

Notes on Programming Multigrid

I find it easiest to use two grids (named `grid` and `new` in Figure 11.2) and to do multigrid in place in these grids. These grids contain `meshSize+1` rows and columns, including the boundaries. I like to declare these so that the indices range from `0:meshSize`. Hence the boundaries are in rows (and columns) `0` and `meshSize`, and the interior points are in rows (and columns) `1:meshSize-1`.

Let `n` be equal to `meshSize-1` below. Let `h` be the "distance" between points. At the tops of the V cycle, `h` is 1 (every point). The other values for `h` are 2, 4, and 8 in the lower levels.

For the compute phases of the V cycle, write a procedure that has argument `h` and that steps through the grids using every `h`'th point:

```
for [i = top + h to bottom by h]
  for [j = h to n by h]
    new[i,j] = average of (old) grid points [i-h,j],
      [i+h,j], [i,j-h] and [i,j+h]
```

Above, `top` is the index of the top boundary of the current strip and `bottom` is the index of the bottom row in the current strip. (In the sequential multigrid program, `top` is 0 and `bottom` is `n`.) Because of the key assumption that `meshSize` is a multiple of `16*numWorkers`, the top boundary for any strip is the bottom row of the strip above it, and the bottom boundary for any strip is the top row of the strip below it. Moreover, the bottom row in a strip is **IN THE SAME PLACE** for all four meshes. Hence you can set `top` and `bottom` **ONCE** in each worker and use it for all meshes.

For the restriction and interpolation steps, let `c` be the distance between points in a coarse mesh and let `f` be the distance between points in the next finer mesh. For example, if `c` is 2, then `f` is 1; if `c` is 4, then `f` is 2; and so on. The keys to implementing restriction and interpolation are to use `c` and `f` and to get the right starting rows, columns, and step sizes.

To implement restriction, just step through the points in the coarse mesh and use the corresponding fine mesh points:

```
for [i = top + c to bottom by c] # step through coarse
  for [j = c to n by c] # rows and columns
    grid[i,j] = 0.5*grid[i,j] + 0.125*(sum of points
      grid[i-f,j], grid[i+f,j], ...)
```

Interpolation is the trickiest operator. If the meshes all lay on top of each other, then the first step is trivial: the fine grid point on top of each coarse grid point is already computed. The second interpolation step is to update all the fine grid points between coarse grid points in either the rows or columns of coarse grid points. The text describes using columns, but with `C` and `MPD` it is better to use rows as follows:

```
for [i = top + c to bottom by c] # step through coarse rows &
  for [j = f to n by c]          # columns with fine points
    grid[i,j] = (grid[i,j-f] + grid[i,j+f]) * 0.5
```

The last step is to iterate through all points in the other rows, which contain only fine grid points:

```
for [i = top + f to bottom by c] # step through rows with
  for [j = f to n by f]          # only fine points
    grid[i,j] = (grid[i-f,j] + grid[i+f,j]) * 0.5
```

I suggest that you take a look at the grids in Figure 11.7 (page 550) and trace out the actions of the above steps. With luck everything will come together and you'll see how this seemingly magic stuff actually works. Good luck!