

LA-UR 97-3595

Approved for public release;
distribution is unlimited

Title:

Patched Based Methods for Adaptive Mesh
Refinement Solutions of Partial Differential Equations

Author(s):

Jeffrey Saltzman

Submitted to:

External distribution.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED 

MASTER

Los Alamos
National Laboratory

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. The Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible electronic image products. Images are produced from the best available original document.

Patched Based Methods for Adaptive Mesh
Refinement Solutions of Partial Differential
Equations

Jeffrey Saltzman ¹

September 2, 1997

¹Los Alamos National Laboratory, MS B256, Los Alamos, NM, 87544.

Contents

1	Mathematical Foundations	8
1.1	Hyperbolic Systems	8
1.1.1	Linear Equations	9
1.1.2	Nonlinear Equations	11
1.2	Hyperbolic-Parabolic Systems	18
1.2.1	Linear Advection Diffusion Equations	19
1.2.2	Navier-Stokes Equations	19
1.3	Hyperbolic-Elliptic Systems	21
1.3.1	Boundary Layers and Vortices	22
2	Numerical Methods	25
2.1	Finite Difference Grids and Notation	25
2.1.1	Grids and Mappings	25
2.1.2	Notation	26
2.2	An Array Class in C++	27
2.3	Upstream Centered Difference Methods	29
2.3.1	1-D Methods	29
2.3.2	2-D Unsplit Methods	35
2.3.3	Linear Systems	38
2.3.4	Nonlinear Equations	41
3	Adaptive Mesh Refinement for Conservation Laws	44
3.1	Moving Mesh Methods	44
3.2	Structured Adaptive Mesh Refinement	46
3.2.1	Rectangular Tensor Product Grids	46
3.2.2	Curvilinear Meshes	55
3.2.3	Moving Meshes	56
3.2.4	Overset Meshes	57
3.3	Parallel Strategies	58
3.3.1	Knapsack Load Balancing	58
3.3.2	Data Parallel AMR	60
3.3.3	Multilevel Load Balancing	62

4	Adaptive Elliptic Methods	64
4.1	Rectangular AMR Grids	64
4.2	Embedded Boundaries	69
4.3	Overset Grids	73
5	Adaptive Hyperbolic-Elliptic Solvers	77
5.1	Incompressible, Inviscid Flow Solution	77
5.1.1	Advection	78
5.1.2	MAC Projection	79
5.1.3	Corrector Step	80
5.1.4	Full Projection	80
5.2	An Adaptive Incompressible Method	81
5.2.1	Single Level Algorithm	82
5.2.2	Multilevel Algorithm	84

List of Figures

2.1	A geometric picture of the evolution of a piecewise constant solution one time step.	30
2.2	A geometric picture of the evolution of a piecewise linear solution one time step.	33
2.3	A geometric picture of the evolution of a limited piecewise linear solution one time step.	35
2.4	A geometric picture of the evolution of a 2-D piecewise constant solution one time step.	36
3.1	Examples of a adaptive mesh hierarchies in one and two dimensions.	47
3.2	Proper nesting insures filling boundary ghost cells using physical boundaries, data from adjacent patches at the same level or interpolation from the next coarsest level.	47
3.3	A sequence of pictures of a flagged region illustrating box selection using the Berger-Rigoutsos algorithm.	50
3.4	An example of a 2-D patch that has its ghost cells filled by physical boundary conditions, interpolation from the next coarsest level or from data of adjacent patches at the same level.	50
3.5	The recursive advance of three levels is illustrated with the coarsest mesh advanced first, then fine grids are advanced using coarser grid data interpolated in space and time.	51
3.6	A coarse cell and 4 fine cells generated using a cylindrical mapping.	55
3.7	Volume flux construction from moving edges or faces.	56
3.8	Flagged cells add buffer cells across components of an overset mesh through the interpolation stencil.	57
3.9	Tiling algorithm for fixed size patches.	61
4.1	Interpolants between coarse and fine grids are created to compute an approximate normal gradient.	67
4.2	Cartesian mesh cut by an embedded boundary creating irregular cells.	69
4.3	Interpolants used in construction of a gradient centered at an irregular edge of a cut cell.	71

4.4	Nonboundary edges of cut cells need to interpolate gradients to compensate for loss of centering in order to preserve second order accuracy.	71
4.5	The higher ranking diamond grid cuts away at the lower ranking Cartesian grid. The detail shows how the diamond grid acts like an embedded boundary in cutting away the Cartesian grid. . . .	74
4.6	Two interpolants on the cutting mesh with one on the cut mesh are used to supply data to a fourth interpolant in order to compute a gradient normal to a cutting edge.	76
5.1	The coarse and fine operators ∇_{coarse} and ∇_{fine} zero out contributions from fine and coarse cells respectively.	85

Abstract

This manuscript contains the lecture notes for a course entitled "Patched Based Methods for Adaptive Mesh Refinement Solutions of Partial Differential Equations" taught from July 7th through July 11th at the 1997 Numerical Analysis Summer School sponsored by C.E.A., I.N.R.I.A., and E.D.F. The subject area was chosen to support the general theme of that year's school which is "Multiscale Methods and Wavelets in Numerical Simulation". The first topic covered in these notes is a description of the problem domain. This coverage is limited to classical PDEs with a heavier emphasis on hyperbolic systems and constrained hyperbolic systems. The next topic is difference schemes. These schemes are the foundation for the adaptive methods. After the background material is covered, attention is focused on a simple patched based adaptive algorithm and its associated data structures for square grids and hyperbolic conservation laws. Embellishments include curvilinear meshes, embedded boundary and overset meshes. Next, several strategies for parallel implementations are examined. The remainder of the notes contains descriptions of elliptic solutions on the mesh hierarchy, elliptically constrained flow solution methods and elliptically constrained flow solution methods with diffusion.

Introduction and Curriculum Summary

Adaptive meshes are conceptually easy to understand. It is known when approximating the solution of Partial Differential Equations (PDEs) the overall accuracy of the solution is determined by local truncation error. Since local truncation error can be explicitly written down as a sum of higher order derivatives of the solution and some known functions it is easily seen that in many instances the truncation error is very nonuniform. For example, many problems with fronts have the largest truncation error at these fronts and small errors elsewhere. However, only relatively recently have people tried to exploit this observation. The reason for this is programming complexity. Because of limitations with programming languages and computer architectures, developing an adaptive mesh algorithm for a multidimensional PDE was a daunting task.

Currently there are a variety of adaptive methods under development or in use — so much so that we can easily spend the time allocated for this course surveying all the developments in this area. Instead, we focus on a particular class of problems with a specific mathematical structure. We will further restrict ourselves to a small collection of difference methods and data structures. Our focus will be on hyperbolic conservation laws that may be augmented with constraints or “small” perturbations on the underlying solution. Constraints will be exclusively elliptic such as is found in incompressibility conditions in fluid dynamics. Diffusion terms may also be introduced (formally changing the system type to parabolic), but the magnitude of the diffusion coefficients will mostly be small preserving the sharp fronts found in problems of interest.

This manuscript has five chapters. Each chapter corresponds to a lecture. However, the material covered in each chapter exceeds what is covered in a lecture because of time constraints of the later. Below is a list of chapter themes:

- **Mathematical Preliminaries:** Here we describe three classes of PDEs that are covered in the remaining lectures. These classes are strictly hyperbolic equations, parabolic equations that are *singular perturbations* of hyperbolic systems, and constrained hyperbolic systems where the constraints are elliptic.

- **Notation, Computer Languages and Finite Difference Schemes:** In order to discuss adaptive mesh algorithms we start with a description of the components. These components include a collection of different types of structured grids that are used, a C++ array class that maps well to these grids (in fact the class was designed to facilitate development of finite difference schemes for these grids), and a collection of finite difference methods. We briefly discuss moving mesh methods as a precursor to adaptive methods.
- **Patched Based Algorithms and Parallel Strategies:** We outline the basic steps in a patch based adaptive mesh refinement (AMR) algorithm for hyperbolic systems of conservation laws on square grids. Data structures and implementation tips are then given. Finally, the basic methodology is generalized to more complicated structured grids. Adaptive methods can be seen as a way of saving memory and CPU time. However, the additional complexity of adaptive methods may lead to problems with implementation on a parallel machine. We describe parallel algorithms currently in use or under development.
- **Elliptic Constraints:** We briefly describe how to solve elliptic problems on adaptive mesh hierarchies. These techniques are tailored for singular perturbation problems and patched grids.
- **Hyperbolic-Parabolic-Elliptic Methods:** Often applications are of mixed type. There may be a combination of elliptic, hyperbolic and parabolic parts of a system. Here we describe algorithms that combine different types of equations within a single application. In particular, the global influence of elliptic constraints will be explored as well as its impact on adaptive mesh algorithms.

Chapter 1

Mathematical Foundations

In this chapter we describe sets of equations that fall into the category of classical PDEs. The central focus of our work is hyperbolic systems. Although our development will be general, our applications will focus on hyperbolic conservation laws. The conservation laws will be seen as a limiting case of a parabolic system where the physical dissipation tends toward zero. This naturally leads to the next topic which is labeled hyperbolic-parabolic systems. Mathematically, these systems are simply parabolic. We use the phrase hyperbolic-parabolic systems to emphasize that our interest will be towards the zero dissipation limit. However, global coupling of field variables is still present and we will move from explicit to implicit finite difference techniques. The final topic in this chapter is elliptically constrained hyperbolic systems. Here, too, global coupling of field variables will be present. In contrast with parabolic systems, elliptically constrained systems much more tightly couple the field variables. This coupling will lead to significant changes in adaptive mesh algorithms.

1.1 Hyperbolic Systems

Adaptive methods are highly effective in approximating solutions of hyperbolic systems. Discontinuities travel at finite speeds and are highly localized. Therefore explicit finite difference methods can be used to approximate solutions and mesh refinements, too, will be highly localized. In this section we describe linear and nonlinear scalar and systems of equations. Our coverage is limited to equations that are written in conservation form. We further will limit our discussion to features of these equations that will be useful to their numerical analysis. The classic reference for hyperbolic systems and conservation laws is Lax [29]. An excellent introductory text on numerical analysis of these systems is Le Veque [32].

1.1.1 Linear Equations

The simplest linear equation is the scalar equation

$$u_t + cu_x = u_t + (cu)_x = 0 \quad (1.1)$$

where

$$u = u(x, t), \quad c = \text{const.}, \quad \text{and } u(x, 0) = u_0(x).$$

Although this problem is trivial (the solution $u(x, t) = u_0(x - ct)$ is found by inspection), it displays many properties that are relevant to systems that are more interesting. The two most important properties that are easily seen are conservation and the existence of characteristics.

To expose the conservation properties of this equation rewrite the wave equation as

$$u_t + cu_x = u_t + f(u)_x = 0$$

where $f(u) = cu$. The function $f(u)$ is called a flux function. Integrating the differential equation over a box in space and time $[x_1, x_2] \times [t_1, t_2]$ yields

$$\int_{t_1}^{t_2} \int_{x_1}^{x_2} u_t + f_x = 0.$$

This can be rewritten as

$$\int_{x_1}^{x_2} u(t_2, y) dy - \int_{x_1}^{x_2} u(t_1, y) dy = \int_{t_1}^{t_2} f(u(x_1, t)) dt - \int_{t_1}^{t_2} f(u(x_2, t)) dt.$$

If u is considered as a density (mass per unit volume) then the above relation implies that the amount of mass at time level t_2 is equal to the initial mass at time level t_1 plus the mass flux flowing into the box at x_1 minus the mass flux leaving the box at x_2 . This property has both engineering, physical and mathematical significance. From the engineering and physical viewpoint conservation is the most fundamental starting point for many mathematical models. From the mathematical viewpoint, conservation represents an important constraint that helps to insure that discontinuities in a solution (later we will call these shocks and contacts) move at the proper speed and are calculated in the correct physical manner. Note that f need not be linear in the solution but only differentiable.

Characteristics are special directions in space-time that represent how information propagates over the evolution of the problem. The special direction for the linear equation is easily seen as $x - ct = \text{const.}$ If we define

$$\frac{d}{dt} = \frac{\partial}{\partial t} + c \frac{\partial}{\partial x}$$

then

$$\frac{du}{dt} = u_t + cu_x = 0.$$

In other words the solution is invariant on a characteristic path in space-time. The concept of characteristics becomes more complicated when we talk about systems.

Linear systems have the form

$$U_t + AU_x = 0 \quad (1.2)$$

where $U(x, t) \in R^m$ is a vector and $A \in R^m \times R^m$ is a constant matrix. The system is hyperbolic if A has real, distinct eigenvalues. An example system is the system formulation of the second order wave equation

$$\begin{pmatrix} u \\ v \end{pmatrix}_t + \begin{pmatrix} 0 & c \\ c & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}_x = 0 \quad (1.3)$$

where c is the wave speed. The above is equivalent to the scalar second order equation

$$u_{tt} - c^2 u_{xx} = 0.$$

Since the matrix A is constant the linear system of equations can be seen to be conservative for each component of the solution vector. This is easily accomplished by moving A inside of the x derivative in equation (1.2).

Let E be a matrix whose columns are the eigenvectors of A in (1.2). Since the eigenvalues are distinct, the eigenvectors are independent and E is invertible. Therefore a mapping of U into a new space V can be defined as

$$U = EV.$$

Substituting the mapping into the linear system (1.2) gives

$$EV_t + AEV_x = EV_t + E\Lambda V_x = 0$$

where Λ is a diagonal matrix whose nonzero elements are the eigenvalues of A . Since E is nonsingular the above system can be multiplied through by its inverse. What is left is a diagonalized system

$$V_t + \Lambda V_x = 0.$$

Following the same procedure for the wave equation (1.3) leads to a system of the form

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}_t + \begin{pmatrix} c & 0 \\ 0 & -c \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}_x = 0$$

which is nothing more than two decoupled waves propagating in opposite directions with like speeds.

The ability to so easily decouple and solve systems of equations component by component is lost with nonlinear systems of equations. However, conservation and characteristics in space-time still remain. Linear systems are useful in their own right. In sections that follow we will describe a system of equations called the Euler equations that describe inviscid fluid flow and are nonlinear. It is easy to find constant solutions to these equations. By perturbing around these constant states with small excursions we find what are called acoustic equations. The acoustic equations are quite close to the wave equations described above. These linear equations in one or more dimensions are the basis for the field of acoustics. Applications range from concert hall design to the study of abatement of noise generated from aircraft flying over populated areas near airports.

1.1.2 Nonlinear Equations

The study of nonlinear hyperbolic equations often starts with Burgers' equation

$$u_t + uu_x = u_t + (u^2/2)_x = 0. \quad (1.4)$$

The flux function for this equation is seen by inspection as $f(u) = u^2/2$. As before we may formally define a directional derivative and express (1.4) as

$$\frac{du}{dt} = \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0.$$

If $u_0(x)$ is the initial condition for (1.4) then it is easy to formally write, again by inspection, a solution of the form

$$u(x, t) = u_0(x - u(x, t)t).$$

Several problems occur with this solution. Most obvious is the solution is expressed implicitly meaning u_0 must be inverted. But what can be seen from this solution or the formal directional derivative is the solution is constant along a characteristic $x - ut$. So once u is known at one point along a characteristic then it is known everywhere. We attempt to cover the $x - t$ plane with characteristic lines using the slopes of the lines starting at $t = 0$ and $x = x_0$

$$x = tu_0(x_0) + x_0.$$

If $u_0(x) = x$ then a fan-like system of characteristics emanate from the x axis at $t = 0$. This solution is often suggestively referred to as an *expansion* or *rarefaction* solution. The expansion solution nicely covers the plane for $t \geq 0$. However, if the initial conditions are $u_0(x) = -x$ then there is a problem. The characteristic crossing the x axis at $x = -1$ will intersect another characteristic crossing the x axis at $x = 1$. The crossing point will be at time $t = 1$. Prior to this time we can find unique solutions along characteristic lines. The collision of characteristics is an example of the formation of what is called a *shock*. The characteristics for a general scalar conservation can be seen from the derivative along the characteristic direction, by inspection, as

$$\frac{du}{dt} = \frac{\partial u}{\partial t} + a(u)u_x = \frac{\partial u}{\partial t} + f'(u)u_x = \frac{\partial u}{\partial t} + f(u)_x = 0$$

where $a(u) = f'(u) = dx/dt$ is the characteristic velocity.

We will introduce the idea of a weak solution in order to accommodate shock solutions. Indeed, a combination of shocks and smooth solutions will suffice to describe a complete solution of the problem. To understand shocks it is best to look at an integral form of a conservation law. First integrate the equation on an interval $[a, b]$.

$$0 = \int_a^b u_t + f(u)_x dx = \frac{d}{dt} \int_a^b u dx + f(u)|_b - f(u)|_a \quad (1.5)$$

The relation simply states that the amount of mass on the interval $[a, b]$ only changes from the mass flux entering and leaving through the endpoints. A key observation is that even if a shock exists within an interval mass is neither created or destroyed. The only way mass is added into the interval is through the fluxes at the endpoints. We take this as a starting point and work towards the differential equation. Suppose there is a shock in the interval $[a, b]$ but the rest of the mass distribution is a smooth function. Let the shock have a location $\xi(t)$ in that interval. Using Leibnitz's rule on the time derivative of the mass on the interval leads to

$$\begin{aligned} \frac{d}{dt} \int_a^b u dx &= \frac{d}{dt} \int_a^{\xi(t)} u dx + \frac{d}{dt} \int_{\xi(t)}^b u dx \\ &= \int_a^{\xi(t)} \frac{\partial u}{\partial t} dx + u_- \dot{\xi}(t) + \int_{\xi(t)}^b \frac{\partial u}{\partial t} dx - u_+ \dot{\xi}(t) \\ &= - \int_a^{\xi(t)} f(u)_x dx + u \dot{\xi}(t) - \int_{\xi(t)}^b f(u)_x dx - u \dot{\xi}(t) \\ &= f(u)|_a - f(u)|_b + f(u_+)|_{\xi(t)} - f(u_-)|_{\xi(t)} + u_- \dot{\xi}(t) - u_+ \dot{\xi}(t) \end{aligned}$$

where the u_- indicates a limiting value from the left of the discontinuity and u_+ indicates a limiting value from the right of the discontinuity. Using the integral of the conservation law (1.5)

$$0 = \frac{d}{dt} \int_a^b u dx - f(u)|_a + f(u)|_b = f(u_+)|_{\xi(t)} - f(u_-)|_{\xi(t)} + u_- \dot{\xi}(t) - u_+ \dot{\xi}(t).$$

gives a the shock jump condition of the form

$$s[u_+ - u_-] = f(u_+) - f(u_-) \quad (1.6)$$

where $s = \dot{\xi}(t)$.

The shock jump relation, sometimes called the Rankine-Hugoniot relation, expresses a change in density must be accompanied by a change in fluxes in order that no mass is lost across a jump. For linear systems and nonlinear systems the above relations hold on a component by component basis. For linear systems the shock values simply turn out to be the same as the wave speeds. For nonlinear systems a set of algebraic equations must be solved to find the shock speeds. An important question that can now be asked is the following: Is every jump that satisfies the Rankine-Hugoniot condition physically correct? The answer is no! In addition to satisfying a jump condition, the solution must also satisfy an entropy condition.

The entropy condition for a scalar hyperbolic conservation law states that characteristics must enter a shock from the left and right side

$$\lambda_- > s > \lambda_+$$

where $\lambda_- = f'(u_-) = a(u_-)$ and $\lambda_+ = f'(u_+)$. Returning to Burgers' equation let us examine the associated *Riemann problem*. The Riemann problem is the solution of a conservation law with piecewise constant initial conditions. To the left of the origin the initial state is constant and is denoted u_l while to the right of the origin the initial state is also constant and is denoted u_r . According to the entropy condition if $u_l > u_r$, then a simple shock propagates with speed $(u_r + u_l)/2$. On the other hand if $u_l < u_r$ a *rarefaction* solution of the form

$$u(x, t) = \begin{cases} u_l & \text{for } x/t \leq u_l \\ x/t & \text{for } u_l < x/t < u_r \\ u_r & \text{for } u_r \leq x/t \end{cases} \quad (1.7)$$

must be used. The rarefaction solution is easily verified to satisfy Burgers' equation.

The entropy condition for systems of conservation laws is more complex than the scalar case. Consider the system

$$U_t + F(U)_x = 0$$

where U is the solution vector and F is the flux vector of size n . Define

$$A = \frac{\partial F}{\partial U}$$

as the *Jacobian* of the flux vector. Assuming the system is hyperbolic, the Jacobian has distinct eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$. For systems it is required that for some index k , $1 \leq k \leq n$,

$$\lambda_k(U_-) > s > \lambda_k(U_+)$$

while

$$\lambda_{k-1}(U_-) < s < \lambda_{k+1}(U_+).$$

The entropy condition guarantees that k characteristics impinge on a shock from the left and $n - k + 1$ from the right. Coupled with the shock jump relations and assuming s is fixed the characteristic equations give $2n$ relations for the two states on either side of the shock.

Euler Equations

Nonlinear systems of equations are richer in complexity and behaviors than scalar equations. Instead of developing a general theory, which only exists in the small, we will focus our attention on specific systems such as the Euler Equations. Our goal is to analyze these equations to show a representative set of solutions representing discontinuous behavior in the solutions and/or their derivatives. The Euler equations describe the time evolution of an inviscid fluid and is most easily written in the compact Lagrangian form suggestive of the discussion of scalar equations

$$\begin{aligned} \frac{d\rho}{dt} + \rho \nabla \cdot \vec{u} &= 0 \\ \frac{d\vec{u}}{dt} + \frac{1}{\rho} \nabla p &= 0 \\ \rho \frac{de}{dt} + p \nabla \cdot \vec{u} &= 0 \\ p &= p(\rho, e). \end{aligned}$$

The first equation of the above set is called the mass equation where ρ is the density of the material, \vec{u} is the vector velocity and t is time. The second equation is called the momentum equation and uses p as the scalar pressure. The third equation is the energy equation where e is the specific internal energy. Finally, the pressure is written as a function of the density and energy and is called the equation of state (EOS). The Lagrangian derivative

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \vec{u} \cdot \nabla f$$

shows how a function, f , evolves along a particle path. Like scalar equations, this path, as will be seen later, is a characteristic path. There are many ways to analyze this system. Define a one-dimensional mass coordinate

$$m(x) = \int_{-\infty}^x \rho dx.$$

In one-dimensional Cartesian coordinates the evolution equations take on the form

$$\frac{\partial \tau}{\partial t} - \frac{\partial u}{\partial m} = 0 \quad (1.8)$$

$$\frac{\partial u}{\partial t} + \frac{\partial p}{\partial m} = 0 \quad (1.9)$$

$$\frac{\partial E}{\partial t} - \frac{\partial(\rho u)}{\partial m} = 0 \quad (1.10)$$

where

$$E = \frac{1}{2}u^2 + e \text{ and } \tau = \frac{1}{\rho}$$

is the total energy and specific volume of the particle respectively. It is seen that the equations are in conservation form. We will use these equations to find shock solutions.

A second form of the 1-D Cartesian Euler equations is

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} &= 0 \\ \frac{\partial \rho u}{\partial t} + \frac{\partial(\rho u u + p)}{\partial x} &= 0 \\ \frac{\partial \rho E}{\partial t} + \frac{\partial(\rho u E)}{\partial x} &= 0. \end{aligned}$$

Once again the system of equations are in conservation form. This form is most useful for numerical computation. A final form of the equations is called the primitive equations. The following set of equations can be derived from any of the above sets of equations by making two observations. First, from thermodynamics, the pressure can be written in terms of the density and entropy S : $p = p(\rho, S)$. Second, the entropy is constant along a smooth particle path or equivalently

$$\frac{dS}{dt} = 0.$$

The primitive equations are

$$\begin{aligned}\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} &= 0 \\ \rho \frac{\partial u}{\partial t} + \rho u \frac{\partial u}{\partial x} + \frac{\partial p}{\partial x} &= 0 \\ \frac{\partial p}{\partial t} + u \frac{\partial p}{\partial x} + \rho c^2 \frac{\partial u}{\partial x} &= 0\end{aligned}$$

where

$$p = p(\rho, S) \text{ and } c^2 = \left. \frac{\partial p}{\partial S}(\rho, S) \right|_S.$$

This form of the equations will be used to find rarefaction or simple wave solutions.

Using the Lagrangian form of the Euler equations, we can examine the system of algebraic equations that describes a shock jump. Using (1.8), (1.9) and (1.10) along with the jump relation (1.6) on each component gives

$$\begin{aligned}s[\tau] &= -[u] \\ s[u] &= [p] \\ s[E] &= [pu].\end{aligned}$$

where $[u] = u_+ - u_-$ and s is the shock speed. Defining $\bar{a} = (a_+ + a_-)/2$ and using $\bar{a}[a] = [a^2]$

$$s[e] = \bar{p}[u].$$

Eliminating the shock speed we have

$$[\tau] = \frac{-[u]^2}{[p]} \text{ and } [e] = \frac{\bar{p}[u]^2}{[p]}.$$

Using any equation of state of the form $p = p(\tau, e)$ we get a relationship strictly between p_+ and u_+ given τ_- , e_- , u_- and $p_-(\tau_-, e_-)$ as

$$p_+ = p\left(\tau_- - \frac{[u]^2}{[p]}, e_- + \frac{\bar{p}[u]^2}{[p]}\right).$$

For an ideal equation of state

$$p = (\gamma - 1)\rho e = (\gamma - 1)e/\tau \quad (1.11)$$

we get a quadratic relation for either p_+ or u_+ in terms of the other. We will use this information later for the solution of the Riemann problem. Notice that two arbitrary states cannot be connected by a single shock. Also be aware that a solution where the relative pressure and velocity increase through a shock is viable. This is called an expansion shock and is unphysical. Like the nonlinear hyperbolic scalar equations, entropy conditions must be imposed to rule out unphysical shocks. For the fluid equations a physical entropy is known and the entropy condition that must be satisfied is the physical entropy must increase across a shock. A characteristic entropy condition generalizing the scalar case has also been given. The equivalence of the two entropy conditions is proven in Lax [29].

Let us now look at some other special solutions of the Euler equations called simple waves (also rarefactions or expansions). The primitive equations can be written in matrix form as

$$U_t + AU_x = 0$$

where

$$U = \begin{pmatrix} \rho \\ u \\ p \end{pmatrix} \text{ and } A = \begin{pmatrix} u & \rho & 0 \\ 0 & u & 1/\rho \\ 0 & \rho c^2 & u \end{pmatrix}.$$

We look for similarity solutions of the form

$$U = U(\xi) = U(x/t).$$

Note that shocks are also similarity solutions of a very simple type — the discontinuity travels in a single direction in space time ($x/t = s$). Substitute the similarity solution into the primitive equations yields a nonlinear eigenvalue problem of the form

$$(A - I\xi)U'(\xi) = 0$$

where the wave speed, ξ , is an eigenvalue. Because the system is hyperbolic, the wave speeds exist and are u , $u + c$ and $u - c$. For the wave speeds $u \pm c$, the corresponding relations from the eigenvector are

$$\frac{d\rho}{d\xi} = \alpha, \quad \frac{du}{d\xi} = \pm \frac{\alpha c(\rho)}{\rho(\xi)} \text{ and } \frac{dp}{d\xi} = \alpha c^2(\rho(\xi)).$$

α is an arbitrary normalization constant for the eigenvector and must be determined along with the other physical quantities. The independence of these

equations is called *genuine nonlinearity*. Again, like the shock case, with an equation of state for gamma law gases of the form

$$p = p_0 \left(\frac{\rho}{\rho_0} \right)^\gamma \quad \text{and} \quad c^2 = \frac{\gamma p}{\rho}$$

we can use these relations to find a direct relationship between the pressure p and the velocity u

$$p = p(u) = p_0 \left(1 \pm \frac{(\gamma - 1)}{2c_0} (u - u_0) \right)^{\frac{2\gamma}{\gamma - 1}}$$

where p_0 , u_0 , c_0 and ρ_0 are known at some reference state.

The eigenvalue corresponding to $\xi = u$ yields what is called a *linearly degenerate* or *contact* solution. That is the corresponding eigenvector only determines that pressure p and velocity u are constant. Using the equation of state (1.11), density and internal energy can be increased and decreased respectively without changing the pressure. Therefore, as long as pressure and velocity are constant contact solutions may appear with related jumps in internal energy and density.

We have outlined the algebraic relations for shocks, simple waves and contacts. Given two constant arbitrary states, these three types of solutions can be pieced together to connect the given states. The connecting solution is also called the Riemann problem. The exact solution of the Riemann problem is complicated by the number of permutations of shocks, contacts and rarefactions¹ A complete description of this problem is found in Courant and Friedrichs [18]. In the numerical solution of hyperbolic equations Riemann solvers are frequently used but these solvers are approximate and extremely simplified.

Because of the existence of discontinuities in hyperbolic systems as shown above the mathematical theory for hyperbolic systems is limited. In particular, most results for general systems are in the small — solutions near constant states. A good example of theoretical work in this area is by Glimm [24].

1.2 Hyperbolic-Parabolic Systems

In this part of the chapter we discuss the consequences of adding diffusive terms to the hyperbolic equations and systems described above. Although equations change type and have infinite signal speeds, we demonstrate that as the diffusive terms become small solution behavior approaches that of hyperbolic systems. For nearly singular equations we make the argument that parabolic equations can be efficiently approximated using AMR.

¹Finding the solution involves connecting the two states through matching pressure and velocity from a combination of shocks and expansion solutions. The contact solution is used to mediate jumps in internal energy and density between the two solutions.

1.2.1 Linear Advection Diffusion Equations

Consider the problem

$$u_t + cu_x = \kappa u_{xx}$$

where

$$u(x, 0) = g(x), \quad c, \kappa > 0 \quad \text{and} \quad -\infty < x < \infty.$$

It is easy to see that the solution, for integrable g , is

$$u(x, t) = \frac{1}{(4\pi\kappa t)^{\frac{1}{2}}} \int_{-\infty}^{\infty} e^{-((x-ct)-\xi)^2/(4\kappa\pi t)} g(\xi) d\xi.$$

The kernel of the integral operator is a translating delta function for with speed c and thickness proportional to κt . This tells us for small κ the advection-diffusion equation behaves in a wavelike manner for bounded times. Alternatively, for small κ the advection-diffusion equation is a singular perturbation problem with small parameter κ . For initially sharp fronts, regions away from fronts are treated using the wave equation and sharp boundary layers are constructed to connect the solutions. Using either interpretation fronts that are initially sharp remain so.

Now consider the related problem

$$u_t + u_x = \kappa u_{xx}$$

on a closed interval where

$$c, \kappa > 0, \quad 0 \leq x \leq 1, \quad u(0, t) = 1, \quad u(1, t) = 0 \quad \text{and} \quad u(x, 0) = g(x).$$

For long times the steady state solution has the form

$$u_s(x) = \frac{1 - e^{-(1-x)c/\kappa}}{1 - e^{-c/\kappa}}.$$

This is called a boundary layer solution. It is characterized by a rapid change on a small interval near 1. The boundary layer can be quite thin if c/κ is very large or holding c fixed and making κ small. Thus, in addition to supporting sharp solutions, the advection-diffusion equation can also support boundary layers.

1.2.2 Navier-Stokes Equations

A nonlinear system of parabolic equations that has all of the features described in the previous section plus many new singularities not found in linear equations

is the Navier-Stokes Equations. We write the Navier-Stokes equations in vector form as

$$\begin{aligned}\frac{d\rho}{dt} + \rho \nabla \cdot \vec{u} &= 0 \\ \rho \frac{d\vec{u}}{dt} + \nabla \cdot \vec{\mathbf{P}} &= 0 \\ \rho \frac{de}{dt} + \vec{\mathbf{P}} : (\nabla \vec{u}) &= \nabla \cdot \vec{q}\end{aligned}$$

where a constitutive relation is defined as

$$\vec{\mathbf{P}} = [p + (2/3\mu - \kappa)(\nabla \cdot \vec{u})]I - \mu[(\nabla \vec{u}) + (\nabla \vec{u})^T],$$

and a Fick's law for energy is assumed as

$$\vec{q} = -\lambda \nabla T,$$

where closure is achieved with thermodynamic relations

$$p = p(\rho, T) \quad T = T(\rho, e).$$

The Navier-Stokes equations differ from the Euler equations in several ways. First, the quantity $\vec{\mathbf{P}}$ is called a pressure tensor. It is composed of the thermodynamic pressure p and two other terms which are often called the bulk and shear viscosity. The bulk viscosity term, the second part of the coefficient of the identity tensor, contains the bulk viscosity coefficient κ . The shear viscosity term, the last part of the tensor pressure expression, contains the shear viscosity coefficient μ . Besides diffusion present in the momentum equation there is energy diffusion in the internal energy equation. The energy flux is proportional to the gradient in temperature T . The proportionality constant is λ . Usually temperature or internal energy is eliminated through the second thermodynamic relationship above. For polytropic gases

$$T = c_v e$$

where c_v is the specific heat at constant volume.

The Navier-Stokes equations are of parabolic type. As λ , μ and κ approach zero the Euler equations are recovered in the limit. Further, shocks satisfying entropy conditions for the Euler equations are the limits of smooth jumps in the Navier-Stokes equations. The mathematical foundations for Navier-Stokes is somewhat limited. It can be shown using standard Cauchy-Kovalewky arguments that smooth initial conditions lead to smooth short time solutions. However, long time results for Navier-Stokes has eluded researchers.

1.3 Hyperbolic-Elliptic Systems

Hyperbolic systems with elliptic constraints, like parabolic systems, have infinite signal speeds. Unlike parabolic equations or systems, large changes can be triggered in one part of a domain from changes in another. However, from elliptic regularity, the changes are smooth. Once again, hyperbolic-elliptic systems may make good candidates for efficient solution approximation using AMR embedded solution algorithms.

The full Navier-Stokes equations are difficult to analyze. However, an important simplification is much more accessible. This simplification is the incompressible Navier-Stokes equations

$$\begin{aligned}\bar{u}_t + \bar{u} \cdot \nabla \bar{u} + \nabla p &= \mu \Delta \bar{u} \\ \nabla \cdot \bar{u} &= 0.\end{aligned}$$

These equations are derived by expanding the compressible set of equations around an assumed solution. The expansion parameter is the Mach number M which is the ratio of the fluid speed $|\bar{u}|$ divided by the sound speed c .

$$\begin{aligned}\rho &= \rho_0 + M^2 \rho_1 + M^4 \rho_2 + \dots \\ \bar{u} &= \bar{u}_0 + M^2 \bar{u}_1 + M^4 \bar{u}_2 + \dots \\ p &= \frac{p_0}{M^2} + P_1 + M^2 p_2 + \dots \\ T &= T_0 + M^2 T_1 + M^4 T_2 + \dots\end{aligned}$$

The coefficient for p_0 is such as to keep the physical scale lengths fixed while letting the pressure go to infinity. Alternatively, in the Mach number M the velocity $|\bar{u}|$ remains fixed while the sound speed goes to infinity. The use of the square of the Mach is often justified by experimentalist observing incompressible behavior of fluids with Mach numbers as high as .3. A final assumption that must be made to insure the correct limit is met is the heat conduction, λ , must be large to equilibrate the temperature on a time scale much smaller than the one of interest.

Upon examination of the incompressible equations, it is not clear how pressure p is calculated. The pressure acts as a constraint on the fluid to keep it divergence free. By computing the divergence of (1.3) we have

$$-\nabla \cdot (\bar{u} \cdot \nabla \bar{u}) = \Delta p. \tag{1.12}$$

By solving the associated Laplace equation, pressure can be computed to within a constant.

Chorin [15] formalized this approach by using the Hodge decomposition. This decomposition is a pair of orthogonal projections that split a given vector field in a domain Ω into a divergence free field and potential field

$$\vec{u} = \vec{u}_d + \nabla\phi = \vec{u}_d + \vec{u}_p$$

The decomposition is easily found by computing the divergence of the above equation

$$\Delta\phi = \nabla \cdot \vec{u}$$

Within a given domain Ω . The boundary condition for ϕ is a homogeneous Neumann condition

$$\frac{\partial\phi}{\partial\hat{n}} = 0$$

on the boundary of the domain $\partial\Omega$ where \hat{n} is the normal to the boundary. The projection operators are then defined as

$$\begin{aligned}\vec{u}_p = Q\vec{u} &= \nabla(\Delta^{-1}(\nabla \cdot \vec{u})) \\ \vec{u}_d = P\vec{u} &= (I - Q)\vec{u}\end{aligned}$$

Applying P to the incompressible equations gives a pure evolution equation for the momentum

$$\vec{u}_t = P(-\vec{u} \cdot \nabla\vec{u} + \mu\Delta\vec{u}).$$

Using (1.12) the pressure p , to within a constant, can be found. Equivalently the gradient of the pressure may be found from the Hodge decomposition

$$\nabla p = Q(-\vec{u} \cdot \nabla\vec{u} + \mu\Delta\vec{u}).$$

Later we will exploit these relationships to formulate a numerical method.

1.3.1 Boundary Layers and Vortices

In addition to shocks, contacts and rarefactions seen in the inviscid limit of the Navier-Stokes equation, boundary layers, vortices and vortex sheets are highly localized phenomena observed in the incompressible limit of the equations. Boundary layers have the same exponential structure seen in the model advection-diffusion equations. Vortices and vortex sheets can easily be seen through some simple analysis.

Consider the upper half plane problem

$$\begin{aligned}u_t + uu_x + vv_y + p_x &= \mu \Delta u \\v_t + uv_x + vv_y + p_y &= \mu \Delta v \\u_x + v_y &= 0\end{aligned}$$

for $y \geq 0$ and

$$u = v = 0$$

for $y = 0$ and

$$u = u_0 \text{ and } v = 0$$

at $y = \infty$. There is a flow that is independent of x . Make the assumption that

$$u = u(y, t) \text{ and } v \equiv 0$$

which directly leads to

$$u_t = \mu u_{yy}.$$

The heat equation for u has a classical solution of the form

$$u(y, t) = \frac{2u_0}{\sqrt{\pi}} \int_0^{y/2\sqrt{\mu t}} e^{-s^2} ds$$

where the integral is the error function. Like the advection-diffusion equation, it is readily observed that for fixed y and small time t a boundary layer is seen.

The 2-D steady state *inviscid* incompressible equations can be written in cylindrical coordinates as

$$\begin{aligned}uu_r + \frac{v}{r}u_\theta - \frac{v^2}{r} + p_r &= 0 \\uv_r + \frac{v}{r}v_\theta + \frac{uv}{r} + \frac{1}{r}p_\theta &= 0 \\\frac{1}{r}(ru)_r + \frac{1}{r}v_\theta &= 0\end{aligned}$$

where u is the radial velocity, v is the angular velocity and p is, again, the pressure. By inspection

$$u = 0, \quad v = v(r) \text{ and } p = p_0 + \int_{r_0}^r \frac{v(R)^2}{R} dR.$$

satisfy this equation for differentiable v . A perfectly reasonable solution is

$$v(x, t) = \begin{cases} 1 & \text{for } 0 \leq r < r_0 \\ \left(\frac{r-r_0}{\epsilon}\right)^3 - \left(1 - \frac{r-r_0}{\epsilon}\right)^3 - \frac{3}{\epsilon}(r-r_0) + 2 & \text{for } r_0 \leq r < r_0 + \epsilon \\ 0 & \text{for } r \geq r_0 + \epsilon. \end{cases}$$

This profile is a rotating cylinder of fluid that is local and can have an arbitrarily steep edge. This cylinder of fluid is called a vortex and its behavior is decoupled in a manner reminiscent of a contact discontinuity.

Another inviscid solution in Cartesian coordinates (u is now the velocity in the x direction) is

$$u(x, t) = \begin{cases} 1 & \text{for } y > \epsilon/2 \\ 2\left(1 - \frac{y+\epsilon/2}{\epsilon}\right)^3 - 2\left(\frac{y+\epsilon/2}{\epsilon}\right)^3 - \frac{6}{\epsilon}y & \text{for } -\epsilon/2 \leq y \leq \epsilon/2 \\ -1 & \text{for } y < -\epsilon/2. \end{cases}$$

This can be seen from inspection of the steady state Cartesian equations. The solution structure is called a shear layer. Again the solution can be made arbitrarily steep by making ϵ go to zero.

In both examples a cubic polynomial was chosen in order to have one degree of differentiability to smoothly connect constant states. Any smooth function with the same matching conditions at the endpoints would be sufficient. What is important to notice is the incompressible inviscid equations support steep localized solutions. In general, if the viscosity μ is small enough, the inviscid profiles used as initial conditions for the viscous case persist. Again, the localized structure of solutions like those mentioned above point to numerical solution methods embedded within an AMR framework.

Chapter 2

Numerical Methods

In this chapter we discuss several topics that lay the foundations for a patched based adaptive mesh algorithm. First, grids and their associated mappings are described. The mappings will also allow an opportunity to describe the notation we will use for writing down finite difference schemes. In a following section we describe a C++ array class that facilitates the implementation of the numerical methods that will be used on these mesh domains. Finally we describe a range of finite difference methods that will be the basis for the algorithms described in the following chapters.

2.1 Finite Difference Grids and Notation

2.1.1 Grids and Mappings

The nonlinear systems of equations described in the previous chapter most often have no closed form solution. They are solved numerically using finite difference techniques. All structured finite difference techniques use a lattice or mesh of points on some domain of interest. At each mesh point an algebraic equation called a *discretization* is written that approximates the original partial differential equations or their boundary conditions. For the initial value problem the initial conditions are sampled at the mesh points and are then advanced in time through the solution of the discretization.

We use *logically* rectangular meshes or grids. In general, points are generated from a smooth mapping from the unit interval, square or cube into the appropriate 1-, 2- or 3-D domain. In two dimensions a mapping will have the form

$$\begin{aligned}x_{i,j} &= y(i\Delta\xi, j\Delta\eta) \\ y_{i,j} &= y(i\Delta\xi, j\Delta\eta).\end{aligned}$$

where $i = 0, \dots, M - 1$; $j = 0, \dots, N - 1$ and

$$\Delta\xi = \frac{1}{M-1} \text{ and } \Delta\eta = \frac{1}{M-1}.$$

Cartesian grids are simply orthogonal mappings

$$\begin{aligned} x(\xi, \eta) &= a_1\xi + b_1 \\ y(\xi, \eta) &= a_2\eta + b. \end{aligned}$$

It is easy to derive discretizations on these lattices. Cartesian grids can only be used for a limited set of computational domains and historically analysts have turned to more complicated mappings as the complexity of computational domains increased. As domains became even more complicated, particularly in the aerodynamics community, it became increasingly more difficult to find a single mapping that would conform to a given computational domain. Therefore multiple domains or components were patched together to cover a region of space

$$\begin{aligned} {}_k x_{i,j} &= {}_k x(i\Delta\xi, j\Delta\eta) \\ {}_k y_{i,j} &= {}_k y(i\Delta\xi, j\Delta\eta) \end{aligned}$$

where $i = 0, \dots, M_k - 1$; $j = 0, \dots, N_k - 1$; and $k = 1, \dots, C$. C is the number of component grids. These grids are called *multiblock* grids. Where domains are adjacent to each other in physical space, mesh points must be collocated although faces of these mapped domains need not be completely shared.

Overset grids are also multiple component meshes. What differentiates overset grids from multiblock grids is that alignment constraints set in multiblock grids are relaxed. Overset grids are only required to overlap so that no part of the computational domain is left uncovered. We will see later that the discretization becomes more complicated at overlap boundaries of overset grids, but the flexibility of having overlapping grids seems to be worthwhile. A good source book on grids and grid generation is by Thompson [41]. The article by Chesshire and Henshaw [14] gives a complete description of overset grids.

An alternative to overset and multiblock grids are embedded boundary grids. Here a single Cartesian grid is used but irregular boundaries are placed within the mesh. Special techniques are then applied to the irregular cells created near the embedded boundaries. These same special techniques will be used for internal boundaries of overset grids when they are hybridized with embedded boundary cutaways at internal boundaries.

2.1.2 Notation

A discrete *grid function* on a given grid domain reflects the grid topology. For example a scalar grid function u is represented on the grid as

$${}_k u(x_{i,j}, y_{i,j}) = {}_k u_{i,j}.$$

We use the convention that integer values of indices indicate cell centers, one half integer value mixed with whole integer values indicates edges and all half integer values indicate nodes. Often continuous and discrete function names will be the same. This will not cause confusion because it will always be clear from context whether a discrete or continuous function is being used. When possible, subscripts and/or superscripts may be omitted. When they are left out, default values can be assumed. For example

$$\begin{aligned} {}_k u_{i,j} &= u \\ {}_k u_{i+1,j} &= u_i \\ {}_k u_{i,j+\frac{1}{2}} &= u_{j+\frac{1}{2}} \\ &\vdots \end{aligned}$$

leaves out all values of indices that are not offset by an integer or half integer value. Like the reuse of names for both continuous and discrete functions, context will always make it clear what meaning a discretization has.

2.2 An Array Class in C++

The implementation language that is used for the exercises for this course is C++. C++ has several advantages over FORTRAN that are exploited. First, C++ is extensible through the definition of structures called classes. Classes are entities that may contain both data and functions, called members, that have access to and manipulate the class data. Putting together functions and data is called *encapsulation*. Classes are generically referred to as objects. Second, the language facilitates application of a broader class of data structures than FORTRAN through the use of pointers and recursion.

To handle the compute intensive part of our work we will use a predefined library called A++/P++¹. A++/P++, developed by Quinlan [31, 36], is a collection of classes that facilitate the manipulation of multidimensional arrays of floating point or integer type. The syntax is suggestive of FORTRAN 90 but has features not found in that language. Perhaps the most significant feature of this library is the ability to run on serial or parallel computers with little modification of a program. In fact single grid codes developed in a serial environment require only a recompilation with a new library in order to run in parallel.

¹"A" in A++ is used to signify array while "P" in P++ is used to signify parallel

We first describe how logically rectangular domains can be described in A++/P++. We start with a *Range* object example:

```
Range I(1,10,1);
```

Here the starting point of a domain is the index 1 and the endpoint is 10. The stride is 1 and could, in this case, be left out for the default stride is 1. Using Range objects an array can easily be specified:

```
Range I(1,10), J(1,10);  
doubleArray A(I,J);
```

The array A is an 10 x 10 double precision array having indices ranging between 1 and 10 in both coordinate directions. The Range object and double precision array are overloaded. Overloaded means that operations typically reserved for simple arithmetical operations have meaning for array and Range objects. For example, the following code fragment implements a simple relaxation scheme:

```
Range IB(0,11), JB(0,11);  
Range I (1,10), J (1,10);  
  
doubleArray A(IB,JB), B(IB,JB), C(IB,JB);  
  
.  
.  
.  
  
C(I,J) = 0.5*A(I,J) + 0.125*(B(I+1,J) + B(I,J+1) +  
B(I-1,J) + B(I,J-1) );
```

In this example, I+1 is a new Range object varying from 2 to 11 with stride 1. Notice also, that the "+" operator between array expressions signifies that all 100 array elements are added simultaneously. Further notice the scalar 0.5 is promoted to an array that multiplies all 100 elements of array A. Conditional expressions are available. The following example gives an example of conditional syntax:

```
intArray A(10), B(10);  
  
.  
.  
.  
  
where(A > 0){  
    B = 1;  
}  
elsewhere(){  
    B = -1;  
}
```

Wherever the integer array A is positive, then B is assigned the value 1, otherwise B is assigned -1. Again all conditionals, like arithmetic operations, take place element by element within the block conditional. For the conditional and array expressions to function properly all the array expressions must *conform* or have the same shape. Also notice that A and B were constructed using the integer 10. The integer 10 is promoted to a Range object from 0 to 9 with stride 1. This part of the design mimics C array syntax.

A large collection of functions are overloaded for array operations. For example in the following fragment

```
floatArray A(20,20), B(20,20);
double pi = 3.1415926;

B = sin(pi*max(0.0, min(A, 1.0)));
```

the 20 element single precision array, B, received the sin of π times the values of A constrained to a range of 0.0 to 1.0.

A more complete description of the A++/P++ library can be found on the World Wide Web at <http://www.c3.lanl.gov/~dquinlan>. The serial and parallel libraries are in the public domain and are highly portable. The library distribution can be found on the World Wide Web at the same locations.

2.3 Upstream Centered Difference Methods

2.3.1 1-D Methods

We use upstream centered finite difference methods as a basis for many of our AMR algorithms. The simplest example of such a method is the Courant, Issacson and Rees [19] scheme for the scalar wave equation (1.1)

$$\frac{u^{n+1} - u}{\Delta t} + c \frac{u - u_{i-1}}{\Delta x} \text{ for } c > 0. \quad (2.1)$$

We have introduced a superscript n to denote the time level, Δt is a time step and Δx is the mesh spacing. Notice that superscripts and subscripts have been left out. In particular if a superscript is missing the time level n is assumed and if a subscript is missing the spatial index i is used. No grid component index is used as there is only one grid. The above equation can be solved, explicitly, for u^{n+1} in terms of the u_i^n thereby enabling the advance of the solution a time step Δt

$$\begin{aligned} u^{n+1} &= u - \frac{c\Delta t}{\Delta x}(u - u_{i-1}) \\ &= u - \lambda(u - u_{i-1}) \\ &= u(1 - \lambda) + \lambda u_{i-1}. \end{aligned}$$

λ is called the Courant Friedrichs Levy (CFL) number. If $\lambda \leq 1$ then the finite difference scheme is *stable*. Stability in the case of the wave equation is that the solution's maximum value does not increase. Remember that all the wave equation does is translate the initial conditions. Therefore the continuous solution and, as we shall see, the discrete solution has a maximum principle. The maximum principle for the discrete case is easily seen because u^{n+1} is a linear combination of positive weighted values of u_i^n where the positive weights add up to one.

$$\begin{aligned} |u_i^{n+1}|_\infty &= \max_i |u_i^{n+1}| \\ &\leq (1 - \lambda)|u_i^n|_\infty + \lambda|u_{i-1}^n|_\infty \\ &= |u_i^n|_\infty \end{aligned}$$

Another interpretation of this scheme is pictorial. Each cell value represents a piecewise constant density with the integrated area under this value representing a mass. For $\lambda \leq 1$ the scheme has a geometric interpretation illustrated by figure 2.1. The piecewise constant solution in (a) is translated a distance

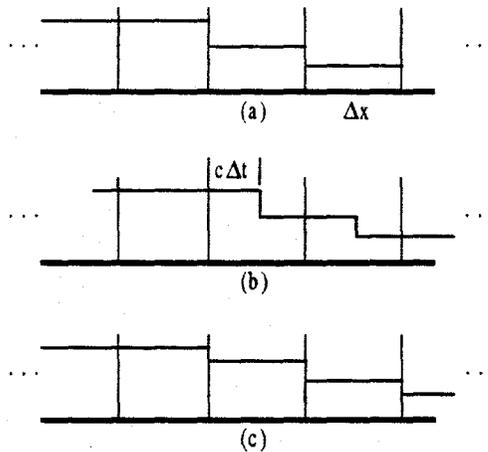


Figure 2.1: A geometric picture of the evolution of a piecewise constant solution one time step.

$c\Delta t < \lambda\Delta x$ in (b). The density profiles are then integrated over the interval and a new average value is placed in each cell (c). Also note that

$$\Delta x u^{n+1} = \Delta x u - (\Delta t c u - \Delta t c u_{i-1})$$

is equivalent to the original scheme. The *mass* of a cells at time level $n + 1$ is equal to the mass at time level n plus the differences of the mass fluxes entering and leaving the cell.

Another important property of scheme (2.1) is it is consistent. Using Taylor series expansions for a solution $u(x, t)$ of the scalar equation at the point $(n\Delta t, i\Delta x)$

$$\begin{aligned} u((n+1)\Delta t, i\Delta x) &= u(n\Delta t, i\Delta x) + \Delta t \frac{\partial u}{\partial t}(n\Delta t, i\Delta x) + \\ &\quad \frac{\Delta t^2}{2!} \frac{\partial^2 u}{\partial t^2}(n\Delta t, i\Delta x) + \mathcal{O}(\Delta t^3) \\ u(n\Delta t, (i-1)\Delta x) &= u(n\Delta t, i\Delta x) - \Delta x \frac{\partial u}{\partial x}(n\Delta t, i\Delta x) + \\ &\quad \frac{\Delta x^2}{2!} \frac{\partial^2 u}{\partial x^2}(n\Delta t, i\Delta x) + \mathcal{O}(\Delta x^3). \end{aligned}$$

Substituting these expansions into the difference scheme yields

$$\begin{aligned} &\frac{u((n+1)\Delta t, i\Delta x) - u(n\Delta t, i\Delta x)}{\Delta t} + c \frac{u(n\Delta t, i\Delta x) - u(n\Delta t, (i-1)\Delta x)}{\Delta x} \\ &= \frac{\Delta t}{2} \frac{\partial^2 u}{\partial t^2}(n\Delta t, i\Delta x) - c \frac{\Delta x}{2} \frac{\partial^2 u}{\partial x^2}(n\Delta t, i\Delta x) + \mathcal{O}(\Delta t^3) + \mathcal{O}(\Delta x^3). \end{aligned}$$

Since $\Delta t = \lambda \Delta x / c$, the right hand side of the above relation, called the *truncation error*, has the form

$$\frac{\lambda \Delta x}{2c} \frac{\partial^2 u}{\partial t^2}(n\Delta t, i\Delta x) - c \frac{\Delta x}{2} \frac{\partial^2 u}{\partial x^2}(n\Delta t, i\Delta x) + \mathcal{O}(\Delta x^3) = \mathcal{O}(\Delta x)$$

assuming that the second derivatives of the solution are bounded. In other words the truncation error goes to zero as $\Delta x \rightarrow 0$. The power of the rate at which the truncation error goes to zero is the *order* of the approximation. This *scheme* is first order. If the truncation error goes to zero as the mesh is refined we call the scheme consistent.

Given a consistent and stable difference scheme it can be proven for a large class of evolution equations that the difference solution converges to the analytic solution with the order of the difference scheme. We prove convergence of the scalar scheme. Let the error at a cell be defined as

$$e_i^n = u_i^n - u(n\Delta t, i\Delta x),$$

then we can write a difference scheme for the error as

$$\frac{e^{n+1} - e}{\Delta t} + c \frac{e - e_{i-1}}{\Delta x} = \tau_i^n.$$

The truncation error can be bounded such that

$$|\tau_i^n| \leq C_1 \Delta x$$

if the initial values have bounded second derivatives. The error estimates at various time levels can be easily listed using the maximum principle and truncation error:

$$\begin{aligned} |e_i^0|_\infty &= 0 \\ |e_i^1|_\infty &\leq C_1 \Delta x^2 \\ |e_i^2|_\infty &\leq |e_i^1|_\infty + C_1 \Delta x^2 \\ &\leq 2C_1 \Delta x^2 \\ &\vdots \\ |e_i^n|_\infty &\leq |e_i^{n-1}|_\infty + (n-1)C_1 \Delta x^2 \\ &\leq nC_1 \Delta x^2 \end{aligned}$$

Using the definition of the CFL number and letting $T = n\Delta t$

$$|e_i^n|_\infty \leq \frac{Tc}{\lambda} C_1 \Delta x.$$

Fixing T the estimate bounds the error so that it diminishes linearly with the mesh spacing Δx . Therefore the scheme is convergent. The stability (maximum principle) and consistency (truncation error bounds) were both used in the proof of convergence.

In practice, linearly convergent schemes are not as efficient as higher order methods. We now introduce a two step scheme called a predictor-corrector method with second order convergence properties. The method is referred to as the Fromm scheme [22]. As before we have a flux form of the scalar equation which is now called the corrector step

$$\frac{u^{n+1} - u}{\Delta t} + c \frac{u_{i+\frac{1}{2}}^{n+\frac{1}{2}} - u_{i-\frac{1}{2}}^{n+\frac{1}{2}}}{\Delta x} = 0.$$

If the predictor yields $u_{i+\frac{1}{2}}^{n+\frac{1}{2}} = u$ then the first order scheme is recovered. The Fromm scheme uses a predictor of the form

$$u_{i+\frac{1}{2}}^{n+\frac{1}{2}} = u + \frac{1}{4}(1-\lambda)(u_{i+1} - u_{i-1})$$

that contributes to the overall second order convergence of the scheme. This can be verified by expanding the analytic solution in a Taylor series around the point $((n + \frac{1}{2})\Delta t, i\Delta x)$. It is easily seen that the approximate time derivative term is second order, but some work is necessary to understand the spatial truncation error. Naively the spacial approximation is only first order because the second order approximation of the derivative in space is divided by Δx . The functional form of the $u_{i+\frac{1}{2}}^{n+\frac{1}{2}}$ terms only vary by $\mathcal{O}(\Delta x)$ giving an extra order to the spatial difference. This is often called Lax-Wendroff cancelation.

The Fromm scheme has a geometric interpretation just like the first order method. Figure 2.2 shows in a) reconstruction of the linear profiles from a cell

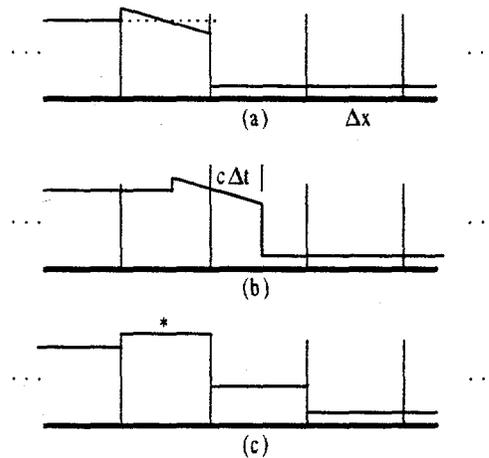


Figure 2.2: A geometric picture of the evolution of a piecewise linear solution one time step.

density value and two neighboring cell values. In b) the profile is translated and c) the areas are integrated to get new cell density values.

It should be obvious that the form of the first and second order difference schemes are dependent upon the sign of the coefficients of the differential operator. For example, if the wave speed c is negative the Fromm scheme has the predictor form

$$u_{i+\frac{1}{2}}^{n+\frac{1}{2}} = u_{i+1} - \frac{1}{2}(1 + \lambda)\Delta_i u_{i+1}$$

and the first order scheme samples the data in the other direction

$$u_{i+\frac{1}{2}}^{n+\frac{1}{2}} = u_{i+1}.$$

The schemes are called *upstream-centered* to indicate that they adapt to the characteristic direction of the differential equation. This sensitivity to characteristic direction will become more apparent for nonlinear equations.

Observe that the cell in figure 2.2 with a "*" over it in (c) has a value that exceeds the values to the left or right of it and the values at the previous time step. This scheme violates the maximum principle. These local violations of the maximum principle are sometimes called *undershoots* or *overshoots*. Godunov (proof given in [32]) proved that there is no linear scheme that is second order and has a maximum principle. However, van Leer [42] discovered a nonlinear scheme that is second order and has no overshoots. Define

$$u_{i+\frac{1}{2}}^{n+\frac{1}{2}} = u + \frac{1}{2}(1 - \lambda)\Delta_i u$$

where $\Delta_i u$ is called a *slope* or *tilt*. The *unlimited* slope operator is defined as

$$\Delta_i u = \frac{1}{2}(u_{i+1} - u_{i-1}).$$

Define a *limited* slope operator as

$$\bar{\Delta}_i u = \delta \text{sign}(u_{i+1} - u_{i-1})$$

such that

$$\delta = \begin{cases} \min(\frac{1}{2}|d_0 u|, 2|d_+ u|, 2|d_- u|) & \text{for } (d_+ u)(d_- u) \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

and

$$d_+ u = u_{i+1} - u, \quad d_- u = u - u_{i-1}, \quad \text{and } d_0 u = u_{i+1} - u_{i-1}.$$

The Fromm scheme using limited slopes in the predictor step

$$u_{i+\frac{1}{2}}^{n+\frac{1}{2}} = u + \frac{1}{2}(1 - \lambda)\bar{\Delta}_i u$$

is *monotone* (Creates no new maxima or minima). This is observed most easily using the geometric interpretation. In the first part of figure 2.3 the slopes are reconstructed so they don't force the reconstruction lines to exceed the average values of the neighboring cells. Once the reconstruction is completed, the translation and integration is identical to the unlimited case. It can also be shown algebraically that the van Leer limiter is optimal — anything less will

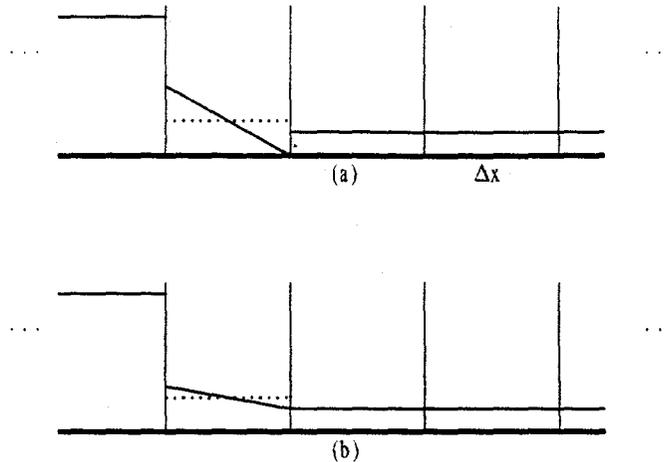


Figure 2.3: A geometric picture of the evolution of a limited piecewise linear solution one time step.

result in overshoots or undershoots. At the same time van Leer limiting effects the overall order of the convergence of the scheme since it reduces the order of the predictor to first order near local minima or maxima. Therefore van Leer limiting should only be used with data that has discontinuous initial conditions or is known to develop discontinuous solutions.

2.3.2 2-D Unsplit Methods

1-D methods described previously may be used in multiple dimensions through splitting techniques such as by Strang [40] or Samarski [39]. Our focus will be on unsplit schemes. To this end we consider an unsplit scheme published by Colella [17] generalizing the geometric interpretation found in the previous section of these notes. 3-D generalizations can be found in Saltzman [38]. Consider a scalar advection equation in 2-D

$$\frac{du}{dt} = u_t + au_x + bu_y = 0, \quad a, b > 0.$$

By inspection the solution, with initial conditions $f(x, y)$, is

$$u(x, y, t) = f(x - at, y - bt).$$

The characteristic direction is a constant vector, $(a, b, 1)$, in space-time. The

analogue of constructing profiles, translating them and integrating over the cell areas is only slightly more complicated in two dimensions. In figure 2.4 a rectangle is translated in the negative characteristic direction to determine where the solution is translated from. A first order scheme is constructed by multiplying

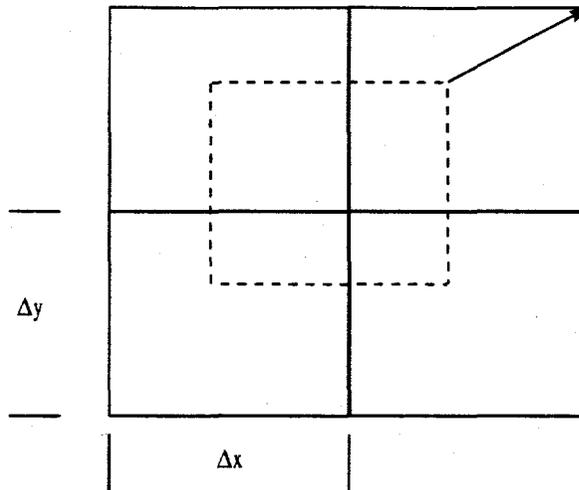


Figure 2.4: A geometric picture of the evolution of a 2-D piecewise constant solution one time step.

the piecewise constant values in the four rectangles intersected by the translated rectangle with the areas of those intersection. The products are summed and divided by the area of a complete cell $(\Delta x, \Delta y)$. This can be expressed in two ways. First

$$u^{n+1} = \frac{((\Delta x - a\Delta t)(\Delta y - b\Delta t)u + (a\Delta t)(\Delta y - b\Delta t)u_{i-1} + (b\Delta t)(\Delta x - a\Delta t)u_{j-1} + (a\Delta t)(b\Delta t)u_{i-1,j-1})}{(\Delta x\Delta y)}$$

shows that u^{n+1} is written as a sum of values of u at time level n with positive weights summing to one. This is only true if the CFL condition

$$\max\left(\frac{a\Delta t}{\Delta x}, \frac{b\Delta t}{\Delta y}\right) \leq 1.$$

Therefore the scalar equation, has a maximum principle and is monotone like

the continuous equation. A second, equivalent, way of expressing the scheme is the predictor-corrector form

$$\frac{u^{n+1} - u}{\Delta t} + a \frac{u_{i+\frac{1}{2}}^{n+\frac{1}{2}} - u_{i-\frac{1}{2}}^{n+\frac{1}{2}}}{\Delta x} + b \frac{u_{j+\frac{1}{2}}^{n+\frac{1}{2}} - u_{j-\frac{1}{2}}^{n+\frac{1}{2}}}{\Delta y} = 0 \quad (2.2)$$

$$\begin{aligned} u_{i+\frac{1}{2}}^{n+\frac{1}{2}} &= u - \frac{b\Delta t}{2} \left(\frac{u - u_{j-1}}{\Delta y} \right) \\ u_{j+\frac{1}{2}}^{n+\frac{1}{2}} &= u - \frac{a\Delta t}{2} \left(\frac{u - u_{i-1}}{\Delta x} \right). \end{aligned}$$

What is apparent from this form is how to make the scheme second order. As in one dimension, $u_{i+\frac{1}{2}}^{n+\frac{1}{2}}$ and $u_{j+\frac{1}{2}}^{n+\frac{1}{2}}$ must be second order in time and space. The Lax-Wendroff cancellation will maintain second order spatial differences. Using a Taylor series expansion (we use the finite difference notation for the continuous functions for brevity)

$$u_{i+\frac{1}{2}}^{n+\frac{1}{2}} = u + \frac{\Delta x}{2} \frac{\partial u}{\partial x} + \frac{\Delta t}{2} \frac{\partial u}{\partial t} + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)$$

and then substituting the differential operator into the expansion yields

$$u_{i+\frac{1}{2}}^{n+\frac{1}{2}} = u + \frac{\Delta x}{2} \frac{\partial u}{\partial x} - \frac{\Delta t}{2} \left(a \frac{\partial u}{\partial x} + b \frac{\partial u}{\partial y} \right) + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2).$$

Approximating the derivatives with differences obtains

$$u_{i+\frac{1}{2}}^{n+\frac{1}{2}} = u + \frac{1}{2} \left(\left(1 - \frac{a\Delta t}{\Delta x} \right) \Delta_i u - \frac{b\Delta t}{2\Delta y} (u - u_{j-1}) \right) + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2).$$

The predictor step then has the form

$$\begin{aligned} u_{i+\frac{1}{2}}^{n+\frac{1}{2}} &= u + \frac{1}{2} (1 - \lambda_x) \Delta_i u - \frac{1}{2} \lambda_y (u - u_{j-1}) \\ u_{j+\frac{1}{2}}^{n+\frac{1}{2}} &= u + \frac{1}{2} (1 - \lambda_y) \Delta_j u - \frac{1}{2} \lambda_x (u - u_{i-1}) \end{aligned}$$

where $\lambda_x = a\Delta t/\Delta x$ and $\lambda_y = b\Delta t/\Delta y$ are the CFL numbers in each coordinate direction. To minimize overshoots and undershoots limited slopes $\bar{\Delta}_i$ and $\bar{\Delta}_j$ may be used instead of the centered differences Δ_i and Δ_j .

2.3.3 Linear Systems

Approximating the solution of linear systems of hyperbolic PDEs follows the treatment of scalar equations. As was seen in the first chapter, systems have multiple characteristic directions so the predictor step must be modified. A clue as to how to do this modification comes from the associated diagonalized systems in the first chapter. Diagonalization would lead to a direct component by component treatment of the system using the scalar schemes developed in 1-D. However, diagonalization will not work for multidimensional systems of nonlinear systems.

Instead we use the solution of a Riemann problem as a basis for our numerical methods. To see this we approximate system (1.2) in the following manner. The corrector step has the form

$$\frac{U^{n+1} - U}{\Delta t} + A \frac{U_{i+\frac{1}{2}}^{n+\frac{1}{2}} - U_{i-\frac{1}{2}}^{n+\frac{1}{2}}}{\Delta x} = 0$$

where U is a vector. As in the scalar case we leave out unnecessary indices. The predictor step is

$$\begin{aligned} U_{i+\frac{1}{2}}^{n+\frac{1}{2}} &= U_L + P^-(U_R - U_L) \\ &= U_R - P^+(U_R - U_L) \\ &= R^{(x)}(U_L, U_R) \end{aligned}$$

where

$$\begin{aligned} U_L &= U + \frac{1}{2} \left(I - \frac{\Delta t A}{\Delta x} \right) \Delta_i U \\ U_R &= U_{i+1} - \frac{1}{2} \left(I + \frac{\Delta t A}{\Delta x} \right) \Delta_i U_{i+1} \end{aligned}$$

and

$$\begin{aligned} P^+ \phi &= \sum_{\Gamma_i > 0} (\phi \cdot e_{\Gamma_i}) e_{\Gamma_i} \\ P^- \phi &= \sum_{\Gamma_i < 0} (\phi \cdot e_{\Gamma_i}) e_{\Gamma_i} \end{aligned}$$

e_{Γ_i} is a normalized right eigenvector of A corresponding to eigenvalue Γ_i . The projections P^+ and P^- break up the jump $U_R - U_L$ into increments that travel from left to right and right to left respectively. This is nothing more than

diagonalizing and solving the system with piecewise initial conditions U_L and U_R . Further, the only part of the Riemann problem we are interested in is at the time level $n + \frac{1}{2}$ and spatial coordinate $i + \frac{1}{2}$. This is sometimes referred to as edge centered (with respect to the discretization cell) and time centered data. The solution we seek is either the left state U_L plus projected increments crossing from the right or the right state U_R minus projected increments crossing from the left. By examining the associated diagonal system, the CFL condition for stability is

$$\max_i \left(\frac{\Gamma_i \Delta t}{\Delta x} \right) \leq 1.$$

It is observed that even though the Riemann solver only solves a problem with two piecewise constant states, the Fromm scheme is still quadratically convergent. The proof involves showing that the Riemann solver only chooses between second order states on the left and right. Therefore the solver does not reduce the order of the fluxes.

Since all the operators are linear, we have actually diagonalized the systems and solved the equivalent set of scalar equations. For multidimensional systems and nonlinear scalar equations and systems there may not exist a diagonalization. The methodology for deriving the difference schemes can still be applied. Let us now approximate a solution of the system

$$U_t + AU_x + BU_y = 0.$$

The corrector step is the same as the unsplit scheme (2.2)

$$\frac{U^{n+1} - U}{\Delta t} + A \frac{U_{i+\frac{1}{2}}^{n+\frac{1}{2}} - U_{i-\frac{1}{2}}^{n+\frac{1}{2}}}{\Delta x} + B \frac{U_{j+\frac{1}{2}}^{n+\frac{1}{2}} - U_{j-\frac{1}{2}}^{n+\frac{1}{2}}}{\Delta y} = 0.$$

Riemann solvers are used to resolve various states

$$\begin{aligned} U_{i+\frac{1}{2}}^{n+\frac{1}{2}} &= R^{(x)}(U_{L,i+\frac{1}{2}}, U_{R,i+\frac{1}{2}}) \\ U_{j+\frac{1}{2}}^{n+\frac{1}{2}} &= R^{(y)}(U_{B,j+\frac{1}{2}}, U_{R,j+\frac{1}{2}}). \end{aligned}$$

How these Riemann solvers are formulated is discussed below. The left and right states are defined below as

$$\begin{aligned} U_{L,i+\frac{1}{2}} &= U + \frac{1}{2} \left(I - \frac{\Delta t}{\Delta x} A \right) \Delta_i U - \frac{\Delta t}{2\Delta y} B (U_{j+\frac{1}{2}}^* - U_{j-\frac{1}{2}}^*) \\ U_{R,i+\frac{1}{2}} &= U_{i+1} - \frac{1}{2} \left(I + \frac{\Delta t}{\Delta x} A \right) \Delta_i U_{i+1} - \frac{\Delta t}{2\Delta y} B (U_{i+1,j+\frac{1}{2}}^* - U_{i+1,j-\frac{1}{2}}^*) \end{aligned}$$

$$\begin{aligned}
U_{B,j+\frac{1}{2}} &= U + \frac{1}{2} \left(I - \frac{\Delta t}{\Delta y} B \right) \Delta_j U - \frac{\Delta t}{2\Delta x} A (U_{i+\frac{1}{2}}^* - U_{i-\frac{1}{2}}^*) \\
U_{T,j+\frac{1}{2}} &= U_{j+1} - \frac{1}{2} \left(I + \frac{\Delta t}{\Delta y} B \right) \Delta_j U_{j+1} - \frac{\Delta t}{2\Delta x} A (U_{i+\frac{1}{2},j+1}^* - U_{i-\frac{1}{2},j+1}^*)
\end{aligned}$$

Finally the $U_{i+\frac{1}{2}}^*$ and $U_{j+\frac{1}{2}}^*$ are computed from Riemann solutions

$$\begin{aligned}
U_{i+\frac{1}{2}}^* &= R^{(x)}(U, U_{i+1}) \\
U_{j+\frac{1}{2}}^* &= R^{(y)}(U, U_{j+1}).
\end{aligned}$$

In practice, a slightly more accurate estimate for $U_{i+\frac{1}{2}}^*$ and $U_{j+\frac{1}{2}}^*$ is used:

$$\begin{aligned}
U_{i+\frac{1}{2}}^* &= R^{(x)}(U_{L,i+\frac{1}{2}}^*, U_{R,i+\frac{1}{2}}^*) \\
U_{j+\frac{1}{2}}^* &= R^{(y)}(U_{B,j+\frac{1}{2}}^*, U_{T,j+\frac{1}{2}}^*).
\end{aligned}$$

where

$$\begin{aligned}
U_{L,i+\frac{1}{2}}^* &= U + \frac{1}{2} \left(I - \frac{\Delta t}{\Delta x} A \right) \Delta_i U \\
U_{R,i+\frac{1}{2}}^* &= U_{i+1} - \frac{1}{2} \left(I + \frac{\Delta t}{\Delta x} A \right) \Delta_i U_{i+1} \\
U_{B,j+\frac{1}{2}}^* &= U + \frac{1}{2} \left(I - \frac{\Delta t}{\Delta y} B \right) \Delta_j U \\
U_{T,j+\frac{1}{2}}^* &= U_{j+1} - \frac{1}{2} \left(I + \frac{\Delta t}{\Delta y} B \right) \Delta_j U_{j+1}.
\end{aligned}$$

There is no additional computational expense² using this modification as $U_{L,i+\frac{1}{2}}^*$, $U_{R,i+\frac{1}{2}}^*$, $U_{B,j+\frac{1}{2}}^*$ and $U_{T,j+\frac{1}{2}}^*$ are computed in the original algorithm. The terms using the above states are called the *transverse* terms. The fluxes affected by these terms are often referred to as high order transverse fluxes (HOTF). Consequently low order transverse fluxes (LOTF) refers to the original fluxes computed by piecewise constant estimates of adjacent states. Although the overall solution is more accurate the HOTF terms don't affect the overall order of accuracy.

The Riemann solvers are used to resolve states moving in the x direction or the y direction. Therefore $R^{(x)}(U_L, U_R)$ need only be the Riemann solution for the system

²The overall stencil size is larger than the original method which may be a consideration when implementing these algorithms on a parallel computer.

$$U_t + AU_x = 0$$

while $R^{(y)}(U_B, U_T)$ need only be the Riemann solution for

$$U_t + BU_y = 0.$$

2.3.4 Nonlinear Equations

The approximate solution of nonlinear scalar and systems of hyperbolic equations is easily generalized from the linear methods. For scalar equations we use the Burger equation (1.4) as a prototype. The corrector step is, once again, time and space centered as

$$\frac{u^{n+1} - u}{\Delta t} + \frac{f(u_{i+\frac{1}{2}}^{n+\frac{1}{2}}) - f(u_{i-\frac{1}{2}}^{n+\frac{1}{2}})}{\Delta x} = 0.$$

The flux function is $f(u) = u^2/2$. As in the linear case is

$$u_{i+\frac{1}{2}} = R^{(x)}(u_L, u_R).$$

The Riemann solution for Burgers' equation is computed using (1.6) or (1.7). $u_{L,i+\frac{1}{2}}$ and $u_{R,i+\frac{1}{2}}$ are from the predictor step such that

$$\begin{aligned} u_{L,i+\frac{1}{2}} &= u + \frac{1}{2} \left(1 - \frac{a(u)\Delta t}{\Delta x} \right) \Delta_i u \\ u_{R,i+\frac{1}{2}} &= u_{i+1} - \frac{1}{2} \left(1 + \frac{a(u_{i+1})\Delta t}{\Delta x} \right) \Delta_i u_{i+1}. \end{aligned}$$

Note that the CFL number $a(u)\Delta t/\Delta x$ is a function of the cell index. Therefore for stability

$$\max_i \frac{a(u_i)\Delta t}{\Delta x} \leq 1.$$

The generalization of scalar equations to systems is also clear. Scalar quantities are replaced by vector or matrix quantities and the stability condition must take into account all the wave speeds at all cells. For a system of the form

$$U_t + F(U)_x = 0$$

the corrector step is

$$\frac{U^{n+1} - U}{\Delta t} + \frac{F(U_{i+\frac{1}{2}}^{n+\frac{1}{2}}) - F(U_{i-\frac{1}{2}}^{n+\frac{1}{2}})}{\Delta x} = 0.$$

where

$$U_{i+\frac{1}{2}}^{n+\frac{1}{2}} = R^{(x)}(U_{L,i+\frac{1}{2}}, U_{R,i+\frac{1}{2}})$$

is some approximation to the Riemann problem. As stated previously, little of the Riemann solution is actually used. There are many articles on the approximate solution of the Riemann problem and approximate flux functions [35]. The predictor step can have various degrees of complexity. First, if we just use the states $U_{L,i+\frac{1}{2}} = U$ and $U_{R,i+\frac{1}{2}} = U_{i+1}$ we have a first order method. This method is called Godunov's method as was first applied to the Euler equations in Lagrangian form [25]. We next describe the simplest second order form of the predictor

$$\begin{aligned} U_{L,i+\frac{1}{2}} &= U + \frac{1}{2} \left(I - \frac{\Delta t}{\Delta x} \frac{\partial F}{\partial U}(U) \right) \Delta_i U \\ U_{R,i+\frac{1}{2}} &= U_{i+1} - \frac{1}{2} \left(I + \frac{\Delta t}{\Delta x} \frac{\partial F}{\partial U}(U_{i+1}) \right) \Delta_i U_{i+1}. \end{aligned}$$

The Jacobian matrix $\frac{\partial F}{\partial U}$ is used like the matrices in the linear system solvers.

Let Γ_i be the eigenvalues of the Jacobian matrix $\frac{\partial F}{\partial U}$ and e_{Γ_i} be the associated eigenvectors. The eigenvalues and eigenvectors are functions of the cell index. From hyperbolicity of the system

$$\begin{aligned} \Delta_i U &= \sum_{\Gamma_i > 0} (e_{\Gamma_i} \cdot \Delta_i U) e_{\Gamma_i} + \sum_{\Gamma_i < 0} (e_{\Gamma_i} \cdot \Delta_i U) e_{\Gamma_i} + \sum_{\Gamma_i = 0} (e_{\Gamma_i} \cdot \Delta_i U) e_{\Gamma_i} \\ &= (P^+ + P^- + P^0) \Delta_i U \end{aligned}$$

Letting only waves moving in the appropriate direction gives a new predictor step

$$\begin{aligned} U_{L,i+\frac{1}{2}} &= U + \frac{1}{2} \left(I - \frac{\Delta t}{\Delta x} \frac{\partial F}{\partial U}(U) \right) P^+ \Delta_i U \\ U_{R,i+\frac{1}{2}} &= U_{i+1} - \frac{1}{2} \left(I + \frac{\Delta t}{\Delta x} \frac{\partial F}{\partial U}(U_{i+1}) \right) P_{i+1}^- \Delta_i U_{i+1}. \end{aligned}$$

This would be redundant for a linear system but has some effect for nonlinear systems. As with the linear systems limited slopes may be applied for solutions expected to have large gradients or discontinuities.

The nonlinear solvers can be directly used in splitting methods for multidimensional computation. The generalizations used in moving from linear to nonlinear 1-D systems will also serve well in modifying unsplit methods for nonlinear systems. Once again matrices are replaced by Jacobians, linear Riemann solvers based on projections of waves are replaced by appropriate approximations of nonlinear waves and the corrector matrix multiplies are replaced by flux functions. The references for unsplit methods [17, 38] describe linear and nonlinear schemes in more detail.

Chapter 3

Adaptive Mesh Refinement for Conservation Laws

In this chapter we describe the basic structured AMR algorithm as applied to a hyperbolic system of conservation laws. By way of introduction we describe predecessors to structured AMR. These include the moving mesh and moving finite element methods. Work in adaptive unstructured mesh generation is beyond the scope of this manuscript.

Following the introduction to the basic algorithm, we discuss the necessary changes needed to handle static mapped and moving mapped grids. Next we examine how to apply AMR techniques to overset grids. The description of the overset grid method will include nonconservative and conservative treatments.

3.1 Moving Mesh Methods

Suppose there is a moving coordinate system described by

$$\begin{aligned}x &= x(\tau, \xi), & 0 \leq \xi \leq 1, & & 0 \leq \tau \leq T \\t &= \tau, & x(\tau, 0) = a, & & x(\tau, 1) = b.\end{aligned}$$

Using this coordinate change the scalar wave equation transforms to

$$u_\tau + (c - v_g)u_\xi / x_\xi = 0$$

where

$$v_g = \frac{\partial x}{\partial \tau}.$$

By choosing $v_g = c$, $u_\tau = 0$ or u stays constant which is simply the characteristic solution. Of course it is not always clear how to choose an optimal moving coordinate system but mesh generation techniques coupled with error estimators have been effective in reducing error in moving mesh algorithms. Suppose $w(x, t)$ is a function that is strictly positive, bounded away from zero and proportional to the numerical error of a given finite difference scheme when the error is above a given tolerance (to guarantee boundedness away from zero). Minimize the integral

$$I[w, x](t) = \int_a^b F(w, x_\xi) dx = \int_a^b w(x, t) x_\xi dx = \int_a^b w(x, t) (x_\xi)^2 d\xi$$

as a functional of $x(\xi)$ by solving the Euler equation (This Euler equation has nothing to do with fluids. It represents an infinite dimensional derivative, called the Frechet Derivative, set to zero)

$$\frac{\partial F}{\partial x} - \frac{\partial}{\partial \xi} \frac{\partial F}{\partial x_\xi} = \frac{\partial w}{\partial x} - 2 \frac{\partial^2 x}{\partial \xi^2}$$

Since $w = w(x, t)$ then $x = x(\tau, \xi)$ and the solution of Euler equation is therefore capable of generating a moving mesh. The integral that is minimized has the property that wherever the error is large $\frac{\partial x}{\partial \xi}$ is small. Therefore wherever the error is large, the mesh spacing is made small. In fact the error is equidistributed by variational integral and the mesh is optimal in reducing the error. In 1-D the method has had success [45]. The method has been extended to higher dimensions with some success [10]. There are some drawbacks to this method.

- The number of mesh points is fixed.
- The mesh may not be smooth or can tangle in higher dimensions. This leads to adding mesh regularization in the variational principle degrading the effect of the error estimator.
- Implicit methods may be needed to insure the adaptive mesh keeps up with the solution error.
- Mesh generation is computationally expensive for complex regions. Therefore adaptive mesh generation may be prohibitively expensive.

Another approach to moving mesh methods is the moving finite element (MFE) method by Miller [23]. The solution of an evolution equation is approximated with a set of basis functions of compact support

$$u(x, t) = \sum_i a_i(t) \phi(s_i, s_{i\pm 1}, \dots, s_{i\pm n}).$$

The basis functions are defined by the location of the nodes s_i . For example, the linear hat function is defined by three nodes (s_{i-1}, s_i, s_{i+1}) where ϕ_i increases

linearly from zero to one between s_{i-1} and s_i and then decreases linearly back to zero from s_i to s_{i+1} . In the MFE method the nodes are allowed to move as a function of time

$$s_i = s_i(t), \quad i = 1, \dots, n.$$

The residual of the L^2 norm of the evolution equation is minimized and evolution equations for $\dot{a}_i(t)$ and $\dot{s}_i(t)$ arise. The same drawbacks have been encountered for the MFE method as for the moving mesh method. However, in recent years, mesh regularization techniques have been refined enough so that successful multidimensional computations have been accomplished [13].

3.2 Structured Adaptive Mesh Refinement

Adaptive mesh refinement is a concept that probably was apparent with the advent of finite difference methods. However, the implementation of this concept did not appear until computer software and hardware gained some sophistication. Multidimensional static (non moving) adaptive mesh refinement for parabolic problems was first discussed in a publication by Ciment and Sweet [16]. Much work appeared in one dimension. For example Davis and Flaherty [21] developed a finite element method with refinement. Gropp [26] developed a moving refined region to follow shocks in a scalar conservation solution. Berger and Oliger [8] published the first structured adaptive mesh refinement algorithms in two dimensions. Refinements of this method appeared in Berger and Colella [7]. Three dimensional computations were first done by Bell et al. [2].

3.2.1 Rectangular Tensor Product Grids

The structured adaptive mesh algorithms we describe are based on logically rectangular, hierarchical, properly nested grid *patches*. Logically rectangular patches are simply mappings described in the second chapter. We initially talk describe an algorithm for grids which are tensor products of 1-D mappings. The grid hierarchy is a set of grid levels such that the finer grid spacing is an integer multiple of the adjacent coarser level.

Finer grids may cover sets of cells on coarser grids but all cells will have data to minimize data communication between grid levels. To avoid multiple valued data, only the finest level cell covering a given point will be used as a function value.

To further facilitate and simplify communication between grid levels meshes are properly nested. Proper nesting is designed to insure that interpolation to a level n grid patch only occurs from adjacent level n patches or level $n - 1$ patches. Near boundaries this requirement is relaxed since boundary conditions are used instead of interpolation.

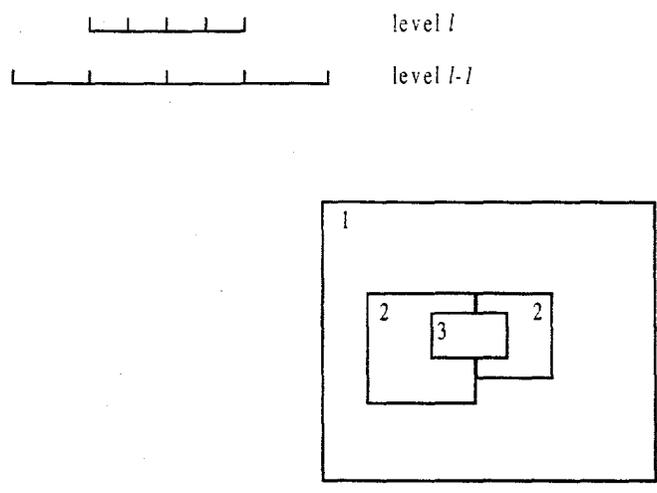


Figure 3.1: Examples of a adaptive mesh hierarchies in one and two dimensions.

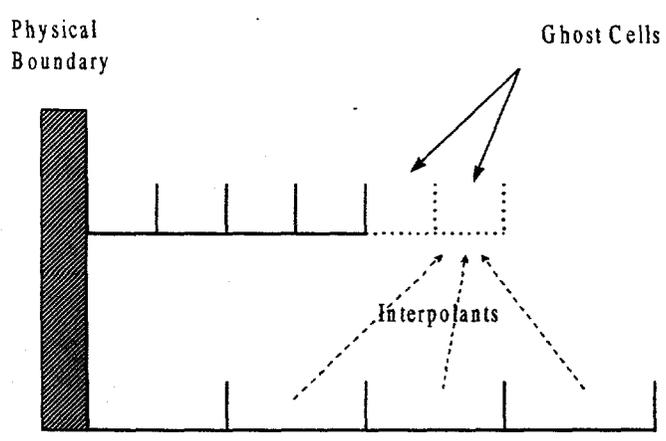


Figure 3.2: Proper nesting insures filling boundary ghost cells using physical boundaries, data from adjacent patches at the same level or interpolation from the next coarsest level.

To generate the grid hierarchy an error estimator is used to flag cells with high error on a given grid level n to generate or update grids on level $n + 1$. Once cells are flagged, they are "boxed" into patches at level $n + 1$ using the Berger-Rigoutsos algorithm [5].

Error estimation can be done using estimates of the local truncation error. The simplest method is to use *gradient lengths*. Often large gradients are a good indicator of large truncation error. In a 1-D difference scheme the gradient length is defined for a positive definite quantity ρ as

$$e_i = \frac{|\rho_{i+1} - \rho_{i-1}|}{\rho_i}$$

Another way of estimating error is to use the leading order truncation error terms of the given difference scheme. There are examples of direct calculation of leading order truncation error in the literature [30]. Two problems are encountered with this approach. First, the complexity and cost of computing the truncation error for a system of equations is large. More significant is truncation errors typically are higher order differentials leading to larger difference stencils and noisier values.

Another way to estimate the leading order truncation error terms is to use Richardson's method. Here the error is estimated by comparing two solutions on two different grids. Let $L_h(\Delta t)$ be an operator that advances a finite difference solution one time step Δt on a mesh with uniform spacing h . Further assume the operator has the same order spatial and temporal truncation error. Truncation error analysis shows for a q^{th} order scheme with a fixed CFL number that

$$u(t_0 + \Delta t) - L_h(\Delta t)u(t_0) = h^{q+1}f(x, t_0) + \mathcal{O}(h^{q+2})$$

where u is a smooth enough solution of the PDE associated with the difference operator $L_h(\Delta t)$. Taking two steps

$$u(t_0 + 2\Delta t) - L_h^2(\Delta t)u(t_0) = 2h^{q+1}f(x, t_0) + \mathcal{O}(h^{q+2}).$$

Applying the difference operator with a time and space grid of size $2\Delta t$ and $2h$ gives

$$u(t_0 + 2\Delta t) - L_{2h}(2\Delta t)u(t_0) = (2h)^{q+1}f(x, t_0) + \mathcal{O}(h^{q+2}).$$

Subtracting and isolating the truncation error gives

$$\begin{aligned} L_h^2(\Delta t) - L_{2h}(2\Delta t) &= 2h^{q+1}f(x, t_0) - (2h)^{q+1}f(x, t_0) + \mathcal{O}(h^{q+2}) \\ &= (2 - 2^{q+1})h^{q+1}f(x, t_0) + \mathcal{O}(h^{q+2}) \\ \frac{L_h^2(\Delta t) - L_{2h}(2\Delta t)}{2 - 2^{q+1}} &= \tau + \mathcal{O}(h^{q+2}) \end{aligned}$$

where τ is the local truncation error. τ can be found if a solution at the previous time level is kept for all grid patches. First coarsen the old time level solution by a factor of two (this implies the refinement factor must have a power of two contained in it). Advance the coarsened solution one time step with the current CFL. Notice this solution will get advanced two time levels because of the coarser mesh spacing. Advance the current mesh solution one step and coarsen it. Compare the two data sets using the above formula to derive an estimate of the truncation error. To leading order the truncation error will have been computed and can be used as an error estimator.

Once cells are flagged several layers of cells next to flagged cells are marked to act as a buffer between regriding steps. Because of the CFL limit imposed on each level, the number of buffer cells, n_b , is equal to the number of cycles between regriding, n_{grad} , divided by the refinement factor r .

$$n_b = \frac{n_{grad}}{r}.$$

Once cells are determined to have large truncation error new grid patches must be formed. The Rigoutsos-Berger boxing algorithm can be used to generate a list of patches. The steps of the algorithm are as follows:

- 1 Find the smallest box containing all the flagged points from error estimation (see part a of figure 3.3).
- 2 Compute all the row sums and column sums of flagged points (see part b of figure 3.3).
- 3 Find the largest difference of all the adjacent row sums and divide the box into two at that point (see part c of figure 3.3).
- 4 If one or two of the boxes generated are satisfactory add them to a list. Otherwise apply steps 1-3 to one or both of the new boxes that are not satisfactory.

Given a hierarchy of adaptive meshes, we now describe how to advance the solutions on all mesh patches. The advance is a recursive algorithm so it will be sufficient to describe it for two adjacent levels. Assume we have a collection of mesh patches at level $l - 1$ and another collection at level l that is refined by a factor r relative to level $l - 1$. That is, if level $l - 1$ has mesh spacing h then level l has mesh spacing h/r .

The first step in this process is to advance the numerical solution on all the level $l - 1$ patches. To do so, boundary conditions are needed for each patch. This is done by adding the appropriate number of *ghost cells* or *border cells* around the edges of each patch. The ghost cells are filled in by using data from other level $l - 1$ patches, interpolating from level $l - 2$ patches, if they exist, or using physical boundary conditions. The proper nesting of the grid hierarchy

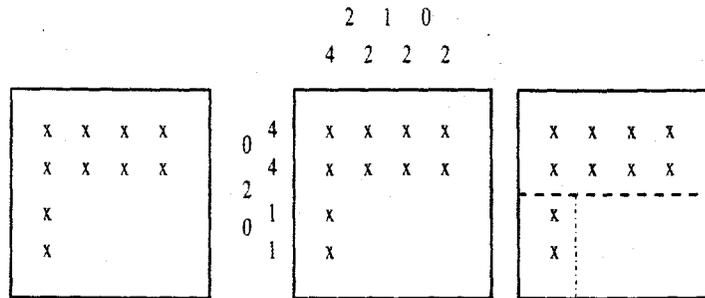


Figure 3.3: A sequence of pictures of a flagged region illustrating box selection using the Berger-Rigoutsos algorithm.

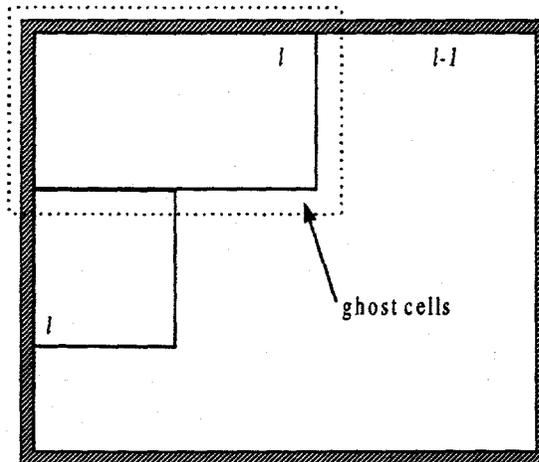


Figure 3.4: An example of a 2-D patch that has its ghost cells filled by physical boundary conditions, interpolation from the next coarsest level or from data of adjacent patches at the same level.

guarantees that the ghost cells can be filled in using only adjacent grids, grids one level lower or from physical boundary conditions.

With the level $l - 1$ mesh patches advanced from time t to time $t + \Delta t$ we can then advance level l patches using the data from level $l - 1$ as boundary conditions. To advance level l at least r steps of size $\Delta t/r$ are taken. The reason for smaller time steps is the CFL stability limit. The method now becomes recursive here. If there is a finer level $l + 1$, for each time step on level l at least r time steps will be taken on level $l + 1$. Returning to level l , ghost cells are again filled from physical boundary conditions, adjacent level l patches and interpolated from level $l - 1$ data. The interpolation from level $l - 1$ data must be done in space and time.

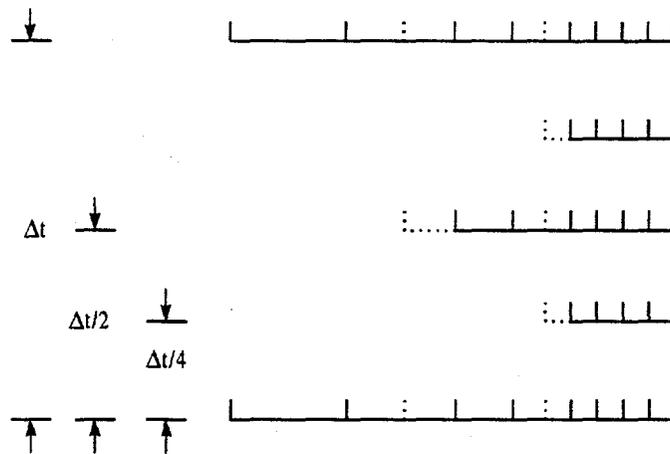


Figure 3.5: The recursive advance of three levels is illustrated with the coarsest mesh advanced first, then fine grids are advanced using coarser grid data interpolated in space and time.

The solutions at level l and $l - 1$ coincide at a new time $t + \Delta t$ after r or more time steps are taken on level l . Once solutions coincide, data from the finer levels are used to update data on coarser levels. This is done by averaging all the cell values in level l that cover $l - 1$ cells. For example, in 1-D, given a cell with index j and value ρ_j at level $l - 1$ and cells at level l with indices i and $i + 1$ that cover cell j ($r = 2$) then

$$\rho_j^{(l-1)} = \frac{1}{2}(\rho_i^{(l)} + \rho_{i+1}^{(l)}).$$

Treating the $\rho^{(l)}$ as conserved quantities, notice the mass $m_j^{(l)}$ in the cell j is

the same as the sum of the masses in cells i and $i + 1$

$$m_j^{(l-1)} = \Delta x \rho_j^{(l-1)} = \frac{\Delta x}{2}(\rho_i^{(l)} + \rho_{i+1}^{(l)}) = m_i^{(l)} + m_{i+1}^{(l)}.$$

Can the adaptive mesh hierarchy be constrained to be conservative? The answer is yes. By averaging fine mesh data onto coarser mesh data the conserved quantities are made identical. Coarse cells that are adjacent to cells covered by a finer grids need some correction. Correction of coarse cells adjacent to fine cells is called *refluxing*. Suppose cell i at level l is adjacent to cell $2i + 1$ at level $l + 1$ where a refinement factor r is assumed. Cell i is advanced using

$$\frac{u^{n+1} - u}{\Delta t} + \frac{f_{i+\frac{1}{2}}^{n+\frac{1}{2}} - f_{i-\frac{1}{2}}^{n+\frac{1}{2}}}{\Delta x} = 0.$$

Cell $2i + 1$ at level $l + 1$ is advanced twice to get to level to the same time level as cell i

$$\begin{aligned} \frac{u^{2n+1} - u^{2n}}{\Delta t/2} + \frac{f_{2i+\frac{3}{2}}^{2n+\frac{1}{2}} - f_{2i+\frac{1}{2}}^{2n+\frac{1}{2}}}{\Delta x/2} &= 0 \\ \frac{u^{2n+2} - u^{2n+1}}{\Delta t/2} + \frac{f_{2i+\frac{5}{2}}^{2n+\frac{3}{2}} - f_{2i+\frac{3}{2}}^{2n+\frac{3}{2}}}{\Delta x/2} &= 0 \end{aligned}$$

The mass flux

$$\Delta t f_{i+\frac{1}{2}}^{n+\frac{1}{2}} \neq \frac{\Delta t}{2}(f_{2i+\frac{1}{2}}^{2n+\frac{1}{2}} + f_{2i+\frac{3}{2}}^{2n+\frac{3}{2}}).$$

For overall conservation on the coarse grid we use the fine mass fluxes

$$\Delta x(u^{n+1} - u^n) + \Delta t(f_{i+\frac{1}{2}}^{n+\frac{1}{2}} - f_{i-\frac{1}{2}}^{n+\frac{1}{2}}) + \Delta t \left(\frac{f_{2i+\frac{1}{2}}^{2n+\frac{1}{2}} + f_{2i+\frac{3}{2}}^{2n+\frac{3}{2}}}{2} - f_{i+\frac{1}{2}}^{n+\frac{1}{2}} \right) = 0.$$

The second term is a mass flux correction. By accumulating the fine grid fluxes at coarse/fine interfaces and keeping the coarse fluxes present the correction can be applied after levels $l - 1$ and l are aligned at the same time level $n + 1$ and $2n + 2$ respectively.

The adaptive algorithm changes the nature of the difference scheme. The boundary values for fine grids sometimes comes from the next coarser grid level. Using a normal mode analysis Berger [6] has shown that for a Lax-Wendroff scheme the boundary conditions are stable. Using sufficient accuracy in the interpolation, the adaptive algorithm also preserves order of accuracy of the integration scheme.

The implementation of AMR methods requires data structures that are not often used with other solution methods for PDEs. As will be seen, object oriented languages, like C++, can be used to great advantage to implement these new structures. Before outlining some of the new data structures, we first describe a way of relating index or domain coordinates at different levels of an adaptive grid. The integer index for the coarsest grid in two dimensions may range over i and j as

$$(i, j): \quad 0 \leq i < M, \quad 0 \leq j < N, \quad \text{for level 0.}$$

Given a refinement factor r , the index space then has indices

$$(i^{(1)}, j^{(1)}): \quad 0 \leq i^{(1)} < rM, \quad 0 \leq j^{(1)} < rN, \quad \text{for level 1.}$$

In general indices at level l have ranges

$$(i^{(l)}, j^{(l)}): \quad 0 \leq i^{(l)} < r^l M, \quad 0 \leq j^{(l)} < r^l N.$$

For adaptive methods in 2-D, a patch can then be uniquely located in domains described above given two index coordinates. One index coordinate contains the pair of integers that are the minimum values in each coordinate direction of a patch. The other index coordinate contains the pair of integers that are the maximum values in each coordinate direction of the same patch. A convenient data structure to handle this pair of coordinates is called a *Box*. Boxes have operations associated with them, called class member functions. For example, a function may intersect two boxes and return and third. Boxes have proven such a fundamental abstraction that a library of classes built upon boxes have been developed called BoxLib [43]. A prototype declaration in C++ for a Box class may be

```
class Box{
public:
    Box(int *idMin, int idMax*, int dim); // constructor
    ~Box(); // destructor
    . // public member
    . // functions follow
    .

private:
    int indexMin[3];
    int indexMax[3];
    int dimension;
    . // private member
    . // functions follow
};
```

The next data structure of interest is called a *Patch*. A Patch contains data used in the time integration of the solution. Patches usually contain a Box, arrays for the solution at its current time level and one time level back. Arrays are also allocated to store fluxes used in the conservative update described above. The patch data structure has many member functions. In particular, a patch should be able to advance the solution data a given time step if it is given boundary conditions. For nonlinear systems the CFL number is solution dependent. Therefore, every patch must compute its own CFL number. A prototype declaration in C++ for a Patch class may look like:

```
class Patch{
public:
    Patch(Box &b);           // constructor using a Box
    ~Patch();               // destructor
    .                       // public member
    .                       // functions follow
    .

private:
    Box patchBox;          //
    doubleArray uOld, uNew; // solutions at two levels
    doubleArray fluxes, fluxRegister; // stored fluxes for
    .                       // conservation

    void advanceSolution(REAL dt); // advance the solution
    REAL computeCFL();           // compute a CFL number
};
```

In the above patch definition a doubleArray called "fluxRegister" is initialized. This is used for accumulating fluxes in time to correct coarse/fine interface disparities. A more sophisticated approach would simply be to define four 1-D arrays (for 2-D). Each array would be for one face of the patch. This saves memory but increase implementation complexity. Each level is composed of patches. Levels are implemented using a container class from what is called the Standard Template Library (STL). The data structure that we use is called a *vector*. A vector of levels of pointers to patches is simply instantiated as:

```
#include <vector.h>

vector<vector<patch *> > patchLists;
```

Vectors are dynamic (they automatically grow in size as needed) and are accessed like C arrays using the square bracket operator. Therefore the fragment

```
Patch *p;
p = patchList[i][j];
```

retrieves the j th patch pointer on level i .

Together, the above data structures comprise what is needed to implement a hyperbolic AMR solver. Although the data structures are relatively simple, their dynamic nature would be difficult to duplicate in an older scientific programming language like FORTRAN 77. FORTRAN 90 has many features, modules and arrays in particular, that would facilitate an AMR implementation. Using an object oriented language also facilitates code reuse. For example, only the patch data structure has solver specific information within it. Therefore the AMR algorithm can be reused except for the Patch class. Even much of the Patch class can be put into a generic patch class called a *base* class. Using an object oriented mechanism called *inheritance* only a small amount of code would be needed to make a complete Patch class.

3.2.2 Curvilinear Meshes

So far, we have only dealt with rectangular grids. If grids are only logically rectangular the AMR algorithm remains nearly the same. A few details need to be addressed. For example, a cell that is refined into an r^2 set of cells may not have the same area as the sum of the r^2 cells.

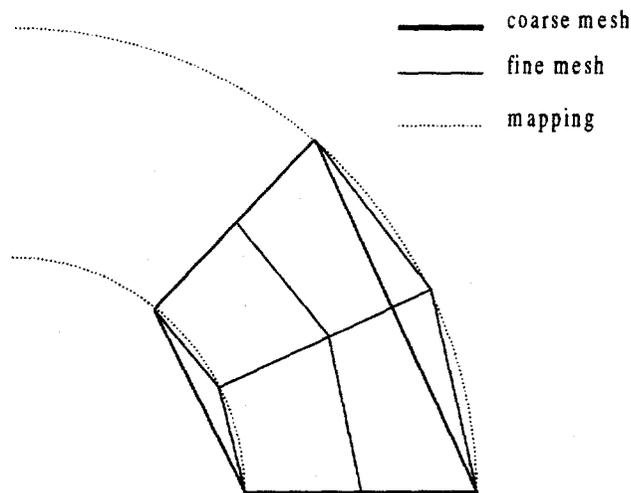


Figure 3.6: A coarse cell and 4 fine cells generated using a cylindrical mapping.

A simple practice to mitigate this problem is to redefine areas on the underlying and adjacent coarse grids in terms of the sums of areas on the fine grids.

Arc lengths at coarse/fine interfaces are also inconsistent. However, this is

taken care of in a similar fashion. The arc lengths are chosen such that they agree with the refined region arc lengths. That is, the arclength of a coarse cell edge adjacent to a number of fine cells is redefined as the sum of the arc lengths of the fine cells. Both of these concepts can be generalized to three dimensions where edge arch lengths become face areas and cell areas become cell volumes. Making both the arch lengths/face areas and the cell areas/volumes consistent leads to what is called *free stream preservation*. Free stream preservation is the property that a constant solution remains constant across coarse/fine grid interfaces.

3.2.3 Moving Meshes

Moving meshes introduce no significant new problems for adaptive meshes. Because grids continually move in time, it is worthwhile having an evolution equation for cell volumes. Following Bell, et al. [4] the volume V^n of a cell is advanced as

$$V^{n+1} = V^n + \sum_{i \in \text{faces}} dV_i. \quad (3.1)$$

The volume fluxes dV_i are computed by taking the face or edge values at new ($n+1$) and old (n) times and computing the volumes or areas formed by the faces and edges respectively.

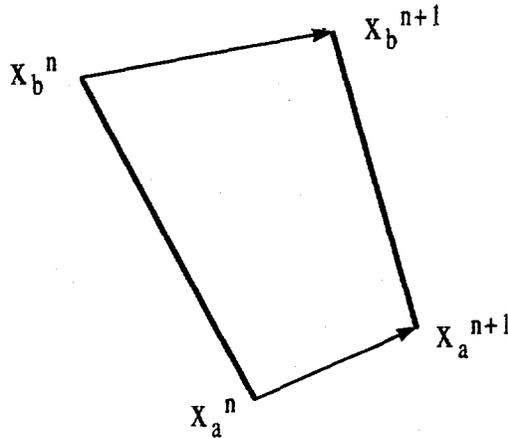


Figure 3.7: Volume flux construction from moving edges or faces.

It is then seen that (3.1) is a conservation equation for V^n .

Like the static case, volumes of cells underlying refinements must be the same as the sum of the volumes of the refined cells. Likewise cells adjacent to refined cells must share the same volume fluxes in order to keep volumes conserved just as the field variables preserve conserved quantities. The conservation of volumes is also a sufficient condition for free stream preservation. An integration scheme that is free stream preserving can be found in the same reference [4].

3.2.4 Overset Meshes

Adaptive meshes on overset grids are implemented in nearly the same way as single curvilinear mesh AMR algorithms. For each component of an overset mesh collection a regular AMR mesh hierarchy can be established in a manner similar to single curvilinear meshes. Where the differences are found is in the error estimation. Grids on like levels of different component AMR hierarchies must communicate in order to propagate buffer cells. This is most easily seen in one dimension in figure 3.8. On grid (a) cells with x's are flagged for refinement.

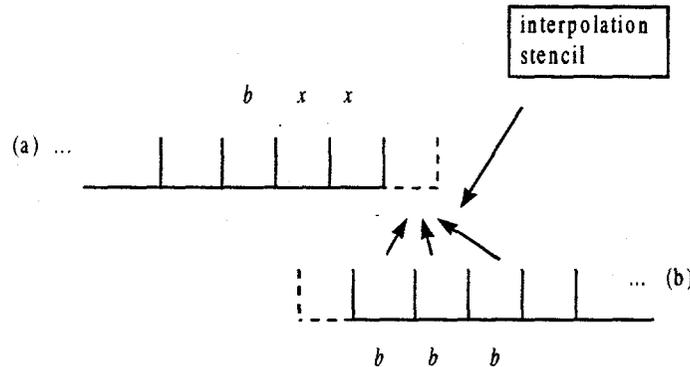


Figure 3.8: Flagged cells add buffer cells across components of an overset mesh through the interpolation stencil.

If a buffer is added next to the flagged points, represented by b's, then it is seen that ghost cells pass through buffer points to adjacent (same refinement level) grids on other components through the interpolation stencil. Refinements can then stay ahead of CFL limited signals across components of overset grids. Details of an overset adaptive algorithm can be found in [12].

3.3 Parallel Strategies

Parallel processors are becoming more available to the scientific community. It is quite common for scientific computing practitioners to have access to multiple CPU workstations, workstation clusters or even more expensive multiple CPU midrange supercomputers. These processors tend to have shared memory between CPUs. Computers with thousands of processors, called massively parallel processors (MPP), are becoming available to a larger community of users. MPPs usually have a distributed memory model. A node within an MPP is a collection of CPUs with shared memory. MPPs still have large numbers of nodes. Network access and resource sharing will significantly enlarge the number of people using MPPs over the remainder of the millennium.

The key to effectively using parallel processors is good load balancing and latency hiding. Load balancing is the process of insuring that every CPU has about the same amount of work to do as every other CPU. If load balancing is not achieved some processors will remain idle waiting for tasks on other processors to complete. Latency hiding is the process of minimizing the effect of relatively slow transit times of data between two or more processors. Latency is the time needed to initiate movement and then move data between one or more processors. Latency does not seem to be much of a problem with shared memory machines but can be a big issue on MPPs. Typical memory access times between nodes of an MPP are 100 - 1000 times slower than memory accesses on a node.

We will describe three parallel strategies that address the issues of load balancing and latency hiding. The first method is called the knapsack method of parallel load balancing by Crutchfield [20]. Here latency is hidden by overlapping computation with communication. Load balancing is achieved, automatically, by finding a uniform distribution of work on all processors. We next describe a data parallel approach [9]. This method is tailored for data parallel machines. These computers execute the same instruction on all CPUs at the same time. They are also called Single Instruction Multiple Data (SIMD) machines or architectures. In this approach, the AMR algorithm is simplified by enforcing the constraint that all patches are the same size. By using a data layout scheme that maximizes data locality (insuring that data is as close to the node that uses it as possible) and has a single fixed communication schedule (enabling smaller latency), a very efficient AMR implementation is implemented. The last approach by Quinlan [37] develops a model of how much work is done as a function of the spatial location on an adaptive mesh hierarchy. The function enables the uniform partitioning of the computational work among the nodes of a parallel machine.

3.3.1 Knapsack Load Balancing

The knapsack load balancing algorithm treats every AMR patch at every level as a single entity. That is, no matter how small or how large the patch is, it will be

resident on a single processor for a single level integration step. Every processor then maintains a list of the patches it has and is responsible for scheduling interchanges of boundary data of its patches with patches on other processors. The question then arises how does one balance the load and hide the latency?

Latency hiding is achieved through overlapping computation with communication. Since each processor has multiple patches, requests for boundary data are sent for some patches while other patches are integrated. In the actual application each patch was characterized as "fat". That is each cell of the patch needed approximately 250 computations to advance one time step. This leaves plenty of time to move data between nodes.

Load balancing is achieved by a heuristic algorithm for solving an NP (non-polynomial time) complete problem. Given N balls of random weights and $M \leq N$ knapsacks, what is the optimal distribution of balls within the knapsacks such that each knapsack has nearly the same load? The obvious application of the knapsack problem is if the balls represent patches and the knapsacks represent processors. The larger the patch size the more work needs to be done. Work then corresponds to the weights of the balls in the problem. It has been proven that this problem is not solvable in polynomial time as the number of balls and knapsacks increase. However, a near-optimal solution to the problem is found through a heuristic approach. The heuristic algorithm is as follows:

- 1 - sort balls by size with the largest first
- 2 - for (each ball), starting with the heaviest,
 - assign the ball to the lightest knapsack
 - end for
- 3 - find the heaviest sack
- 4 - for each ball i in the heaviest sack
 - for each ball j not in the heaviest sack
 - if interchanging i and j improves the load balance
 - perform an interchange of i and j and go to 3
 - end if
 - end for j
 - end for i

Define efficiency of the load balancing as E_{LB} where

$$E_{LB} = \frac{\sum_{\alpha,i} W_i^\alpha}{M \max_{\alpha} (\sum_i W_i^\alpha)}$$

and where W_i^α is the weight of the i 'th ball in knapsack α . Inefficiency I_{LB} is defined as

$$I_{LB} = 1 - E_{LB}.$$

Studies were performed on computer generated distributions of weights. As N/M increased the inefficiency dropped logarithmically. When N/M exceeded 3 inefficiencies were near or below 2%. The load balancing approach was shown to be effective on the 32 processor BBN 2000 parallel computer. Inefficiency was measured at about 5% for N/M near 3.5. The difference between theory and practice is patch size is neither a perfect or all-determining factor of work. Scaling studies for larger numbers of processors are not available.

3.3.2 Data Parallel AMR

The algorithm we describe here is designed for SIMD machines. The first distinguishing characteristic of this algorithm is all patches have the same number of cells and logical dimensions. For example, a patch in 3-D may have 32^3 cells regardless of refinement level. Fixing the patch size has many advantages and some disadvantages. Advantages include easy memory management and grid generation. A key disadvantage is a less flexible grid generation algorithm (a new grid generation algorithm will be needed as the boxing algorithm described above is not applicable).

Grid generation is accomplished by tiling. As with the standard algorithm, mesh points with large errors are flagged and buffer cells are added adjacent to flagged points. For simplicity assume we are working with a 2-D region. The grid generation algorithm is as follows:

- 1) Create a regular rectangular matrix of patches that cover the flagged points.
- 2) If a patch is not covering any flagged points remove it.
- 3) For each column of the matrix shift contiguous parts covering flagged points for higher efficiency by:
 - a) using fewer patches to cover the given flagged points
 - b) centering remaining patches on the flagged points

The memory layout uses a periodic mapping to increase locality. Suppose the domain index coordinates of a patch on the finest level are

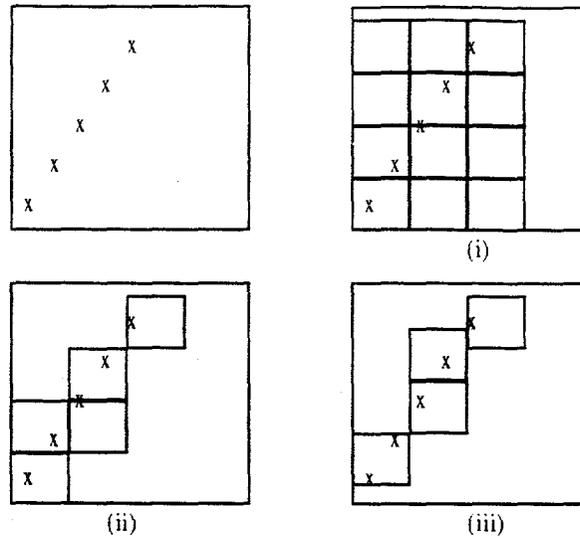


Figure 3.9: Tiling algorithm for fixed size patches.

$$(i, j): \quad i_{min} \leq i \leq i_{max}, \quad j_{min} \leq j \leq j_{max}$$

and let

$$n = i_{max} - i_{min} + 1 = j_{max} - j_{min} + 1.$$

Define a mapping to a computation patch with coordinates

$$(k, l): \quad 1 \leq k \leq n, \quad 1 \leq l \leq n$$

as

$$k = (i - 1) \bmod n + 1, \quad l = (j - 1) \bmod n + 1. \quad (3.2)$$

The computational patch directly relates how one patch is laid out in memory with respect to others. The mappings are many to one but two points on two different patches with the same domain index will have the same computational patch index. Where this is useful is in filling ghost cells of one patch from another. Since a patch ghost cell and patch *donor* cell (the cell that fills in the ghost cell at the same level) have the same domain index then they are in the same node of a parallel processor. Therefore, filling ghost cells of the patches at the same level requires no internode communication.

Coarse grids have a more complicated mapping into the computational patch indices

$$k' = (r * k) \bmod n + (r * k) / n + 1$$

$$l' = (r * l) \bmod n + (r * l) / n + 1$$

where r is the refinement and k and l are mappings described in (3.2). This mapping preserves locality of the coarse grid with respect to the fine grid. A fine grid interpolating data from a coarse grid will only need to gather data from coarse grid points on the same node or from nearby nodes. Nearby means in the neighborhood of a point on the computational patch. An illustrative example is in 1-D where $n = 16$ and r is 4. Let the coarse grid cover the entire domain and the fine grid be embedded in the coarse grid.

comput. patch	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
fine patch	17	18	19	20	5	6	7	8	9	10	11	12	13	14	15	16
coarse patch	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16
fine patch proj.	5	5	5	5	2	2	2	2	3	3	3	3	4	4	4	4

The above table shows how fine patch ghost cells with indices 5 and 6 need to interpolate from coarse cells 1, 2 and 3. Cells 5 and 6 are within Cr memory locations of coarse cells 1, 2 and 3 where $C < 2$. This would be true for larger n .

Lower level meshes need to be advanced. Advancement is achieved by changing the data layout of these lower levels to the simpler periodic mapping and then restoring the more complicated layout when the data is being used for interpolation. Moving data between these two layouts can be made fast since the communication schedule is only a function of the refinement factor r . On the CM-5 an optimum communications schedule was computed once at initialization and then used throughout the computation. This layout strategy is highly effective on SIMD machines that were tested. Typically 75% of all the time spent in test runs was in the integrator.

3.3.3 Multilevel Load Balancing

The last approach to load balancing partitions memory through construction of an integrated work function. Suppose we have a 2-D base grid with m cells in the abscissa (x) direction and n cells in the ordinate (y) direction. The integrated work function is constructed as follows:

- i) Let $IW(m, n + 1)$ be an integer array and initialize the first n rows to one and the $n + 1$ row to zero (The first dimension of IW is the column index and second dimension is the row index).

ii) For every cell of every patch above the base grid determine which cell it covers on the base grid and increment the integer array value corresponding to that cell by r^l . l is the level number with $l = 0$ for the base grid.

iii) Column sum the integer array using the following loop:

```

for i = 1 to m
  for j = 1 to n
    IW(i,n+1) := IW(i,n+1) + IW(i,j)
  end j
end i

```

iv) Create a running sum in the last row of the array:

```

for j = 2 to m
  IW(j,n+1) := IW(j,n+1) + IW(j-1,n+1)
end j

```

If there are p processors then let

$$w = IW(m, n + 1)/p.$$

The partitions of the grids are indices i_p where

$$i_p(j) = \min_k(IW(k, n + 1) \geq jw), \quad j = 1, \dots, p$$

The integrated work function $IW(i, n + 1)$ then yields a partition of the base grid and associated finer grid projections (and thus the fine grids themselves) that have roughly the same amount of work to perform. The higher the level of a cell, the more advances it will go through. This is why the work function is incremented by r^l for cells at level l . Often the work function may need to be constructed at levels higher than the base grid because the base grid is too coarse.

If there are $p \times p = p^2$ processors then for each stripe, an integrated work function can be constructed in the ordinate direction. Notice the similarity of this construction with the tiling algorithm for the SIMD method. It is obvious how a 3-D partitioning would be done with $p \times p \times p = p^3$ processors. With either a 1-, 2- or 3-D integrated work function data communication for interpolation and refluxing is nearly always between adjacent processors.

Chapter 4

Adaptive Elliptic Methods

The numerical solution of elliptic equations will be briefly covered in this chapter. In particular, we concentrate on solving

$$-\Delta u = f \quad (4.1)$$

on a region Ω with Neumann boundary condition

$$\frac{\partial u}{\partial n} = 0, \quad \text{on } \partial\Omega.$$

We describe a basic multigrid algorithm on structured adaptive meshes and the variants discussed in previous chapters. No effort will be expended in discussing the general elliptic problem as this is the focus of another course in this summer school. Our primary objective is to have an understanding of the numerical solution of the Laplace equation in order to apply it to constrained elliptic-hyperbolic computations in the next chapter. For the remainder of the chapter we will cover 1) the numerical solution of the Laplace equation on a simple rectangular grid hierarchy, 2) solving the same problem on a Cartesian grid with embedded boundary conditions and 3) solving the Laplace equation on a hybrid Cartesian overset grid. We will include a short discussion of mesh generation for 3). Our focus is on 2-D and 1-D problems but there is no technical barriers preventing application of the same concepts in 3-D.

4.1 Rectangular AMR Grids

The multigrid v-cycle for a rectangular adaptive mesh is broken into two parts. The first part solves a multigrid problem on the base grid. The other part is the algorithm for the two legs of a v-cycle on the adaptive mesh hierarchy. One leg starts on the finest grid and extends down to the base grid. The other leg starts with the base grid and traverses the grid hierarchy up to the finest level.

The multigrid method on the base grid is an algorithm that is well known. The right hand side of (4.1) for the base grid multigrid solver is constructed from f on the base grid cells not covered by finer meshes. Cells covered by finer meshes will have residuals from finer grids as the right hand side of the Laplacian. A brief summary of the multigrid algorithm follows. A solution guess is given on the base grid and a set of corrections to the solution are computed on coarser grids using residuals from finer grids. The coarse grid corrections are interpolated to the finer grid while the grid residuals are averaged to coarser grids. The multigrid hierarchy is used to simplify computation of corrections by separating Fourier components of the solution onto various parts of the grid hierarchy. Therefore each error correction method, called a smoother, need only solve the highest frequency component on a given mesh. Smoothers are inexpensive and easy to implement. Most smoothers are relaxation methods such as Jacobi, Gauss-Seidel or Gauss-Seidel with red-black ordering. The general multigrid v-cycle can be written recursively.

```

- MG_Recurse(Solution Array, RHS Array)

  - Smooth Solution Array

  - If a coarser level is available

    - Compute a Residual Array  $R = f - L u$ 

    - Coarsen the Residual Array to  $RC = Av(R)$ 

    - Set the Error Correction Array  $EC = 0$ 

    - Call MG_Recurse(EC Array, RC Array)

    - Interpolate the EC Array to the Fine Mesh Array  $E = I(EC)$ 

    - Solution Array := Solution Array + E

  - End If

  - Smooth Solution Array

-End MG_Recurse

```

For the Laplace equation (4.1), Gauss-Seidel red-black iteration is used as a smoother because of its ability to completely eliminate the highest frequency on a grid. On a square $m \times m$ mesh with spacing h and right hand side f , the solution u is first updated on red cells (cells of even parity) as

$$u := \frac{1}{4}(u_{i+1} + u_{i-1} + u_{j+1} + u_{j-1}) - h^2 f$$

$$(i+j) \bmod 2 = 0, \quad 1 \leq i, j \leq m.$$

The same relaxation procedure is applied to update the black (odd parity) for $(i+j) \bmod 2 = 1$. The residual is computed as

$$r = f + \Delta_h u = f - \left(\frac{-u_{i+1} - u_{i-1} + 4u - u_{j+1} - u_{j-1}}{h^2} \right)$$

$$1 \leq i, j \leq m.$$

Ghost cell boundary conditions used are

$$u_{m+1,j} = u_{m,j}, \quad u_{i,m+1} = u_{i,m}, \quad u_{0,j} = u_{1,j}, \quad \text{and} \quad u_{i,0} = u_{i,1}.$$

Given a coarser grid of size $m/2 \times m/2$, coarse grid residuals are computed from fine residuals

$$r_{i,j}^c = Av(r) = \frac{1}{4}(r_{2i,2j} + r_{2i,2j-1} + r_{2i-1,2j} + r_{2i-1,2j-1})$$

$$1 \leq i, j \leq m/2.$$

Interpolation from the coarse grid uses piecewise constant profiles

$$e_{i,j} = \text{Int}(e^c) = e_{(i-1)/2+1, (j-1)/2+1}^c$$

$$1 \leq i, j \leq m.$$

For a more complete treatment of the multigrid algorithm consider the references by Briggs [11] and Wesseling [44].

For a properly nested adaptive grid hierarchy we follow Cartwright and Martin [33]. Operators must be defined at coarse/fine interfaces. Away from boundaries the standard five point operator is used and physical boundary conditions remain unchanged. Referring to figure 4.1 we need to determine an operator for coarse cells (i, j) and fine cells (i', j') . For fine cells we fill in ghost cells using data from a quadratic interpolant through points $(i, j-1)$, (i, j) and $(i, j+1)$ and from a quadratic interpolant using points (i', j') , $(i'+1, j')$ and a point p and its value from the first interpolant.

The coarse cell (i, j) operator is defined as

$$\Delta_h u_{i,j} = \frac{f_{i+\frac{1}{2}} - f_{i-\frac{1}{2}}}{h} + \frac{f_{j+\frac{1}{2}} - f_{j-\frac{1}{2}}}{h}$$

where

$$f_{i-\frac{1}{2}} = \frac{u - u_{i-1}}{h}, \quad f_{j+\frac{1}{2}} = \frac{u_{j+1} - u}{h} \quad \text{and} \quad f_{j-\frac{1}{2}} = \frac{u - u_{j-1}}{h}$$

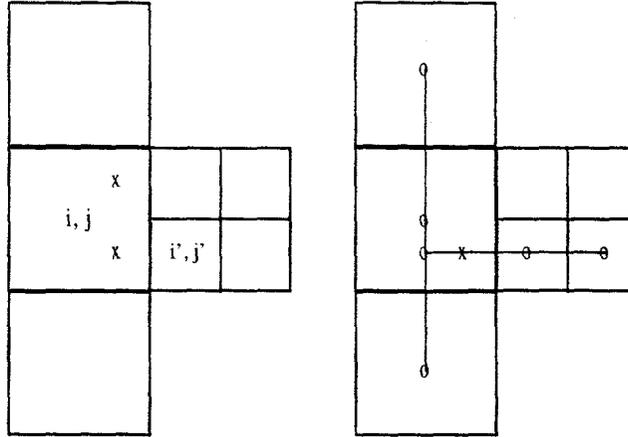


Figure 4.1: Interpolants between coarse and fine grids are created to compute an approximate normal gradient.

and

$$f_{i+\frac{1}{2}} = \frac{1}{2} \left(\frac{u_{i',j'+1} - u_{i'-1,j'+1}}{h/2} + \frac{u_{i',j'} - u_{i'-1,j'}}{h/2} \right).$$

The ghost values $u_{i'-1,j'}$ and $u_{i'-1,j'+1}$ are determined, as above, from the interpolants. Interpolants may be shifted to accommodate extruding corners (270°) of refined grids.

We now describe three new operators that will be used to compute residuals and smooth errors. The first operator is a single smoother which we define as

$$S_{GS}^l = S_{GS}^l(u^l, f^l, h^l)$$

where Gauss-Seidel red-black smoothing is applied to all points u^l of level l using a right hand side f^l and mesh spacing h^l . The smoothing operator S_{GS}^l always uses zero for ghost cell values at boundaries adjacent to coarse cells.

The operator

$$\Delta_{nf}^l = \Delta_{nf}^l(u^l, u^{l-1}, h^l)$$

is a discrete Laplace operator defined on all level l patches. It uses the solution u^{l-1} for boundary conditions discussed above. This operator ignores finer grids. The last operator

$$\Delta_h^l = \Delta_h^l(u^{l+1}, u^l, u^{l-1}, h^l)$$

is a Laplace operator defined only on those points u^l not covered by higher level meshes. It interpolates ghost cells using values u^{l-1} from the coarser level $l-1$ and uses the modified operator at the coarse/fine level $l, l+1$ interfaces with values u^l, u^{l+1} .

The multigrid algorithm has the form:

```

AMG_Recurse(l)
  if(l = lmax)
    elmax = 0
    Rlmax = flmax -  $\Delta_h^{l_{max}}$ 
  endif

  if(l > 0)
    usavel = ul
    el-1 = 0
    el = Sl(el, Rl, hl)
    ul = ul + el
    Rl-1 = Av(Rl -  $\Delta_{nf}^l(e^l, e^{l-1}, h^l)$ )
    Rl-1 = fl-1 -  $\Delta_h^l(u^l, u^{l-1}, u^{l-2})$  on uncovered part
    AMG_Recurse(l - 1)
    el = el + Int(el-1)
    Rl = Rl -  $\Delta_{nf}^l(e^l, e^{l-1}, h^l)$ 
     $\delta e^l = 0$ 
     $\delta e^l = S^l(\delta e^l, R^l, h^l)$ 
    el = el +  $\delta e^l$ 
    ul = usavel + el
  else
    solve/relax (e0, R0) on base grid
  End AMG_Recurse

```

The real complexity in this algorithm resided in keeping clear what are error corrections and what are solution corrections. This differentiation must be made in order to take into account patches that are partially covered by finer level patches.

4.2 Embedded Boundaries

Johansen and Colella in [28] describe an extension of the adaptive method for Cartesian grids with embedded boundaries. The key ingredient in this algorithm is finding an operator for the irregular cells caused by the cutouts. We assume that a single line segment is used to cut each cell. The operator for these

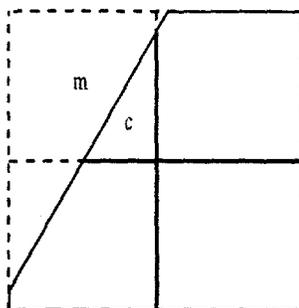


Figure 4.2: Cartesian mesh cut by an embedded boundary creating irregular cells.

irregular cells need only be first order accurate as the global operator will smooth errors to second order. There is also a question of centering of variables. For both uncut and cut cells, u (the solution) is centered at the midpoint (m in figure 4.2) of the entire cell — whether the cell is cut or not. f (the right hand side) is centered at the centroid (c in figure 4.2) of a cell. For an uncut cell the centroid is the midpoint. For cut cells the midpoint of the same uncut cell differs from the centroid. The Cartesian geometry away from cut cells generates a simple five point stencil in 2-D.

The discretization of the operator at cut cells is performed using a finite volume approach. First observe

$$\Delta u = \nabla \cdot \vec{\phi} = f \quad \text{and} \quad \vec{\phi} = \nabla u.$$

The divergence theorem provides a simple way of discretizing the divergence. For smooth vectors $\vec{\phi}$

$$\nabla \cdot \vec{\phi} = \lim_{V(\Omega) \rightarrow 0} \frac{1}{V} \int_{\Omega} \nabla \cdot \vec{\phi} dV = \lim_{V(\Omega) \rightarrow 0} \frac{1}{V} \oint \vec{\phi} \cdot d\vec{S}.$$

For discrete $\vec{\phi}_e$ centered on an edge of a cutaway cell

$$\nabla \cdot \vec{\phi} = \frac{1}{V_c} \sum_e L_e \vec{\phi}_e \cdot \hat{n}_e + O(h^p)$$

where V_c is the area of the cut cell, L_e is a length of an edge with index e and \hat{n}_e is the edge outward pointing normal. If $\vec{\phi}_e \cdot \hat{n}_e$ is known to second order then p is one or the discrete divergence is first order at the centroid of the cell.

Estimating the normal gradients to second order is done in two steps. The first step is discretizing the boundary conditions and the second step is discretizing gradients at cut cell edges. To estimate the normal gradient at a boundary cut cell, a line l_b from the center of the edge and normal to this edge is extended into the computation region (see figure 4.3). Let S be the set of all straight lines connecting 3 adjacent midpoints of cells in the computation region. The midpoint of the cell where the divergence is being computed is excluded from membership in S . Let l_1 and l_2 be the two *closest* lines in the sense that they intersect l_b and the distance along l_b from the edge center to the intersection is the measure of distance. Quadratic interpolants may be constructed with the points in l_1 and l_2 to give third order estimates of the solution at the points of intersection with l_b . A third quadratic interpolant can then be constructed along l_b using the value of the derivative of the solution or the solution value itself at the edge and the two values constructed from the interpolants on l_1 and l_2 . The normal gradient can be computed to second order using the quadratic interpolant.

The second step in computing gradients is on nonboundary edges of cut cells. Here we have two cases as well. If an edge is not truncated then the normal gradient components to second order is one of the following centered differences

$$\phi_{x_{i+\frac{1}{2}}} = \frac{u_{i+1} - u}{h} \quad \text{or} \quad \phi_{y_{j+\frac{1}{2}}} = \frac{u_{j+1} - u}{h}.$$

For nonboundary cut edges, the centers of these cut edges no longer coincide with the centers of the edges if they were not cut. Instead gradients must be interpolated in order to preserve second order accuracy. Suppose a cell with indices (i, j) is cut as illustrated in figure 4.4. then

$$\phi_{y_{j+\frac{1}{2}}} = \frac{1-\alpha}{2} \left(\frac{u_{i+1,j+1} - u_{i+1}}{h} \right) + \frac{1+\alpha}{2} \left(\frac{u_{j+1} - u}{h} \right)$$

where

$$\alpha = l/h$$

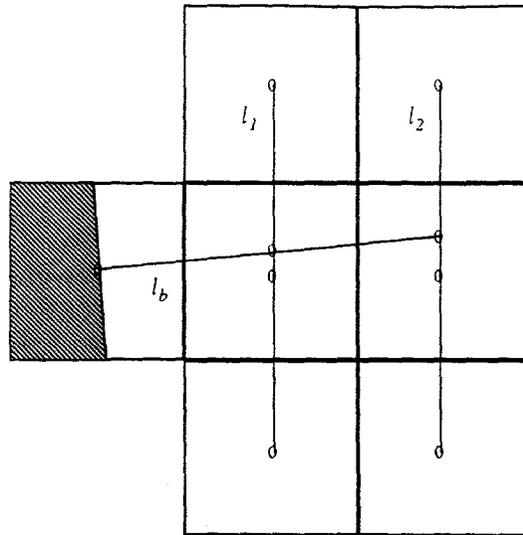


Figure 4.3: Interpolants used in construction of a gradient centered at an irregular edge of a cut cell.

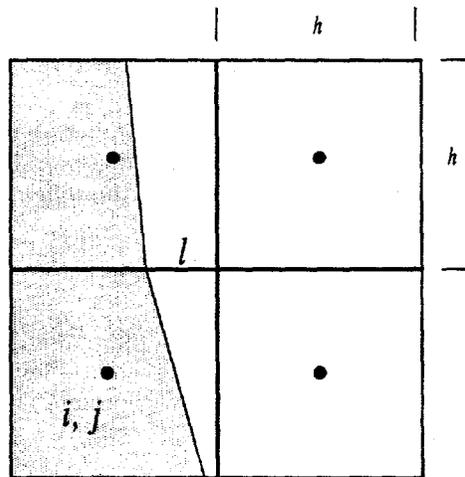


Figure 4.4: Nonboundary edges of cut cells need to interpolate gradients to compensate for loss of centering in order to preserve second order accuracy.

and l is the exposed length of edge $(i, j + \frac{1}{2})$.

With the gradient and divergence operators defined we can now discuss how multigrid and adaptive multigrid extensions can be developed. For multigrid, a coarsened grid hierarchy can be created in the same way as a grid without embedded boundaries. However, partial cells on the coarse grids may no longer have single line segments cutting them. This leads to two rules:

- i) Line segments on the coarse grids are created by taking the two endpoints of the two or more segments on the finer grid.
- ii) The area of the coarse cut cell is set to the sum of the areas of the underlying fine cut and uncut cells.

The second rule changes the averaging operator from fine cell data to coarse. Near boundaries area weights are used

$$r_{i,j}^c = \frac{r_{2i,2j}A_{2i,2j} + r_{2i,2j-1}A_{2i,2j-1} + r_{2i-1,2j}A_{2i-1,2j} + r_{2i-1,2j-1}A_{2i-1,2j-1}}{A_{i,j}^c}$$

where

$$A_{i,j}^c = A_{2i,2j} + A_{2i-1,2j} + A_{2i,2j-1} + A_{2i-1,2j-1}.$$

$A_{i,j}$ is the (i, j) cell area on a fine grid.

With the coarsening operations and the new operator defined, the multigrid algorithm for the uncut grid can be directly applied. Likewise the AMR multigrid algorithm can also be applied directly provided the finest grid in a hierarchy determines the underlying areas of coarser cells covered by the finest grid. We end this section by describing some theoretical and practical issues with the embedded boundary method. These issues involve symmetry, conditioning and some empirical observations.

The interpolation used at coarse/fine interfaces and at embedded boundaries destroy the symmetry of the underlying matrix. This is observed to have little or no impact on the performance of the multigrid algorithm for the AMR hierarchy. In the next section on overset grids we will encounter instances where a conjugate gradient solver will be needed to solve a linear system at the bottom of the V-cycle. Solvers that tolerate asymmetries in the matrix like BiCG-Stab and GMRES are needed. These algorithms take twice or more the memory and time to execute in comparison with conjugate gradient methods for symmetric positive definite matrices. But memory and time are not issues when solving a system on the coarsest available grid of a multigrid hierarchy.

It is apparent that some cut cells in 2-D grids can be very small. The question can then be asked whether the condition of the system blows up? In 1-D, it can be analytically shown that the system remains well conditioned as the boundary cell size goes to zero. The order of convergence can also be shown to be second

order even with a first order discretization of the cut cells. 2-D numerical studies for both adaptive and nonadaptive embedded boundary problems have shown uniform second order convergence and insensitivity to small cell sizes.

One empirical observation that improves convergence is the use of underrelaxation of the smoother near patch boundaries. Underrelaxation with a factor $\lambda = 3/4$

$$e^l = (1 - \lambda)e^l + \lambda S(e^l, R^l, h^l)$$

improves convergence. To get a factor of ten or more reduction in residual per v-cycle, four smoothing iterations per level seems optimal. At cut cells a Jacobi iteration is substituted for the Gauss-Seidel red-black algorithm as a smoother. First the Jacobi iteration is applied on cut cells while holding values of uncut cells fixed. Next the Gauss-Seidel red-black algorithm is applied holding cut cell values fixed.

4.3 Overset Grids

Overset grids use less complex boundary conditions compared with those found in embedded boundary methods. However overset grids have interior coupling conditions (additional interpolation conditions) not found in embedded boundary methods. The relationship can be made more symmetric by using a finite volume approach to connect components of an overset grid. What is meant is instead of two or more grids overlapping, a ranking is established among grids such that each grid with a higher ranking cuts away a grid with lower ranking. Therefore higher ranking grids act as embedded boundaries for lower priority grids as illustrated in figure 4.5. The diamond shaped region has a higher ranking than the uniform background mesh. It then cuts away the background mesh to get a single valued composite mesh. Irregular cells appear near the edges of the diamond shaped grid.

Forming operators for overset grids requires more effort than on uniform Cartesian grids because geometric information must be incorporated. The gradient and divergence operators are formed separately and composed. Function values are cell centered, gradients are edge centered and geometric $((x, y)$ coordinates) are node centered. The 2-D gradient operator is a discretization of the continuum case

$$\nabla u = (u_x, u_y) = \frac{1}{J} (u_\xi y_\eta - u_\eta y_\xi, -u_\xi x_\eta + u_\eta x_\xi)$$

where

$$J = \frac{\partial(x, y)}{\partial(\xi, \eta)} = \begin{vmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{vmatrix}$$

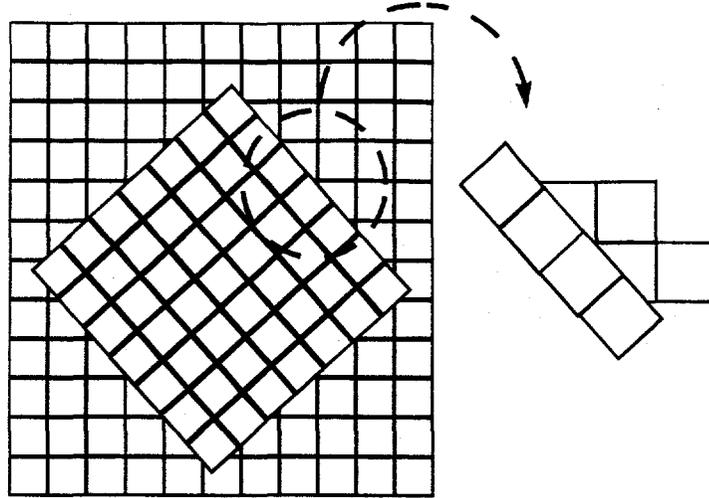


Figure 4.5: The higher ranking diamond grid cuts away at the lower ranking Cartesian grid. The detail shows how the diamond grid acts like an embedded boundary in cutting away the Cartesian grid.

The discretization of the gradient operator is placed at cell edges

$$\begin{aligned}
 (\nabla\phi)_{i+\frac{1}{2},j} &= \frac{1}{J_{i+\frac{1}{2}}} \left((\phi_{i+1} - \phi)(y_{i+\frac{1}{2},j+\frac{1}{2}} - y_{i+\frac{1}{2},j-\frac{1}{2}}) - \right. \\
 &\quad \left. \frac{1}{4}(\phi_{i+1,j+1} - \phi_{i+1,j-1} + \phi_{j+1} - \phi_{j-1})(\bar{y}_{i+1} - \bar{y}), \right. \\
 &\quad \left. -(\phi_{i+1} - \phi)(x_{i+\frac{1}{2},j+\frac{1}{2}} - x_{i+\frac{1}{2},j-\frac{1}{2}}) - \right. \\
 &\quad \left. \frac{1}{4}(\phi_{i+1,j+1} - \phi_{i+1,j-1} + \phi_{j+1} - \phi_{j-1})(\bar{x}_{i+1} - \bar{x}) \right)
 \end{aligned}$$

where

$$J_{i+\frac{1}{2}} = ((\bar{x}_{i+1} - \bar{x})(y_{i+\frac{1}{2},j+\frac{1}{2}} - y_{i+\frac{1}{2},j-\frac{1}{2}}) - ((\bar{y}_{i+1} - \bar{y})(x_{i+\frac{1}{2},j+\frac{1}{2}} - x_{i+\frac{1}{2},j-\frac{1}{2}}))$$

and

$$\begin{aligned}
 \bar{x} &= \frac{1}{4}(x_{i+\frac{1}{2},j+\frac{1}{2}} + x_{i-\frac{1}{2},j+\frac{1}{2}} + x_{i-\frac{1}{2},j-\frac{1}{2}} + x_{i+\frac{1}{2},j-\frac{1}{2}}) \\
 \bar{y} &= \frac{1}{4}(y_{i+\frac{1}{2},j+\frac{1}{2}} + y_{i-\frac{1}{2},j+\frac{1}{2}} + y_{i-\frac{1}{2},j-\frac{1}{2}} + y_{i+\frac{1}{2},j-\frac{1}{2}}).
 \end{aligned}$$

The expression for $(\nabla\phi)_{i,j+\frac{1}{2}}$ is similar. The divergence operator uses the same finite volume formulation as above

$$\nabla \cdot \vec{u} = \lim_{V_c \rightarrow 0} \frac{1}{V_c} \oint_{\partial V_c} \vec{u} \cdot d\vec{S} \simeq \frac{1}{V_c} \sum_s \vec{u}_s \cdot \Delta\vec{S}$$

where the index s denotes a side and $\Delta\vec{S}$ is the length of the side times the normal to the side.

Given an edge centered piecewise constant quantity $(\phi_x, \phi_y)_{i+\frac{1}{2},j}$ direct integration on an edge leads to the following relationship

$$(\vec{u}_s \cdot \Delta\vec{S})_{i+\frac{1}{2}} = \phi_{x_{i+\frac{1}{2}}}(y_{i+\frac{1}{2},j+\frac{1}{2}} - y_{i+\frac{1}{2},j-\frac{1}{2}}) - \phi_{y_{i+\frac{1}{2}}}(x_{i+\frac{1}{2},j+\frac{1}{2}} - x_{i+\frac{1}{2},j-\frac{1}{2}}).$$

The volume of a cell V_c is computed by observing

$$V_c = \int_{V_c} dV = \int_{V_c} \nabla \cdot (x, 0) dV = \oint_{\partial V_c} (x, 0) \cdot d\vec{S}.$$

The volume is computed by summing up line integral contributions on each side.

On irregular cells the divergence operator is still relevant. It should be noticed that irregular cells may have more than five sides. Gradients are problematical. The same kind of trick with polynomial interpolation is used. In figure 4.6 a gradient is found normal to the cutting edge of a cell using four quadratic polynomial interpolants. For conservation purposes, both the uncut and cut cells share the gradients centered at edges.

With operators defined, multigrid hierarchies can be established on each component grid. Operators can be established at each level using the same rules of area conservation as discussed for the embedded boundary case. One problem that is encountered in this case is, in general, a multigrid hierarchy cannot be fully established. Below some coarsening interpolants and operators cannot be found. Even more fundamental is the possibility that different components have varied linear dimension leading to different numbers of cells to coarsen. This implies a linear solver needs to be used to solve for the error as function of the residuals on the coarsest level.

Finally, like the embedded boundary case, an AMR multigrid scheme uses exactly the same algorithm for each component of an overset grid. Residuals are propagated to the base grid where the multigrid solver is called.

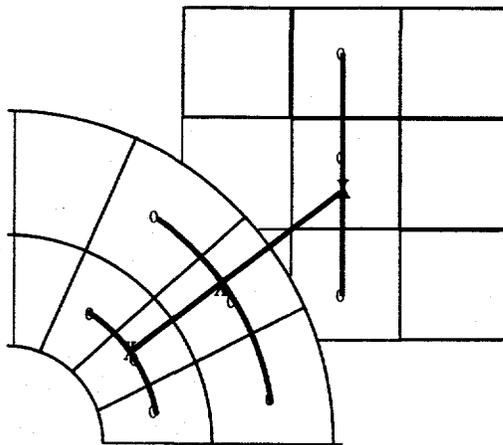


Figure 4.6: Two interpolants on the cutting mesh with one on the cut mesh are used to supply data to a fourth interpolant in order to compute a gradient normal to a cutting edge.

Chapter 5

Adaptive Hyperbolic-Elliptic Solvers

In this chapter we examine adaptive methods for hyperbolic systems that are constrained with an elliptic equation. The model system we use is the incompressible, inviscid Euler equations. We start with a description of a nonadaptive algorithm and then develop two adaptive methods. The first method is a straightforward application of the adaptive multigrid methods described in the previous chapter and the advection methods described in chapter 2. There is no recursive time stepping procedure used. The second method incorporates a recursive time step algorithm with a consequent increase in algorithm complexity.

5.1 Incompressible, Inviscid Flow Solution

For convenience we repeat the equations for incompressible flow

$$U_t + U \cdot \nabla U + \nabla p = 0 \quad (5.1)$$

$$\nabla \cdot U = 0 \text{ in a region } \Omega$$

$$U \cdot \hat{n} = 0 \text{ on } \partial\Omega.$$

where

$$U(x, y, t) = (u(x, y, t), v(x, y, t))$$

and \hat{n} is the normal to the boundary. The numerical method we choose to discuss to solve (5.1) and its constraints is called a projection method. The projection algorithm was first studied by Chorin [15] and the second order extension was developed by Bell, Colella and Glaz [3]. The projection method takes advantage

of the Hodge theory to eliminate the pressure in the time evolution of U . The formal structure of the algorithm is, once again, predictor-corrector.

The variables U and p are cell centered but boundary conditions and intermediate quantities may be centered at edges. The three major steps in the algorithm is: 1) To estimate the velocities at cell edges at half time levels much like the predictor for explicit schemes described in the second chapter. This will be called the advection step. 2) Make the edge centered velocities divergence free. This step uses what is called the MAC projection (described below). 3) Using the edge centered velocities, advance the cell centered quantities to the full time level. In the last step a second or full projection is applied to recover the pressure and insure the velocities are divergence free.

5.1.1 Advection

The $U \cdot \nabla U$ term in (5.1) is approximated using an explicit approximation. Since ∇p is orthogonal to divergence free fields we can ignore it in the predictor because its effect will be felt when the MAC projection is applied. Define

$$\begin{aligned}\hat{U}_{i+\frac{1}{2},L}^{n+\frac{1}{2}} &= U + \frac{1}{2} \left(1 - \frac{\Delta t u}{h}\right) \Delta_i U \\ \hat{U}_{i+\frac{1}{2},R}^{n+\frac{1}{2}} &= U_{i+1} - \frac{1}{2} \left(1 + \frac{\Delta t u_{i+1}}{h}\right) \Delta_i U_{i+1} \\ \hat{U}_{j+\frac{1}{2},B}^{n+\frac{1}{2}} &= U + \frac{1}{2} \left(1 - \frac{\Delta t v}{h}\right) \Delta_j U \\ \hat{U}_{j+\frac{1}{2},T}^{n+\frac{1}{2}} &= U_{j+1} - \frac{1}{2} \left(1 + \frac{\Delta t v_{j+1}}{h}\right) \Delta_j U_{j+1}\end{aligned}$$

where h is the mesh spacing and Δt the time step. Next define Riemann solutions of the form

$$\begin{aligned}\hat{U}_{i+\frac{1}{2}}^{n+\frac{1}{2}} &= R^{(x)}(\hat{U}_{i+\frac{1}{2},L}^{n+\frac{1}{2}}, \hat{U}_{i+\frac{1}{2},R}^{n+\frac{1}{2}}) \\ \hat{U}_{j+\frac{1}{2}}^{n+\frac{1}{2}} &= R^{(y)}(\hat{U}_{j+\frac{1}{2},B}^{n+\frac{1}{2}}, \hat{U}_{j+\frac{1}{2},T}^{n+\frac{1}{2}})\end{aligned}$$

where

$$R^{(x)}(U_L, U_R) = \begin{cases} U_L & \text{if } u_L, u_R > 0 \\ U_R & \text{if } u_L, u_R < 0 \\ \frac{1}{2}(U_L + U_R) & \text{otherwise} \end{cases}$$

and the analogous relation in the y direction is

$$R^{(y)}(U_B, U_T) = \begin{cases} U_B & \text{if } v_B, v_T > 0 \\ U_T & \text{if } v_B, v_T < 0 \\ \frac{1}{2}(U_B + U_T) & \text{otherwise.} \end{cases}$$

Next define

$$\begin{aligned}
\tilde{U}_{i+\frac{1}{2},L} &= \hat{U}_{i+\frac{1}{2},L}^{n+\frac{1}{2}} - \frac{\Delta t}{4}(\hat{v}_{j+\frac{1}{2}}^{n+\frac{1}{2}} + \hat{v}_{j-\frac{1}{2}}^{n+\frac{1}{2}})(\hat{U}_{j+\frac{1}{2}}^{n+\frac{1}{2}} - \hat{U}_{j-\frac{1}{2}}^{n+\frac{1}{2}}) \\
\tilde{U}_{i+\frac{1}{2},R} &= \hat{U}_{i+\frac{1}{2},R}^{n+\frac{1}{2}} - \frac{\Delta t}{4}(\hat{v}_{i+1,j+\frac{1}{2}}^{n+\frac{1}{2}} + \hat{v}_{i+1,j-\frac{1}{2}}^{n+\frac{1}{2}})(\hat{U}_{i+1,j+\frac{1}{2}}^{n+\frac{1}{2}} - \hat{U}_{i+1,j-\frac{1}{2}}^{n+\frac{1}{2}}) \\
\tilde{U}_{j+\frac{1}{2},B} &= \hat{U}_{j+\frac{1}{2},B}^{n+\frac{1}{2}} - \frac{\Delta t}{4}(\hat{u}_{i+\frac{1}{2}}^{n+\frac{1}{2}} + \hat{u}_{i-\frac{1}{2}}^{n+\frac{1}{2}})(\hat{U}_{i+\frac{1}{2}}^{n+\frac{1}{2}} - \hat{U}_{i-\frac{1}{2}}^{n+\frac{1}{2}}) \\
\tilde{U}_{j+\frac{1}{2},T} &= \hat{U}_{j+\frac{1}{2},T}^{n+\frac{1}{2}} - \frac{\Delta t}{4}(\hat{u}_{i+\frac{1}{2},j+1}^{n+\frac{1}{2}} + \hat{u}_{i-\frac{1}{2},j+1}^{n+\frac{1}{2}})(\hat{U}_{i+\frac{1}{2},j+1}^{n+\frac{1}{2}} - \hat{U}_{i-\frac{1}{2},j+1}^{n+\frac{1}{2}}).
\end{aligned}$$

Solve another Riemann problem to get a fully upwinded unsplit time and edge centered estimate of the velocity

$$\begin{aligned}
\tilde{U}_{i+\frac{1}{2}}^{n+\frac{1}{2}} &= R^{(x)}(\tilde{U}_{i+\frac{1}{2},L}^{n+\frac{1}{2}}, \tilde{U}_{i+\frac{1}{2},R}^{n+\frac{1}{2}}) \\
\tilde{U}_{j+\frac{1}{2}}^{n+\frac{1}{2}} &= R^{(y)}(\tilde{U}_{j+\frac{1}{2},B}^{n+\frac{1}{2}}, \tilde{U}_{j+\frac{1}{2},T}^{n+\frac{1}{2}}).
\end{aligned}$$

5.1.2 MAC Projection

The Hodge projection can be applied to $\tilde{U}_{i+\frac{1}{2}}^{n+\frac{1}{2}}$ and $\tilde{U}_{j+\frac{1}{2}}^{n+\frac{1}{2}}$ to get a divergence free field. This is called a MAC projection after the marker and cell method of Harlow and Welch [27]. Its effect is to incorporate the ∇p term into the predictor. The underlying Laplace equation is

$$\Delta_{i,j}\phi = \nabla_{i,j} \cdot \tilde{U} = \frac{1}{h} \left[(\hat{u}_{i+\frac{1}{2}}^{n+\frac{1}{2}} - \hat{u}_{i-\frac{1}{2}}^{n+\frac{1}{2}}) + (\hat{v}_{j+\frac{1}{2}}^{n+\frac{1}{2}} - \hat{v}_{j-\frac{1}{2}}^{n+\frac{1}{2}}) \right]$$

where

$$\Delta_{i,j}\phi = \frac{\phi_{i+1} + \phi_{i-1} - 4\phi + \phi_{j+1} + \phi_{j-1}}{h^2}.$$

The gradient operator at cell edges $(i + \frac{1}{2}, j)$ and $(i, j + \frac{1}{2})$ is

$$\nabla_{i+\frac{1}{2}}\phi = \left(\frac{\phi_{i+1} - \phi}{h}, \frac{\phi_{i+1,j+1} - \phi_{i+1,j-1} + \phi_{j+1} - \phi_{j-1}}{4h} \right)$$

$$\nabla_{j+\frac{1}{2}}\phi = \left(\frac{\phi_{i+1,j+1} - \phi_{i-1,j+1} + \phi_{i+1} - \phi_{i-1}}{4h}, \frac{\phi_{j+1} - \phi}{h} \right).$$

The divergence free part of the edge centered velocities can be computed as

$$\begin{aligned}
U_{i+\frac{1}{2}}^{n+\frac{1}{2}} &= \tilde{U}_{i+\frac{1}{2}}^{n+\frac{1}{2}} - \nabla_{i+\frac{1}{2}}(\phi) \\
U_{j+\frac{1}{2}}^{n+\frac{1}{2}} &= \tilde{U}_{j+\frac{1}{2}}^{n+\frac{1}{2}} - \nabla_{j+\frac{1}{2}}(\phi).
\end{aligned}$$

5.1.3 Corrector Step

To approximate the right hand side of the expression

$$U_t = -U \cdot \nabla U - \nabla p$$

at cell centers and centered in time we use the projected edge velocities in the following expression

$$\begin{aligned}
(U \cdot \nabla U)_{i,j}^{n+\frac{1}{2}} &= \frac{1}{2h} \left((u_{i+\frac{1}{2}}^{n+\frac{1}{2}} + u_{i-\frac{1}{2}}^{n+\frac{1}{2}})(U_{i+\frac{1}{2}}^{n+\frac{1}{2}} - U_{i-\frac{1}{2}}^{n+\frac{1}{2}}) + \right. \\
&\quad \left. (v_{j+\frac{1}{2}}^{n+\frac{1}{2}} + v_{j-\frac{1}{2}}^{n+\frac{1}{2}})(U_{j+\frac{1}{2}}^{n+\frac{1}{2}} - U_{j-\frac{1}{2}}^{n+\frac{1}{2}}) \right).
\end{aligned}$$

5.1.4 Full Projection

Although the edge centered velocities are divergence free it is not sufficient to update to the time level $n+1$. Instead another projection is applied. Let

$$U^* = U^n - \Delta t (U \cdot \nabla U)^{n+\frac{1}{2}}.$$

Following Minion [34] we project the cell centered U^* to get the advanced time level value U^{n+1} . To do this the MAC projection is reused. Cell centered values are averaged onto edges. To preserve accuracy, fourth order interpolants are applied

$$\begin{aligned}
\tilde{u}_{i+\frac{1}{2}} &= \frac{-u_{i-1}^* + 9(u_{i+\frac{1}{2}}^* + u_{i+1}^*) - u_{i+2}^*}{16} \\
\tilde{v}_{j+\frac{1}{2}} &= \frac{-v_{j-1}^* + 9(v_{j+\frac{1}{2}}^* + v_{j+1}^*) - v_{j+2}^*}{16}.
\end{aligned}$$

Define

$$\begin{aligned}
(\bar{\nabla} \phi)_x &= \frac{(\nabla \phi)_{x_{i+\frac{1}{2}}} + (\nabla \phi)_{x_{i-\frac{1}{2}}}}{2} \\
(\bar{\nabla} \phi)_y &= \frac{(\nabla \phi)_{y_{j+\frac{1}{2}}} + (\nabla \phi)_{y_{j-\frac{1}{2}}}}{2}
\end{aligned}$$

then

$$U^{n+1} = U^* - \bar{\nabla}\phi = U^* - ((\bar{\nabla}\phi)_x, (\bar{\nabla}\phi)_y). \quad (5.2)$$

The pressure, to within a constant, can be recovered as

$$p^{n+\frac{1}{2}} = \frac{\phi}{\Delta t} \quad (5.3)$$

since

$$\begin{aligned} \frac{\nabla\phi}{\Delta t} &= \frac{U^* - U^{n+1}}{\Delta t} = \frac{U^n - U^{n+1}}{\Delta t} - P(U \cdot \nabla U)^{n+\frac{1}{2}} \\ &= (I - P)(-U \cdot \nabla U)^{n+\frac{1}{2}}. \end{aligned}$$

In [34], with minor modifications to the advection algorithm described above, Minion implemented a projection method that uses a hierarchical set of adaptive grids. A single time step, controlled by the finest grid, is used to advance all cells on all meshes.

Because of the nonlinear advection terms, the projection method is difficult to analyze. In particular, stability of the evolution operator has not been analytically proven. The MAC projection is stable and idempotent in the sense that

$$\|P\| \leq 1 \quad \text{and} \quad P^2 = P$$

respectively. Unfortunately cell centered projections may lose these properties. These projections are called *approximate projections*. The stability (the projection operator being bounded by unity) of the projection is certainly a necessary but not sufficient condition for the overall method to be stable. Therefore some care must be exercised in formulating approximate projections.

5.2 An Adaptive Incompressible Method

We describe an algorithm by Almgren *et al.* [1] that was developed to adaptively solve variable density incompressible flow in 3-D. The referenced paper discusses a numerical solution for the equation set

$$\begin{aligned} U_t + (U \cdot \nabla)U &= \frac{1}{\rho}(-\nabla p + \mu\Delta U + H_U) \\ \rho_t + \nabla \cdot (\rho U) &= 0 \\ c_t + (U \cdot \nabla)c &= H_c + \kappa\Delta c \\ \nabla \cdot U &= 0 \end{aligned}$$

where additional variables not seen before include H_U an external force, H_c a source for an advected scalar c and κ is the diffusion coefficient for the advected concentration c . Because of time and space constraints, we simplify the above equations to constant density, inviscid incompressible flow (5.1). The equation for the evolution of concentration, c , is ignored.

We first examine how a single level mesh is advanced in time. This allows one to concentrate on the issue of centering and differencing of various quantities. The differencing of the full projection is different from the full projection above. In addition, we will see how flux registers are initialized and updated. The algorithm we describe uses a node based projection method. Therefore, pressure $p^{n+\frac{1}{2}}$ is centered at nodes while the velocity vector $U_{i,j}^n$ continues to be cell centered.

5.2.1 Single Level Algorithm

The first step in a single level, say level l , time advance is the estimate of velocities at cell edges at the half time level. This step is identical to the predictor step described in the first part of this chapter. The predicted velocities are written as $\tilde{U}_{i+\frac{1}{2}}^{n+\frac{1}{2}}$ and $\tilde{U}_{j+\frac{1}{2}}^{n+\frac{1}{2}}$. These velocities are projected using the MAC projection to get normal, divergence free velocities to the cell faces. The normal velocities are written as $U_{i+\frac{1}{2}}^{ADV}$ and $U_{j+\frac{1}{2}}^{ADV}$. With face centered velocity estimates and their projections computed, the corrector step, before a projection, can be written as

$$\frac{U^* - U}{\Delta t} = - \left[\nabla \cdot U^{ADV} \tilde{U} \right]^{n+\frac{1}{2}} - \nabla p^{n-\frac{1}{2}}.$$

The discretization of $\nabla \cdot U^{ADV} \tilde{U}$ requires only the normal velocity components at cell edges for U^{ADV} . Since there is a node centered pressure, a different projection is used for advancing the velocity to time level $n+1$

$$\frac{U^{n+\frac{1}{2}} - U}{\Delta t} = P \left(\frac{U^* - U}{\Delta t} \right)$$

$$\nabla p^{n+\frac{1}{2}} = \nabla p^{n-\frac{1}{2}} + (I - P) \left(\frac{U^* - U}{\Delta t} \right).$$

The nodal projection computes the divergence at nodes away from coarse/fine interfaces or physical boundaries using

$$\nabla_{i+\frac{1}{2}, j+\frac{1}{2}} \cdot U = \frac{u_{i+1} + u_{i+1, j+1} - u - u_{j+1}}{2h} + \frac{v_{i+1, j+1} + v_{j+1} - v_{i+1} - v}{2h}.$$

The adjoint operator, which is the gradient, has the same stencil offset from cell centers to cell nodes. Also notice that the gradient operator is never evaluated using data from more than one level.

$$\nabla_{i,j}\phi = \left(\frac{\phi_{i+\frac{1}{2},j+\frac{1}{2}} + \phi_{i+\frac{1}{2},j-\frac{1}{2}} - \phi_{i-\frac{1}{2},j+\frac{1}{2}} - \phi_{i-\frac{1}{2},j-\frac{1}{2}}}{2h}, \right. \\ \left. \frac{\phi_{i+\frac{1}{2},j+\frac{1}{2}} + \phi_{i-\frac{1}{2},j+\frac{1}{2}} - \phi_{i+\frac{1}{2},j-\frac{1}{2}} - \phi_{i-\frac{1}{2},j-\frac{1}{2}}}{2h} \right).$$

The nodal Laplace operator is derived by using finite element techniques. The weak formulation of the Laplace operator is

$$\int \nabla\phi(x) \cdot \nabla\psi(x) dx = \int \vec{f} \cdot \nabla\psi(x) dx \quad \forall\psi(x)$$

The ψ are piecewise bilinear elements with support over the four adjacent cells of a given node. On the coarse/fine interfaces the basis functions are associated with coarse nodes only, values at intermediate fine nodes are linearly interpolated from the coarse nodes. This insures a continuous basis set. Properly normalized, the right hand side of the weak formulation is a divergence operator. The projection is solved using the time difference of the velocities

$$\Delta_{i+\frac{1}{2},j+\frac{1}{2}}\phi = \nabla_{i+\frac{1}{2},j+\frac{1}{2}} \cdot \left(\frac{U^* - U}{\Delta t} \right).$$

The gradient of ϕ is the subtracted from the velocity difference

$$\frac{U^{n+1} - U}{\Delta t} = \frac{U^* - U}{\Delta t} - \nabla_{i,j}\phi$$

and the pressure is updated as

$$p^{n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \phi.$$

Notice the difference between these relations and (5.2, 5.3). One increments the pressure while the other computes a new pressure. These formulations are equivalent in the continuous limit and stems from deciding whether to project the velocity field or its time derivative.

In discussing the single level algorithm we have avoided discussing boundary conditions. For a single level advance, both for the the MAC and nodal projections, physical boundary conditions dictate the normal derivative of ϕ is set to zero. Otherwise, interpolation is used from a lower level assuming ϕ^{l-1} is piecewise constant.

Like the hyperbolic case, conserved quantities are stored for refluxing. For level l , velocity and advective flux registers at coarse/fine interfaces are initialized as

$$\begin{aligned}\delta u^l &= -A^l U^{ADV,l} \\ \delta F_U^{l,ADV} &= -\Delta t A^l (U^{ADV,l} \tilde{U}^{n+\frac{1}{2},l}).\end{aligned}$$

$(U^{ADV,l} \tilde{U}^{n+\frac{1}{2},l})$ is a velocity tensor but it only has effectively three nonzero components. Both flux registers are initialized to cancel the coarse fluxes. A^l is the face area in 3-D and the edge length in 2-D.

Level $l-1$ flux registers are changed by the advance of the solution on level l .

$$\begin{aligned}\delta U^{l-1} &:= \delta U^{l-1} + \frac{1}{r} A^{l-1} \sum_{faces} U^{ADV,l} \\ \delta F_U^{l-1,ADV} &:= \delta F_U^{l-1,ADV} + A^l \sum_{faces} \Delta t^l (U^{ADV,l} \tilde{U}^{n+\frac{1}{2},l}).\end{aligned}$$

Let

$$V^l = \frac{U^* - U}{\Delta t}$$

then a mismatch of the divergence computed at nodes is stored in registers as

$$\begin{aligned}RHS_{S-P}^l &:= \nabla_{coarse} \cdot (V^l - \nabla \phi^l) \\ RHS_{S-P}^{l-1} &:= RHS_{S-P}^{l-1} + \frac{1}{r} \nabla_{fine} \cdot (V^l - \nabla \phi^l).\end{aligned}$$

The subscript $S-P$ stands for synchronizing projection and RHS indicates that the quantities being saved will be used in a two level projection between levels l and $l-1$ in order to correct inconsistencies in the divergence of the solution at the coarse/fine interface. The operators ∇_{coarse} and ∇_{fine} are node based operators that when combined compute the composite divergence at a given node. The coarse operator zeros out the contribution from the fine cells while the fine node zeros out the contribution from the coarse node as shown in figure 5.1.

5.2.2 Multilevel Algorithm

Unlike the hyperbolic case, advancing level l r steps to coincide with one step of level $l-1$ does not just result in local corrections of cells on level $l-1$ at coarse/fine interfaces. Instead, a global correction must be made on both levels

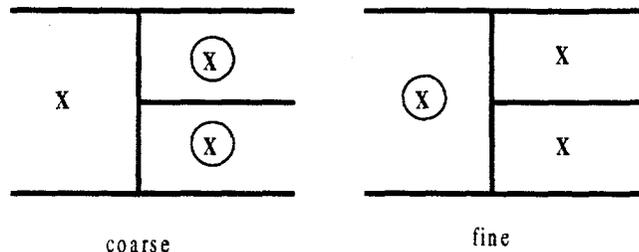


Figure 5.1: The coarse and fine operators ∇_{coarse} and ∇_{fine} zero out contributions from fine and coarse cells respectively.

to satisfy the elliptic constraints. The authors of [1] call this resynchronization. Registers for resynchronization, have already been described in the previous section. We outline the resynchronization process below.

(Step 1) The first step involves averaging down level $l+1$ to level l . Pressure is time averaged (since it is staggered in time) and the velocity U is averaged conservatively. This is much like the hyperbolic case with the enhancement of the pressure averaging in time.

(Step 2) The quantity V_{sync}^l , initially defined at coarse/fine interfaces, is defined as

$$V_{sync}^l = -\frac{1}{\Delta t V_c^l} \delta F_U^{ADV,l}.$$

This is a velocity correction from the momentum flux. V_{sync}^l corresponds to the tensor $U^{ADV,l} \tilde{U}^{n+\frac{1}{2},l}$ contributions to the velocity field not being divergence free. Even though V_i is only nonzero at coarse/fine interfaces it will have nonzero values everywhere later in the algorithm.

(Step 3) Once a mismatch is accumulated in the velocity field δU^l , it is projected using a MAC projection

$$\Delta_{MAC} \delta e^l = \nabla_{MAC} \cdot (\delta U^l).$$

The subscript *MAC* is used to distinguish the Laplace and divergence operators from those used in a nodal projection. Though δU^l is local to the coarse/fine interface δe^l is global to the level l grid.

(Step 4) A velocity correction U_{corr}^l is computed as

$$U_{corr}^l = \nabla_{MAC}(\delta e^l).$$

Again, the subscript *MAC* is used to distinguish the MAC gradient from the nodal gradient. U_{corr}^l is nonzero on all edges of the level l grid.

(Step 5) This correction is used to update the advected momentum correction everywhere

$$V_{sync}^l := V_{sync}^l + \nabla_{mac} \cdot (U_{corr}^l \tilde{U}^{n+\frac{1}{2},l}).$$

Now V_{sync}^l carries corrections from changes in velocity and advection fluxes.

(Step 6) The above correction also must change the flux registers at level $l-1$.

$$\begin{aligned} \delta U^{l-1} &:= \delta U^{l-1} + \frac{1}{r} A^{l-1} \sum_{faces} U_{corr}^l \\ \delta F_U^{l-1,ADV} &:= \delta F_U^{l-1,ADV} + A^l \sum_{faces} \Delta t^l (U_{corr}^l \tilde{U}^{n+\frac{1}{2},l}). \end{aligned}$$

(Step 7) A composite solve is done on level l and level $l+1$

$$\Delta(\phi_{sync}) = RHS_{S-P}^l + \nabla(V_{sync})$$

where V_{sync}^l is interpolated to level $l+1$. The combined interpolated values at $l+1$ and V_{sync}^l is called V_{sync} . This projection corrects discrepancies in the divergence at coarse/fine interfaces due to advective and velocity mismatches.

(Step 8) At level l and level $l+1$ velocities are corrected

$$\begin{aligned} U^{n+1,l} &:= U^{n+1,l} + \Delta t^l (V_{sync}^l - \nabla \phi_{sync}^l) \\ U^{n+1,l+1} &:= U^{n+1,l+1} + \Delta t^l (V_{sync}^{l+1} - \nabla \phi_{sync}^{l+1}) \end{aligned}$$

to get a divergence free composite field. The pressures are also updated as

$$\begin{aligned} p^{n+\frac{1}{2},l} &:= p^{n+\frac{1}{2},l} + \phi_{sync}^l \\ p^{n+1-\frac{1}{2r},l} &:= p^{n+1-\frac{1}{2r},l} + \phi_{sync}^{l+1}. \end{aligned}$$

The corrections must also be introduced to the lower level $l - 1$ as

$$RHS_{S-P}^{l-1} := RHS_{S-P}^{l-1} + \frac{1}{r} \nabla_{fine}(V_{sync}^l - \nabla \phi_{sync}^l).$$

(Step 9) Finally, changes must be propagated upward in the AMR hierarchy. For all levels $l_{max} \geq q > l + 1$:

$$U^{n+1,q} := U^{n+1,q} + \Delta t^l I_{const}(V_{sync}^l - \nabla \phi_{sync}^l)$$

$$p^{n+1-\frac{1}{2r^q-1}} := p^{n+1-\frac{1}{2r^q-1}} + I_{lin}(\phi_{sync}^{l+1})$$

The resynchronization process, with the exception of the last step (9), has a multigrid flavor. The time advance/resynchronization combination is like a coarse to fine to coarse V-cycle. When all levels coincide the resynchronization is like a multigrid V-cycle of a composite projection of all levels.

Bibliography

- [1] A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome. A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. Submitted to the Journal of Computational Physics.
- [2] J. Bell, M. Berger, J. Saltzman, and M. Welcome. Three dimensional adaptive mesh refinement for hyperbolic conservation laws. *SIAM J. Sci. Comput.*, 15:127-138, 1994.
- [3] J. B. Bell, P. Colella, and H. M. Glaz. A second order projection method for the incompressible Navier-Stokes equations. *J. Comp. Phys.*, 85(2):257-283, December 1989.
- [4] J. B. Bell, P. Colella, J. A. Trangenstein, and M. Welcome. Adaptive mesh refinement on moving quadrilateral grids. In *Proceedings of the AIAA 9th Computational Fluid Dynamics Conference*, Lecture Notes in Computer Science, June 1989.
- [5] M. Berger and I. Rigoutsos. An algorithm for point clustering and grid generation. *IEEE Trans. on Systems Man and Cybernetics*, 21(5):1278-1286, 1991.
- [6] M. J. Berger. *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*. PhD thesis, Stanford University, 1982.
- [7] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comp. Phys.*, 82:64-84, 1989.
- [8] M. J. Berger and J. Olinger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comp. Phys.*, 53:561-568, March 1984.
- [9] M. J. Berger and J. Saltzman. AMR on the CM-2. *Applied Numerical Mathematics*, 14:239-253, 1994.
- [10] J. Brackbill and J. Saltzman. Adaptive zoning for singular problems in two dimensions. *J. Comp. Phys.*, 46:342, 1982.
- [11] W. L. Briggs. *A Multigrid Tutorial*. SIAM, Philadelphia, 1987.

- [12] K. D. Brislawn, D. Brown, G. Chesshire, and J. Saltzman. Adaptively-refined overlapping grids for the numerical solution of systems of hyperbolic conservation laws. LANL Unclassified Report LA-UR-257, Los Alamos National Laboratory, 1995. Proceedings of ICASE/Langley Conference on Adaptive Meshes.
- [13] Neil Carlson and Keith Miller. Gradient weighted moving finite elements in two dimensions. In *Finite elements (Hampton, VA, 1986)*, ICASE/NASA LaRC Ser., pages 151-164. Springer, New York, 1988.
- [14] G. Chesshire and W. D. Henshaw. Composite overlapping meshes for the solution of partial differential equations. *J. Comp. Phys.*, 90(1):1-64, 1990.
- [15] A. J. Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comp.*, 22:745-762, 1968.
- [16] M. Ciment and R. Sweet. Mesh refinements for parabolic problems. *J. Comp. Phys.*, 12:513-525, 1973.
- [17] P. Colella. Multidimensional upwind methods for hyperbolic conservation laws. *J. Comp. Phys.*, 87:171-200, 1990.
- [18] R. Courant and K.O. Friedrichs. *Supersonic Flow and Shock Waves*. Springer Verlag, New York, 1986.
- [19] R. Courant, E. Issacson, and M. Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. *Comm. Pure Appl. Math.*, 5:243-255, 1952.
- [20] W. Y. Crutchfield. Load balancing irregular algorithms. UC-LLNL Unclassified Report UCRL-JC-107679, Lawrence Livermore National Laboratory, 1991.
- [21] S. F. Davis and J. E. Flaherty. An adaptive finite element method for initial-boundary value problems for partial differential equations. *SIAM J. Sci. Stat. Comp.*, 3(1):6-27, 1982.
- [22] J. E. Fromm. A method for reducing dispersion in convective difference schemes. *J. Computational Physics*, 3:176-189, 1968.
- [23] R. J. Gelinas, S. K. Doss, and K. Miller. The moving finite element method: Applications to general partial differential equations. *J. Comp. Physics.*, 40(1):202-249, 1980.
- [24] James Glimm. Solutions in the large for nonlinear hyperbolic systems of equations. *Comm. Pure Appl. Math*, 1965.
- [25] S. K. Godunov. A finite difference method for the numerical computation of discontinuous solutions of the equations of fluid dynamics. *Mat. Sb.*, 47:271, 1959.

- [26] W. D. Gropp. A test of moving mesh refinement for 2-d scalar problems. *SIAM J. Sci. Stat. Comp.*, 1(2):191-197, 1982.
- [27] F. H. Harlow and J. E. Welch. Numerical calculations of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids*, 8(12):2182-2189, 1965.
- [28] H. Johansen and P. Colella. A cartesian grid embedded boundary method for poisson's equation on irregular domains. submitted to *SIAM J. Sci. Stat. Comp.*, January 1997.
- [29] Peter D. Lax. *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves*, *SIAM Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, 1972.
- [30] D. Lee and Y. M. Tsuei. A formula for estimation of truncation error of convection terms in a curvilinear coordinate system. *J. Comp. Phys.*, 98(1):90-100, 1992.
- [31] M. Lemke and D. Quinlan. P++, a c++ virtual shared grids based programming environment for architecture-independent development of structured grid applications. In *CONPAR/VAPP V, September 1992, Lyon, France*, Lecture Notes in Computer Science. Springer Verlag, September 1992.
- [32] R. Leveque. *Numerical Methods for Conservation Laws*. Birkhauser Verlag, Boston, 1990.
- [33] D. F. Martin and K. L. Cartwright. Solving poisson's equation using adaptive mesh refinement. Electronics Research Laboratory Memorandum UCB/ERL M96/66, University of California at Berkeley, 1996.
- [34] M. L. Minion. A projection method for locally refined grids. *J. Comp. Phys.*, 127:158-178, 1996.
- [35] K. Pao and J. Saltzman. A comparison of approximate flux functions for 1-D gas dynamics. LANL Unclassified Report LA-UR-90-3290, Los Alamos National Laboratory, 1990.
- [36] R. Parsons and D. Quinlan. A++/p++ array classes for architecture independent finite difference computations. In *Proceedings of the Second Annual Object-Oriented Numerics Conference*, April 1994.
- [37] D. Quinlan and M. Berndt. Mlb: Multilevel load balancing for structured grid applications. In *Proceedings of the 1997 SIAM Parallel Computing Conference*, 1997.
- [38] J. Saltzman. Multidimensional upwind methods for hyperbolic conservation laws. *J. Comp. Phys.*, 115(1):153-168, 1994.

- [39] A. A. Samarskii. Economical difference schemes for systems of equations of parabolic type. *Zh. Vychisl. Mat. i Mat. Fiz.*, 4:927, 1964.
- [40] G. Strang. On the construction and comparison of difference schemes and partial differential equations. *SIAM J. Num. Anal.*, 5:506-517, 1968.
- [41] Joe. F Thompson, Z. U. A. Warsi, and C. Wayne Mastin. *Numerical Grid Generation. Foundations and Applications*. North-Holland Publishing Co., New York, Amsterdam, 1985.
- [42] Bram van Leer. Towards the ultimate conservative difference scheme. iv. a new approach to numerical convection. *J. Computational Physics*, 23:276-299, 1977.
- [43] M. Welcome, W. Crutchfield, C. Rendleman, J. Bell, L. Howell, V. Beckner, and D. Simkins. *BoxLib User's Guide and Manual*. Lawrence Livermore National Laboratory, Lawrence Livermore National Laboratory, Livermore, CA 94550, version 0.02 beta edition, September 1994. A Library for Managing Rectangular Domains.
- [44] P. Wesseling. *An Introduction to Multigrid Methods*. John Wiley and Sons, New York, 1992.
- [45] K. H. Winkler. *A Numerical Procedure for the Calculation of Nonsteady Spherical Shock Fronts with Radiation*. PhD thesis, Max Planck Institute for Physics and Astrophysics, 1977.