SIAM J. SCI. COMPUT.
Vol. 15, No. 1, pp. 127–138, January 1994 008

THREE-DIMENSIONAL ADAPTIVE MESH REFINEMENT FOR HYPERBOLIC CONSERVATION LAWS*

JOHN BELL†, MARSHA BERGER‡, JEFF SALTZMAN§, AND MIKE WELCOME†

Abstract. A local adaptive mesh refinement algorithm for solving hyperbolic systems of conservation laws in three space dimensions is described. The method is based on the use of local grid patches superimposed on a coarse grid to achieve sufficient resolution in the solution. A numerical example computing the interaction of a shock with a dense cloud in a supersonic inviscid regime is presented. Detailed timings are given to illustrate the performance of the method in three dimensions.

Key words. adaptive methods, mesh refinement

AMS subject classifications. 65M06, 65M50, 76N15

1. Introduction.

Advanced finite difference methods, by themselves, are unable to provide adequate resolution of three-dimensional phenomena without overwhelming currently available computer resources. High-resolution three-dimensional modeling requires algorithms that focus the computational effort where it is needed. In this paper we describe an extension of the adaptive mesh refinement (AMR) algorithm for hyperbolic conservation laws originally developed in [2], [3] to three space dimensions. AMR is based on a sequence of nested grids with finer and finer mesh spacing in both time and space. These fine grids are recursively embedded in coarser grids until the solution is sufficiently resolved. An error estimation procedure automatically estimates the accuracy of the solution, and grid generation procedures dynamically create or remove rectangular fine grid patches. Special difference equations are used at the interface between coarse and fine grid patches to insure conservation. This is all handled without user intervention by the AMR program.

Two-dimensional versions of the AMR algorithm described here have been used to solve fluid flow problems in a variety of settings and have enabled the study of fluid flow phenomena not previously possible. For example, the extra resolution provided by AMR enabled the computation of a Kelvin–Helmholtz type instability along the slip line in a study of Mach reflection off an oblique wedge [2] and aided in the resolution of the weak von Neumann paradox in shock reflection [7]. When combined with a multifluid capability, the algorithm was used to compute the interaction of a supernova remnant with an interstellar cloud [8] and to categorize refraction patterns when a shock hits an oblique material interface [13]. When extended to use body-fitted coordinates, AMR was used to study diffraction of a shock over an obstacle [11]. In each of these cases the use of adaptive mesh refinement reduced the cost of the computation by more than an order of magnitude. The improved efficiency associated with using AMR may make similar flows in three dimensions computationally tractable.

There are several alternative approaches to focusing computational effort in three-dimensional flows. One approach uses a logically rectangular grid with moving grid points that adjust to the flow [11], [9]. There are several difficulties with this approach. First, it is hard

*Received by the editors December 17, 1991; accepted for publication (in revised form) January 27, 1993.

†Lawrence Livermore Laboratory, Livermore, California 94550. This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory contract W-7405-Eng-48. Partial support under contract W-7405-Eng-48 was provided by the Applied Mathematical Sciences Program of the Office of Energy Research and by Defense Nuclear Agency contract IACRO 90-871.

‡Courant Institute, New York University, New York, New York 10012. The work of this author was supported in part by Air Force Office of Scientific Research grant 86-0148, Department of Energy grant DE-FG02-88ER25053, and by a PYI National Science Foundation grant ASC-8848101.

§Los Alamos National Laboratory, Los Alamos, New Mexico 87545. The work of this author was supported by the U.S. Department of Energy through Los Alamos National Laboratory contract W-7405-ENG-36.

to implement a three-dimensional high-resolution integration scheme for moving rectilinear grids. In our approach the integration scheme need only be defined for uniform rectangular grids; this avoids the complexity and computational cost associated with metric coefficients in the moving grid approach. Furthermore, in three dimensions it is extremely difficult to effectively cluster points to capture unsteady phenomena while maintaining a grid with sufficient smoothness in both space and time to permit effective computation. Even if acceptable grid motion can be determined, the entire computation is usually performed with a fixed number of gridpoints throughout the computation. In contrast, the local grid refinement approach dynamically adjusts the number of zones to match the requirements of the computation. The timestep used in moving mesh codes is also limited by the smallest cell size unless additional work is done by solving the equations implicitly or using techniques that allow each cell to evolve with its own timestep.

Another approach to three-dimensional computations uses adaptive unstructured grids [1], [14]. Unstructured grids offer the most flexibility in optimally placing the gridpoints. They can treat general geometries in a simple uniform way. They also offer a possibility of anisotropic refinement, as might be used for Navier–Stokes boundary layer calculations. However, we favor the use of locally uniform refined patches (the basis for AMR, described later) for their accuracy and wave propagation properties. The development of discretization techniques that avoid degradation for strong shocks on highly irregular meshes remains an open issue. Our use of uniform grids allows us to directly use much of the high resolution difference scheme methodology developed for this flow regime. Uniform patches also have low overhead, both from the computational and the storage point of view. The extra information that is needed, in addition to the actual solution values, is proportional to the number of grids rather than the total number of grid points.

An indication of the robustness of the mesh refinement algorithm is that very few changes were required in extending it from two to three dimensions. However, time-dependent three-dimensional computations push the limits of current machine resources, both in terms of memory and CPU time. For this reason, particular care was taken in the implementation, and a better grid generation algorithm was used to increase the overall efficiency of the code.

The starting point for this paper is the version of AMR presented in [2] and we assume the reader is familiar with that paper. In §2 we describe the differences in the three-dimensional mesh refinement algorithm which have to do with the grid generation algorithm and the error estimator. In §3 we describe the operator split integration scheme used in the numerical experiment. In particular, this section clarifies the interaction of the grid refinement and operator split boundary conditions on patched grids while maintaining conservation at grid interfaces. We include a brief section on implementation since some simple changes that produce much cleaner code have been incorporated. We are also rewriting the code in a hybrid of C++ and FORTRAN. The results of a numerical experiment of a shock-cloud interaction modeling the laboratory experiments of Haas and Sturtevant [12] are presented in §5. Detailed timings are presented as well as memory usage and grid statistics demonstrating that AMR offers significant savings of computational resources and can be an important tool in the study of three-dimensional fluid dynamics.

2. The adaptive mesh refinement algorithm.

AMR uses a nested sequence of logically rectangular meshes to solve a partial differential equation (PDE). In this work, we assume the domain is a single rectangular parallelepiped although it may be decomposed into several coarse grids. With the new grid generator described below, grids at the same level of refinement do not overlap. We require that the discrete solution be independent of the particular decomposition of the domain into subgrids. Grids must be properly nested, i.e., a fine grid should be at least one cell away from the boundary of the next coarser level unless it is touching the boundary of the physical domain. However, a fine grid can cross a coarser grid boundary and still be properly nested. In this case, the fine grid has more than one parent grid. This is illustrated in Fig. 1 in two dimensions. (This set of grids was created for a problem with initial conditions specifying a circular discontinuity.)

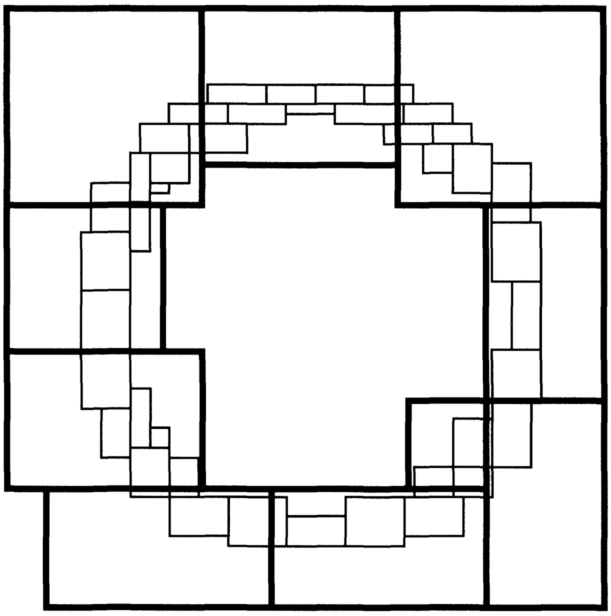

FIG. 1. *A coarse grid with two levels of refined grids. The grids are properly nested, but may have more than one parent grid.*

AMR contains five relatively separate components. The *error estimator* uses Richardson extrapolation to estimate the local truncation error; this determines where the solution accuracy is insufficient. The *grid generator* creates fine grid patches covering the regions needing refinement. *Data structure* routines manage the grid hierarchy allowing access to the individual grid patches as needed. *Interpolation* routines initialize a solution on a newly created fine grid and provide the boundary conditions for integrating the fine grids. *Flux correction* routines insure conservation at grid interfaces by modifying the coarse grid solution for coarse cells that are adjacent to a fine grid.

When all these components are assembled, a typical integration step proceeds as follows. The integration steps on different grids are interleaved so that before advancing a grid, all the finer level grids have been integrated to the same time. One coarse grid cycle is then the basic unit of the algorithm. The variable r denotes the mesh refinement factor in both space and time (typically 4), and *level* refers to the number of refinements (the coarsest grid is at level 0). The regridding procedure is done every few steps, so any particular step may or may not involve regridding.

```
Recursive Procedure Integrate ($lev$)
  Repeat $r^{lev}$ times
    Regridding time? - error estimation and grid generation
                          for level $lev$ grids and finer
    step $\Delta t_{lev}$ on all grids at level $lev$
    if ($lev$+1) grids exist
    then begin
          integrate ($lev$+1)
          conservation_fixup($lev$, $lev$+1)
         end
  end.

$lev = 0$ (* coarsest grid level *)
Integrate ($level$)
```

Single Coarse Grid Integration Cycle

All of these steps are described fully in [2], with the exception of the new grid generation algorithm.

2.1. Grid generation. The grid generation algorithm takes a list of coarse grid points tagged as needing refinement and groups them in clusters. Fine grids are then defined by fitting the smallest possible rectangle around each cluster. The grid generator should produce efficient grids, i.e., rectangles containing a minimum number of cells that are not tagged, without creating a large number of small grids with poor vector performance and excessive boundary overhead.

The grid generator described in [2] uses a simple bisection algorithm. If a single enclosing rectangle is too inefficient, it is bisected in the long direction, the tagged points are sorted into their respective halves, and new enclosing rectangles are calculated. The efficiency is measured by taking the ratio of tagged points to all points in a new fine grid. The procedure is repeated recursively if any of the new rectangles is also inefficient. Since this algorithm uses no geometric information from the tagged points, it often results in too many tiny subgrids and is followed by a merging step. Unfortunately, this results in overlapping grids. Since the memory usage in three-dimensional calculations is at a premium, we want to avoid overlapping. Furthermore, we expect that there will be large numbers of grids in three dimensions which makes the merging step costly.

We now use a new clustering algorithm that uses a combination of signatures and edge detection. Both techniques are common in the computer vision and pattern recognition literature. After much experimentation, described in [4], we settled on what amounts to a "smart bisection" algorithm. Instead of cutting an inefficient rectangle in half, we look for an "edge" where a transition from a flagged point region to a nonflagged one occurs. The most prominent such transition represents a natural line with respect to which the original grid can be partitioned.

We describe the procedure in two dimensions for purposes of illustration. First, the signatures of the flagged points are computed in each direction. Given a continuous function $f(x, y)$, the horizontal and vertical signatures Σ_x and Σ_y are defined as

$$\Sigma_x = \int_y f(x, y)\, dy$$

and

$$\Sigma_y = \int_x f(x, y)\, dx,$$

respectively. For discrete binary images, this is just the sum of the number of tagged points in each row and column. If either signature contains a zero value, then clearly a rectangle can be partitioned into two separate clusters in the appropriate direction. If not, an edge is found by looking for a zero crossing in the second derivative of the signature. If there is more than one such zero crossing, the largest one determines the location for the partitioning of the rectangle. If two zero crossings are of equal strength, we use the one closest to the center of the old rectangle to prevent the formation of long thin rectangles with poor vectorization. This procedure is also applied recursively if the resulting rectangles do not meet the efficiency criterion, with the exception that if no good partition is found and the efficiency is at least 50%, the rectangle is accepted; otherwise, it is bisected in the long direction as a last resort. In computational experiments in two space dimensions, for the same level of accuracy the new algorithm reduces the CPU time by approximately 20%.

Figure 2 illustrates the entire procedure on a sample set of points. The first column on each side is the signature, and next to it is the second derivative. After each partitioning of the points, a new enclosing rectangle is calculated around the tagged points. In this figure, after three steps, the fine grids are acceptably efficient and the procedure stops.

2.2. Error estimation. The second improvement over the basic approach in [2] is the addition of a purely spatial component to the error estimation process to supplement Richardson extrapolation. In Richardson extrapolation, the data on the grid where the error is being estimated is coarsened and then integrated for a timestep. That result is then compared to the result of integrating first and then coarsening. For smooth solutions, the difference in these two results is proportional to the truncation error of the scheme. The motivation for including an additional error measure is to identify structures that are missed by the averaging process associated with the coarsening in the Richardson extrapolation. For example, in gas dynamics a slow-moving or stationary contact surface generates little or no error in the Richardson extrapolation process. In fact, the integrator is not generating any error in this case. However, failure to tag the contact will cause it to be smeared over a coarser grid. The error associated with deciding whether or not to refine the contact surface is associated only with the spatial resolution of the discontinuity, not with errors in integration. Should the contact need to be refined later (for example, if it interacts with another discontinuity), the initial conditions are no longer available to provide higher resolution. For certain special cases a similar phenomenon can also occur for shocks. These problems can be avoided by providing the error estimation routine with the unaveraged grid data so that spatial resolution can also be measured.

Along with the addition of a purely spatial component to the error estimation, we also directly control the process of tagging (and untagging) cells for refinement. For example, a user can insist that only a certain part of the domain is of interest and that the error estimator should be ignored if it says refinement is needed outside of the interesting regions. Similarly, it can force refinement in a particular region independent of the error estimation result.

3. Integration algorithm. For the computational examples presented in §5, we use an operator-split second-order Godunov integration scheme. However, the particular form of the integration scheme is independent of the remainder of the AMR shell. Other integration methods and, in fact, other hyperbolic systems can be easily inserted into the overall AMR framework. The only requirement for the integration scheme is that it be written in flux form, i.e.,

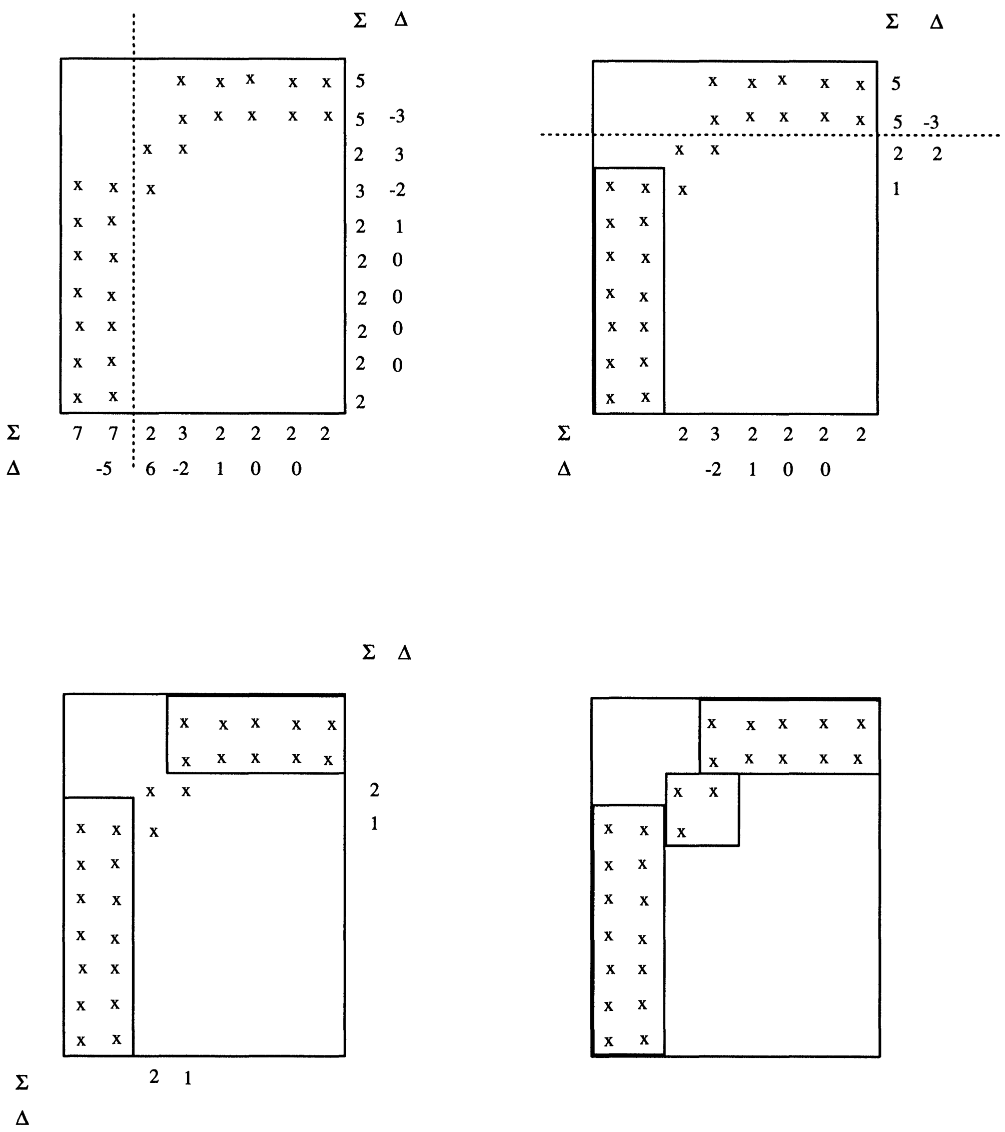

FIG. 2. *The signature arrays Σ_x, Σ_y and the second derivatives Δ_x, Δ_y used to partition the clusters.*

$$U_{ijk}^{n+1} = U_{ijk}^{n} - \Delta t \left(\frac{F_{i+1/2,j,k} - F_{i-1/2,j,k}}{\Delta x} + \frac{G_{i,j+1/2,k} - G_{i,j-1/2,k}}{\Delta y} + \frac{H_{i,j,k+1/2} - H_{i,j,k-1/2}}{\Delta z} \right), \tag{1}$$

where F, G, H are the numerical fluxes in the x, y, z directions, respectively. In its current form, these numerical fluxes are assumed to be explicitly computable from the values in cell ijk and a localized collection of its neighbors, as is typical of conventional explicit finite difference methods. When the integrator is invoked, it is provided with data on the grid to be integrated as well as sufficient boundary data (based on the scheme's stencil) to advance the solution on the given grid. No special stencils are used at fine/coarse interfaces. Instead, coarse grid data is linearly interpolated to the fine grid resolution to provide a border of boundary cells, which is provided to the integrator along with the grid data itself. After the integration step, fluxes are adjusted to insure conservation at grid interfaces.

When operator splitting is used with local grid patches, the only thing to note is that extra boundary cells must be integrated during the first sweep to provide accurate boundary values for subsequent sweeps. For example, for a scheme with four points to the side in the stencil, four entire rows of dummy cells along the top and bottom of the grid must be advanced in the x sweep, so that four points are available for the y sweep at the next stage. For very small grids, this additional boundary work can dominate the computational cost of advancing a grid (particularly for difference methods having a broad stencil such as the Godunov algorithm we are using). This additional boundary work, as well as the vectorization issues, places a premium on generating large but efficient grids during regridding.

As an example, if one $60 \times 60 \times 60$ grid were replaced by two $30 \times 60 \times 60$ grids, both grids would redundantly integrate the overlapping boundary cells. This causes approximately 4% of the total computation to be redundant.

4. Implementation. Two simple decisions had a large impact on the implementation of AMR. The first concerned the organization and separation of the problem-dependent and problem-independent parts of the AMR code. The problem-dependent parts include the particular hyperbolic systems to be solved (and a suitable integration scheme), the initial and boundary conditions, the problem domain, and the error estimation routines. When a new problem is being set up, the changes required to the code are localized to a small number of subroutines. The integration subroutine advances the solution of the particular differential equations for a single timestep on a rectangular subgrid and returns fluxes that are required to insure conservation at coarse-fine boundaries. Consequently, adapting an existing integration module for use with the AMR algorithm is routine. The remainder of the AMR shell treats the data in terms of conserved variables where the number of variables is specified as a parameter. Thus, the data structures, memory management, grid generation algorithms, the logic controlling the timestepping and subcycling on subgrids, interior boundary conditions, and the interfacing between grids that insure conservation are completely divorced from the particular system being solved.

The second implementation detail that simplified the programming of AMR and resulted in much cleaner code than in [2] is the use of a global integer index space covering the domain. These integers are used in describing the location of the grids. Based on the initial (user-specified) domain, given in floating point numbers, an integer lattice based on the number of cells in each dimension (nx, ny, and nz) is determined. The domain may then be partitioned into several coarse grids, each located in subcubes with indices between $(1, 1, 1)$ and (nx, ny, nz). If the refinement ratio is r between level l and level $l + 1$, then the fine grid cells corresponding to coarse grid cell i, j, k are $r \cdot i + p_i, r \cdot j + p_j, r \cdot k + p_k$, where the $0 \leq p_{i,j,k} \leq r - 1$. This completely eliminates round-off error problems that would otherwise require careful coding to determine whether two grids overlap or whether a coarse grid is a parent to a fine grid.

We are currently rewriting the AMR algorithm in C++ with calls to FORTRAN routines for the numerically intensive parts. By constructing the appropriate classes we are able to define a grid calculus in which the computation of intersections and, in fact, the entire regridding process is greatly simplified. Using the built-in macro preprocessor, we are able to implement a large portion of the code in a dimension-independent manner with the dimension as a compile time parameter. With the data hiding inherent in C++ we are able to implement AMR in such a way that the underlying data representation is restricted to a few model dependent classes. Changing the data representation, such as to a sparse data representation for a multifluid version of the code, would be restricted to the member functions of these classes. Since the majority of the run time for AMR is spent integrating large rectangular meshes, the overhead experienced by doing the data management in C++ is just a few percent over an implementation done entirely in FORTRAN. Further optimization of the most important member functions reduces the running time to a few percent less than the pure FORTRAN code.

5. Numerical example. To test the performance of the three-dimensional AMR algorithm, we have modeled the interaction of a Mach 1.25 shock in air hitting an ellipsoidal bubble of Freon-22. The case being modeled is analogous to one of the experiments described by Haas and Sturtevant [12]. The Freon is a factor of 3.0246 more dense than the surrounding air, which leads to a deceleration of the shock as it enters the cloud and a subsequent generation of vorticity that dramatically deforms the bubble.

The computational domain is a rectangular region with length (x) 22.5 cm and width (y) and height (z) of 8.9 cm. The major axis of the bubble is 3.5 cm and is aligned with the z-axis. The minor axes are 2.5 cm with circular cross sections of the bubble in the $x - y$ plane. The bubble is centered at the point $(x, y, z) = (10 \text{ cm}, 0 \text{ cm}, 0 \text{ cm})$. The shocked fluid is placed at points less than or equal to 14.5 cm in the x direction. The shock moves in the direction of increasing x. We use the operator split second-order Godunov method of [6], with Strang splitting. Reflecting boundary conditions are set on the constant z and constant y planes. The inflow and outflow velocities on the constant x planes, as well as the interior fluid velocities, are set so the frame of reference is shifted to one in which the post-shock velocity is zero in order to minimize the x extent of the problem. The numerics preserve a four-fold symmetry in y and z, so we only compute on a quarter of the physical domain. (We have reflected the data in the renderings so that the entire domain is shown.)

We use a simplified treatment of the equations of multifluid gas dynamics. The features of this method include using a single fluid solver, advecting only one additional quantity, and solving a set of equations in conservation form.

We use a γ-law equation of state for each gas with $\gamma_a = 1.4$ for air and $\gamma_f = 1.25$ for Freon. Mixtures of the two gases are modeled with an equation of state defined using both γ's,

$$\Gamma_c = \frac{1}{\frac{f}{\gamma_f} + \frac{(1-f)}{\gamma_a}}$$

for sound speeds, and

$$\Gamma_e = 1 + \frac{1}{\frac{f}{\gamma_f - 1} + \frac{(1-f)}{\gamma_a - 1}}$$

for energy. Here, Γ_c is used to compute an effective sound speed for the mixture with

$$c = \sqrt{\frac{\Gamma_c p}{\rho}}, \tag{2}$$

and f is the mass fraction of the Freon. The harmonic average used to compute Γ_c, used by Colella, Ferguson, and Glaz for multifluid computations [5], expresses the net volume change of a mixture of the gases in terms of their individual compressibilities. The sound speed defined by (2) is used in the integration routine for defining characteristic speeds and for approximate solution of the Riemann problem. We also assume that the two components of a mixed fluid cell are at the same pressure. Pressure is computed from density and internal energy using Γ_e, namely,

$$p = (\Gamma_e - 1)\,\rho\, e.$$

The harmonic average used to compute Γ_e insures that mixing of the two fluids at the same pressure does not result in a pressure and internal energy change of the composite fluid.

We ran the AMR algorithm with an initial coarse grid of $80 \times 16 \times 16$ with two levels of refinement, each by a factor of four, for 100 coarse grid timesteps. The integration used six conserved quantities (mass, momentum, energy, and mass of Freon). The addition of this extra conserved variable besides the usual five found in three-dimensional gas dynamics does not change the AMR structure. The error estimation procedure was modified so that the finest level grids (level 3) only existed in a neighborhood of the bubble and so that acoustic waves away from the bubble were not refined once the incident shock was well past the bubble. The computation was performed on a Cray-2 and required about 20 hours of CPU time. In Fig. 3, we show volume renderings of the density field at four times during the evolution of the cloud. For the renderings, we interpolated all the data onto a uniformly fine grid. The effective result of the volume rendering is to yield an isosurface of the interface between air and Freon. The earliest frame is shortly after the incident shock has completely passed through the bubble. The bubble evolution qualitatively agrees with the experimental results of Haas and Sturtevant. The dominant features of the flow are a strong jet coming from the back of the bubble and the unstable evolution of the roll-up of the outer portion of the bubble.

During the computation, a total of 1.77×10^9 cells were advanced by the integration routine (not including boundary advancements required by operator splitting) for an effective time of 40 μ-seconds per zone including all of the AMR overhead. Figures 4 and 5 present a more detailed breakdown of the algorithm performance. Figure 4 shows a histogram of the dimensions of the level 3 grids during the entire calculation. In this figure, each dimension is counted separately in adding up the total number of grids of a given size. For this computation the maximum grid dimension was limited to 50 and most grids were at least 24 cells wide in each direction. This was done to limit an individual grid size to 125000 zones times six variables per zone. Since the integration algorithm is operator split and each dimension determines the vector length for one step, this gives a measure of the quality of the grids.

Figure 5 shows the number of level 3 cells as a function of time and indicates a growth in memory requirements as the solution complexity grows. The maximum memory required at any point during the run was 22.6 Mwords.

Independent measurements of the integration algorithm on a single large grid $160\times64\times64$ indicated a time of 30 μ-seconds per zone. Thus, the additional boundary work, smaller vector lengths and AMR overhead increased the time per zone by 33%. However, to achieve the same resolution, a uniform grid $1280 \times 256 \times 256$ would be required. Such a computation would require 500 Mwords of storage and 1100 hours of CPU time. The net speedup with AMR is a factor of 55.

Although it is difficult to predict in advance, only 40 of the 80 zones in the x direction played a significant role in the computation. The difficulty in prediction of the extent of the problem is shown by cruder mesh calculations, which indicated a larger computation region. An outflow boundary condition to handle the reflected shock off of the bubble could have eliminated a region of length 40 zones in the x direction. Such a computation would require 251 Mwords of storage and 560 hours of CPU time. Alternatively, we estimate that with a carefully designed, exponentially stretched computation mesh, the fixed grid computational cost also could be halved; however, such an approach also requires substantial knowledge of the solution in designing the mesh. Thus, even from a conservative viewpoint, AMR reduced the computational cost by more than a factor of 20 for this problem.

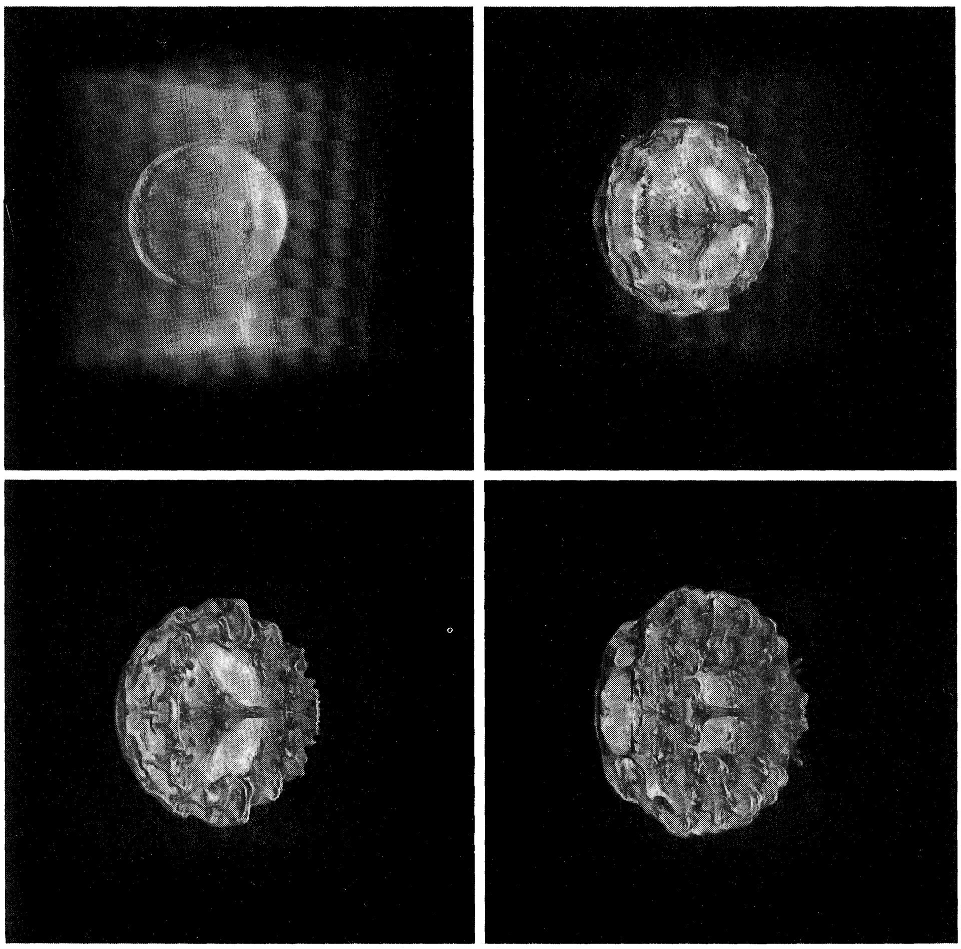
FIG. 3. *Volume renderings of the density during the evolution of the bubble after being hit with a shock wave.*

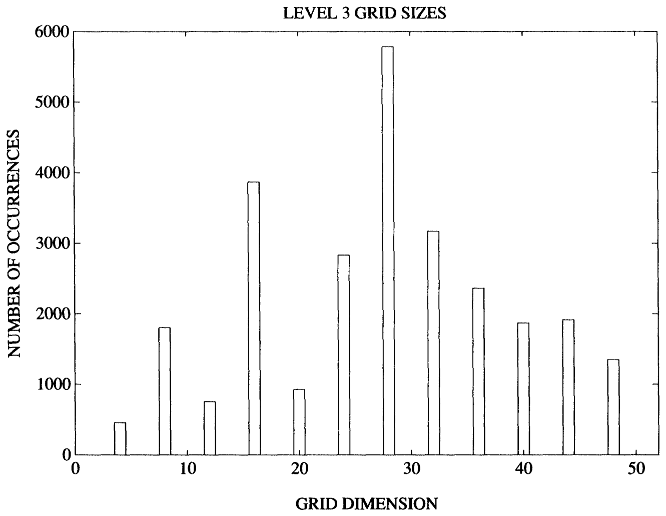

FIG. 4. *Histogram of the linear dimensions of all the level 3 grids.*

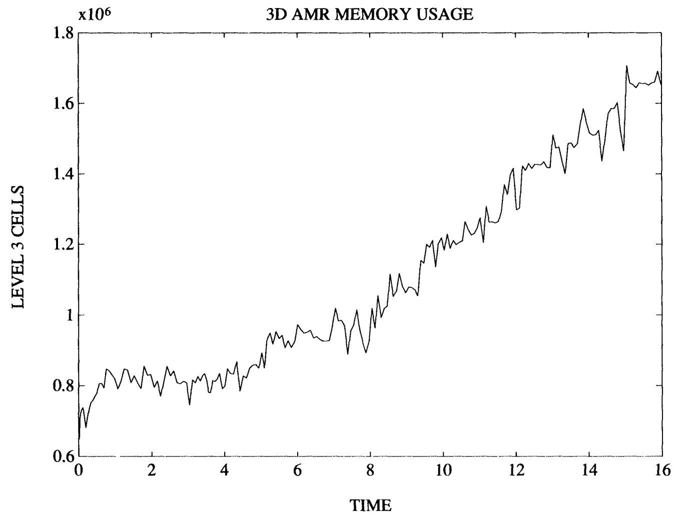

FIG. 5. *The number of level* 3 *cells as a function of time during the calculations.*

Of course, the performance of AMR is highly problem dependent. For some problems the cost reduction may be greater and for some problems it may be less. However, AMR will be cost effective as long as the average number of coarse cells per timestep that require the finest level of refinement throughout the computation is less than 75% of the total number. Although sophisticated grid placement strategies can reduce the advantage of using AMR, these strategies require considerable knowledge of the solution to be effective and may add considerable difficulties to problem setup. AMR provides a high level of performance while making problem setup routine.

REFERENCES

[1] J. D. BAUM AND R. LÖHNER, *Numerical design of a passive shock deflector using an adaptive finite element scheme on unstructured grids*, AIAA-92-0448 (1992).

[2] M. J. BERGER AND P. COLELLA, *Local adaptive mesh refinement for shock hydrodynamics*, J. Comput. Phys., 82 (1989), pp. 64–84.

[3] M. J. BERGER AND J. OLIGER, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys., 53 (1984), pp. 484–512.

[4] M. J. BERGER AND I. RIGOUTSOS, *An algorithm for point clustering and grid generation*, IEEE Trans. Systems Man and Cybernet, 21 (1991), pp. 1278–1286.

[5] P. COLELLA, R. E. FERGUSON, AND H. M. GLAZ, *Multifluid algorithm*, to appear.

[6] P. COLELLA AND H. M. GLAZ, *Efficient solution algorithms for the Riemann problem for real gases*, J. Comput. Phys., 59 (1985), pp. 264–289.

[7] P. COLELLA AND L. F. HENDERSON, *The von Neumann paradox for the diffraction of weak shock waves*, J. Fluid Mech., 213 (1990), pp. 71–94.

[8] P. COLELLA, R. KLEIN, AND C. MCKEE, *The nonlinear interaction of supernova shocks with galactic molecular clouds*, in Proc. Physics of Compressible Turbulent Mixing Intl. Workshop, Princeton, NJ, 1989.

[9] G. S. DIETACHMAYER AND K. K. DROEGEMEIER, *Application of continuous dynamic grid adaption techniques to meteorological modeling. Part i: Basic formulation and accuracy*, Mon. Wea. Rev., 120, (1992), pp. 1675–1706.

[10] H. A. DWYER, M. SMOOKE, AND R. J. KEE, *Adaptive gridding for finite difference solutions to heat and mass transfer problems*, in Numerical Grid Generation, J. Thompson, ed., Elsevier Science Publishing Co., New York, 1982.

[11] I. I. Glass, J. Kac, D. L. Zhang, H. M. Glaz, J. B. Bell, J. A. Trangenstein, and P. Collins, *Diffraction of planar shock waves over half-diamond and semicircular cylinders: an experimental and numerical comparison*, in Proc. 17th International Symposium on Shock Waves and Shock Tubes, Bethlehem, PA, 1989.

[12] J.-F. Haas and B. Sturtevant, *Interaction of weak shock waves with cylindrical and spherical gas inhomogeneities*, J. Fluid Mech., 181 (1987), pp. 41–76.

[13] L. F. Henderson, P. Colella, and E. G. Puckett, *On the refraction of shock waves at a slow-fast gas interface*, J. Fluid Mech, to appear.

[14] D. J. Mavripilis and A. Jameson, *Multigrid solution of the Euler equations on unstructured and adaptive meshes*, in Proc. Third Copper Mountain Conf. Multigrid Methods, Lecture Notes in Pure and Applied Mathematics, 1987; ICASE Report 87-53.