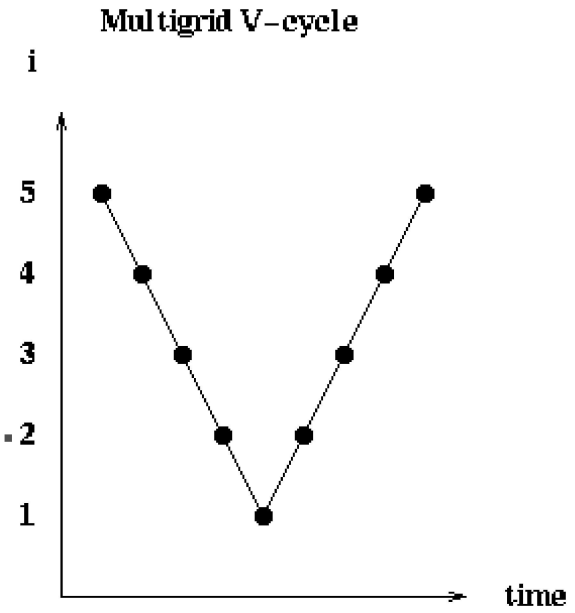
Multigrid Operators (2)

- 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
  - $x_{i, improved} = S_i (b_i , x_i)$
- Details of these operators follow after describing overall algorithm

Multigrid V-Cycle Algorithm (recursive)



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

Complexity of a V-Cycle

° On a serial machine

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

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

° On a parallel machine (PRAM)

- with one processor per grid point and free communication, each step in the V-cycle takes constant time, O(1)
- Total V-cycle time is $O(m) = O(\log \#\text{unknowns})$

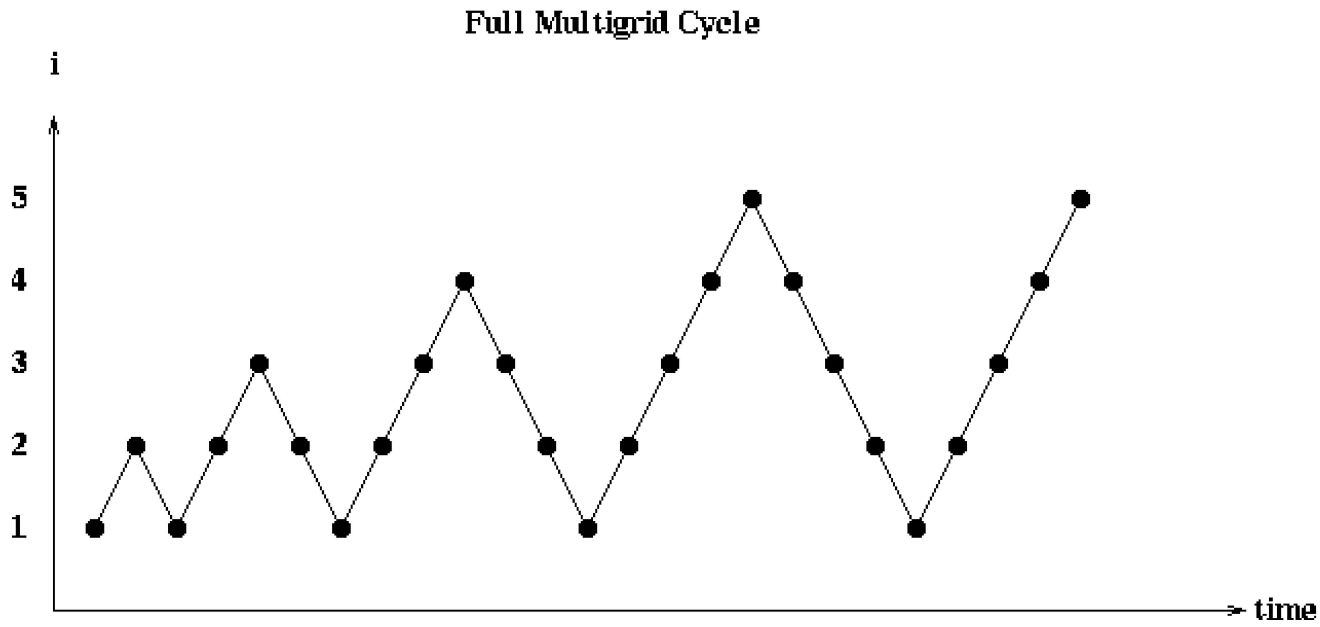
Full Multigrid (FMG)

- 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 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

- one also use "W"s and other compositions

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

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

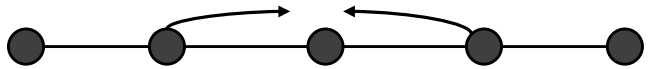
Complexity of Solving Poisson's Equation

- **Theorem: error ε after one FMG call is ≤ c times the error before, where c < 1/2, and independent of # unknowns**
- **x^k = FMG (b, x^{k-1}) ==> $\varepsilon(x^k) < 1/2\ \varepsilon(x^{k-1})$ (i.e., at least 1 bit per FMG iteration. x^k =solution after k^{th} FMG iteration)**
- **Corollary: We can make the error ε < tol, for any fixed tolerance in a fixed number of steps, independent of size of the finest grid**
- **This is the most important convergence property of MG, distinguishing it from other methods, which converge more slowly for large grids**
- **Total complexity is just proportional to the cost of one FMG call**

The Solution Operator S_i - Details

° The solution operator S_i , is a weighted Jacobi op.

° 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:

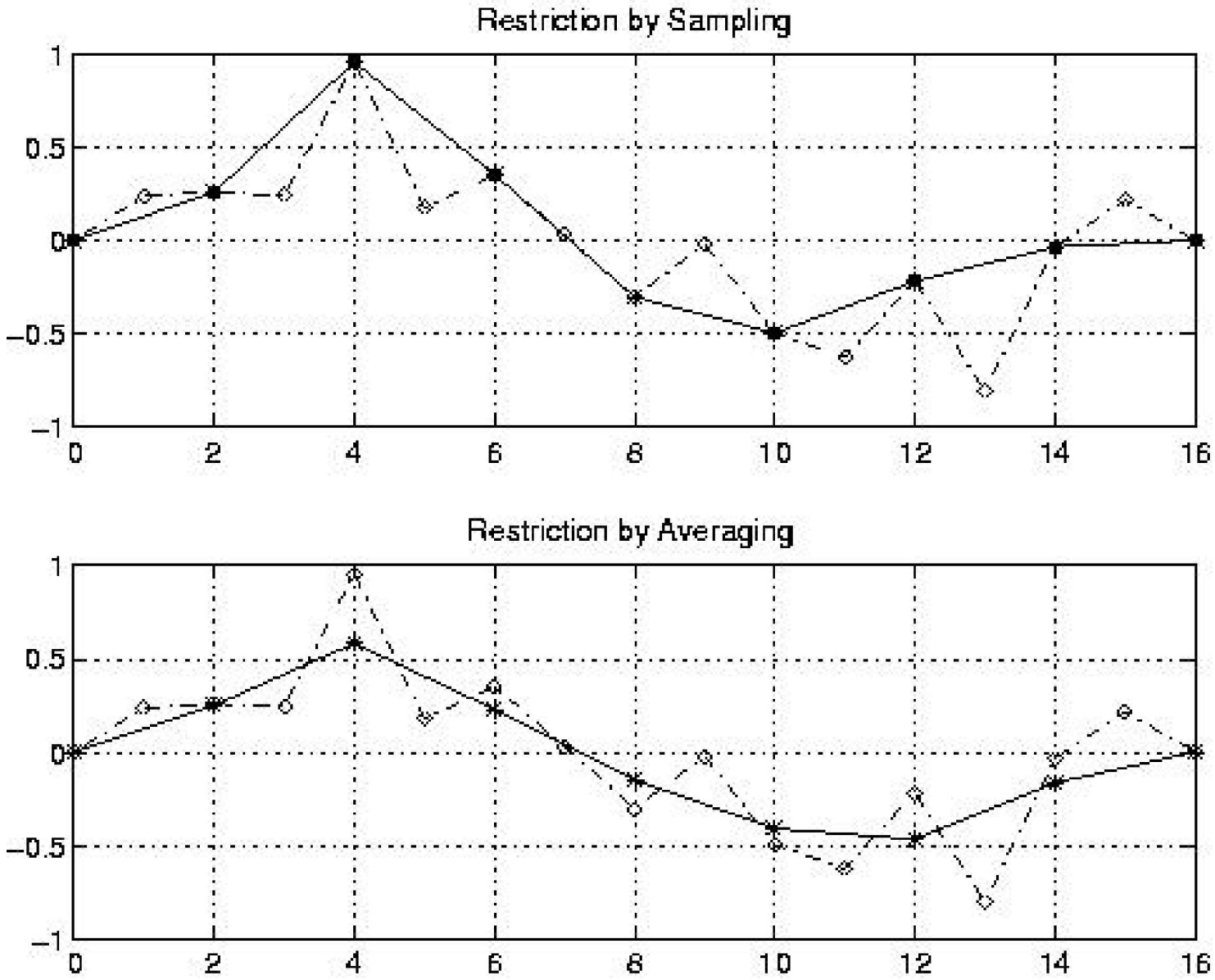
$$x(j) := 1/3\ (x(j-1) + x(j) + x(j+1) + b(j))$$

° In 2D, similar average of nearest neighbors

The Restriction Operator R_i - Details

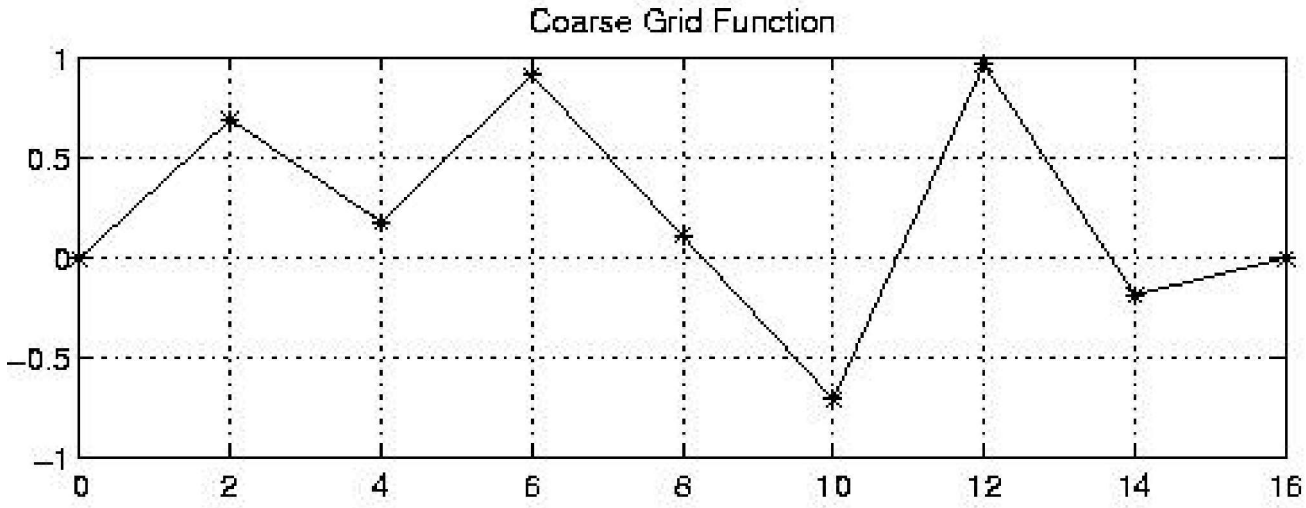
- The restriction operator R_i takes
  - a problem $P^{(i)}$ with RHS b_i and
  - maps it to a coarser problem $P^{(i-1)}$ with RHS b_{i-1}
- In 1D, average values of neighbors
  - $x_{coarse}(j) = 1/4 * x_{fine}(j-1) + 1/2 * x_{fine}(j) + 1/4 * x_{fine}(j+1)$

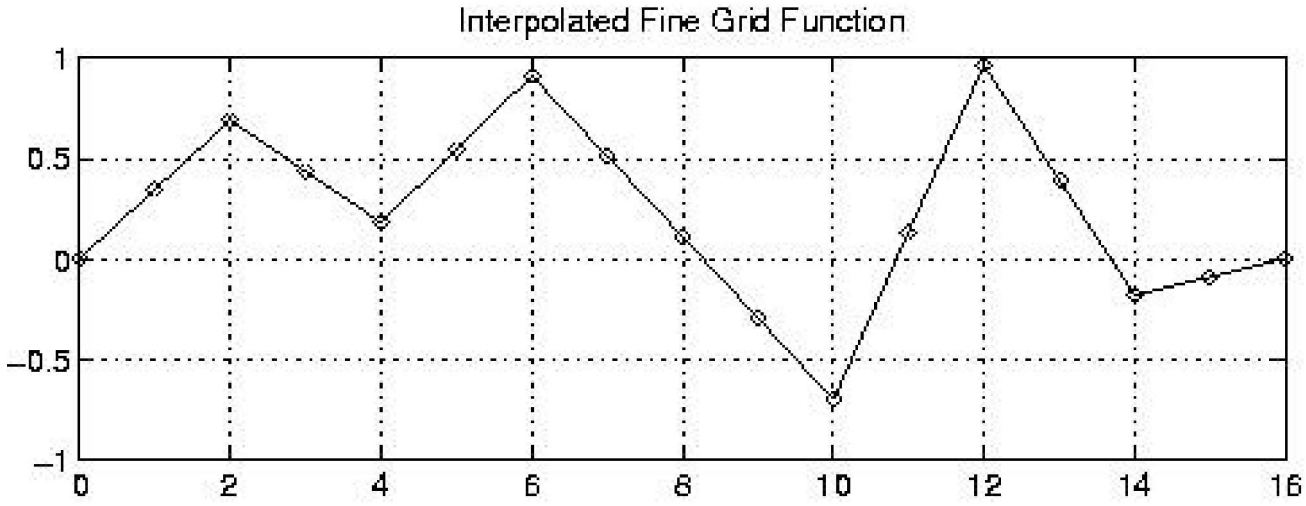
The Restriction Operator R_i - Details



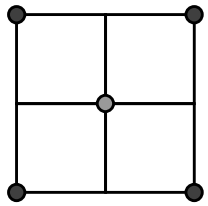
° **In 2D, average with all 8 neighbors (N,S,E,W,NE,NW,SE,SW)**

Interpolation Operator In_i





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



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

Performance Model of parallel 2D Multigrid (2)

° **Cost of one level (j) or $P^{(j)}$ in V-cycle**

- **If level j >= k, then cost =**

  $O(4^{j-k})$ …. **Flops, proportional to the number of grid points/processor**

  + O(1) α …. **Send a constant # messages to neighbors**

  + $O(2^{j-k})$ β …. **Number of words sent**

- **If level j < k, then cost =**

  O(1) …. **Flops, proportional to the number of grid points/processor**

  + O(1) α …. **Send a constant # messages to neighbors**

  + O(1) β …. **Number of words sent**

° **Sum over all levels in all V-cycles in FMG to get complexity**

Comparison of Methods (using p processors)

	# 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 4*sqrt(1000) ~= 128, or 13%
  - In 3D, the surface is $1000\text{-}8^3$ ~= 500, or 50%
  - Data locality ratio α (large α is preferred):

$$\alpha = \frac{\text{computation time between two communication steps}}{\text{communication time}}$$

- Apply communication avoiding Jacobi iterations!
- Dealing with coarse meshes efficiently
  - Should we switch to using fewer processors on coarse meshes?
  - Should we switch to another solver on coarse meshes?