

## Topic 4: Multigrid Methods (continued)

This lecture covers some variations on the basic V-cycle multigrid algorithm and various issues involved with programming the multigrid on a parallel processing system.

Let's recall some notation which will help summarize the algorithm variations and their implementation in a parallel program:

- The equation to be solved:

$$\mathcal{L}u = -f ,$$

where  $u$  is an approximation to the unknown function,  $\mathcal{L}$  is a differential operator, e.g.,  $\partial^2/\partial x^2 + \partial^2/\partial y^2$  for Poisson's equation in 2-D, and  $f$  is the given source function.

- The residual:

$$r = \mathcal{L}u + f .$$

- The correction:

$$v = u_{\text{exact}} - u ,$$

which obeys

$$\mathcal{L}v = -r .$$

- The restriction  $\mathcal{R}$  and prolongation  $\mathcal{P}$  operators which transfer a function between coarse and fine grids, e.g.,

$$u_{\text{coarse}} = \mathcal{R}u_{\text{fine}}, \quad u_{\text{fine}} = \mathcal{P}u_{\text{coarse}}.$$

- The smoothing operation:

$$u \leftarrow \mathcal{S}u,$$

where  $\mathcal{S}$  can for example be taken to be the one Gauss-Seidel sweep of the grid using checkerboard updating.

### *Multigrid schedules*

The multigrid method uses a series of grids or lattices. Approximate solutions and auxiliary functions are *smoothed* on a particular grid and then *transferred* to the next coarser grid (restriction) or to the next finer grid (prolongation). The order in which the grids are visited is called the *multigrid schedule*.

The multigrid method is usually based on an elementary *two-grid* step:

1. If this is the coarsest grid, e.g., Poisson's equation with only one interior grid points, then solve the equation exactly and return. Otherwise, continue with the following steps.
2. Do  $\nu_{\text{pre}}$  pre-smoothing steps  $u \leftarrow \mathcal{S}u$ .

3. Find the residual  $r = \mathcal{L}u + f$ .
4. Restrict it to the coarser lattice  $R = \mathcal{R}r$ .
5. Approximate the correction on the coarser lattice  $V = 0$ .
6. Solve  $\mathcal{L}V = -R$  by calling the two-grid routine recursively  $\gamma$  times.
7. Prolongate the correction to the finer lattice  $v = \mathcal{P}V$ .
8. Apply the correction  $u \leftarrow u + v$ .
9. Do  $\nu_{\text{post}}$  post-smoothing steps  $u \leftarrow \mathcal{S}u$ .

Note in step number 6 that the two-grid routine can be called more than once depending on the integer parameter  $\gamma$ . Two popular choices for this parameter are

- $\gamma = 1$  gives the simplest V-cycle schedule.
- $\gamma = 2$  gives a variation called the W-cycle.

If the parameters  $\gamma$ ,  $\nu_{\text{pre}}$  and  $\nu_{\text{post}}$  are held fixed, then the schedule is called a *fixed schedule*. However, the parameters can be varied in the course of the computation

based for example on some convergence criterion: this is referred to as an *adaptive schedule*.

A multigrid program based on these types of schedules can be constructed as follows:

- Choose an approximation to  $u$  on the finest grid.
- Perform the following steps until the desired accuracy has been achieved:
  - Select the parameters  $\gamma$ ,  $\nu_{\text{pre}}$  and  $\nu_{\text{post}}$ .
  - Call the basic recursive two-grid algorithm.

See Chapter 19, Section 6 of *Numerical Recipes* for pictures of the V- and W-cycles, and also for the *Full Multigrid Algorithm*:

1. Solve the problem exactly on the coarsest grid.
2. Repeat the following until the finest grid has been reached:
  - Prolongate to the next finer grid  $u \leftarrow \mathcal{P}u$ .
  - Call the basic two-grid routine with this  $u$  as input.

This algorithm can be pictured as a series of increasingly taller N-cycles. One advantage of the full multigrid algorithm is that a solution is generated on each grid. In

the schedules discussed previously, only the *residuals*  $r$  and *corrections*  $v$  are generated on the coarser grids and propagated between grids.

### *Parallel multigrid: choice of grids*

Let's consider the 2-D Poisson equation with Dirichlet boundary conditions  $\Phi(x, y) = 0$  on the perimeter of a unit square  $0 \leq x, y \leq 1$ .

The simplest type of grid to implement in a parallel program is a uniform vertex-centered grid. On the finest grid, there are

$$N = 2^\ell - 1,$$

interior points in each dimension  $x$  and  $y$ , plus two boundary points. Here,  $\ell$  is the number of multigrid levels. For example, with  $\ell = 3$ , there are  $N + 2 = 2^3 - 1 + 2 = 9$  grid points in each dimension; the next coarser level has 5 points; and the coarsest level has 3 points, i.e., one interior point and two boundary points. With this convention, the number of multigrid levels is  $\log_2(N + 1)$ .

### *Parallel multigrid: domain decomposition*

We now wish to partition the grid among some number of parallel processes. In *domain decomposition* the grid points are assigned to the different processes by partitioning the spatial domain into regions.

A parallel algorithm will usually run most efficiently if it is *load balanced*, i.e., the amount of work done by each process is approximately the same at any given step in the computation. It is therefore reasonable to try to assign

- the same number of grid points to each process, and
- ensure that each domain has the same shape.

It is easy to see that these conditions cannot be satisfied in general. For example with 4 processes and  $N = 7$ , there are 81 grid points, which cannot be divided evenly among the different processes!