CS 267 Applications of Parallel Computers

Lecture 25:

Solving Linear Systems arising from PDEs - II

James Demmel

http://www.cs.berkeley.edu/~demmel/cs267_Spr99

Review of Last Lecture and Outline

- 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

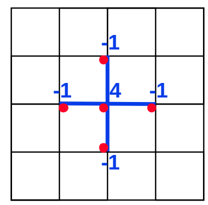
- Multigrid
- Comparison of methods

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 Poisson Equation with N unknowns

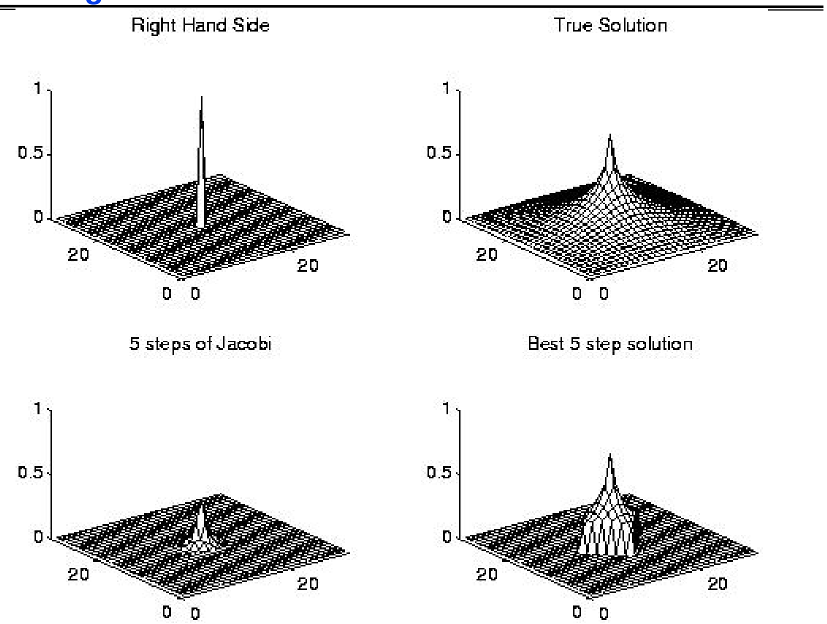
Algorithm	Serial	PRAM	Memory	#Procs
Dense LU	N^3	N	N^2	N^2
Band LU	N^2	N	$N^{3/2}$	N
Jacobi	N^2	N	N	N
Explicit Inv.	N^2	log N	N^2	N^2
Conj.Grad.	$N^{3/2}$	$N^{1/2}$ *log N	N	N
RB SOR	$N^{3/2}$	$N^{1/2}$	N	N
Sparse LU	$N^{3/2}$	$N^{1/2}$	N*log N	N
FFT	N*log N	log N	N	N
Multigrid	N	$\log^2 N$	N	N
Lower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

PRAM is an idealized parallel model with zero cost communication

Multigrid Motivation

- Recall that Jacobi, SOR, CG, or any other sparse-matrix-vector-multiply-based algorithm can only move information one grid call at a time
- Can show that decreasing error by fixed factor c<1 takes Ω(log n) steps
- Convergence to fixed error < 1 takes Ω(log n) steps
- Therefore, converging in O(1) steps requires moving information across grid faster than to one neighboring grid cell per step

Multigrid Motivation



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 before

° **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):**

- Approximate leaf node of QuadTree containing many particles by Center-of-Mass and Total-Mass (or multipole expansion)
- Approximate internal nodes of QuadTree by combining expansions of children
- Force in any particular node approximated by combining expansions of a small set of other nodes

° **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

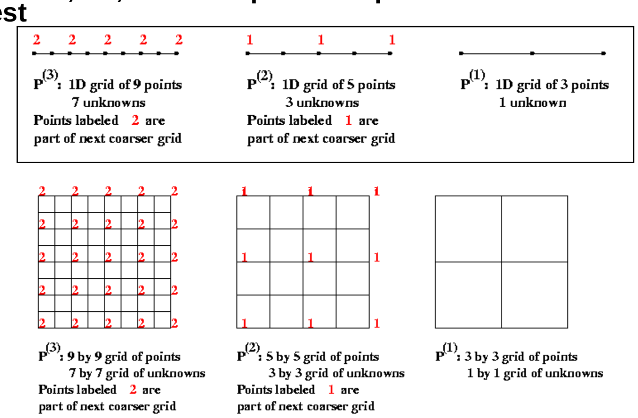
° **Second way:**
- Think of error as a sum of sine curves of different frequencies
- Same idea as FFT solution, but not explicit in algorithm
- Each call to Multgrid responsible for suppressing coefficients of sine curves of the lower half of the frequencies in the error (pictures later)

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



Multigrid Operators

- For problem $P^{(i)}$:
  - b(i) is the RHS and
  - x(i) is the current estimated solution
  - (T(i) is implicit in the operators below.)

  both live on grids of size 2^i-1
- All the following operators just average values on neighboring grid points
  - Neighboring grid points on coarse problems are far away in fine problems, so information moves quickly on coarse problems
- The restriction operator R(i) maps $P^{(i)}$ to $P^{(i-1)}$
  - Restricts problem on fine grid $P^{(i)}$ to coarse grid $P^{(i-1)}$ by sampling or averaging
  - b(i-1)= R(i) (b(i))
- The interpolation operator In(i-1) maps an approximate solution x(i-1) to an 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 computes an improved solution x(i) on same grid
  - Uses "weighted" Jacobi or SOR
  - $x_{improved}$ (i) = S(i) (b(i), x(i))
- Details of these operators after describing overall algorithm

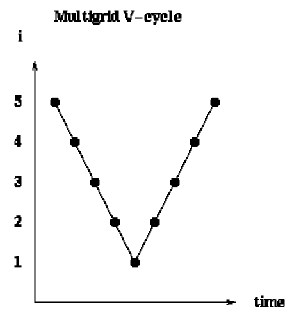
Multigrid V-Cycle Algorithm

```
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
     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



Complexity of a V-Cycle

° On a serial machine

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

° 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 #unknowns)

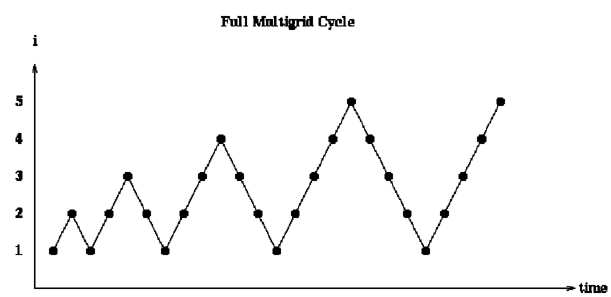
Full Multigrid (FMG)

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

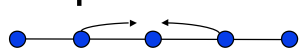
° **PRAM time:** $\Sigma_{i=1}^{m}$ $O(i) = O(m^2) = O(\log^2 \#\text{ unknowns})$

Complexity of Solving Poisson's Equation

- Theorem: error after one FMG call <= 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

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



- At level i, pure Jacobi replaces:

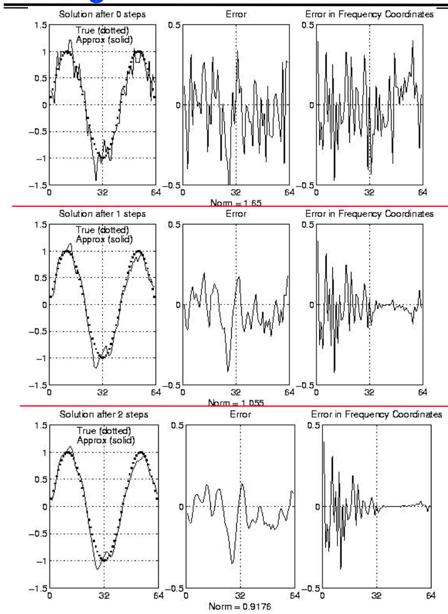
  x(j) := 1/2 (x(j-1) + x(j+1) + b(j))

- Weighted Jacobi uses:

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

- In 2D, similar average of nearest neighbors

Weighted Jacobi chosen to damp high frequency error



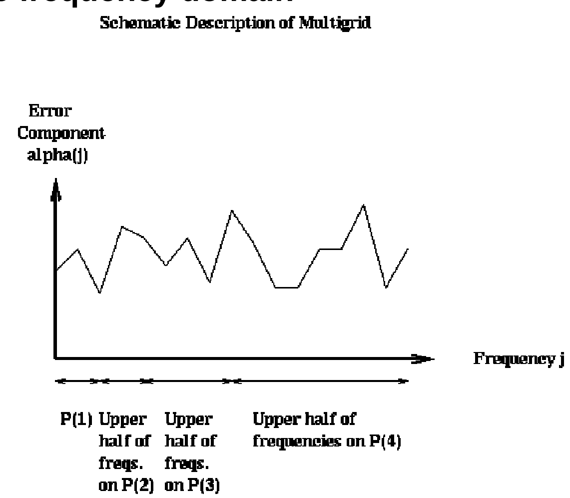
Initial error
"Rough"
Lots of high frequency components
Norm = 1.65

Error after 1 Jacobi step
"Smoother"
Less high frequency component
Norm = 1.055

Error after 2 Jacobi steps
"Smooth"
Little high frequency component
Norm = .9176,
won't decrease much more

Multigrid as Divide and Conquer Algorithm

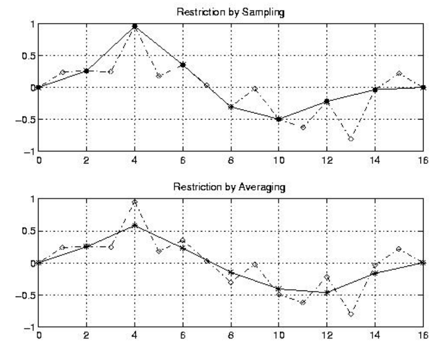
° **Each level in a V-Cycle reduces the error in one part of the frequency domain**



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)

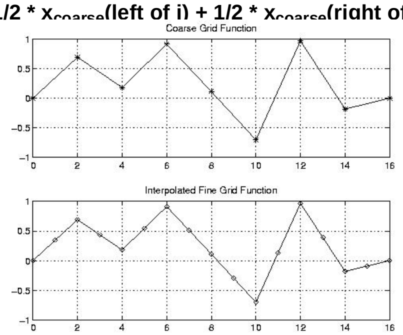
- In 1D, average values of neighbors
  - $x_{coarse}(i) = 1/4 * x_{fine}(i-1) + 1/2 * x_{fine}(i) + 1/4 * x_{fine}(i+1)$



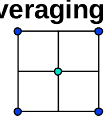
- In 2D, average with all 8 neighbors (N,S,E,W,NE,NW,SE,SW)

Interpolation Operator

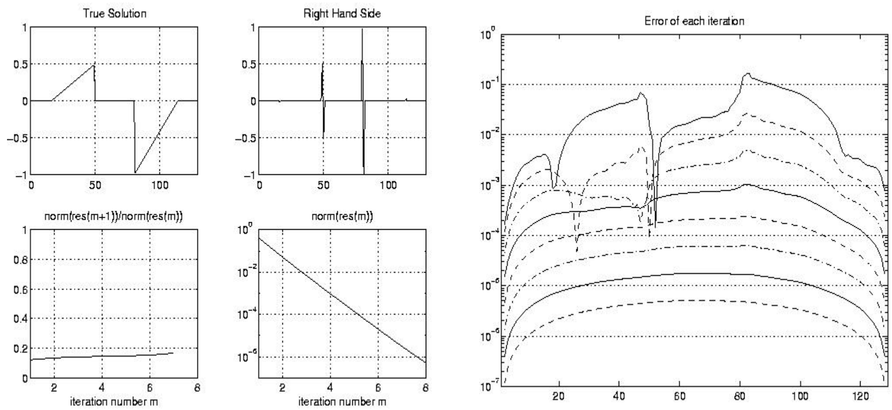
- The interpolation operator In(i-1), takes a function on a coarse grid $P^{(i-1)}$, and produces a function on a fine grid $P^{(i)}$
- In 1D, linearly interpolate nearest coarse neighbors
  - $x_{fine}(i) = x_{coarse}(i)$ if the fine grid point i is also a coarse one, else
  - $x_{fine}(i) = 1/2 * x_{coarse}(\text{left of } i) + 1/2 * x_{coarse}(\text{right of } i)$



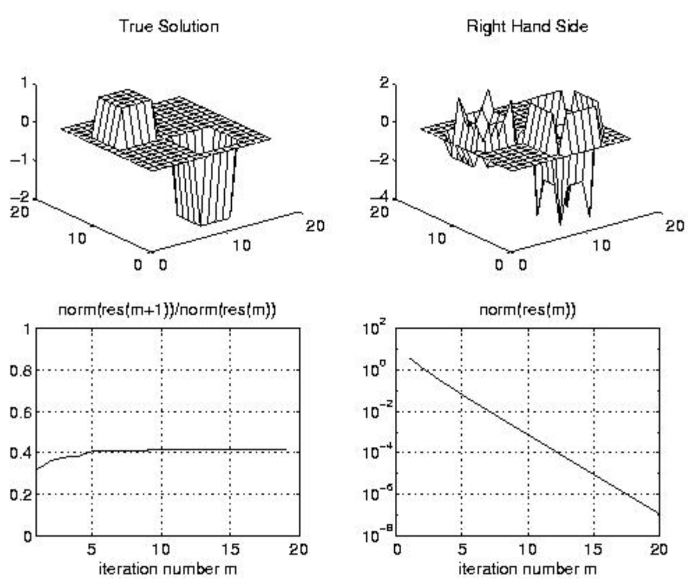
- In 2D, interpolation requires averaging with 2 or 4 nearest neighbors (NW,SW,NE,SE)



Convergence Picture of Multigrid in 1D

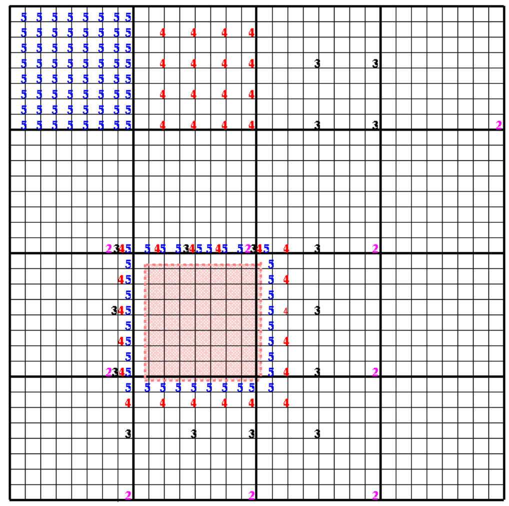


Convergence Picture of Multigrid in 2D



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)



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

- Assume 2^m+1 by 2^m+1 grid of unknowns, $n= 2^m+1$, $N=n^2$
- Assume $p = 4^k$ processors, arranged in 2^k by 2^k grid
  - Each processor starts 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

	# 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 $\alpha >> \beta$

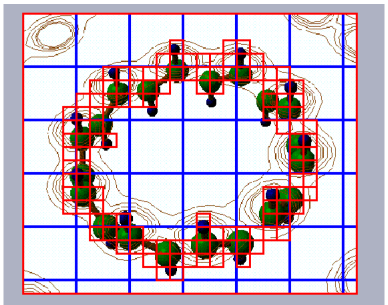
Practicalities

- 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)

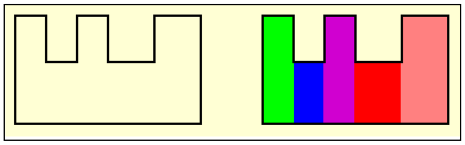
° Can't afford to use finest mesh everywhere

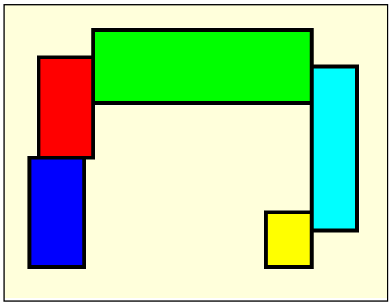
Multiblock Applications

° Solve system of equations on a union of rectangles

- subproblem of AMR

° E.g.,



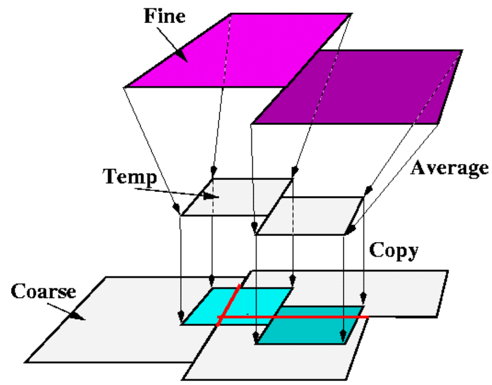


Adaptive Mesh Refinement

° Data structures in AMR

° Usual parallelism is to deal grids on each level to processors

° Load balancing is a problem

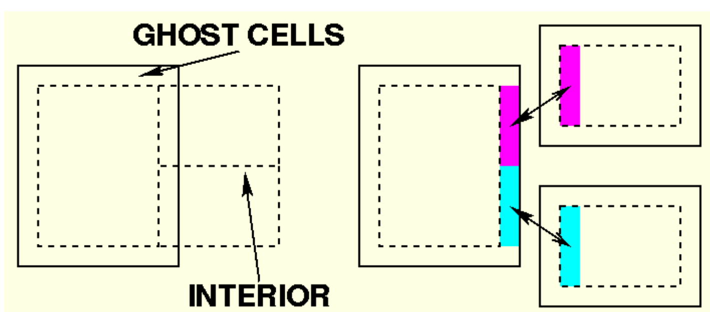


Support for AMR

° Domains in Titanium designed for this problem

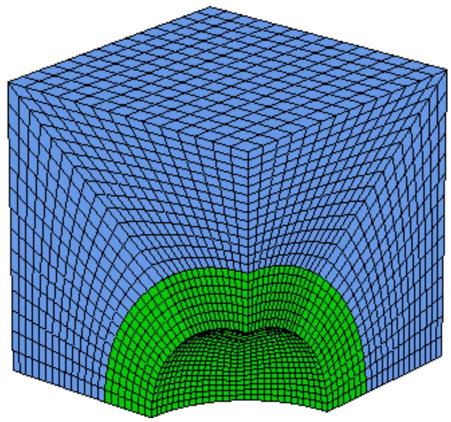
° Kelp, Boxlib, and AMR++ are libraries for this

° 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
- Controversy over adaptive rectangular vs. unstructured
  - Numerics easier on rectangular
  - Supposedly easier to implement (arrays without indirection) but boundary cases tend to dominate code



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 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?
- Next Lecture by Mark Adams on unstructured Multigrid
  - Solved up to 39M unknowns on 960 processors with 50% efficiency