem P(i) with RHS b(i) and
  - maps it to a coarser problem P(i-1) with RHS b(i-1)

- In 1D, average values of neighbors
  - \( x_{\text{coarse}}(i) = \frac{1}{4} x_{\text{fine}}(i-1) + \frac{1}{2} x_{\text{fine}}(i) + \frac{1}{4} x_{\text{fine}}(i+1) \)

- In 2D, average with all 8 neighbors (N,S,E,W,NE,NW,SE,SW)
Interpolation Operator \text{In}(i-1): details

- The interpolation operator \text{In}(i-1), takes a function on a coarse grid \(P(i-1)\), and produces a function on a fine grid \(P(i)\).

- In 1D, linearly interpolate nearest coarse neighbors
  - \(x_{\text{fine}}(i) = x_{\text{coarse}}(i)\) if the fine grid point \(i\) is also a coarse one, else
  - \(x_{\text{fine}}(i) = 1/2 \times x_{\text{coarse}}(\text{left of } i) + 1/2 \times x_{\text{coarse}}(\text{right of } i)\)

- In 2D, interpolation requires averaging with 4 nearest neighbors (NW, SW, NE, SE)
Convergence Picture of Multigrid in 1D

- **True Solution**
  - Graph showing a step function.

- **Right Hand Side**
  - Graph showing a step function.

- **Error of each iteration**
  - Graph showing a comparison of errors over iterations.

- **norm(res(m+1))/norm(res(m))**
  - Graph showing the ratio of residual norms for consecutive iterations.

- **norm(res(m))**
  - Graph showing the residual norm over iterations.
Convergence Picture of Multigrid in 2D
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 \( s=4 \)).
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 $j \geq k$, then cost =  
    \[
    O(4^j) \text{ Flops, proportional to the number of grid points/processor}  
    + O(1) \alpha \text{ Send a constant # messages to neighbors}  
    + O(2^j) \beta \text{ Number of words sent}  
    \]  
  • If level $j < k$, then cost =  
    \[
    O(1) \text{ Flops, proportional to the number of grid points/processor}  
    + O(1) \alpha \text{ Send a constant # messages to neighbors}  
    + O(1) \beta \text{ Number of words sent}  
    \]  
° Sum over all levels in all V-cycles in FMG to get complexity
### Comparison of Methods (in O(.) sense)

<table>
<thead>
<tr>
<th></th>
<th># Flops</th>
<th># Messages</th>
<th># Words sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>MG</td>
<td>$\frac{N}{p} + \log p \times \log N$</td>
<td>$(\log N)^2$</td>
<td>$(\frac{N}{p})^{1/2} + \log p \times \log N$</td>
</tr>
<tr>
<td>FFT</td>
<td>$\frac{N \log N}{p}$</td>
<td>$p^{1/2}$</td>
<td>$\frac{N}{p}$</td>
</tr>
<tr>
<td>SOR</td>
<td>$\frac{N^{3/2}}{p}$</td>
<td>$N^{1/2}$</td>
<td>$\frac{N}{p}$</td>
</tr>
</tbody>
</table>

- 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 $4\sqrt{1000} \approx 128$, or 13%
  • In 3D, the surface is $1000-8^3 \approx 500$, or 50%

  • 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 V-cycle) 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

° 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)
Multigrid on an Unstructured Mesh

° Need to partition graph for parallelism

° 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
  • Use “maximal independent sets” again
  • How to make coarse graph approximate fine one

° Need to define $R()$ and $\text{In}()$
  • How do we convert from coarse to fine mesh and back?

° Need to define $S()$
  • How do we define coarse matrix (no longer formula, like Poisson)

° Dealing with coarse meshes efficiently
  • Should we switch to using fewer processors on coarse meshes?
  • Should we switch to another solver on coarse meshes?
Source of Unstructured Finite Element Mesh: Vertebra

Source: M. Adams, H. Bayraktar, T. Keaveny, P. Papadopoulos, A. Gupta
Multigrid for nonlinear elastic analysis of bone

Mechanical testing for material properties

Micro Computed Tomography @ 22 μm resolution

3D image

μFE mesh
2.5 mm³
44 μm elements

Up to 537M unknowns
4088 Processors (ASCI White)
70% parallel efficiency

Source: M. Adams et al
Extra Slides
Preconditioning

° One can change $A$ and $b$ by preconditioning

$$M_1^{-1} A (M_2^{-1} M_2)x = M_1^{-1} b$$

° is same equation as before for any choice of matrices $M_1$ and $M_2$

° All these choices are designed to accelerate convergence of iterative methods

° $A_{new} = M_1^{-1} A M_2^{-1}$

° $x_{new} = M_2 x$

° $b_{new} = M_1^{-1} b$

° We choose $M_1$ and $M_2$
to make our standard methods perform better

° There are specialized preconditioning ideas and perhaps better general approaches such as multigrid and Incomplete LU (ILU) decomposition

$A_{new} x_{new} = b_{new}$

has same form as above and we can apply any of the methods that we used on $A x = b$
Multigrid Philosophically

° Suppose we have a finest level $M(1)$ with $N$ by $N$ points (in 2D)
° Then the $k$’th coarsest approximation $M(k)$ to this has $N/2^k$ by $N/2^k$ points
° One way to think about Multigrid is that $M(k+1)$ can form a preconditioner to $M(k)$ so that one can replace natural matrix $A(k)$ by $A^{-1}(k+1)A(k)$
  • $A^{-1}(k+1)A(k)$ is a nicer matrix than $A(k)$ and iterative solvers converge faster as long wavelength terms have been removed
° Basic issue is that $A(k)$ and $A(k+1)$ are of different size so we need prolongation and restriction to map solutions at level $k$ and $k+1$
  • We apply this idea recursively
Multigrid Algorithm: procedure MG(level, A, u, f)

° if level = coarsest then
  • solve coarsest grid equation by another method “exactly”
° else
  • smooth $A^{\text{level}}u = f \quad (m_1 \text{ times})$
  • Compute residual $r = f - A^{\text{level}}u$
  • Restrict $F = Rr \quad (R \text{ is Restriction Operator})$
  • Call MG( level + 1, $A^{(\text{level}+1)}$, V, F) \quad (m_c \text{ times})
  • Interpolate $v = P V$ (Interpolate new solution at this level)
  • correct $u_{\text{new}} = u + v$
  • smooth $A^{\text{new}}u = f \quad (m_2 \text{ times})$ and
  • set $u = u_{\text{new}}$
° endif
° endprocedure
Multigrid Cycles

° The parameter $m_c$ determines the exact strategy for how one iterates through different meshes.

° One ends up with a full cycle as shown.

Slide source: Geoffrey Fox