JOURNAL OF COMPUTATIONAL PHYSICS 82, 64–84 (1989)

Local Adaptive Mesh Refinement for Shock Hydrodynamics

M. J. BERGER

Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, 10012 New York

AND

P. COLELLA

Lawrence Livermore Laboratory, Livermore, 94550 California

Received September 8, 1987; revised May 20, 1988

The aim of this work is the development of an automatic, adaptive mesh refinement strategy for solving hyperbolic conservation laws in two dimensions. There are two main difficulties in doing this. The first problem is due to the presence of discontinuities in the solution and the effect on them of discontinuities in the mesh. The second problem is how to organize the algorithm to minimize memory and CPU overhead. This is an important consideration and will continue to be important as more sophisticated algorithms that use data structures other than arrays are developed for use on vector and parallel computers.

1. INTRODUCTION

In this paper, we present computations that use adaptive mesh refinement to solve multidimensional, time dependent shock hydrodynamics problems. Complicated structures such as multiple Mach reflections arise in these problems. Adaptive techniques are essential for our computations in order to adequately resolve features in the solution within today's computer limitations.

Our starting point will be the algorithms in [6] for adaptive mesh refinement for hyperbolic equations on rectangular grids. In this approach, the refined regions consist of a small number of rectangular grid patches with finer mesh spacing than the underlying global coarse grid. These rectangular subgrids contain points where the error in the coarser grid solution is too high, and other points as well. We use rectangular subgrids so that we can use integration methods for rectangular grids whose convergence properties are well understood. These methods can be made quite efficient on vector and parallel computers. In addition, rectangular grids have a simple user interface. We can use the same integrator on fine and coarse grids. By separating the integrator from the adaptive strategy, an off-the-shelf integrator can

0021-9991/89 $3.00

be used without modification. This eliminates much of the problem specific work in doing adaptive calculations.

The present work differs from that in [6] in several respects. The main one is that we are computing unsteady flows with shocks, so that maintaining global conservation form is a primary consideration. The second difference is that the nested refinements we use have boundaries coinciding with the grid lines of the underlying coarse mesh. This greatly simplifies the maintenance of conservation over the former approach, where the refined subgrids were allowed to be rotated with respect to the coarse grid. Third, great care was taken to obtain an efficient implementation on a supercomputer. The main difficulty with adaptive methods is the need for data structures not usually found in numerical software. We felt the program complexity was high enough to justify the effort of devising as general and automatic an approach as possible.

Earlier work along the lines of the present work was done by [7] in one dimension, and [14] for scalar problems in two dimensions. [19] have also computed transonic flow in two dimensions with grid embedding. However, in the latter two approaches, the grids were not restricted to rectangles. The data structures, and therefore the efficiency of such an approach, are quite different. Our method of adaptivity through grid refinement is in contrast to methods that adapt the grid by moving grid lines into one region, leaving a coarser region somewhere else [18, 1, 15, 13, 20, 8]. Such methods try to get the most accurate solution for a fixed cost, whereas our approach tries to attain a fixed accuracy for a minimum cost. Both approaches have their advantages and disadvantages. The so-called moving grid point methods often have trouble maintaining a smooth grid. Regularity terms and penalty functions used to regularize the grid add overhead and reduce the simplicity of these methods. Local grid refinement, on the other hand, has the drawback of needing special equations at grid interfaces. In a method where a fixed number of grid points are used during a computation, the user must initially guess at what will be an adequate number of points to resolve features in the solution that may arise later. With local grid refinement, grid points are added or removed as necessary.

In the numerical experiments shown below, we have combined this adaptive mesh refinement strategy with the high resolution difference scheme of [10] to develop an almost automatic software tool for solving gas dynamics problems in two space dimensions. A reasonable question is, why is an adaptive method needed, given that the difference scheme used, a second-order Godunov-type method, already has quite high resolution? Conversely, if adaptive methods are used, are such complicated and expensive difference schemes really necessary? The answer is that both components are necessary to obtain well-resolved results for shock hydrodynamics. It has been demonstrated [22] that the more complicated Godunov-type schemes give more resolution per computational dollar than simpler schemes such as Lax–Wendroff. Given that a high quality scheme is necessary, adaptive mesh refinement can then concentrate the computational effort in regions where it is most useful. Since Godunov-type methods are more expensive than simple schemes, the computational savings of selective refinement can be substan-

tial. Some of the computations presented here could not have been done reasonably without the use of an adaptive solver.

In the sections that follow, we describe the adaptive mesh refinement (AMR) algorithm for integrating a general hyperbolic system of conservation laws

$$u_t + f(u)_x + g(u)_y = 0 \qquad \text{on} \quad D \subset R^2$$
$$Bu = b \qquad \text{on} \quad \partial D.$$

Our numerical examples involve the Euler equations for gas dynamics, where

$$u = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \qquad f(u) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uE + up \end{pmatrix}, \qquad g(u) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vE + vp \end{pmatrix}.$$

and

$$p = (\gamma - 1)\left(\rho E - \rho\left(\frac{u^2 + v^2}{2}\right)\right).$$

Although our work to date is in two space dimensions, all the algorithms extend to three dimensions, and in fact it seems possible to implement a general code where the number of dimensions is input.

In the next sections we describe in detail the adaptive mesh refinement algorithm and its implementation. We give enough detail for users interested in modifying the algorithm, using our code, or writing their own. First, we discuss the structures that define our grid hierarchy. Next, we describe the integration scheme for such a (static) grid hierarchy. Third, the grid generation and error estimation procedures used to generate the grid hierarchy itself are presented. Our error estimation procedure is theoretically justifiable only for smooth solutions. We discuss variations of it that may prove useful for problems with shocks. In the last section we present numerical experiments along with a detailed timing analysis of the runs. This program is being used to study Mach reflection in two dimensions with resolution not previously possible. New results, a triple Mach stem configuration at low γ, have been observed.

2. Grid Description

AMR is based on using a sequence of nested, logically rectangular meshes on which the pde is discretized. In this work, we require the domain D to be a finite union of rectangles whose sides lie in the coordinate directions. We assume here that all the meshes are physically rectangular as well, although this is not essential. The method discussed here can be implemented on a general quadrilateral mesh.

(See, for example, [5]). We define a sequence of levels $l = 1, ..., l_{\max}$. A grid $G_{l,k}$ has mesh spacing h_l, with level 1 coarsest, and define

$$G_l = \bigcup_k G_{l,k}.$$

With an abuse of terminology describing a grid and the domain it covers, we have $G_1 = \bigcup_k G_{1,k} = D$, the problem domain. If there are several grids at level 1, the grid lines must "align" with each other; that is, each grid is a subset of a rectangular discretization of the whole space.

We may often have overlapping grids at the same level, so that $G_{l,j} \cap G_{l,k} \neq 0$, $j \neq k$. However, we require that the discrete solution be independent of how G_l is decomposed into rectangles.

Grids at different levels in the grid hierarchy must be "properly nested." This means

(i) a fine grid starts and ends at the corner of a cell in the next coarser grid.

(ii) There must be at least one level $l-1$ cell in some level $l-1$ grid separating a grid cell at level l from a cell at level $l-2$, in the north, south, east, and west directions, unless the cell abuts the physical boundary of the domain.

Note that this proper nesting is not as stringent as requiring a fine grid to be contained in only one coarser level grid. For example, in Fig. 2.1, there is one grid at level 3, $G_{3,1}$. Every grid point in $G_{3,1}$ is contained in one of the two grids at level 2, $G_{2,1}$ or $G_{2,2}$.

Grids will be refined in time as well as space, by the same mesh refinement ratio r, where $r = \Delta x_{l-1}/\Delta x_l$. Thus,

$$\frac{\Delta t_l}{\Delta x_l} = \frac{\Delta t_{l-1}}{\Delta x_{l-1}} = \cdots = \frac{\Delta t_1}{\Delta x_1}$$

and so the same explicit difference scheme is stable on all grids. This means more

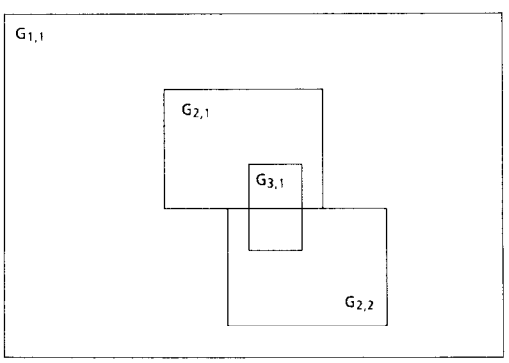


FIG. 2.1. Grid $G_{3,1}$ spans two coarser grids but is properly level nested.

time steps are taken on the finer grids than on the coarser grids. This is a reasonable requirement from the point of view of accuracy, since for many difference schemes, the leading terms in the spatial and temporal truncation error are of the same order. In addition, the smaller time step of the fine grid is not imposed globally. In this implementation we only allow an even refinement ratio. This simplifies the error estimation procedure described later, by avoiding the need of distinguish between an odd and even number of grid points in a grid.

At discrete times the grid hierarchy may be modified. The finest grids need to be changed ("moved," deleted if necessary) most often. When grids at level l are changed, all finer level grids are changed as well, but the coarser grids may remain fixed. New grids at level l may replace the old ones, but they are still subject to the same "proper nesting" requirement.

A point $(x, y) \in D$ may be contained in several grids. The solution $u(x, y)$ will be taken from the finest grid containing the point. If there are several equally fine grids containing the point, any fine grid value will suffice, since the solution on the intersection of overlapping fine grids will be identical.

3. Integration Algorithm

AMR assumes there is a basic, underlying, conservative, explicit finite difference scheme of the form

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \frac{\Delta t}{\Delta x}(F_{i+1/2,j} - F_{i-1/2,j}) - \frac{\Delta t}{\Delta y}(G_{i,j+1/2} - G_{i,j-1/2}). \tag{1}$$

The values $u_{i,j}$ are cell-centered quantities. Each cell is defined by its four corner grid points. If there are no refined regions, then Eq. (1), augmented by the discretized physical boundary conditions, defines the time evolution on a single grid.

With multiple grids, each grid is separately defined and has its own solution vector, so that a grid can be advanced independently of other grids, except for the determination of its boundary values (see Section 4). The integration steps on different grids are interleaved, so that before advancing a grid to time $t + \Delta t$, all the finer level grids have been integrated to time t. Scheme (1) is still initially applied on every grid at every level, but the results will need to be modified in case

(i) the cell is overlayed by a finer level grid; or

(ii) the cell abuts a fine grid interface but is not itself covered by any fine grid.

In case (i), the coarse grid value at level $l-1$ is defined to be the conservative average of the fine grid values at level l that comprise the coarse cell. After every coarse integration step, the coarse grid value is simply replaced by this conservative

average, and the value originally calculated using (1) is discarded. For a refinement ratio of r, we define

$$u_{l,j}^{\text{coarse}} \leftarrow \frac{1}{r^2} \sum_{p=0}^{r-1} \sum_{q=0}^{r-1} u_{k+p,m+q}^{\text{fine}},$$

where the indices refer to the example in Fig. 3.1. We could define a coarse cell $u_{i,j}$ at multiples of the fine time step in the same way, but this is not necessary. This is equivalent (within roundoff error) to redefining the coarse fluxes around the overlayed coarse grid point to be the sum over the fine time steps of all fine grid fluxes calculated on any boundary segment for that cell. However, this implementation would use extra storage to save the fine grid fluxes. By updating the solution values themselves, no extra flux storage is needed.

In case (ii), the difference scheme (1) itself that is applied to the coarse cell must be modified. According to (1), the fine grid abutting the coarse cell has no effect. However, for the difference scheme to be conservative on this grid hierarchy, the fluxes into the fine grid across a coarse/fine cell boundary must equal the flux out of the coarse cell. (This conservative procedure has been discussed by [17]. A fuller discussion of conservation at grid interfaces is in [4].) We use this to redefine the coarse grid flux in case (ii). For example, in Fig. 3.2, the difference scheme at point i, j should be

$$\begin{aligned} u_{i,j}(t+\Delta t_{\text{coarse}}) &= u_{i,j}(t) - \frac{\Delta t_{\text{coarse}}}{\Delta x}\left[F_{i+1/2,j}(t) - \frac{1}{r^2}\sum_{q=0}^{r-1}\sum_{p=0}^{r-1} F_{k+1/2,m+p}(t+q\,\Delta t_{\text{fine}})\right] \\ &\quad - \frac{\Delta t_{\text{coarse}}}{\Delta y}\left[G_{i,j+1/2}(t) - G_{i,j-1/2}(t)\right], \end{aligned} \tag{2}$$

where Δx and Δy are coarse spatial step sizes. The double sum is due to the refinement in time: for a refinement ratio r, there are r times as many steps taken on the fine grid as the coarse grid. If the cell to the north of (i, j) were also refined, the flux $G_{i,j+1/2}$ would be replaced by the sum of fine fluxes as well.

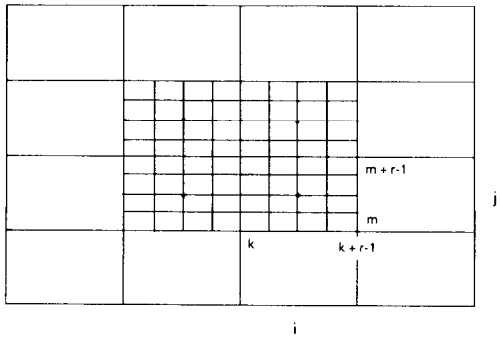


FIG. 3.1. The coarse cell value is replaced by the average of all the fine grid points in that cell.

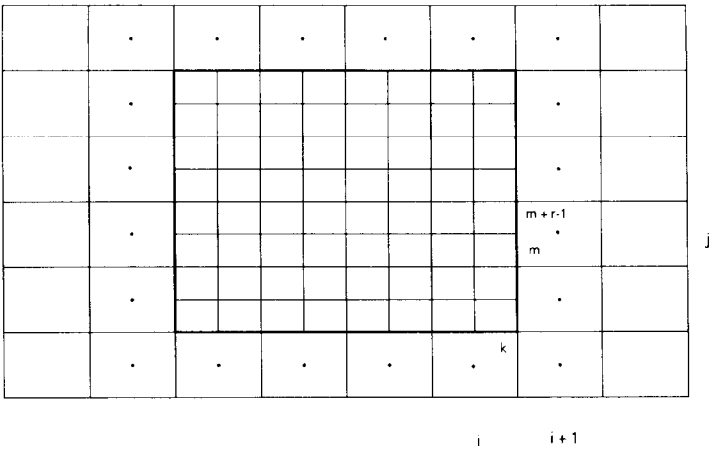


FIG. 3.2. The difference scheme is modified at a coarse cell abutting a fine grid.

This modification is implemented as a correction pass applied after a grid has been integrated using scheme (1) and after the finer level grids have also been integrated, so that the fine grid fluxes in (2) are known. The provisional coarse flux used in (1) is subtracted from the solution $u_{i,j}^{\text{coarse}}(t+\Delta t_{\text{coarse}})$, and the fine fluxes are added using Eq. (2). To implement this modification, we save an array δF of fluxes at coarse grid edges corresponding to the outer boundary of each fine grid. After the coarse grid fluxes have been calculated by (1), we initialize δF with

$$\delta F_{i+1/2,j} := -F_{i+1/2,j}^{\text{coarse}}. \tag{3}$$

At the end of each fine grid time step, we add to $\delta F_{i+1/2,j}$ the sum of the fine grid fluxes along the $(i+1/2, j)^{\text{th}}$ edge,

$$\delta F_{i+1/2,j} := \delta F_{i+1/2,j} + \frac{1}{r^2}\sum_{p=0}^{r-1} F_{k+1/2,m+p}^{\text{fine}}.$$

Finally, after r fine grid time steps have been completed, we use $\delta F_{i+1/2,j}$ to correct the coarse grid solution so that the effective flux is that of (2). For example, for cell $(i+1, j)$, we make the correction

$$u_{i+1,j}^{\text{coarse}} := u_{i+1,j}^{\text{coarse}} + \frac{\Delta t_{\text{coarse}}}{\Delta x_{\text{coarse}}}\delta F_{i+1/2,j}.$$

If the cell $i+2, j$ were refined, we would also make the correction

$$u_{i+1,j}^{\text{coarse}} := u_{i+1,j}^{\text{coarse}} - \frac{\Delta t_{\text{coarse}}}{\Delta x_{\text{coarse}}}\delta F_{i+3/2,j},$$

and similarly for the vertical fluxes.

The boundary fluxes δF are stored in a vector associated with every fine grid. In the initialization step (3), there may be several coarse grids that set δF. Since all fluxes calculated at a given edge and level are identical (up to roundoff error) and are independent of the particular grid on which they are calculated, we simply use the last value assigned. At the end of a time step, we may have several fine grids available to update a given coarse cell edge, since overlapping grids are permitted. For this reason, we use a matrix to indicate the edges of a coarse cell that have already been updated and only perform the update once for each edge. As before, it does not matter which fine grid actually performs the update for any given edge, so the result is independent of the order in which the fine grid list is traversed. This modification is a negligible amount of work, taking approximately 0.3% of a typical run time. On machines with a scatter/gather operation, this should proceed even faster.

We emphasize that this work is done as a "fix up" step after each grid is updated using scheme (1). In this way, the integrator can be separated from the additional work which is needed because of the grid hierarchy. A new difference scheme can be substituted by a user unfamiliar with and not interested in the inner workings of the AMR program.

4. Boundary Conditions

A discussion of boundary conditions completes the description of the integration procedure on a multiple grid hierarchy. Let the interior integration scheme have a stencil which is centered in space, with d points to each side. To compute the new time step, AMR provides solution values at the old time step on a border of cells of width d intersected with the physical domain. The user must supply the code to compute any additional information needed to implement the boundary conditions. For example, if boundary conditions are imposed by extrapolation, the user would provide the extrapolated values for points outside the domain.

For a grid at level l, the bordering cell values are provided using values from adjacent level l grids where they are available; otherwise, the AMR algorithm computes boundary values using bilinear interpolation from coarser level solution values. If necessary, we also interpolate linearly in time.

It may happen that a point (x, y) is inside the domain D, but one or more surrounding coarse grid points needed for the bilinear interpolation are outside. As before, we assume there is a user-supplied routine that can provide exterior coarse grid points given some interior points.

Our implementation partitions the required border cells at level l into rectangular boundary patches. For each rectangular piece we:

(i) find solution values from level $l-1$ grids on a slightly larger rectangular piece enclosing the border cells;

(ii) linearly interpolate for the border values;

(iii) if there are fine grids at level l that could supply some values (say an adjacent fine grid), overwrite the linearly interpolated values from step (ii).

In step (i), most of these coarser level values are found by intersecting the rectangular patch with level $l-1$ grids and by filling the overlapping pieces. However, it may be necessary to go to even coarser grids to supply these level $l-1$ values. This is done by applying (i) to (iii) recursively to the smallest rectangular patch containing the unfilled cells.

For efficiency, it is important that the boundary values are supplied on a rectangular patch at one time and not computed a point at a time. Even though the amount of work is proportional to the boundary of each grid, our initial implementation took 40% of the run time and had to be rewritten. By working on grid patches, the bulk of the memory transfers are done in blocks, and the number of subroutine calls is minimized. This is particularly important on the Cray, where there is a substantial performance penalty for single word accesses and subroutine calls.

5. Creating the Grid Hierarchy

At specified time intervals, an error estimation procedure is invoked, and a new grid structure is determined. If there are several nested levels of refined grids, the error estimation and grid generation procedures are recursively applied on each level, from finest to coarsest, to (re-)create the fine grids at the next level. The error estimation procedure (see Section 6) produces a list of coarse grid points with large error estimates, indicating that a fine grid patch is needed in that region. Every flagged coarse grid point should be included in a finer grid. Our grid generation algorithms try to produce grids that have as little overlap as possible, so that the area that is unnecessarily refined is as small as possible. The algorithm also strives for a small number of patches that are as large as possible, to reduce the computational overhead. It is difficult to find a foolproof algorithm that satisfies these often conflicting goals. However, we have developed heuristics that have been successfully tested in many different applications. A much fuller discussion of grid generation is in [3]. Here, we will describe the particular set of algorithms that produced the numerical results in Section 7.

Suppose there is a base level, l_{base}, where grids will stay fixed, but that the finer levels from $l_{\text{base}}+1$ to l_{finest} may be "moved." Starting with the finest level grids, we estimate the error, using a procedure described next. If there are points where the error estimate is too high, these points are flagged, and a level $l_{\text{finest}}+1$ grid will be needed. Next, we estimate the error on the existing $l_{\text{finest}}-1$ grids. If there are flagged points, a different level l_{finest} grid will be created, making sure that if there are any level $l_{\text{finest}}+1$ grids, they are properly contained in the level l_{finest} grids. This continues until the error is estimated on the base level grids. Thus, it is only possible to add one new level at a time, although many levels may be removed

during a single regridding operation. (At the initial time however, where the initial data is known for all x, y and not just at coarse grid points, it is possible to add many levels at a time. This is essential for some problems, where the error can depend entirely on the initial conditions.)

In more detail, our regridding algorithm performs the following steps:

(1) *Adds the buffer zone.* A buffer zone of unflagged points is added around every grid. This ensures that discontinuities or other regions of high error do not propagate out from a fine grid into coarser regions before the next regridding time. This is possible because of the finite propagation speed of hyperbolic systems. The larger the buffer zone, the more expensive it is to integrate the solution on the fine grids, but the less often the error needs to be estimated on the coarse grids and the fine grids moved. The buffer zone is added by flagging all coarse grid points that are sufficiently close to flagged points with high error estimates. A buffer zone of two cells in each direction is typical. By flagging neighboring points, instead of enlarging grids at a later step, the area of overlap between grids is reduced.

(2) *Flags every cell at level l corresponding to an interior cell in a level $l+2$ grid.* This will maintain proper nesting, by ensuring that there will be a new level $l+1$ grid containing every point in the level $l+2$ grid, even if the level l grid error estimation did not report a high error. This procedure ensures that the fine grid error estimates are used instead of the coarse grid estimates at the same point. To ensure proper nesting, points within one cell of a non-physical (interior) boundary of G_l are deleted from the list of flagged points.

(3) *Creates rectangular fine grids.* The grid generator takes all the flagged points as input, and outputs a list of corners of rectangles that are the level $l+1$ grids. Nearby points are clustered together, and a fine grid patch spanning each cluster is formed. These clustering algorithms use heuristic procedures described separately below.

(4) *Ensures proper nesting.* The new fine grids are checked to ensure that they are properly contained in the base level grids. If they are not, the new grid is repeatedly subdivided until each piece does fit. Since the flagged points originally were inside the base grid, at least one cell from the boundary, the new grid containing the flagged points must eventually lie inside as well. Since the base level grids did not move, step (2) cannot be used to ensure the proper nesting of this level. This problem only arises when the base grids are a non-convex union of rectangles.

Step (3) is the difficult one. Since problems in gas dynamics develop 1-dimensional discontinuities, we have streamlined the more general grid generation procedures of [3] for this particular application. The procedure we use here includes a *bisection* step and a *merging* step. Initially, a grid patch is formed around the entire list of flagged cells on a given level. The *efficiency* of the patch is measured by taking the ratio of flagged cells to the total number of cells in the new grid. If this efficiency rating is less than an input minimum efficiency (e.g., 60%), the long direction of the rectangular grid is bisected, and the flagged points are

sorted into two clusters depending on which half they are in. The process is repeated on the two clusters. The bisection steps ends when each cluster has an acceptable efficiency rating.

The bisection step uses no geometric information, so although each grid may be "acceptable" by itself, the resulting grid hierarchy may not be optimal. For this reason, the bisection is followed by a merge step. In addition to an absolute efficiency criterion, grids are merged if the new grid is relatively more efficient than the two smaller grids. The cost function we use to measure this is proportional to the cost of an integration step on each grid. On an m by n grid, $(m+1)$ by $(n+1)$ fluxes are calculated, with perhaps 1000 vector operations per flux. In addition there is a cost associated with the perimeter of each grid: finding interface conditions, conservative updating of coarser grids, and special slope calculations that are done only for boundary fluxes. Some of this work uses scalar arithmetic, at least on machines such as the Cray 1 that does not vectorize indirect addressing and

$mn+m+n$. Grids are merged if the single resulting grid has a smaller cost. The merging step ends when no pair of grids can be successfully merged.

Although this procedure is somewhat ad hoc, it has been successfully used on several different types of problems. The grid generation routines, not including the solution initialization on each grid or the error estimation to produce the flagged points, account for approximately 1.7% of the CPU time for a typical run.

6. Error Estimation

In [6], estimates of the local truncation error were used to select those grid points on a given level with unacceptably large errors. If the solution $u(x, t)$ is smooth enough, the local truncation error $u(x, t+k) - Qu(x, t)$ on a mesh with spatial step h and time step k satisfies

$$u(x, t+k) - Qu(x, t) = k(c_1(x, t)k^q + c_2(x, t)h^q) + kO(k^{q+1} + h^{q+1})$$
$$\equiv \tau(x, t) + kO(k^{q+1} + h^{q+1}),$$

where the leading term is denoted by τ. Here we assume our difference method Q has order of accuracy q in both time and space. If u is smooth enough, then if we take two time steps with the method Q, to leading order the error is 2τ ,

$$u(x, t+2k) - Q^2u(x, t) = 2\tau + kO(k^{q+1} + h^{q+1}).$$

Let Q_{2h} denote the same difference method as Q but based on a mesh widths of $2h$ and $2k$. Then

$$u(x, t+2k) - Q_{2h}u(x, t) = 2^{q+1}\tau + O(h^{q+2}).$$

By taking two steps with the regular integration scheme, and one "giant" step using every other grid point, the difference

$$\frac{Q^2u(x, t) - Q_{2h}u(x, t)}{2^{q+1} - 2} = \tau + O(h^{q+2})$$

gives an estimate of the local truncation error at time t. We emphasize that it is not necessary to know the exact form of the truncation error (e.g., h^2u_{xxx}), only its order.

This procedure is easily implemented in AMR for the conservative finite difference methods presented here. The values on a grid at a given level are projected onto a virtual grid coarsened by a factor of two in each direction. The solution on both grids is advanced in time: the original grid for two time steps, the coarsened grid for one step using a time step twice as large. The difference between the solutions obtained on the two grids at each point is proportional to the local truncation error at that point. At coarse cells where the difference between the two sets of values exceed some tolerance, all four cells contained in the real grid are flagged as requiring refinement. Notice that this estimation procedure is independent of the finite difference method actually used, as well as the pde. One disadvantage of this procedure is that it always predicts a large error in the neighborhood of captured discontinuities. It is easy to construct examples for which the procedure outlined above will give values on the coarsened grid which differ pointwise by an amount independent of the mesh spacing in the neighborhood of a shock. In general, this leads to refinement of the mesh at all discontinuities with strength greater than some minimum.

Theoretically, one could define a distributional error by averaging the difference between the two solutions over some region centered at the given cell whose size is $O(1)$ relative to the mesh spacing. We have devised various techniques for carrying out such a procedure, all of which are equivalent to ignoring the pointwise error estimate in the neighborhood of those discontinuities which, by some other criteria, are considered adequately resolved. For problems in shock hydrodynamics, only shock discontinuities, and not slip surfaces or contact discontinuities, are likely to satisfy any such criteria. This is because conservative finite difference methods applied to shocks mimic the convergence of characteristics in the analytic solution, so that the shock spreads only over a fixed number of zones independent of the mesh spacing and time. Thus, for example, it is unnecessary to refine the grid in the neighborhood of a shock separating two constant states. In contrast, it is usually necessary to refine at linearly degenerate discontinuities since the number of cells over which they spread is an increasing function of time.

There is a second set of difficulties with refinement in the presence of strong shocks. There is evidence that shock-capturing methods are zeroth-order methods, i.e., that the fluxes computed in the neighborhood of the shock differ by $O(1)$ from the exact fluxes at a given time step [21, 16]. These $O(1)$ errors could generate

waves associated with the characteristic families crossing the shock, sending $O(1)$ pointwise errors into the postshock region. For shocks computed on a single uniform grid, this does not occur, because the same $O(1)$ errors are committed on successive cell edges with a phase lag, so that errors in the time integral of successive fluxes cancel upon differencing. However, when a shock intersects an interface between two grids with different mesh spacings, the $O(1)$ errors in the time integrals of the fluxes on each of the two grids will be different, generating $O(1)$ errors in the solution propagating into the postshock state. In practice, we have observed that the amplitude of the spurious waves generated in this fashion is proportional to the amplitude of the jump in the characteristic quantities carried by characteristics crossing the shock. In practice, then, there is usually a threshold shock strength below which the errors generated by a shock crossing a grid discontinuity are acceptable and above which the errors generated are too large. For the latter shocks, they must be refined everywhere, if they are to be refined anywhere.

We illustrate this with an example where we force the algorithm not to refine the grid above a certain height. This forces the strong incident shock, with a shock Mach number of 10, to pass through a fine grid boundary into the coarse grid. The oscillations caused by this are apparent in the contour plots of Fig. 6.1 and the plot of Fig. 6.2 for a fixed value of y.

Combining the two considerations given above, a fairly general refinement strategy is to use the local truncation error estimate described above, but ignore it in the neighborhood of gas-dynamic shocks whose strength lies below some predetermined threshold. This has the effect of refining, possibly unnecessarily, all shocks whose strength is above the threshold. We have found that this strategy works acceptably well if the coarsened base grid is sufficiently fine, so that the waves are well separated. A much simpler strategy, which is applicable in a large class of problems, is simply to use the user's knowledge of the problem instead of the truncation error estimates. For example, in the shock reflection problems given below, the solution is made up entirely of smooth waves and weak shocks a certain distance behind the incident shock. Consequently, we simply ignore the local truncation error estimate in that region.

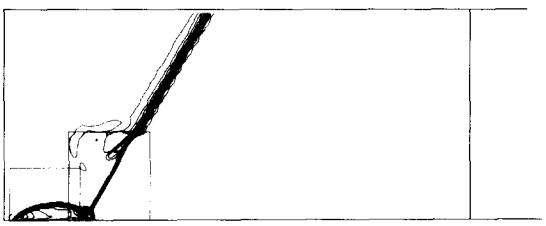


FIG. 6.1. Contour plot showing the effects of a strong shock passing through a grid boundary.

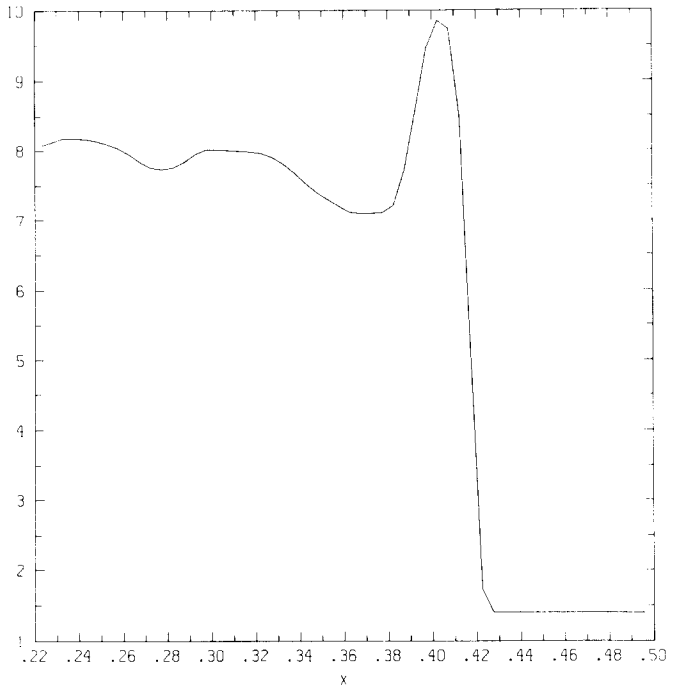


FIG. 6.2. Density profile for a horizontal line slightly below the grid interface intersecting the shock.

7. NUMERICAL RESULTS

We choose as our test problem reflection of a planar shock in an ideal gas by an oblique surface. In this problem, a straight shock is incident on a perfectly reflecting surface. At later times, a reflected wave pattern is generated, depending only on α, M_s, and γ, where α is the angle between the direction of propagation and the reflecting surface, M_s is the incident shock Mach number, and γ is the ratio of specific heats. The solution to this problem is formally self-similar, depending on (x, y, t) only in the combination $(x/t, y/t)$. Thus, the time dependence of the solution is given by a linear scaling of a fixed wave pattern with time. We are particularly interested in values of the problem parameters for which very complicated small scale structures are observed.

The underlying integration method in our AMR calculations is a second-order Godunov method described in [10]. In our calculations, we make use of the fact that the regions where the small scale behavior can appear are localized in the vicinity of the reflection point. Our procedure for estimating the error is to measure the local truncation error of the density, except that we do not tag points beyond a certain distance behind the incident shock. This effectively restricts our refinement

TABLE I

Breakdown of Computational Times Obtained Using FLOWTRACE

Grid integration	78.29
Interpolation	12.95
Output	2.87
Grid updates	2.78
Grid generation	1.71
Memory management	0.59

to be in a window moving with the reflection point, and shuts off the grid refinement ground the (weak) reflected shock.

The results presented here were calculated on the Cray XMP 22 at the LLNL NMFECC, using the CFT 1.14 compiler. To obtain detailed diagnostic information about where most of the time is spent in the calculation, we used the FLOWTRACE option of the CFT compiler in the first calculation below. The total time spent was 5674 seconds of CPU time. Table I shows a breakdown of the calculation time into six categories: the integration routine, the interpolation routines (for constructing boundary conditions and initializing new fine grids), the updating routines (fine grids updating coarse grids and for maintaining conservation across grid interfaces), the grid generation routines, output routines, and memory management routines.

The main result is that the integration step takes about 80% of the computational time. This figure includes integration steps needed for the error estimation. However, measurements show that the latter is only 3% of the integration cost, with actual useful integration steps accounting for 97% of the integration time. Note that the error estimation cost is very small despite the fact the error is estimated at every other coarse time step. There are two reasons for this. First, over 90% of the cells being integrated belong to the finest level grids (level 3 in both calculations) and the error is not estimated there. Second, since refinements are performed in time as well as space, the overwhelming majority of the work is done on the finest grid. Table II shows the number of cell updates done on each grid level, as well as the total number of cell updates done for error estimation.

We can thus obtain a rough estimate of the efficiency of AMR relative to computing on a uniform grid. About 80% of the run time is spent integrating grids. The

TABLE II

Number of Cell Updates at Each Level

Level 1	2.98×10^5
Level 2	4.59×10^6
Level 3	1.13×10^8
Error estimation	3.06×10^6

finest level grids occupy only about 10% of the domain. Thus an equivalent uniform grid computation would require a factor of 8 more CPU time. Of course in this particular problem, one could omit computations ahead of the incident shock, since the solution there is constant, saving approximately half the uniform grid time. In general, this could not be done.

It is more difficult to compare memory usage with that of a uniform grid calculation. The integration algorithm used here requires five 2-dimensional grid arrays for scratch space, in addition to the four required to store the conserved quantities, so that the memory requirements for a uniform grid calculation would be $9 \cdot 320 \cdot 1600 \approx 4.5 \times 10^6$ words. In contrast, the maximum storage used in the calculation performed here was 8.94×10^5 words. Much of the memory use in AMR is due to saving two time levels of the solution on each grid. It is possible to avoid the memory overhead of having full grid scratch arrays by breaking the calculation into pieces. (In fact, we effectively do this in the AMR calculations by restricting the size of any grid to be less than some pre-determined maximum, subdividing grids that are too large.) However, this would introduce overheads and programming complexities in the uniform grid calculation similar to those in AMR. In any case, even if those overheads could be neglected and only four full grid arrays were required, the memory required would be 2.0×10^6 words, a factor of 2.2 larger than that required by AMR.

In Fig. 7.1, we show results for the case $M_s = 10$, $\alpha = 30°$, $\gamma = 1.4$. The domain is a rectangle of length 2.0 by 0.4, with initial coarse grid spacing $\Delta x = \Delta y = 0.02$. The calculation ran for 149 coarse grid time steps. The error was estimated every other step, with a buffer zone of one cell and a grid efficiency of 65%. The error tolerance was 0.02. The mesh was refined by a factor of 4 in each direction at each grid level. The finest grids in this calculation represent a factor of 4 increase in resolution in each spatial direction over the finest grid calculation in [22]. Figure 7.1a shows the location of the level 2 and 3 grids at time $t = 1.20$. In displaying the solution, we show two sets of plots. One is a contour plot of the full flow field. The other is an enlargement of the region around the reflection point. This is the part of the domain covered by the level two and three grids. In both cases, the contour plots are made using the finest available grid in the subregion. Due to the increased resolution, we can now observe a non-self-similar Kelvin–Helmholtz rollup along the principal slip line. This is to be expected, since this slip line is instable. The Kelvin–Helmholtz rolls are formed near where the weak shock emanating from the second triple point impinges on the slip line. They then propagate along the slip line and are eventually swept up into a large rollup at the tip of the jet, along the bottom wall.

Finally, in Fig. 7.2 we present results for $M_s = 8$, $\alpha = 35°$, $\gamma = 1.107$. It has been noticed [11] that the wave patterns associated with double Mach reflection become increasingly complex as γ is reduced. The jet along the reflecting wall formed by the slip line from the principal Mach triple point is more and more strongly accelerated, pushing the Mach stem out ahead of it. This leads to strongly rotational supersonic flow and the formation of multiple Mach triple points. The pre-

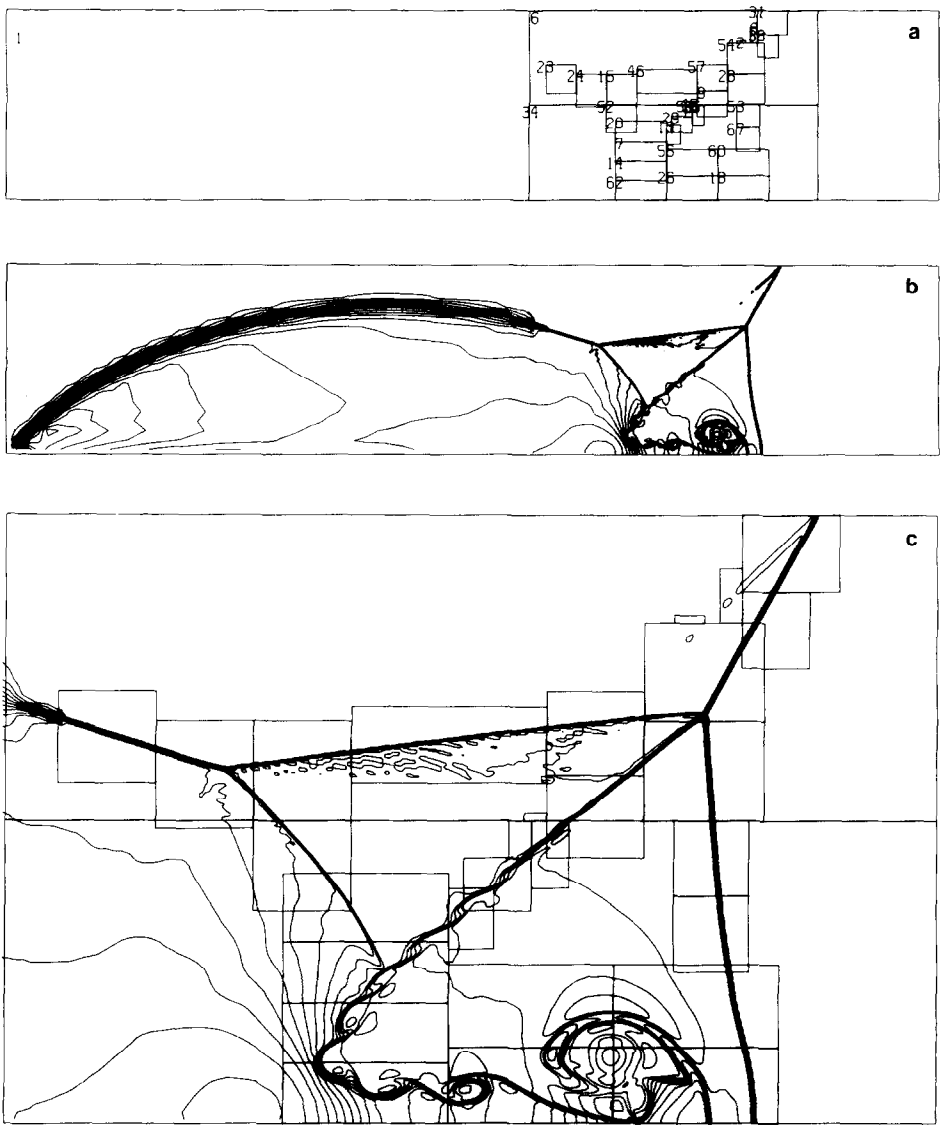
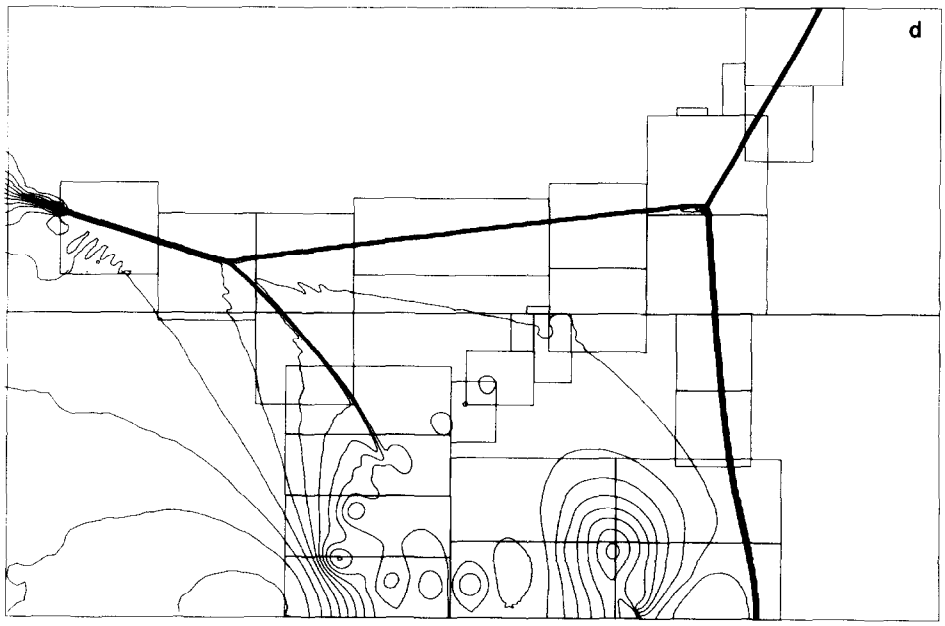


FIG. 7.1. Shock reflection off an oblique wedge with $\gamma = 1.4$, $M_s = 10$, $\alpha = 30°$: (a) Shows the grid hierarchy; grid 1 is a level 1 grid, grids 6 and 34 are level 2 grids, and the rest are level 3 grids, at time $t = 0.12$. (b) Density, full flow field. (c) Density, level 2 and 3 grids only. (d) Pressure, level 2 and 3 grids only. (e) Entropy, level 2 and 3 grids only.



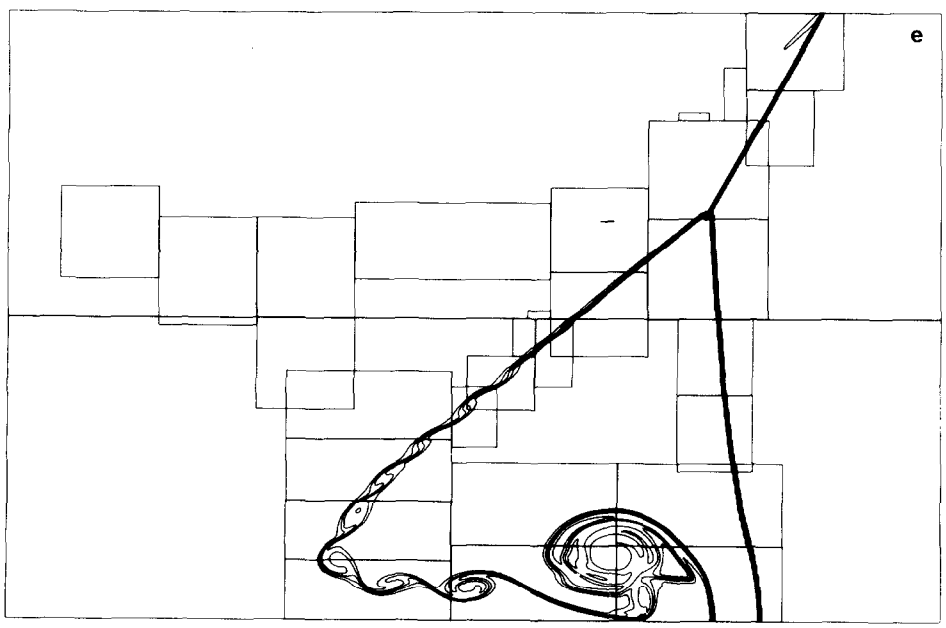


Fig. 7.1—*Continued*

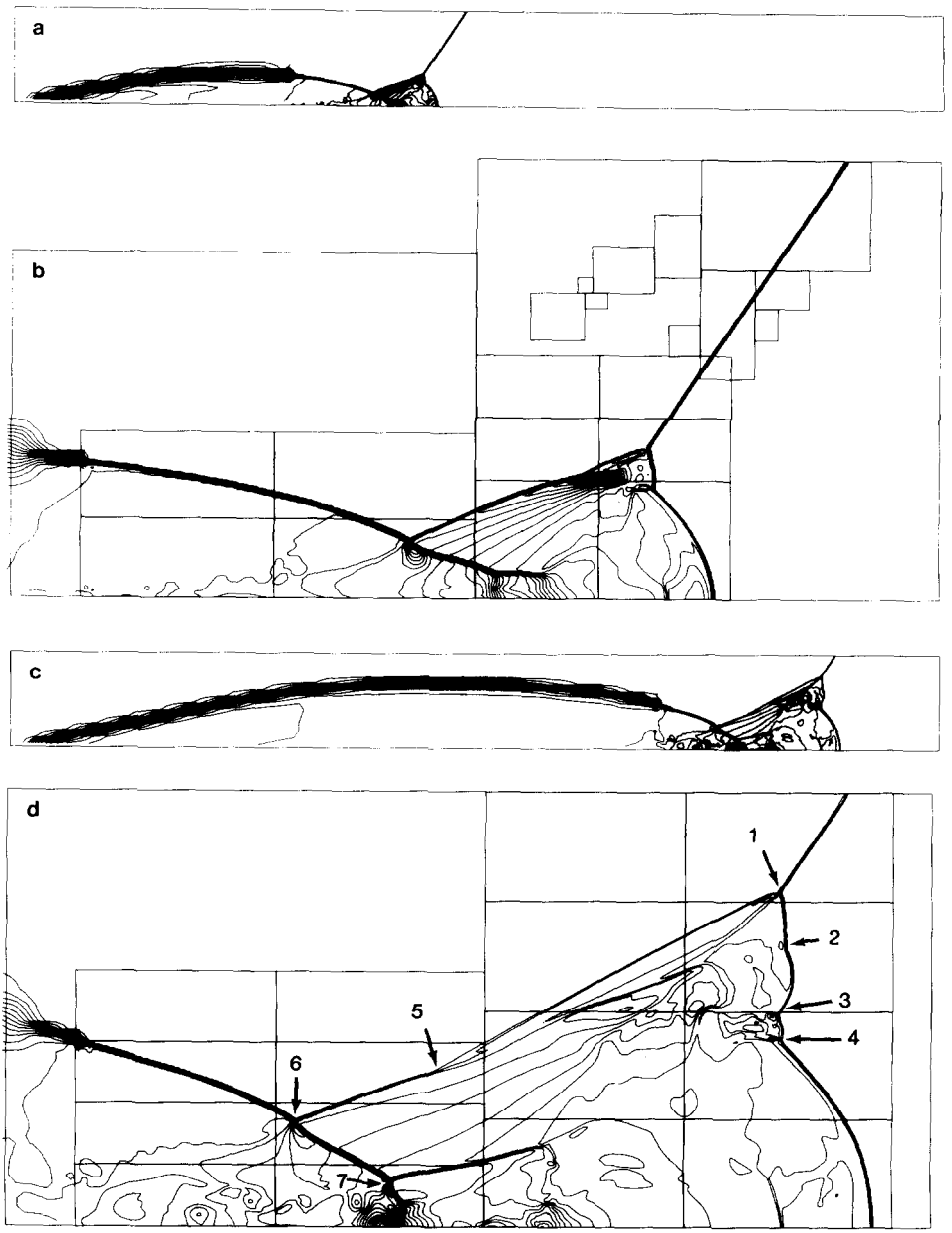


FIG. 7.2. Shock reflection off an oblique wedge with $\gamma = 1.107$, $M_s = 8$, $\alpha = 35°$: (a) Density, full flow field, at $t = 0.115$. (b) Density, level 2 and 3 grids only, $t = 0.115$. (c) Density, full flow field, at $t = 0.230$. (d) Density, level 2 and 3 grids only, $t = 0.230$.

sent results represent a rather extreme example of this, with a total of seven Mach triple points in the double Mach region, including a third triple point along the top shock. The seven triple points are marked in Fig. 7.2d. This calculation gives some indication of why adaptive mesh refinement is an important tool in these problems. As γ approaches one, the distance between the leading edge of the wall jet and the main Mach stem becomes smaller and smaller, requiring more grid resolution in that region. The value of γ in this calculation represents the limit of resolution, given the resources available on the Cray XMP. The calculations in [11] using uniform grids without mesh refinement, were not fully resolved for $\gamma < 1.25$. Even with mesh refinement, at this low value of γ the flow field is not fully resolved at time $t = 0.115$, in Fig. 7.2a and b. Only after running to time $t = 0.230$, which by self-similarity corresponds to increased grid resolution, is the solution adequately resolved.

8. Conclusions

The complexity of our AMR code might be intimidating to a new user. Not counting the integration routine, our program consists of 3000 lines of Fortran. However, a big code is not necessarily a fragile code. We have been careful to develop AMR to make it automatic and robust. In addition, a user should be able to use AMR without having to understand it all. This makes it important to develop AMR in a modular way. A user should be able to plug in an integrator for a new problem without knowing details about how the more computer science oriented parts of AMR work, but knowing that these other parts will indeed work. We have already demonstrated this modularity by using AMR to compute transonic flow in conjunction with FLO52 [5] and to compute a combustion problem with a simple induction time model for chemistry [2].

The most difficult problems will best be solved by combining several adaptive techniques. Despite its more complicated data structures, AMR has already been combined with the conservative front-tracking scheme of [9]. This enables tracking

This combined approach is being used to study transition from regular to Mach reflection. Finally, we intend to couple this method with the variational technique of [8]. Their mesh-moving technique would allow the underlying mesh geometry to be approximately aligned with global features in the flow, leading to more efficient refined meshes. However, the actual mesh refinement for error reduction would be done with AMR, so the global time step penalty of moving mesh methods is not incurred. Lastly, a major open question is how to use implicit difference schemes with embedded grids for a time-dependent calculation. This will be needed to compute solutions to hyperbolic–parabolic problems, such as the Navier–Stokes equations at high Reynolds number.

ACKNOWLEDGMENTS

We thank Michael Welcome for assembling the graphics program for multiple grids used to display the numerical results. We thank John Bolstad for a careful reading of the manuscript. The first author's work was supported in part by the Department of Energy Contract DEAC0276ER03077-V and by the Air Force Office of Scientific Research under Contract AFORSR-86-0148. The second author's work was supported in part by the Office of Energy Research of the U.S. Department of Energy at the Lawrence Livermore National Laboratory under Contract W-7405-ENG-48 and at the Lawrence Berkeley Laboratory under Contract DE-AC03-76SF00098 and by the Air Force Office of Scientific Research under Contract AFOSR-ISSA-870016. Phillip Colella wishes to thank the Courant Institute, which he visited for five months under Department of Energy Contract DEAC0276ER03077-V.

REFERENCES

1. S. ADJERID AND J. FLAHERTY, RPI Computer Science Report No. 85-21 (unpublished).
2. J. BELL, P. COLELLA, J. TRANGENSTEIN, AND M. WELCOME, presented at the 8th AIAA CFD Conference, June 1987, Honolulu, HI; Lawrence Livermore Report UCRL-96443 (unpublished).
3. M. J. BERGER, *SIAM J. Sci. Statist. Comput.* **7**, 904 (1986).
4. M. J. BERGER, *SIAM J. Num. Anal.* **24**, 967 (1987).
5. M. J. BERGER AND A. JAMESON, *AIAA J.* **23**, 561 (1985).
6. M. J. BERGER AND J. OLIGER, *J. Comput. Phys.* **53**, 484 (1984).
7. J. BOLSTAD, Ph. D. thesis, Department of Computer Science, Stanford University, California, 1982 (unpublished).
8. J. U. BRACKBILL AND J. S. SALTZMAN, *J. Comput. Phys.* **46**, 342 (1982).
9. I. CHERN AND P. COLELLA, *J. Comput. Phys.*
10. P. COLELLA, Lawrence Berkely Laboratory Report LBL-17023; *J. Comput. Phys.*
11. P. COLLELA AND H. GLAZ, in *Proceedings, 9th Intl. Conf. Numerical Methods in Fluid Dynamics*, June 1984; Lecture Notes in Physics Vol. 218 (Springer-Verlag, New York/Berlin, 1985).
12. P. COLLELA AND P. WOODWARD, *J. Comput. Phys.* **59**, 264 (1985).
13. H. A. DWYER, AIAA Paper No. 83-0449 (unpublished).
14. W. D. GROPP, *SIAM J. Sci. Statist. Comput.* **4**, 191 (1980).
15. K. MILLER AND R. N. MILLER, *SIAM J. Num. Anal.* **18**, 1033 (1981).
16. W. NOH, Lawrence Livermore National Laboratory Report No. UCRL-52112, June 1976 (unpublished).
17. S. OSHER AND R. SANDERS, *Math. Comput.* **41**, 321 (1983).
18. M. M. RAI AND D. A. ANDERSON, *J. Comput. Phys.* **43**, 327 (1981).
19. W. J. USAB AND E. M. MURMAN, AIAA Paper No. 83-1946-CP, July 1983 (unpublished).
20. K. H. WINKLER, Ph. D. thesis, University of Göttingen, 1976 (unpublished).
21. P. WOODWARD, in *Proceedings Nato Workshop in Astrophysical Radiation Hydrodynamics, Munich, W. Germany, Nov. 1983;* Lawrence Livermore Report UCRL-90009, August 1982 (unpublished).
22. P. WOODWARD AND P. COLELLA, *J. Comput. Phys.* **54**, 115 (1984).