Solving Poisson's Equation using Adaptive Mesh Refinement

D. F. Martin*and K. L. Cartwright†

October 24, 1996

Abstract

This report discusses an implementation [1] of an Adaptive Mesh Refinement (AMR) Poisson solver which solves Poisson's equation using multigrid relaxation. Local refinement introduces several added issues. Special care has been taken to match the solution across coarse/fine interfaces so that the solution maintains global second order accuracy. The nested mesh hierarchy can be defined by the user, Richardson extrapolation, or a user supplied criterion. Extensive use of Boxlib [2] has reduced the bookkeeping needed by the authors.

1 Motivation

Many physical problems have variations in scale. When solving these problems numerically, high grid resolution is needed to adequately solve the equations. However, there are often also large portions of the domain where high levels of refinement are not needed; using a highly refined mesh in these regions represents a waste of computational effort. Limitations on computational resources often force a compromise on grid resolution, resulting in under-resolution of the problem. By locally refining the mesh only where needed, Adaptive Mesh Refinement (AMR) allows concentration of effort where it is needed, allowing better resolution of the problem.Solutions to Poisson's equation, for example, can exhibit this type of variation, due to local concentration of charges or boundary conditions. A familiarity with multigrid methods for solving linear problems is assumed; a good reference is Briggs [3].

*Ph.D. candidate in mechanical engineering, U.C. Berkeley, supported by the Computational Science Graduate Fellowship Program of the Department of Energy

†Graduate student in physics, U.C. Berkeley, supported by ONR-AASERT N100014-94-1-1033

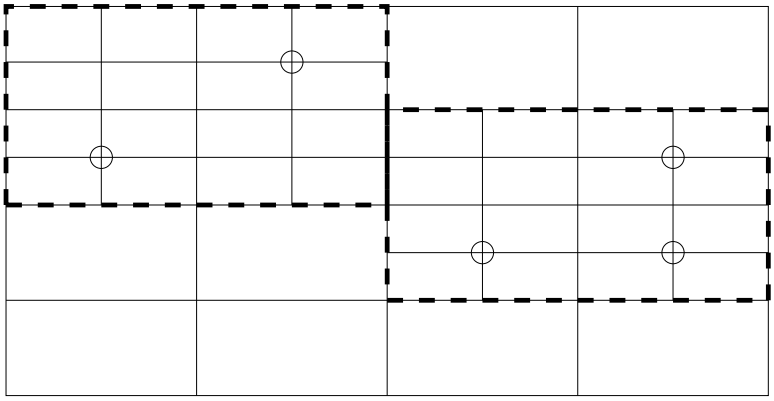

Figure 1: Block structured refinement of cells.

2 Grid Structure

We wish to compute solutions to Poisson's equation,

$$\Delta\phi = \rho \quad \text{on} \quad \Omega. \tag{1}$$

However, this does not fully state the problem because the grid structure has not been defined.

The basic idea will be to start with a uniform coarse mesh which covers the entire domain Ω and compute a solution on this grid. We decide where the error is high in this solution and where we would like increased resolution. We then locally refine in these regions. If desired, these refined regions may be refined still further where necessary to adequately resolve the solution. In principle, there is no limit to the number of times an area can be refined. Any number of methods can be used to identify regions where refinement is desired. We have found that using Richardson extrapolation as an estimator of the local truncation error of the scheme works well for solutions to Poisson's equation.

We are going to use a block structured approach, which is shown in Figure 1. In a block structured approach, patches or sub-domains are refined instead of individual cells; this way areas that need more accuracy can be covered with a relatively small number of refined blocks. This results in more internal area for the amount of coarse/fine interface processing. While a block structured approach results in some unnecessary refinement, the advantages of this approach are better efficiency of data access and better amortization of the overhead costs of irregular operations (like interlevel communication) over a larger number of regular operations on a grid.

This block structured approach was first developed by Berger and Oliger [4] for hyperbolic

PDE's, and was extended to shock hydrodynamics by Berger and Colella [5]. Bell et al [6] found this approach to be highly efficient for three-dimensional systems of hyperbolic conservation laws. Also, Almgren et al [7] have extended this approach to the solution of the variable density incompressible Navier-Stokes equations.

In general, we will have a number of different levels of refinement, $0 \leq \ell \leq \ell_{max}$. Level 0 is the coarsest level, and the level 0 grid covers the entire domain. Finer levels will be made up of one or more refined patches which will not, in general, cover the entire domain (We would hope they do not – that is the entire point of local refinement). Ω^ℓ denotes the union of these patches. $\partial\Omega^\ell$ will denote the boundary of level ℓ. For simplicity, we will often use two levels as an example. In this case, the coarse and fine domains will be denoted by Ω^c and Ω^f, respectively.

Where they do not extend all the way to a physical boundary, refined patches will need to use information from coarser grids to provide boundary conditions. To simplify the passing of information between refinement levels, we will require that the refined grids be "properly nested". Any refined grid must be bordered by either a physical boundary (where the normal physical boundary conditions will apply), another grid at the same level of refinement (where boundary conditions will simply be copied from the adjoining grid), or grids with only one level of refinement less (where boundary conditions will be interpolated from the coarser level). Written formally:

$$R(P(\Omega^{\ell+1})) = \Omega^{\ell+1}, \tag{2}$$

$$P(\Omega^{\ell+1}) \subset (\Omega^\ell) \tag{3}$$

where P is the projection operator, $\ell \rightarrow \ell - 1$ and R is the refining operator $\ell \rightarrow \ell + 1$. Boundaries of $\Omega^{\ell+1}$ can intersect boundaries of Ω^ℓ only at the boundary of the domain. An example of a properly nested grid is shown in Figure 2. In the following algorithm, we will restrict refinements to be by a factor of two for simplicity. This is not essential, and different refinement ratios have been used in other applications of the AMR approach.

3 Discretization

For ease of formulation, ϕ^ℓ and ρ^ℓ on a level ℓ are cell – centered. This simplifies the book-keeping involved in maintaining a solution with different levels of refinement. In contrast, in a nodal point scheme the points on the boundary belong to both the fine grid and the coarse grid. In this cell – centered scheme ϕ^f is only defined on Ω^f and ϕ^c is defined on $\Omega^c - P(\Omega^f)$, where P is the projection operator. In other words, ϕ and ρ are only defined on a grid where it is not overlain by a finer grid. So, we are computing a solution on a composite of valid

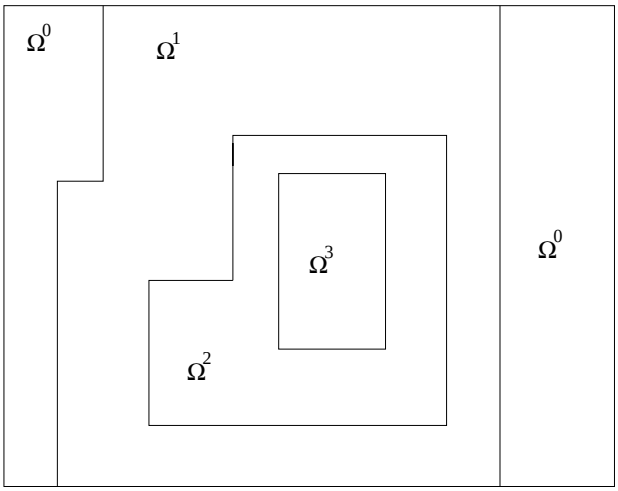

Figure 2: A properly nested grid.

regions on different levels. We will call this union of valid regions the composite grid:

$$\Omega = \sum_{\ell=0}^{\ell_{max}} (\Omega^\ell - P(\Omega^{\ell+1})). \tag{4}$$

Care must be taken to make the coarse/fine grid interfaces work seamlessly and to make sure we get the accuracy we expect. This is a case where the naive approach will not work. As an example, solve $\Delta^c \phi^c = \rho^c$, where ρ^c is the average of ρ^f. Interpolate the boundary values on the border cells of Ω^f to obtain the boundary conditions for the fine mesh. Then solve $\Delta^f \phi^f = \rho^f$. As it turns out, this gives the same accuracy as the coarse grid alone. What happened? ϕ is continuous at the interface between the two domains, Ω^c and Ω^f; however, $\frac{\partial \phi}{\partial n}$ is not continuous across $\partial\Omega^f$. The jump in $\frac{\partial \phi}{\partial n}$ is $O(h^c)$. This discontinuity looks like a charge on the interface. The issue is that this approach only transfers information from the coarse grid to the fine but not back again. The coarse grid passes information in the form of a Dirichlet boundary condition. The fine grid solution uses the the Dirichlet boundary condition; however, it changes the slope of the solution along the interface. Not passing this change of the solution back to the coarse grid is the reason this method fails. In other words, the coarse grid solution never sees the effect of the finer grids. So, we need to enforce both Dirichlet and Neumann matching at the coarse-fine interface. This is the elliptic matching condition that must be met at coarse-fine interfaces.

Because of this, we must construct a composite Laplacian operator $L(\phi)$ which is defined for the different regions in the composite domain. For the region away from $\partial\Omega^f$ in Ω^c the operator is simply the normal coarse grid stencil:

$$(\Delta^c \phi^c)_{ij} = (\phi^c_{i+1,j} + \phi^c_{i-1,j} + \phi^c_{i,j+1} + \phi^c_{i,j-1} - 4\phi^c_{i,j})/(h^c)^2. \tag{5}$$

For regions away from $\partial\Omega^f$ in Ω^f the operator is the usual fine grid stencil:

$$(\Delta^f \phi^f)_{ij} = (\phi^f_{i+1,j} + \phi^f_{i-1,j} + \phi^f_{i,j+1} + \phi^f_{i,j-1} - 4\phi^f_{i,j})/(h^f)^2. \tag{6}$$

In the boundary region we will need to use flux differencing ($\Delta\phi = \nabla \cdot \mathbf{f}$, where $\mathbf{f} = \nabla\phi$) :

$$\begin{aligned} (\Delta\phi)_{ij} &= \nabla \cdot \mathbf{f} \\ &= (f_{i+\frac{1}{2},j} - f_{i-\frac{1}{2},j} + f_{i,j+\frac{1}{2}} - f_{i,j-\frac{1}{2}})/h, \end{aligned} \tag{7}$$

$$\begin{aligned} f &= \nabla\phi : \\ f_{i+\frac{1}{2},j} &= \frac{1}{h}[\phi_{i+1,j} - \phi_{i,j}], && (8) \\ f_{i-\frac{1}{2},j} &= \frac{1}{h}[\phi_{i,j} - \phi_{i-1,j}], && (9) \\ f_{i,j+\frac{1}{2}} &= \frac{1}{h}[\phi_{i,j+1} - \phi_{i,j}], && (10) \\ f_{i,j-\frac{1}{2}} &= \frac{1}{h}[\phi_{i,j} - \phi_{i,j-1}]. && (11) \end{aligned}$$

Using a control volume, the discrete form of the Laplacian operator can be found. For example, consider a fine grid to the left of a coarse grid, as in Figure 3. Placing our control volume around the coarse cell adjacent to the coarse/fine interface and summing the fluxes passing in and out of the box,

$$(\Delta\phi)_{ij} = (f_{i+\frac{1}{2},j} - f^{ave}_{i-\frac{1}{2},j} + f_{i,j+\frac{1}{2}} - f_{i,j-\frac{1}{2}})/h^c, \tag{12}$$

$$f^{ave}_{i-\frac{1}{2},j} = \frac{1}{2}(f^f_{top} + f^f_{bottom}), \tag{13}$$

where

$$f^f_{top/bottom} = \frac{1}{h^f}(\phi^{interpolated} - \phi^f). \tag{14}$$

On the boundary between coarse and fine grids, $\phi^{interpolated}$ is calculated along $\partial\Omega^f$ by interpolation, shown in Figure 3. First, quadratic interpolation is used parallel to the interface to get the intermediate points above and below the coarse grid location. Then another quadratic interpolation is used normal to the interface to get values in the fine grid "ghost cell" positions outside the fine grid. $\Delta^f\phi^f$ for fine cells adjacent to the coarse/fine interface can be computed with the normal stencil using these border values, since this will be equivalent to enforcing the flux matching condition across the interface.

Quadratic interpolation is the minimum necessary to maintain second order accuracy globally. Since the Laplacian is a second derivative operator, it involves a division by h^2. If

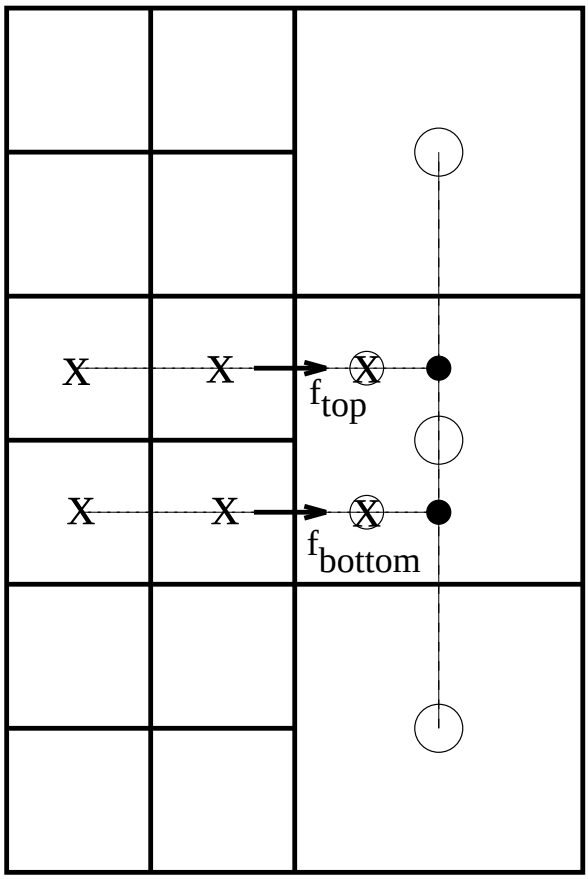

Figure 3: Interpolation on the coarse fine boundary: Use quadratic interpolation through coarse cells (open circles) to get intermediate values (solid circles), then use intermediate value with fine cells (X's) to get "ghost cell" values (circled X's) for computing coarse-fine fluxes (arrows).

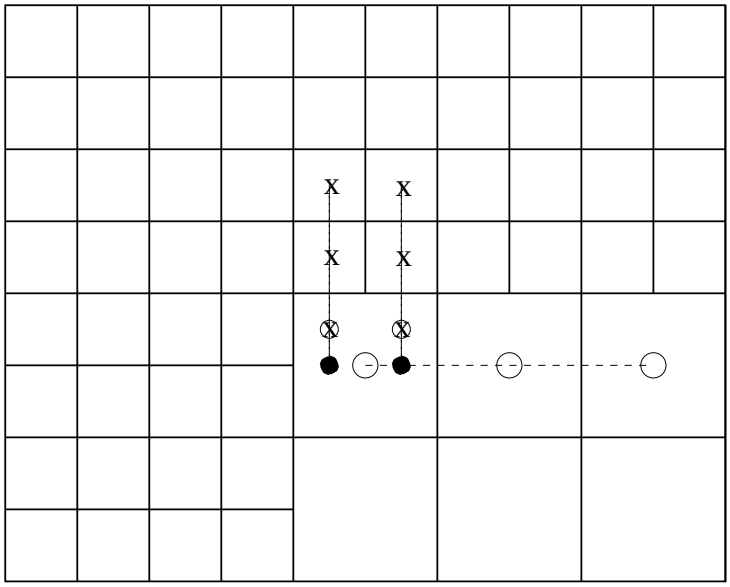

Figure 4: Modified interpolation stencil: Since left coarse cell is covered by a fine grid, use shifted coarse grid stencil (open circles) to get intermediate values (solid circles), then perform final interpolation as before to get values for "ghost cell" values (circled X's). Note that to perform interpolation for the vertical coarse-fine interface, we will need to shift coarse stencil down.

an interpolated quantity has an error of $O(h^p)$, dividing by h^2 in the derivative results in an error of $O(h^{p-2})$. So, with quadratic interpolation ($p = 3$), there is still an $O(h)$ error at the coarse-fine interface. However, since the coarse-fine interface is a set of codimension one, we can drop one order of accuracy and still be $O(h^2)$ globally. Since we want to use this type of quadratic interpolation wherever possible to link the coarse and fine grids, we must use different interpolation stencils for special cases like fine grid corners (Figure 4). If one of the coarse grid cells in the usual stencil is covered by a finer grid, we then shift the stencil so that only coarse cells in $(\Omega^c - P(\Omega^f))$ are used in the interpolation parallel to the coarse-fine interface. If a suitable coarse grid stencil parallel to the interface does not exist, we then drop the order of interpolation and use whatever coarse cells we do have.

Interpolating using both coarse and fine grids and using the same fluxes for both coarse and fine grids links the two grids enough to fix the coarse-fine interface problem. We now can attain the improved accuracy expected from refinement.

4 Multigrid for AMR

The algorithm described here is a cell-centered extension of the node-centered algorithm of Almgren, et al [8]. A similar algorithm was used for steady incompressible flow for by Thompson and Ferziger [9] We want to solve

$$L(\phi) = \rho \quad \text{on} \quad \Omega \tag{15}$$

where $L(\phi)$ is the composite Laplacian operator described above.

For each refinement level from $\ell = 0$ to ℓ_{max}, we will obviously need to store Ω^ℓ, ϕ^ℓ, and ρ^ℓ, where ϕ^ℓ and ρ^ℓ are only defined on $\Omega^\ell - \mathrm{P}(\Omega^{\ell+1})$, that is, wherever Ω^ℓ is not over lain by a finer grid. Since we are using the residual-correction formulation, for each level we will also have to define the residual R^ℓ and the correction e^ℓ on the entire Ω^ℓ.

Now that we have defined the variables which exist on each level, we need to define the operators that we will use to generate the solution. First, the simplest operator is $\Delta^{h_0}(\Psi)$, which is simply the five-point Laplacian on the physical domain Ω^0.

The second operator we will define is the Laplacian on a level, $L^\ell(\Psi)$, which is defined on $\Omega^\ell - \mathrm{P}(\Omega^{\ell+1})$. Away from the boundaries of Ω^ℓ, $L^\ell(\Psi)$ is simply the normal $\Delta^{h_\ell}\Psi^\ell$ that we are used to dealing with. In cells which abut the physical boundary or the $\Omega^\ell/\Omega^{\ell-1}$ boundary, we interpolate values into the border cells, and then evaluate Δ^{h_ℓ} as usual. Finally, for cells adjacent to the $\Omega^\ell/\Omega^{\ell+1}$ boundary, we use our flux matching condition to generate the fluxes across the boundary. Thus, we always interpolate coarse grid information as mentioned earlier, and we always use the flux matching condition to represent the influence of the finer grids.

Third, we define the "no fine" Laplacian, $L^{nf,\ell}(\Psi^\ell, \Psi^{\ell-1})$, where Ψ^ℓ is defined on Ω^ℓ and $\Psi^{\ell-1}$ is defined on $\Omega^{\ell-1} - \mathrm{P}(\Omega^\ell)$. This operator, which is defined on Ω^ℓ, resembles the L^ℓ operator, except that we assume that there is no finer grid information. So, away from the boundaries of Ω^ℓ, it is just the normal five point stencil. At the boundaries, we either use interpolated Dirichlet boundary conditions as in Figures 3 and 4 or physical boundary conditions where Ω^ℓ meets a physical boundary. On the coarsest level, this is the same as the $\Delta^{h_0}(\Psi)$ operator described above.

Finally, we will need something which performs a point relaxation for Poisson's Equation. So, we define GSRB_LEVEL(e^ℓ, R^ℓ, h^ℓ) on Ω^ℓ. This does one iteration of Gauss-Seidel with Red-Black ordering on the data on level ℓ. This operation should know nothing about other levels, although it should know the appropriate operators and boundary conditions to relax on each level. Therefore, this operator looks like:

$$e^\ell_{ij} := e^\ell_{ij} + \lambda\{L^{nf,\ell}(e^\ell, e^{\ell-1} = 0) - R^\ell_{ij}\} \tag{16}$$

with red-black ordering. Red-black ordering means that we relax using two passes through the domain in a checkerboard pattern: on the first pass, we relax on points where (i+j) is even (the RED pass); on the second, we relax on points where (i+j) is odd (the BLACK pass). Note that, because the GSRB_LEVEL is designed to be unaware of both coarser and finer levels, the relaxation uses $L^{nf,\ell}$ with all the coarse grid information set to 0. For interior cells, we use the normal relaxation parameter $\lambda_{interior} = \frac{h^2}{4}$. For boundary cells, we will use $\lambda_{boundary} = \frac{3}{4}\lambda_{interior}$ as the relaxation parameter.

For each level, the residual will contain two components. First, as in normal multigrid relaxation, the residual on the level ℓ contains the residual from higher (finer) levels, mostly the low wavenumber error that is not damped out by the GSRB iterations at the finer levels. In addition, there is the residual from the operators on level ℓ where there are no overlying finer grids. If there is no overlying fine grid, then we are starting our multigrid V cycle on this level; otherwise, we are simply continuing the multigrid relaxation which was begun on the finer levels.

The basic algorithm is structured like a basic multigrid Poisson solve – we start at the finest levels, then progressively coarsen and relax our way down the V-cycle, then solve on the coarsest level, then interpolate and relax our way back up the V-cycle. The difference is that in this case, the grids we're solving on may not encompass the entire physical domain.

First, we define the residual on the finest level:

$$R^{\ell_{max}} := \rho^{\ell_{max}} - L^{\ell_{max}}(\phi) \tag{17}$$

Note that since this is occurring at the finest level, the only boundary conditions are interpolated ones from either the $\ell_{max} - 1$ grid or the physical boundary.

Then, starting at level ℓ_{max}, we relax on each level on the way down to the coarsest level. For each individual level relaxation on the way down, this is the algorithm. First, set the correction for next coarsest level $e^{\ell-1}$ equal to 0 (to provide a boundary condition for the current level.) Also, save a copy of the current version of ϕ^ℓ for later. Then, perform a GSRB_LEVEL operation on the current level:

$$e^\ell := \text{GSRB\_LEVEL}(\mathrm{e}^\ell, \mathrm{R}^\ell, \mathrm{h}^\ell) \tag{18}$$

Then, use this correction to update ϕ on this level:

$$\phi^\ell := \phi^\ell + e^\ell \tag{19}$$

We will need this current version of ϕ to enforce the flux-matching conditions at the interface between this level and the next coarser one. Then, define the residual on the next coarsest level. Where the current finer grid overlays the coarser grid, average the residual (after correcting it to include the current correction) on this level down to the next coarsest level.

Otherwise, use the coarser grid information:

$$R^{\ell-1} := \begin{cases} \text{Average}(\text{R}^\ell - \text{L}^{\text{nf},\ell}(\text{e}^\ell, \text{e}^{\ell-1})) & \text{on P}(\Omega^\ell) \\ \rho^{\ell-1} - L^{\ell-1}(\phi) & \text{on } \Omega^{\ell-1} - \text{P}(\Omega^\ell) \end{cases} \tag{20}$$

Then, repeat this process on the next coarsest $(\ell - 1)$ level. When we arrive at the coarsest level, we have one domain. We can then iterate on $\Delta^{h_0} e^0 = R^0$ on Ω^0. Once that is done, update the coarse level solution, $\phi^0 := \phi^0 + e^0$, and start back up the V-cycle.

The procedure on the way up is slightly different. First, we update the fine grid (level ℓ) correction:

$$e^\ell = e^\ell + \text{Interpolate}(\text{e}^{\ell-1}) \tag{21}$$

However, now we cannot blithely go directly to a GSRB_LEVEL iteration, because we now have a coarse grid correction which we will need to use as a boundary condition. We handle this the same way we handle any problem with inhomogeneous boundary conditions: we precondition the problem to make it homogeneous. So, we first must modify the residual:

$$R^\ell := R^\ell - L^{nf,\ell}(e^\ell, e^{\ell-1}) \tag{22}$$

We then define a correction to the correction, δe^ℓ, set it to 0, and then perform a GSRB_LEVEL operation on it:

$$\delta e^\ell := \text{GSRB\_LEVEL}(\delta \text{e}^\ell, \text{R}^\ell, \text{h}^\ell) \tag{23}$$

Then, we update first the correction, and then the copy of ϕ^ℓ which we had saved:

$$e^\ell := e^\ell + \delta e^\ell \tag{24}$$

$$\phi^\ell := \phi^{\ell,save} + e^\ell \tag{25}$$

We continue this process on the next level up (level $\ell+1$), until we have updated all the levels up to the finest level. This V-cycle is repeated until we have reduced the residual below our tolerance for each level ($\|R^\ell\| < \epsilon\|\rho^\ell\|$ for $\ell = 0, ..., \ell_{max}$, where ϵ is our tolerance.).

Summarizing, the algorithm is:

$R := \rho - L(\phi)$.
While ($|R| > \epsilon|\rho|$)

$R := \rho - L(\phi)$.

MGRelax($\ell^{\max}$).
EndWhile

Procedure MGRelax(ℓ):

if ($\ell = \ell^{max}$) then $R^\ell := \rho^\ell - L^{nf,\ell}(\phi^\ell, \phi^{\ell-1})$
if ($\ell > 0$) then

$\phi^{\ell,save} := \phi^\ell$

$e^{\ell-1} := 0$

$e^\ell := \mathrm{GSRB\_LEVEL}(\mathrm{e}^\ell, \mathrm{R}^\ell, \mathrm{h}^\ell)$

$\phi^\ell := \phi^\ell + e^\ell$

$R^{\ell-1} := \mathrm{Average}(\mathrm{R}^\ell - \mathrm{L}^{\mathrm{nf},\ell}(\mathrm{e}^\ell, \mathrm{e}^{\ell-1}))$ on $\mathrm{P}(\Omega^\ell)$

$R^{\ell-1} := \rho^{\ell-1} - L^{\ell-1}(\phi)$ on $\Omega^{\ell-1} - \mathrm{P}(\Omega^\ell)$

$\mathrm{MGRelax}(\ell - 1)$

$e^\ell := e^\ell + \mathrm{Interpolate}(\mathrm{e}^{\ell-1})$

$R^\ell := R^\ell - L^{nf,\ell}(e^\ell, e^{\ell-1})$

$\delta e^\ell := 0$

$\delta e^\ell := \mathrm{GSRB\_LEVEL}(\delta \mathrm{e}^\ell, \mathrm{R}^\ell, \mathrm{h}^\ell)$

$e^\ell := e^\ell + \delta e^\ell$

$\phi^\ell := \phi^{\ell,save} + e^\ell$

else

solve/relax $\Delta^{h_0} e^0 = R^0$ on Ω^0

$\phi^0 := \phi^0 + e^0$

Endif

5 Error Estimation

Once a solution has been constructed on the existing hierarchy of grids, we may need to construct new refined levels, or we may decide to alter existing levels in order to improve our solution. Before constructing new grids, we must decide where they are needed. To do this, we go through each existing level and "tag" cells where refinement is desired. In this process, we may also decide that previously refined cells were unnecessary and deallocate portions of refined domains. There are many possible criteria for deciding where refinement is necessary. In many physical problems, physical quantities like sharp density gradients or large charge distributions may provide indicators for refinement. The current implementation of the code

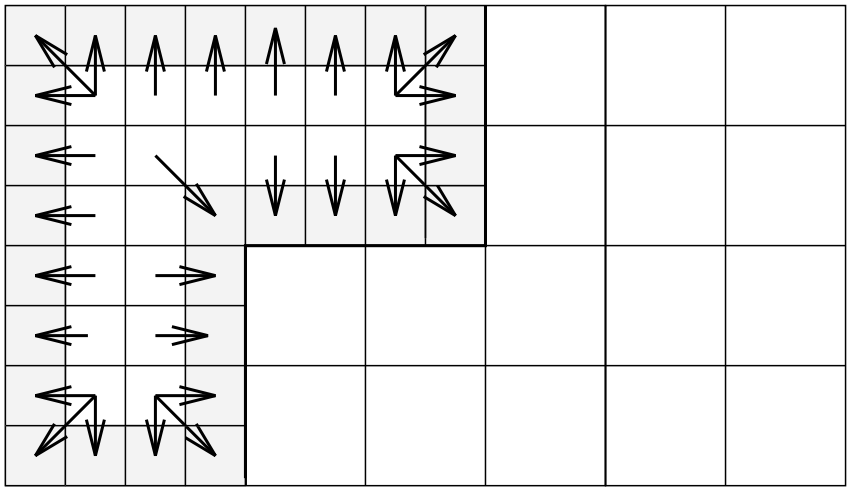

Figure 5: Replacing error in fine grid boundary cells (shaded) with adjacent values.

is capable of using such user-supplied criteria to trigger refinement.

In the current implementation, Richardson extrapolation can also be used to find areas where the local truncation error is large, indicating a need for refinement. The basic idea is to apply the operator (in this case, the Laplacian) to the existing solution, coarsen it, and then compare it to the operator applied to a coarsened version of the solution. It can be shown that the difference between the two is proportional to the local truncation error. In terms of the existing operators,

$$Error^{\ell} = \text{Average}(\mathrm{L}^{\mathrm{nf},\ell}(\phi^{\ell})) - \mathrm{L}^{\mathrm{nf},\ell-1}(\text{Average}(\phi^{\ell})) \tag{26}$$

Since we want to estimate the error on all existing levels, including those partially over lain by refined grids, we need to modify this procedure slightly. Where a grid is covered by a refined patch, we use the error computed on the refined level. Since we know that the error is proportional to h^2, we can rescale the fine error by multiplying by $(\frac{h^c}{h^f})^2$ (the square of the refinement ratio) and average it onto the coarser grid. This gives a reasonable approximation of what the error in a refined region would be if there was no refinement.

As mentioned before, we expect that the scheme will lose accuracy at coarse-fine interfaces and that the local truncation error will be $O(h)$ (one order lower than the rest of the scheme) due to the coarse-fine interpolation error. For the same reasons, the scheme will also lose accuracy at physical boundaries, since we're using quadratic interpolation there as well. So, if we naively use the error computed using equation (26) here, we will see a large error, which will appear in a single layer of cells on both the coarse and fine sides of the interface. Both in theory and in practice, however, this error on the coarse-fine interface does not affect the

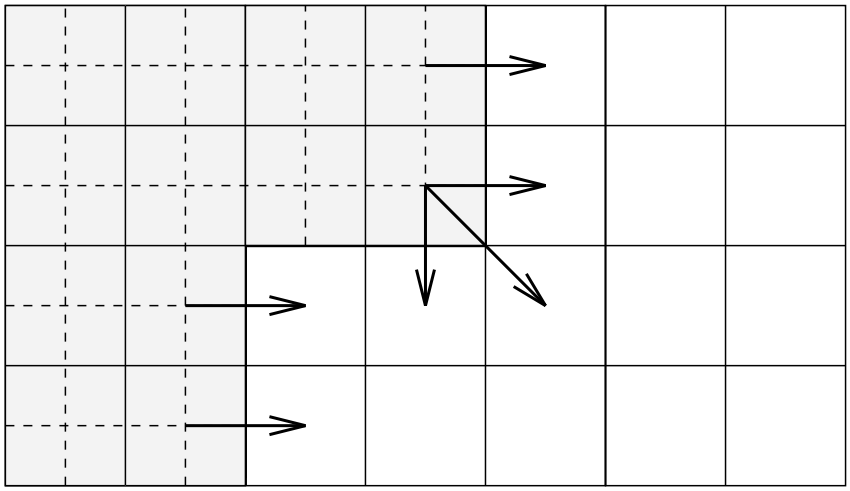

Figure 6: Replacing error on coarse side of coarse-fine interface with boundary cells from averaged fine grid values (shaded).

global accuracy of the scheme, since it is on a set of one dimension less than the problem space.

So, we do not want to use the local truncation error computed on these cells as a refinement criterion; if we did, refined grids would simply expand until they reached the physical boundaries. We don't, however, want to ignore the possibility that it may be desirable to refine these boundary cells. So, for each boundary cell, we copy the error from an adjacent cell which is untainted by the interpolation error. Because areas of high error tend to be patches, rather than single cells, this works to indicate places where we want to refine. Since the interpolation error manifests itself only in cells adjacent to the coarse/fine interface, only one layer of fine cells and one layer of coarse cells need be overwritten. In all cases, we copy along an outward normal to the interface. On the outer boundaries of fine levels and at physical boundaries, this is straightforward, as shown in Figure 5. On coarse grids, this is simplified by the fact that we have already rescaled and averaged the corrected fine grid error onto the coarse grid. So, we can simply copy the error outward from the interior boundary cells of the projection of the fine level, as shown in Figure 6.

Once we have compensated for the effects of refined grids and interpolation errors, we can step through each grid and tag cells wherever the error is greater than the desired tolerance. In our implementation, we tag cells whenever the error is greater than a user-supplied tolerance ϵ_{error} multiplied by $\max(\rho)$.

6 Creation of the Grid Hierarchy

Once we have decided which cells are to be refined, we need to use this information to create a hierarchy of levels. We use the algorithm of Berger and Rigoutsos [10], in which tagged points are clustered into efficient boxes. The efficiency of a grid is defined as the number of cells in the grid which were tagged for refinement divided by the total number of cells in the grid. The grid generator takes a list of tagged points and draws the smallest possible box around them. If this grid does not meet an efficiency criterion (typically $eff_{grid} \leq 0.7$ or so), the grid generator will look for the best way to subdivide the tagged points in order to create more efficient boxes.

The grid generator is designed to be as independent of the rest of the code as possible. In the current implementation it is a separate object, which only needs the currently existing grids, the base domain at each existing level's resolution, and an array of tagged cells.

7 Implementation

In the actual implementation of the AMR algorithm, a domain at level ℓ (Ω^ℓ) is made up of a union of rectangular domains, so

$$\Omega^\ell = \cup_k \Omega^{\ell,k}. \tag{27}$$

The rectangular domains $\Omega^{\ell,k}$ can overlap. Since there is no unique decomposition of Ω^ℓ into its constituent rectangles, one design point of the algorithm will be that the results should be independent of the particular decomposition chosen. In regions where the different $\Omega^{\ell,k}$'s overlap, the values must be the same, if there is to be any hope of meeting the goal of decomposition independence. (Note: while the algorithm will allow overlapping rectangles, the grid generator used will not create them.)

In actuality, all quantities will be defined on each subdomain $\Omega^{\ell,k}$, rather than only the valid regions $\Omega^\ell - \mathrm{P}(\Omega^{\ell+1})$. In the covered regions $\mathrm{P}(\Omega^{\ell+1})$ the grid quantities will be defined as the averaged fine grid values: $\phi^\ell = \mathrm{Average}(\phi^{\ell+1})$ on $\mathrm{P}(\Omega^{\ell+1})$.

When a new level is created, it is useful to initialize it with the best guess at a solution possible, to speed multigrid convergence. To this end, whenever a new level Ω^ℓ is created (or recreated, in the case of regridding) we first copy the solution from previously existing grids at level ℓ wherever possible (this is relevant whenever we are changing the grids on a preexisting level, rather than creating a completely new one), and then use piecewise constant interpolation of the next coarser level $(\ell - 1)$ solution to fill in the rest of the new Ω^ℓ. In this way, the new grids start off containing an accurate representation of the current solution.

In its current implementation, the AMRPoisson code is written in C++ and Fortran. The

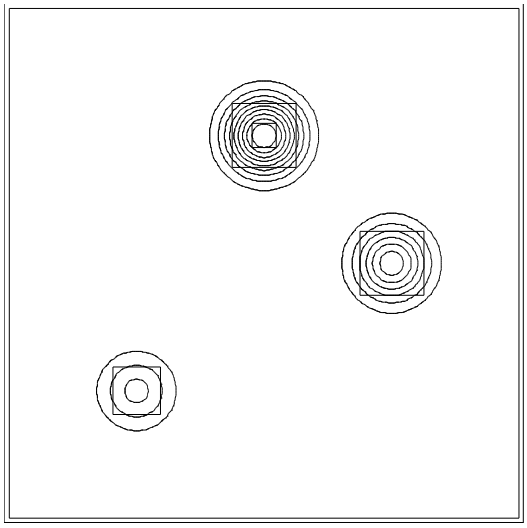

Figure 7: Sample ρ.

basic organization and data structures are in C++, while the actual numerics are performed in Fortran. There are three main data structures. The class levelAMR contains all data and operations for a single refinement level, which is defined on a union of rectangular grids. The class AMRPoisson is a parent class which contains and manages the entire hierarchy, which includes an array of levelAMR's. User interfaces are built into this class, as well. Finally, the class GridGenerator is basically the grid generator object described earlier. These classes are built upon Boxlib, which is a library of C++ classes for managing rectangular domains developed by the Center for Computational Science and Engineering at the Lawrence Livermore National Laboratory [2].

8 Timing and Convergence Results

The AMRPoisson code was tested on a sample problem with ρ equal to three Gaussian charges, as shown in Figure 7. To give an idea of grid placement, the grids used for a solution with two levels of refinement are shown as well. To judge the effects of adaptivity, we solved this problem with a series of coarser base grids, but with the same error tolerance. By doing this, we solve the problem to the same level of accuracy each time, but more levels of refinement become necessary as the base grid becomes coarser. By setting the error tolerance ϵ_{error} for the Richardson extrapolation error tagging routine to be 0.0005, no refinements were needed for a base grid of 1024 X 1024, while one level of refinement was needed for a 512 X 512 base grid (and two levels were needed for a 256 X 256 base grid, etc.). For all of the solutions, the maximum error on the finest grid as computed using Equation (26) was

Base Grid Size	h = 1/64	1/128	1/256	1/512	1/1024	total
1024 x 1024	—-	—-	—-	—-	1048576	1048576
512 x 512	—-	—-	—-	262144	2304	264448
256 x 256	—-	—-	65536	10496	2304	78336
128 x 128	—-	16384	17280	10496	2304	46464
64 x 64	4096	15360	17152	10496	2304	49408

Table 1: Number of cells at each grid resolution, tabulated for different base grid sizes when solving sample problem

4.27E-4. To show the effects of adaptivity on the resulting grid hierarchy, the total number of cells on each level is tabulated in Table 1. It is worth noting that, in every solution where refinement is employed, the number of cells at the finest resolution is constant at 2304, while the number of cells at the second finest resolution is constant at 10496. This points to the effectiveness of Richardson extrapolation as a consistent indicator of the necessary resolution for attaining a given level of accuracy in the solution.

Adding local refinement to the solution did slow down the convergence rates of the multigrid cycle somewhat. The convergence results are shown in Table 2. With no refinement, the Max(residual) was reduced by an average factor of 18.6 per multigrid cycle. In other words, the maximum of the residual after one full multigrid V-cycle was, on average, $\frac{1}{18.6}$ times the maximum residual at the start of the V-cycle. When one or two levels of refinement (with a refinement ratio of 2) were added to the solution, the convergence rate dropped to an average factor 10.1 reduction per multigrid cycle. When a fourth level was added, however, the convergence rate increased to an average factor 15.2 reduction in the max(residual). With a refinement ratio of 4, the convergence appears to be somewhat better; one level of refinement converged with an average factor 18.4 reduction in the residual, while for two levels, the max(residual) was reduced by an average factor of 15.8 per multigrid iteration. Using a refinement ratio of 8 led to a markedly poorer performance, however. One level of factor 8 refinement only showed an average reduction in the max(residual) of a factor of 5.6 per multigrid cycle. For the rest of this section, all timing results use a refinement ratio of two.

To easily judge the effects of adaptivity, timings were normalized by the timing for the unrefined 1024 X 1024 solution, which was 58.11 sec on an SGI Power Challenge (using O1 optimization). Also, the total number of cells for all levels in each solution (including the non-valid portions where grids are over lain by finer grids) was recorded and likewise normalized by the total number of cells for the unrefined grid, 1048576. The total number of cells is an indicator of how much memory was used in the solution. A log plot of these normalized results appear in Figure 8. As can be seen, the total number of cells in the solution decreases with the number of refinement levels, ranging from 30.6% of the base number of cells with one

Refinement Ratio	1 level (no refinement)	2 levels	3 levels	4 levels
2	18.61	10.1	10.1	15.2
4	18.61	18.4	15.8	—-
8	18.61	5.59	—-	—-

Table 2: Convergence rates (average factor by which max(residual) is reduced for each multigrid iteration), tabulated for different refinement ratios and number of levels of refinement

level of refinement to 4.5% of the base number of cells with three levels of refinement. Adding a fourth level of refinement actually *increases* the total number of cells because so much of the base level is being refined (in this case, 93.75 of the domain is refined to level 1). The timings initially decrease strongly with additional levels of refinement, up to two levels of refinement, then level off at around 15% of the CPU time for the unrefined solution and actually rise slightly. This leveling off is due to the need when using Richardson extrapolation to compute a solution with $\ell - 1$ levels before generating a ℓth level. In other words, to compute a solution with two levels of refinement, first a single grid solution must be computed, then the error estimator defines the level 1 grids, which are then used to compute a two level solution, and then the error estimator is able to create the level 2 grids based on the error computed in the two level composite solution. Since the coarser levels are, in general, small in comparison to a finer base domain, this is generally inexpensive. However, with more levels of refinement, this can begin to offset the savings in computational time. It should be noted, however, that even once the break even point has been reached on CPU time, the additional refinement in this case still represents a savings in memory. Obviously, when the total number of cells increases, as is the case between three and four levels of refinement, CPU time will increase faster than the number of cells in the solution, due to the overhead of generating and managing the grid hierarchy.

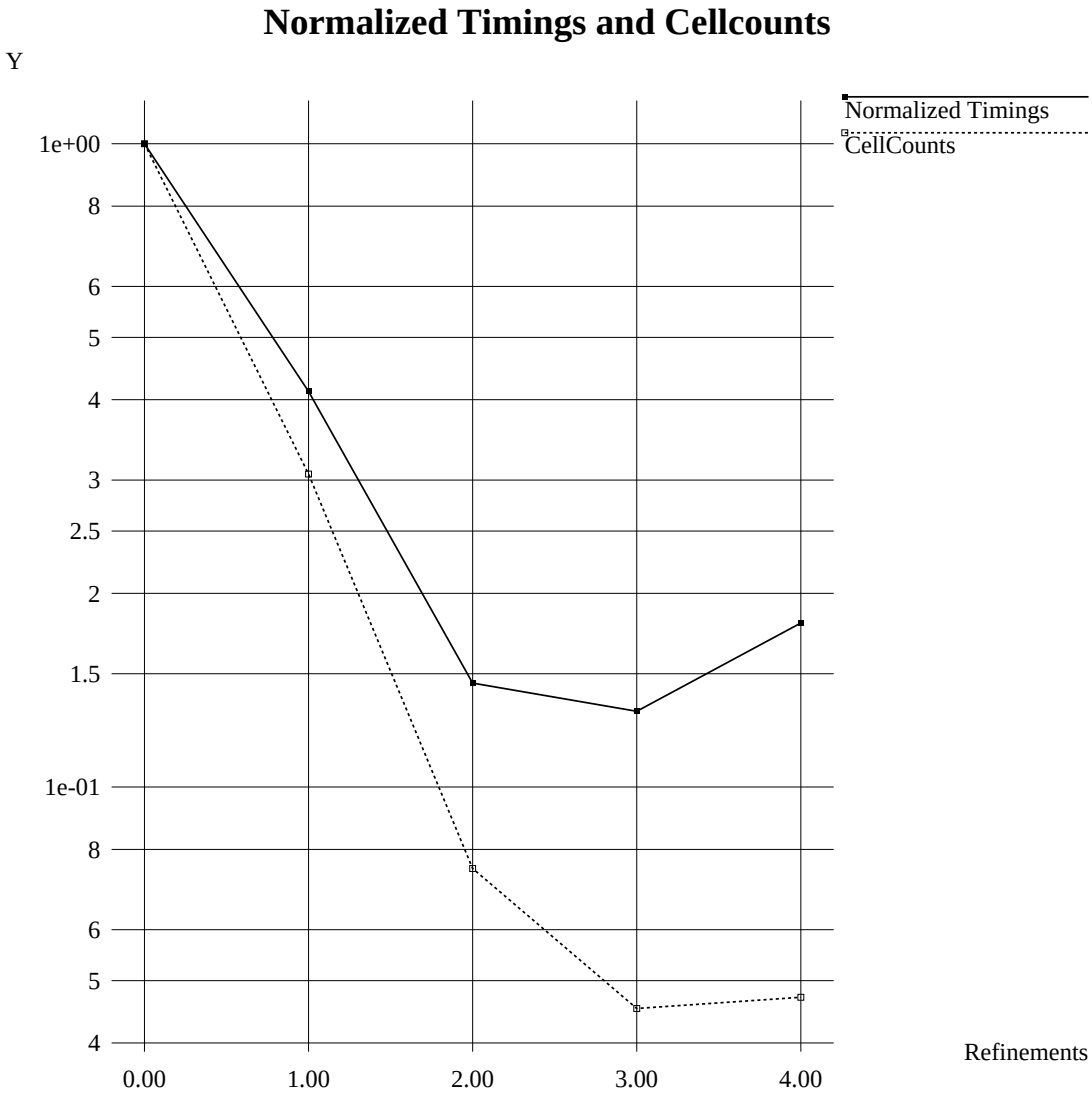

Figure 8: Normalized timings.

References

[1] D. F. Martin and K. L. Cartwright. *AMRPossion*. University of California, Berkeley, version 2.0 edition, 1996. http://barkley.me.berkeley.edu/˜ martin/AMRPoisson.html.

[2] Chuck Rendleman, Vince Beckner, John Bell, Bill Crutchfield, Louis Howell, and Mike Welcome. *Boxlib User's Guide and Manual: A Library for Managing Rectangular Domains*. Center for Computational Science and Engineering, National Energy Research Supercomputing Center, Lawrence Berkeley National Laboratory, edition 2.0, for boxlib version 2.0 edition, 1996. http://www.nersc.gov/research/CCSE/software/software.html.

[3] William L. Briggs. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, Philadelphia, 1987.

[4] M.J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. of Comput. Phys.*, 53:484–512, 1984.

[5] M.J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodyamics. *J. Comput. Phys.*, 82(1):64–84, May 1989.

[6] John Bell, Marsha Berger, Jeff Saltzman, and Mike Welcome. Three dimensional adaptive mesh refinement for hyperbolic conservation laws. *J. Sci. Comput.*, 15(1):127–138, January 1994.

[7] Ann Almgren, John Bell, Phillip Colella, Louis Howell, and Michael Welcome. A conservative adaptive projection method for the variable density incompressible navier-stokes equations. Lbnl-39075, Computing Sciences Directorate, Lawrence Berkeley National Laboratory, July 1996. Submitted to Journal of Computational Physics.

[8] Ann Almgren, Thomas Buttke, and Phillip Colella. A fast adaptive vortex method in three dimensions. *Journal of Computational Physics*, 113:177–200, 1994.

[9] M.C. Thompson and J.H. Ferziger. An adaptive multigrid technique for the incompressible navier-stokes equations. *Journal of Computational Physics*, 82:94–121, 1989.

[10] M. J. Berger and I. Rigoutsos. An algorithm for point clustering and grid generation. *IEEE Transactions Systems, Man and Cybernetics*, 21(5):1278–1286, 1991.