THE ASTROPHYSICAL JOURNAL SUPPLEMENT SERIES, 131:273–334, 2000 November

FLASH: AN ADAPTIVE MESH HYDRODYNAMICS CODE FOR MODELING ASTROPHYSICAL THERMONUCLEAR FLASHES

B. FRYXELL,[1,2] K. OLSON,[1,2] P. RICKER,[2,3] F. X. TIMMES,[2,3] M. ZINGALE,[2,3] D. Q. LAMB,[1,2,3] P. MACNEICE,[4] R. ROSNER,[1,2,3] J. W. TRURAN,[1,2,3] AND H. TUFO[2,5]

Received 1999 November 9; accepted 2000 April 13

ABSTRACT

We report on the completion of the first version of a new-generation simulation code, FLASH. The FLASH code solves the fully compressible, reactive hydrodynamic equations and allows for the use of adaptive mesh refinement. It also contains state-of-the-art modules for the equations of state and thermonuclear reaction networks. The FLASH code was developed to study the problems of nuclear flashes on the surfaces of neutron stars and white dwarfs, as well as in the interior of white dwarfs. We expect, however, that the FLASH code will be useful for solving a wide variety of other problems. This first version of the code has been subjected to a large variety of test cases and is currently being used for production simulations of X-ray bursts, Rayleigh-Taylor and Richtmyer-Meshkov instabilities, and thermonuclear flame fronts. The FLASH code is portable and already runs on a wide variety of massively parallel machines, including some of the largest machines now extant.

Subject headings: equation of state — hydrodynamics — methods: numerical — nuclear reactions, nucleosynthesis, abundances — stars: general

1. INTRODUCTION

The Center for Astrophysical Thermonuclear Flashes, or Flash Center, was founded at the University of Chicago in 1997 under contract to the US Department of Energy as part of its Accelerated Strategic Computing Initiative (ASCI). The goal of the Center is to advance significantly the solution to several problems related to thermonuclear flashes on the surfaces and in the interiors of compact stars (neutron stars and white dwarfs), in particular X-ray bursts, Type Ia supernovae, and classical novae. These problems are remarkable for the breadth of physical phenomena involved. They range from accretion flow onto the surfaces of the compact stars to shear flow and Rayleigh-Taylor instabilities on the stellar surfaces, ignition of nuclear burning under conditions leading to convection, deflagration and detonation flame fronts, and stellar envelope expansion. The physical processes include convection and turbulence at large Reynolds and Rayleigh numbers, convective penetration of stable matter at very high densities, equations of state for relativistic and degenerate matter, thermonuclear burning, mixing instabilities, burning front propagation, and radiation hydrodynamics. Few astrophysical problems present a substantially greater level of physical complexity.

Astrophysical thermonuclear flashes are fascinating in and of themselves, but they are also important for the information they yield on other fundamental questions in astrophysics, for example, X-ray bursts for what they can tell us about the masses and radii of neutron stars, classical novae for the contribution they make to the abundances of intermediate-mass isotopes in the Galaxy and for what they say about how the masses of white dwarfs change with time in close binary systems, and Type Ia supernovae for the contribution they make to the iron group nuclei in the chemical evolution of the Galaxy and for their crucial role as "standard candles" in determining the Hubble constant.

While seemingly diverse phenomena, X-ray bursts, classical novae, and Type Ia supernovae all involve the ignition of a nuclear fuel under degenerate conditions. It is likely that ignition of the nuclear fuel occurs at a single point (or at a few discrete points) in a turbulent environment as a result of the long timescale over which the nuclear fuel is accreted and the extremely short nuclear burning timescales. These ignition points are quickly followed by the propagation of a thermonuclear burning front, a deflagration and/or a detonation, in the lateral directions and the radial direction. Thus, the thermonuclear flash problem is at least two-dimensional and probably three-dimensional, even in the absence of a magnetic field.

Although there have been a few three-dimensional studies (Khokhlov 1995, 1997; García-Senz, Bravo, & Woosley 1999; Kercek, Hillebrandt, & Truran 1999), most attacks on the problems have been in two dimensions (Fryxell & Woosley 1982; Steinmetz, Müller, & Hillebrandt 1992; Shankar, Arnett, & Fryxell 1992; Shankar & Arnett 1994; Livne 1993; Glasner & Livne 1995; Glasner, Livne, & Truran 1997; Khokhlov 1995; Boisseau et al. 1996; Kercek, Hillebrandt, & Truran 1998). The ASCI program provides a unique opportunity to attack the full physics of these problems in three dimensions.

To advance the solution of these astrophysical flash problems requires the development of new simulation tools capable of handling the extreme resolution and physical requirements imposed by these thermonuclear flashes and to do so while making efficient use of the parallel supercomputers developed by the ASCI project, the most powerful constructed to date. The FLASH code represents an important step along the road to this goal. FLASH is a modular, adaptive, and parallel simulation code capable of handling general compressible flow problems in astrophysical environments. It is modular:

[1] Enrico Fermi Institute, University of Chicago, Chicago, IL 60637.
[2] Center on Astrophysical Thermonuclear Flashes, University of Chicago, Chicago, IL 60637.
[3] Department of Astronomy and Astrophysics, University of Chicago, Chicago, IL 60637.
[4] NASA Goddard Space Flight Center, Greenbelt, MD 20771.
[5] Department of Computer Science, University of Chicago, Chicago, IL 60637.

FLASH has been designed to allow users to configure initial and boundary conditions, change algorithms, and add new physical effects with minimal effort. It is adaptive: FLASH uses a block-structured adaptive grid, placing resolution elements only where they are needed most. It is parallel: FLASH uses the Message-Passing Interface (MPI) library to achieve portability and scalability on a variety of different message-passing parallel computers.

The purpose of this paper is to describe the adaptive mesh procedures, hydrodynamic algorithms, physics modules, test cases, and overall organization of the FLASH code. The user's guide that accompanies the FLASH code is more pragmatic; it is designed to enable one to become quickly acquainted with running and customizing FLASH. This paper and the user's guide compliment each other. This paper assumes that the reader has some familiarity both with the basic physics involved and with numerical hydrodynamics methods. This familiarity is absolutely essential in using FLASH (or any other simulation code) to arrive at meaningful solutions to physical problems.

In § 2 we discuss the salient features of the adaptive mesh procedures. The hydrodynamic algorithms are presented in § 3, and the equations of state are addressed in § 4. In § 5 the nuclear reaction networks and their solution method are discussed. The overall organization of the FLASH code and its development architecture are presented in § 6, while in § 7 we present the results of using the FLASH code on a few standard test problems. Some performance benchmarks of the FLASH code are presented in § 8. Finally, in § 9 we summarize the capabilities and present limitations of the FLASH code.

2. ADAPTIVE MESH REFINEMENT

Many scientific modeling challenges today attempt to simulate processes that span very large ranges in spatial scale. These challenges have reached a point where the finest uniform meshes that can be run on the largest computers do not provide sufficient resolution. Larger dynamic ranges in spatial resolution are required, and for this researchers are looking to adaptive mesh refinement (AMR) techniques. At the same time, the largest computers are now highly parallel, distributed memory machines, which provide a challenging programming environment. Few genuinely shared memory machines exist, and those that do exist can perform inefficiently unless the programmer takes aggressive control of decomposing their computational domain. A major reason is that most shared memory machines are actually distributed memory machines with globally addressable memory. Data locality is often critical for good performance because memory access times are not uniform, and fetching data from more remote memory can be relatively expensive.

Many aspects of the problems that the Flash Center will address are expected to benefit from AMR techniques. Further, one of the goals of the ASCI project is to construct and use highly parallel, distributed memory computer architectures. Therefore, we have included parallel, dynamic adaptive mesh refinement as part of the FLASH code. To that end we have used an already existing parallel, AMR package known as PARAMESH (MacNeice et al. 1999, 2000).

PARAMESH is a package of FORTRAN 90 subroutines. It is designed to provide an application developer with an easy route to extend an existing serial code that uses a logically Cartesian structured mesh into a parallel code with AMR. Alternatively, in its simplest use and with minimal effort it can operate as a domain decomposition tool for users who want to parallelize their serial codes but who do not wish to use adaptivity. The package can provide them with an incremental evolutionary path for their code, converting it first to uniformly refined parallel code and then later, if they so desire, adding adaptivity.

There are a number of different approaches to AMR in the literature. Most AMR treatments have been in support of finite-element models on unstructured meshes (e.g., Löhner 1987). These have the advantage that they can be shaped most easily to fit awkward boundary geometries. However, unstructured meshes require a large degree of indirect memory referencing, which can lead to relatively poor performance.

In another version of AMR, Khokhlov (1997) has developed a strategy that refines individual Cartesian grid cells. These cells are managed as elements of a tree data structure. This approach has the advantage that it can produce very flexible adaptivity, in much the same way that the finite-element AMR does. It also avoids the guard cell overhead associated with the subgrid approaches discussed below. However, just as with the unstructured finite-element AMR, it requires a large degree of irregular memory referencing and so can be expected to produce slowly executing code. Also, the code that updates the solution at a grid cell is more labor intensive, and in some cases much more so, than in the subgrid approach. This is because it must constantly use an expensive general interpolation formula to evaluate the terms in the difference equations from the data in the neighboring grid cells, which can be arranged in many different spatial patterns.

PARAMESH falls into a third class of AMR techniques known as block-structured AMR. Berger and coworkers (Berger 1982; Berger & Oliger 1984; Berger & Colella 1989) have pioneered the use of block-structured AMR, using a hierarchy of logically Cartesian grids and subgrids to cover the computational domain. They allow for logically rectangular subgrids, which can overlap, be rotated relative to the coordinate axes, have arbitrary shapes, and be merged with other subgrids at the same refinement level whenever appropriate. This is a flexible and memory-efficient strategy, but the resulting code is very complex and has proven to be very difficult to parallelize. Quirk (1991) has developed a somewhat simplified variant of this approach. De Zeeuw & Powell (1993) implemented a still more simplified variant that develops the hierarchy of subgrids by bisecting grid blocks in each coordinate direction when refinement is required and linking the hierarchy of subgrids developed in this way as the nodes of a data tree. This is similar to the approach used by PARAMESH.

Most of these AMR schemes have been developed within application codes to which they are tightly coupled. Some have been distributed as packages to enable other users to develop their own applications. Serial examples include HAMR (Neeman 1999)[6] and AMRCLAW (LeVeque & Berger 1999)[7]. We are currently aware of three packages that support the

[6] http://zeus.ncsa.uiuc.edu:8080/hneeman/hamr.html.

[7] http://www.amath.washington.edu:807rjl/amrclaw.

subgrid class of AMR on parallel machines. DAGH (Parashar 1999)[8] is a package written in C and C++ that can interface with a user's FORTRAN routines. It executes in parallel using MPI. An object-oriented AMR library called AMR++ is currently under development (Quinlan 1999)[9], and a third package known as SAMRAI (Kohn et al. 1999)[10] is also under development. All the AMR schemes in this class use guard cells at subgrid boundaries as a means of providing needed information to the subgrids that surround it. This can add a significant memory overhead and in most cases a computational overhead also.

The PARAMESH, DAGH, AMR++, and SAMRAI packages are similar in many respects but do have some differences. PARAMESH and SAMRAI have additional support routines for conservation laws and storing variables on cell faces and allow the integration time step to vary with spatial resolution. DAGH and SAMRAI enable error estimation by comparison of the solution at two different refinement levels at each spatial grid point, a feature not (currently) supported by PARAMESH. Perhaps the most significant difference is that DAGH, AMR++, and SAMRAI are constructed and are described in terms of highly abstracted data and control structures. PARAMESH was designed and is described with much less abstraction. This difference will have some impact on the speed with which a user can learn to use each package and also how fast each executes on a given computer. We make no claims here as to which is the best approach.

The philosophy adopted in constructing PARAMESH was to remove from the application developer as much of the burden of interblock and interprocessor communication as possible. We hope the result is that the application developer can focus on writing code to advance their solution on one generic structured grid block that is not split across processors.

The parallel structure that PARAMESH assumes is a single program multiple data (SPMD) approach. In other words, the same code executes on all the processors being used, but the local data content modifies the program flow on each processor. It is the message-passing paradigm, but with the burden of message-passing removed from the application developer by the package.

There are two critical data structures maintained by PARAMESH, one to store the model solution and the other to store the tree information describing the numerical grid. The data that constitute the solution can include data located at the center point of grid cells and data located at the centers of the grid cell faces. The PARAMESH package builds a hierarchy of subgrids to cover the computational domain, with spatial resolution varying to satisfy the demands of the application. These subgrid blocks form the nodes of a tree data structure (quad-tree in two dimensions or oct-tree in three dimensions). PARAMESH supports one-, two-, 2.5- (such as is used frequently in magnetohydrodynamics applications where a magnetic field pointing out of the two-dimensional plane is kept), and three-dimensional models. The FLASH code uses only cell-centered data.

The grid blocks in PARAMESH are assumed to be logically Cartesian (or structured). By this we mean that within a block a cell whose first dimension index is i lies between cells $i - 1$ and $i + 1$. The actual physical grid geometry can be Cartesian, cylindrical, spherical, polar (in two dimensions), or any other metric that enables the physical grid to be mapped to a Cartesian grid. The metric coefficients that define quantities such as cell volumes are assumed to be built into the user's algorithm. Each block has $nxb \times nyb \times nzb$ cells. Blocks are refined by halving a block along each dimension and creating a set of new subblocks. In two dimensions this process would result in the creation of four new blocks, each of which is itself a Cartesian mesh of $nxb \times nyb$ cells but with a spatial resolution twice that of the parent block that is being refined. PARAMESH also restricts these newly created blocks to fit exactly within the borders of their parent block. In this manner the entire domain of the calculation is covered with blocks of varying spatial resolution.

Figure 1 shows the structure of a single PARAMESH block. Each grid block has a user-prescribed number of guard cell layers at each of its boundaries. These guard cells are filled with data from the appropriate neighbor blocks or by evaluating user-prescribed boundary conditions, if the block boundary is part of a boundary to the computational domain. These guard cells are the overlapping regions of the local block with any of its neighboring blocks. The piecewise parabolic method (PPM) algorithm used for the hydrodynamics portion of the FLASH code requires four guard cells at each spatial boundary. The guard cells for each block must be filled with data from neighboring blocks in order for the solution to be advanced on that block. PARAMESH provides a routine for guard cell filling. If a block boundary occurs at a jump in refinement, these cells are filled via interpolation from their neighbor. Otherwise they are filled via a direct copy. The FLASH code employs quadratic interpolation for the guard cell filling operation. Once the guard cell region for a given block is filled, that block can have its variables updated to the next time level in parallel and independently of any other blocks.

Requiring that all grid blocks have an identical logical structure may, at first sight, seem inflexible and therefore inefficient. In terms of memory use this is certainly true, although even in extreme cases the associated memory overhead is rarely more than about 30%. However, this approach has two significant advantages. The first and most important is that the logical structure of the code is considerably simplified, which is a major advantage in developing robust parallel software. The second is that the data structures are defined at compile time, which gives modern optimizing compilers a better opportunity to manage cache use and extract superior performance.

PARAMESH manages the creation of the grid blocks, builds and maintains the tree structure that tracks the spatial relationships between blocks, distributes the blocks among the available processors, and handles all interblock and interprocessor communication. It can distribute the blocks in ways that maximize block locality and so minimize interprocessor communications. It also keeps track of physical boundaries on which particular boundary conditions are to be enforced, ensuring that child blocks inherit this information when appropriate.

[8] http://www/caip.rutgers.edu/parashar/DAGH.
[9] http://www.c3.lanl.gov/dquinlan/AMR++.html.
[10] http://www.llnl.gov/CASC/SAMRAI.

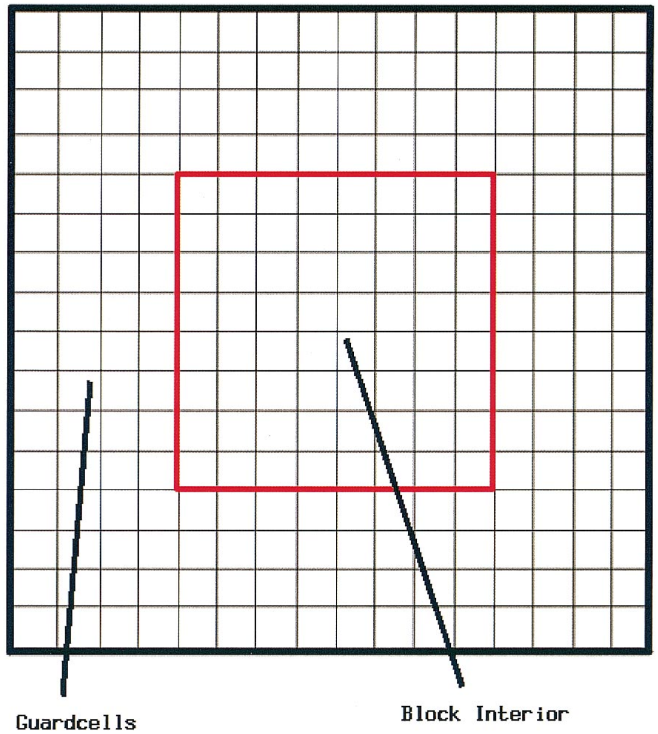

FIG. 1.—Structure of a single block is a logically Cartesian and structured mesh. The interior of the block is highlighted by the interior bold box and encompasses a region that is 8 × 8 cells large in the FLASH code. The guard cell region surrounding the block interior has four guard cells and represents those cells that overlap with the interior domains of neighboring blocks. The size of the block interior and the number of guard cells are adjustable parameters of PARAMESH.

To organize the blocks across a multiprocessor machine, each block is stored at one of the nodes of a fully linked tree data structure (i.e., the locations of the neighbor blocks are "known" by each block). To create the tree, each block stores its parent, any child blocks it might have, and a list of its neighboring blocks. The neighbor list stores the blocks that are the same level of refinement in the north, south, east, west, up, and down directions. If a neighboring block does not exist at the same level of refinement, a null value is stored in the appropriate place in the list of neighboring blocks. A convention of PARAMESH is that if a block is located at the boundary of the physical domain, the neighbor link in that direction is given a specific value to indicate this condition. Marking the neighbor values in this way provides an easy way to detect if a block lies on the domain boundary and which face of that block lies on the boundary so that problem-specific boundary conditions can be set. All the tree links (i.e., children, parents, and neighbors) are stored as integers that indicate a local identification into a processor's memory and a second number that indicates on which processor that child, parent, or neighbor is located. PARAMESH also stores the x, y, z coordinates and bounding box information for each block at each node in the tree. A typical tree and its relation to a set of blocks are shown for a two-dimensional case in Figure 2. For most applications the blocks at the leaves of the tree will represent those blocks that contain the current solution. As the mesh of blocks is refined or derefined, PARAMESH ensures that all the tree links are reset accordingly.

Refinement and derefinement of the set of blocks is accomplished by looping through the local list of on-processor blocks and computing a user-defined error criterion. Based on this criterion, each block is marked for refinement or derefinement. When a block is refined, two, four, or eight child blocks are created, depending on the selected dimensionality. As these blocks are created, they are temporarily placed at the end of the list located on the processor of the block that is refining. When a set of sibling blocks derefines, those blocks are simply removed from the tree structure, and the memory locations of those blocks are overwritten by packing the list of remaining on-processor blocks. After the refinements and derefinements are completed, the redistribution of blocks for load balancing is performed using a work-weighted, Morton space-filling curve in a manner similar to that described in Warren & Salmon (1993) for a parallel, gravitational N-body tree code. Such a Morton curve is shown in Figure 3. During the distribution step each block is assigned a work value (which can change as a function of time and/or position), the Morton number is computed by interleaving the bits of the integer coordinates of each block, the Morton number is sorted using a parallel bitonic sort (e.g., Dorband 1988 and references therein), and finally the total list is cut into a number of sublists equal to the number of processors such that each sublist has roughly the same amount of work. Note that only the list of Morton numbers is sorted, and the blocks of data at each node of the tree are *not* moved during the sorting step. The sort only returns ranks for each key in the sorted list. Only after the sort is accomplished are the blocks of data moved to their new locations in the redistributed list. This step is only done as a refinement or derefinement occurs and typically does not happen at every time step. This grid redistribution step being accomplished, each processor holds a fraction of the global list of blocks. An additional restriction that PARAMESH enforces (independent of any refinement pattern requested by a user) is that a jump of no more than one refinement level across adjacent blocks at any location in the spatial domain is permitted.

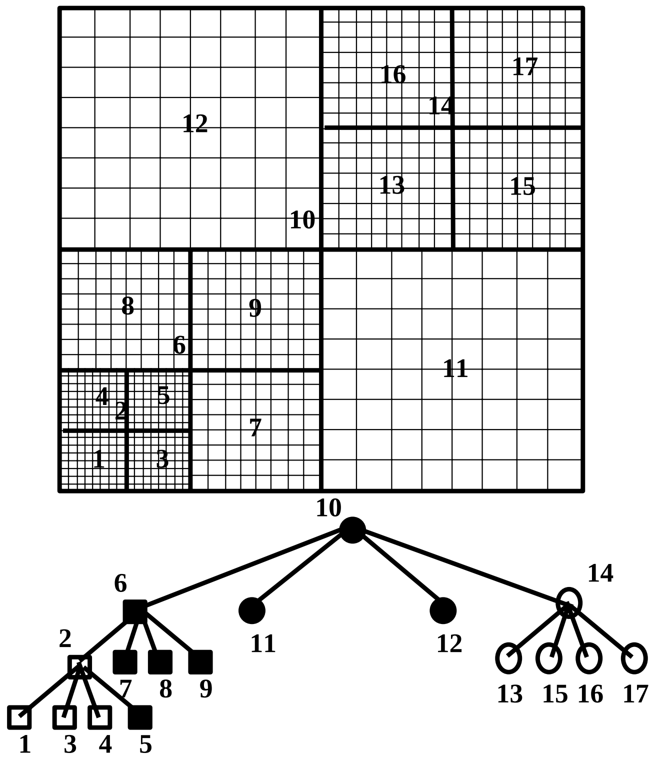

FIG. 2.—Simple set of blocks (*outlined in bold*) that cover a fixed domain. The interior cells within the blocks are also shown. Each block in this example has 8 × 8 cells. In more refined regions the cell size is a factor of 2 smaller in each dimension when compared to its parent block, but each block has identically the same number of cells. The numbers near the centers of each block indicate where that block falls in the Morton ordered list of blocks, while the lower plot shows the resulting tree structure and the positions of each block. The tree symbols indicate on which processor the block would be located on a four processor parallel machine: open squares on processor 0, filled squares on processor 1, filled circles on processor 2, and open circles on processor 3.

The FLASH code is a conservative hydrodynamics code. It requires that the fluxes entering or leaving a grid cell through a common cell face shared with four cells of a more refined neighbor block equal the sum of the fluxes across the appropriate faces of the four smaller cells. PARAMESH provides a routine that enforces these constraints. At jumps in refinement, fluxes of mass, momentum, energy, and composition across the fine cell faces are added and provided to the corresponding coarse face on the neighboring block. These added fluxes are then used to update the values of the coarse cell's local flow variables rather than the fluxes that were computed through the coarse face. This is shown in Figure 4.

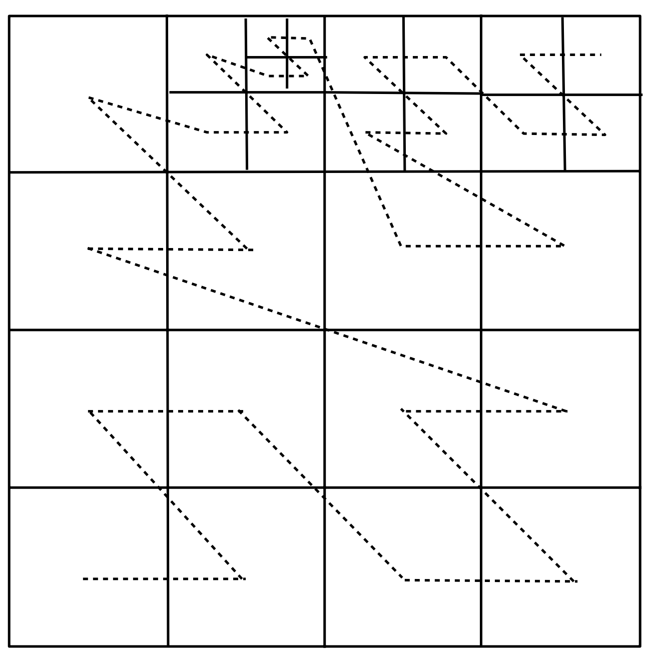

FIG. 3.—Morton space-filling curve for an arbitrary set of blocks of differing spatial resolution

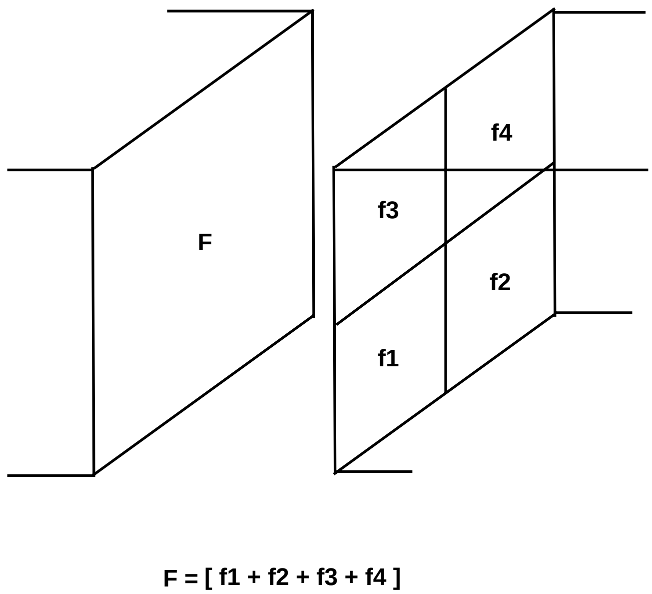

FIG. 4.—To ensure flux conservation at boundaries between blocks where there is a jump in spatial refinement, the fluxes through fine cells (labeled f1, f2, f3, and f4) are added and replace the course cell flux (F).

When there is significant variation in spatial resolution within the computational domain, the time step computed using a fixed Courant number will probably also vary significantly. With PARAMESH, the user can choose to use a uniform time step or can vary the time step from grid block to grid block provided certain restrictions are satisfied. These restrictions are that any two blocks with the same refinement level must use the same time step, that a block cannot use a longer time step than any more coarsely resolved blocks, and that all time steps are integer multiples of the value used for the finest active refinement level.

There are two reasons why we might want to allow the solution on coarser blocks to be advanced with longer time steps, but it is not clear that these reasons are compelling for all cases. If we use a large number of very fine time steps to advance the solution on the more coarsely refined grid blocks, we introduce the possibility that the accumulated effects of numerical diffusion will become significant. A counter argument to this possibility is that the reason these blocks were coarsely refined is that there was not very much structure there anyway, and presumably accuracy increases as the time step gets smaller.

The second reason to use variable time steps is to save the computational effort associated with advancing the solution on the coarser blocks with unnecessarily fine time steps. However, to enable the use of variable time steps, extra memory must be allocated to store temporary copies of the solution. Also, because most real applications have restrictive synchronization constraints, enabling variable time steps tends to force ordering of the way the solution must be advanced on the different grid blocks, and this can have a damaging effect on load balance. The FLASH code does not currently take advantage of this copability of PARAMESH, but it is being considered for future implementations.

PARAMESH is written such that the criteria to determine whether and where to refine or derefine the mesh of blocks are to be supplied by the user. The criteria provided as a default with the FLASH code are based on the work described in Löhner (1987). This technique was originally developed for finite-element applications and has the advantage that it is an entirely local calculation and does not require any global operations. Further, the error indicator that is computed is dimensionless and can be applied in complete generality to any of the field variables of the simulation or any combination of them. A numerical finite-difference estimate of the second derivative in one dimension on a uniform mesh of spacing Δx can be computed from

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1} + 2u_i + u_{i-1}}{\Delta x^2} . \tag{1}$$

In one dimension on a uniform mesh, the modified second-derivative error norm is given by

$$E_i = \frac{|u_{i+1} - 2u_i + u_{i-1}|}{|u_{i+1} - u_i| + |u_i - u_{i-1}| + \epsilon(|u_{i+1}| + 2|u_i| + |u_{i-1}|)} . \tag{2}$$

This error criterion amounts to computing second derivatives of the quantity that is chosen for refinement and normalizing it by averaged gradients of the quantity over the width of a computational cell. The last term in the denominator of the above expression acts as a filter and prevents refinement in regions of small ripples. The constant ϵ is given a value of 0.01. When extending this criterion to multidimensions, all cross derivatives are computed, and the following generalization of equation (2) is

$$E_i = \left\{ \sum_{k,l} \frac{(\partial^2 u/\partial x_k \, \partial x_l)}{[(|\partial u/\partial x_k|_{i+1} + |\partial u/\partial x_k|_i)/\Delta x_l + \epsilon |(\partial^2/\partial x_k)\partial x_l||u|]^2} \right\}^{1/2} , \tag{3}$$

where the sums are carried out over the coordinate directions. If the maximum error norm (E_i) on a block is larger than an adjustable constant, *CTORE*, the block is marked for refinement. If the error norm is less than a second constant, *CTODE*, the block is marked for derefinement. Currently we set $CTORE = 0.8$ and $CTODE = 0.2$. We compute the error norm for a region of each block that includes all interior cells of the block as well as a region two guard cells wide surrounding the block. The frequency of refinement testing is an adjustable parameter, defaulting to once every four time steps. It is important to ensure that discontinuities are detected by a block before they move into the interior cells of that block. For our current calculations, we compute the error norm for the density, pressure, and temperature to detect areas needing refinement. We are considering augmenting this technique with a convergence test similar to that described in Berger & Colella (1989), since there is nothing in the PARAMESH structure that would prevent advancing the solution on meshes that are the parents of the leaf blocks in the tree data structure.

In general, refinement detection techniques used in combination with any AMR package are not guaranteed by themselves to give better results (i.e., better time to solution while using less computer memory). For instance, one can envisage problems where a large fraction of the computational domain contains a great deal of structure and would trigger refinement if the above described refinement criterion is used. Arbitrary refinement according to an error estimate in one or several of the flow variables may not be necessary in all parts of the domain. The reasons for this are problem dependent and are best known by the person running the code for a particular problem. Therefore, the FLASH code allows the user to easily modify the criteria for refinement or to add a completely new criterion by including the refinement marking routine as a module in the overall code architecture (see discussion below on code architecture).

3. HYDRODYNAMICS

The hydrodynamic module in the current version of the FLASH code is based on the PROMETHEUS code (Fryxell, Müller, & Arnett 1989). The module solves Euler's equations for compressible gas dynamics in one, two, or three dimensions. The equations can be written in conservative form as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \boldsymbol{v} = 0 , \tag{4}$$

$$\frac{\partial \rho \boldsymbol{v}}{\partial t} + \nabla \cdot \rho \boldsymbol{v}\boldsymbol{v} + \nabla P = \rho \boldsymbol{g} , \tag{5}$$

$$\frac{\partial \rho E}{\partial t} + \nabla \cdot (\rho E + P)\boldsymbol{v} = \rho \boldsymbol{v} \cdot \boldsymbol{g} , \tag{6}$$

where ρ is the fluid density, $\boldsymbol{v}$ is the fluid velocity, P is the pressure, E is the sum of the internal energy ϵ and kinetic energy per unit mass

$$E = \epsilon + \tfrac{1}{2}v^2 , \tag{7}$$

$\boldsymbol{g}$ is the acceleration due to gravity, and t is the time coordinate. The pressure is obtained from the internal energy and the density by an equation of state (EOS). For the case of a simple gamma-law EOS, the pressure is given by

$$P = (\gamma - 1)\rho\epsilon , \tag{8}$$

where γ is the ratio of specific heats. More general equations of state are discussed later in this section and in § 4.

For reactive flows, a separate advection equation must be solved for each chemical or nuclear species

$$\frac{\partial \rho X_\ell}{\partial t} + \nabla \cdot \rho X_\ell \boldsymbol{v} = 0 , \tag{9}$$

where X_ℓ is the mass fraction of the ℓth species, with the constraint that

$$\sum_\ell X_\ell = 1 . \tag{10}$$

The quantity ρX_ℓ represents the partial density of the ℓth fluid. The code does not explicitly track interfaces between the fluids, so that a small amount of numerical mixing can be expected during the course of the calculation.

The equations are solved using a modified version of the PPM, which is described in detail in Woodward & Colella (1984) and Colella & Woodward (1984). PPM is a higher order version of the method developed by Godunov (Godunov 1959; Godunov, Zabrodin, & Prokopov 1961). In Godunov's method, the flow is divided into a series of slabs, each of which occupies a single zone of the computational grid. The discontinuities between these slabs are treated by solving Riemann's shock tube problem at each zone interface. This treatment has the effect of introducing explicit nonlinearity into the difference equations and permits the calculation of sharp shock fronts and contact discontinuities without introducing significant nonphysical oscillations into the flow. Since the value of each variable in each slab is assumed to be constant, this method is limited to first-order accuracy in both space and time. The conserved quantities are advanced in time by solving conservative finite-difference equations with fluxes computed using the values obtained from the Riemann problem solution.

A formalism for extending Godunov's method to higher order was developed with the MUSCL scheme of van Leer (1979), in which second-order accuracy in both space and time is achieved by representing the flow variables as piecewise linear instead of piecewise constant functions. This is analogous to switching from the rectangle rule to the trapezoidal rule for numerical integration of a function. Another major advance of MUSCL was the incorporation of monotonicity constraints instead of artificial viscosity to eliminate oscillations in the flow. PPM takes the next logical step by representing the flow variables as piecewise parabolic functions. This corresponds to switching to Simpson's rule for numerical integration. Although this could lead to a method that is accurate to third order, PPM is formally accurate to only second order in both space and time. A fully third-order method provides only a slight additional improvement in accuracy but results in a significant increase in the computational cost of the method. However, the most critical steps are performed to third- or fourth-order accuracy, resulting in a method that is considerably more accurate and efficient than most second-order codes using typical grid sizes.

PPM is particularly well suited to flows involving discontinuities, such as shocks and contact discontinuities. The method also performs extremely well for smooth flows, although other schemes, which do not perform the extra work necessary for the treatment of discontinuities, might be more efficient in these cases. The high resolution and accuracy of PPM are obtained by the explicit nonlinearity of the scheme and through the use of smart dissipation algorithms, which are much more effective in stabilizing shock waves than the more traditional approach of using an artificial viscosity. Typically, shocks are spread over only one to two grid points, and in most cases, postshock oscillations are virtually nonexistent. Contact discontinuities and interfaces between different fluids create special problems for Eulerian hydrodynamic codes. Unlike shocks, which contain a self-steepening mechanism, contact discontinuities spread diffusively during a calculation. The farther they propagate through the computational grid, the broader they become. PPM contains a special algorithm that prevents contact discontinuities from spreading more than one to two grid points, no matter how far they propagate.

Colella & Woodward (1984) describe two different formulations of PPM. One solves Euler's equations directly. The other solves Lagrange's equations and then remaps the results back onto the original Eulerian grid at the end of each time step. The second approach can also be thought of as extracting the advection terms out of the hydrodynamic equations and computing them in a separate step. Although the two formulations produce slightly different results, their overall accuracy and efficiency are comparable. The current version of the FLASH code uses the direct Eulerian approach, which will be described below. However, both options may be implemented in future versions.

3.1. *Solution of One-dimensional Equations*

We begin by describing the details of PPM for solving the one-dimensional Euler equations. Modifications for solving the equations in more than one spatial dimension will be discussed below. The equations are solved in the form

$$\frac{\partial \rho}{\partial t} + \frac{\partial A \rho u}{\partial V} = 0 , \tag{11}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial A \rho u^2}{\partial V} + \frac{\partial P}{\partial \xi} = \rho g , \tag{12}$$

$$\frac{\partial \rho v}{\partial t} + \frac{\partial A \rho v u}{\partial V} = 0 , \tag{13}$$

$$\frac{\partial \rho w}{\partial t} + \frac{\partial A \rho w u}{\partial V} = 0 , \tag{14}$$

$$\frac{\partial \rho E}{\partial t} + \frac{\partial A(\rho E + P)u}{\partial V} = \rho u g , \tag{15}$$

$$\frac{\partial \rho X_\ell}{\partial t} + \frac{\partial A \rho X_\ell}{\partial V} = 0 . \tag{16}$$

Here ξ is the spatial coordinate, which, for example, in Cartesian coordinates takes the values (x, y, z); u is the component of the velocity in the ξ direction; v and w are the two components of velocity normal to the ξ direction; V is the volume coordinate; A is the cross-sectional area through which the material is flowing; and g is the gravitational acceleration in the ξ direction. For the case of curvilinear coordinates, additional terms appear in these equations representing centrifugal and coriolis accelerations. These terms are to be treated in exactly the same way as the gravitational acceleration in the following discussion. These additional terms affect only the momentum equations and do not appear in the energy equation.

For a nonideal gas EOS, there are four different adiabatic indices. In this case, we parameterize the EOS using the method of Colella & Glaz (1985), which requires only two of the adiabatic indices. One is the value of γ defined in equation (8), which involves the ratio of the energy and pressure. The other, which is required to compute the adiabatic sound speed, is given by

$$\Gamma = \frac{\rho}{P} \frac{\partial P}{\partial \rho} , \tag{17}$$

where the derivative is taken at constant entropy. Changes to the basic algorithm that are required for the general case are described later in this section.

PPM can be characterized as a finite-volume method, rather than as a finite-difference method, since the initial conditions for a problem are specified as the average of each variable in each computational element rather than as values at particular points in space, such as zone centers or zone interfaces. Typically, the variables that are specified include the gas density, the three components of the fluid velocity, the specific internal energy, and the mass fraction of each nuclear species. Additional variables that are required, such as the zone average values of the pressure, temperature, and adiabatic indices, can be obtained from these using the EOS. This approximation is accurate to second order. The spatial average for any variable U in zone i at time step n is given by

$$\langle U \rangle_i^n = \frac{1}{\Delta\xi_i} \int_{\xi_{i-1/2}}^{\xi_{i+1/2}} U(\xi) d\xi , \tag{18}$$

where $\Delta\xi_i = \xi_{i+1/2} - \xi_{i-1/2}$. For the case of a parabolic distribution of variable U, equation (18) becomes

$$\langle U \rangle_i^n = \tfrac{1}{6}(U_{i-1/2}^n + 4U_i^n + U_{i+1/2}^n) , \tag{19}$$

where the values on the right-hand side of the equation represent the values of U at the zone center and the two zone interfaces at the beginning of time step n. This is just the expected result for integration by Simpson's rule.

The procedure for updating the initial conditions can be divided into two separate parts, the reconstruction step and the solution step. Since the values of the flow variables are not known at any given point in space, it is first necessary to reconstruct the requisite point values at the zone interfaces from the zone average values. The solution step then uses the zone interface and zone average values to construct fluxes through the zone interfaces, which are used in conservative difference equations to update the conserved variables.

3.1.1. *Reconstruction Step*

The reconstruction for PPM is quite complex and is what differentiates it from Godunov's method and MUSCL. The solution step for all three methods is essentially the same. The first step is to perform a high-order interpolation to obtain preliminary values for the primitive variables $U = (\rho, u, v, w, P, \gamma, X_\ell)$ at the zone interfaces using the zone average values of these variables. For PPM we use a cubic polynomial to obtain fourth-order accuracy. In order to interpolate the interface value at $\xi_{i+1/2}$, for example, the zone average values at ξ_{i-1}, ξ_i, ξ_{i+1}, and ξ_{i+2} are required. However, since the average values are not associated with a particular location in space, it is not correct to just pass an interpolation polynomial through them assuming that they are located at the zone centers. What is needed instead is a polynomial that has the correct average values in these four zones.

In the following discussion, the subscripts R and L will denote the right and left interfaces of zone i, e.g., $\rho_R = \rho_{i+1/2}^n$ and $\rho_L = \rho_{i-1/2}^n$. Except where noted, the other primitive variables are to be treated analogously. The right-hand interface value, of the density, for example, is obtained from the equations

$$\begin{aligned}\rho_R = {} & \langle \rho \rangle_i^n + \frac{\Delta\xi_i}{\Delta\xi_i + \Delta\xi_{i+1}} (\langle \rho \rangle_{i+1}^n - \langle \rho \rangle_i^n) \\ & + \frac{1}{\Delta X}\left[\frac{2\,\Delta\xi_{i+1}\,\Delta\xi_i}{\Delta\xi_{i+1} + \Delta\xi_i}(Z_1 - Z_2)(\langle \rho \rangle_{i+1}^n - \langle \rho \rangle_i^n) - \Delta\xi_i Z_1\, \delta\rho_{i+1} + \Delta\xi_{i+1} Z_2\, \delta\rho_i\right] ,\end{aligned} \tag{20}$$

where

$$Z_1 = \frac{\Delta\xi_{i-1} + \Delta\xi_i}{2\,\Delta\xi_i + \Delta\xi_{i+1}} , \tag{21}$$

$$Z_2 = \frac{\Delta\xi_{i+2} + \Delta\xi_{i+1}}{2\,\Delta\xi_{i+1} + \Delta\xi_i} , \tag{22}$$

$$\Delta X = \Delta\xi_{i-1} + \Delta\xi_i + \Delta\xi_{i+1} + \Delta\xi_{i+2} , \tag{23}$$

and

$$\delta\rho_i = \frac{\Delta\xi_i}{\Delta\xi_{i-1} + \Delta\xi_i + \Delta\xi_{i+1}}\left[\frac{2\,\Delta\xi_{i-1} + \Delta\xi_i}{\Delta\xi_{i+1} + \Delta\xi_i}(\langle \rho \rangle_{i+1}^n - \langle \rho \rangle_i^n) + \frac{\Delta\xi_i + 2\,\Delta\xi_{i+1}}{\Delta\xi_{i-1} + \Delta\xi_i}(\langle \rho \rangle_i^n - \langle \rho \rangle_{i-1}^n)\right] . \tag{24}$$

The above equations retain their fourth-order accuracy even in the presence of nonuniform zoning. The complexity of the interpolation polynomial is due, in large part, to this feature. For the case of uniform zoning, it reduces to the much simpler expression

$$\rho_R = \tfrac{1}{12}\,(-\langle \rho \rangle_{i+2}^n + 7\langle \rho \rangle_{i+1}^n + 7\langle \rho \rangle_i^n - \langle \rho \rangle_{i-1}^n) . \tag{25}$$

As with any high-order polynomial, the results of the interpolation are not guaranteed to be monotonic. In other words, an interface value calculated using the above equations may not fall between the average values in the two neighboring zones. Any nonmonotonic behavior in the interpolation can lead to unphysical oscillations in the solution and must be prevented. Fortunately, this is easily corrected. The quantity $\delta\rho_i$ in equation (24) is the average slope in zone i of the parabola that has zone averages of $\langle \rho \rangle_{i-1}^n$, $\langle \rho \rangle_i^n$, and $\langle \rho \rangle_{i+1}^n$. If this slope is limited in a way similar to the monotonicity criterion used for the slopes in MUSCL (van Leer 1979), the result is guaranteed to be monotonic. The procedure is to replace the quantity $\delta\rho_i$ by

$$\delta_m \rho_i = \begin{cases} \min(|\delta\rho_i|, 2|\langle\rho\rangle_i^n - \langle\rho\rangle_{i-1}^n|, 2|\langle\rho\rangle_i^n - \langle\rho\rangle_{i+1}^n|) \operatorname{sgn}(\delta\rho_i) & \text{if } (\langle\rho\rangle_{i+1}^n - \langle\rho\rangle_i^n)(\langle\rho\rangle_i^n - \langle\rho\rangle_{i-1}^n) > 0 , \\ 0 & \text{otherwise} . \end{cases} \quad (26)$$

After completion of the interpolation subject to the monotonicity constraint, each primitive variable in each zone has associated with it a zone average value and left- and right-interface values. It is now possible to construct a parabola that passes through the two interface values and that also has the correct average value for the zone. The parabolic distribution for the density in zone i can be represented by

$$\rho(\alpha) = \rho_L + \alpha[\Delta\rho_i + \rho_{6i}(1 - \alpha)] , \quad (27)$$

where

$$\alpha = \frac{\xi - \xi_{i-1/2}}{\xi_{i+1/2} - \xi_{i-1/2}} , \qquad 0 \le \alpha \le 1 , \quad (28)$$

$$\Delta\rho_i = \rho_R - \rho_L , \quad (29)$$

and

$$\rho_{6i} = 6[\langle\rho\rangle_i^n - \tfrac{1}{2}(\rho_L + \rho_R)] . \quad (30)$$

A different form of equation (27) that will also be useful later is

$$\rho(\beta) = \rho_R - \beta[\Delta\rho_i - \rho_{6i}(1 - \beta)] , \quad (31)$$

where

$$\beta = \frac{\xi_{i+1/2} - \xi}{\xi_{i+1/2} - \xi_{i-1/2}} , \quad 0 \le \beta \le 1 . \quad (32)$$

The above procedure is used to construct the interface values and parabolic distributions of all the primitive variables. Note that there are no discontinuities in the piecewise parabolic distributions of the flow variables at this point. In other words, ρ_R in zone i is equal to ρ_L in zone $i + 1$, for example.

The interface values and parabolae calculated above are only preliminary. There are a number of situations in which they must be modified in order to obtain acceptable results. The first modification is performed in regions near contact discontinuities. In order to prevent these sharp structures from spreading diffusively as they propagate across the grid, the interface values of density and composition are altered to provide steeper slopes within the relevant zones. Second, because of the self steepening mechanism in shocks and the low dissipation in PPM, shock fronts tend to become too narrow, frequently less than a zone wide. When this situation arises, unphysical oscillations develop behind the shock, and a procedure, know as flattening, is used to provide extra dissipation. The algorithm looks for regions in the flow where the shock width is too small and flattens the parabolic distribution in those zones by taking a linear combination of the interface values obtained from the above interpolation procedure with those that would have been produced using the first-order Godunov reconstruction. The final step is to check for monotonicity of the parabolae within each zone. Even though the interface values have been subjected to a monotonicity constraint, it is still possible for the parabolae to overshoot or undershoot within the zone. If this happens, one or both of the interface values must be modified. All three of these procedures will be discussed below.

3.1.2. *Contact Discontinuity Steepening*

The procedure used by PPM to keep contact discontinuities sharp is perhaps the most controversial feature of the method. Its use requires some care, since under certain circumstances it can produce incorrect results. For example, it is possible for the code to interpret a very steep (but smooth) density gradient as a contact discontinuity. When this happens, the gradient is usually turned into a series of contact discontinuities, producing a stair-step appearance in one-dimensional flows, or a series of parallel contact discontinuities in multidimensional flows. The most likely cause of this error is that the calculation is underresolved in the vicinity of the steep gradient. The algorithm could be disabled in this region, but the results would still be very inaccurate, since there would not be sufficient grid points in the gradient for the code to produce a good representation of the flow. Since it is almost always completely obvious when something has gone wrong, the algorithm can be used as a diagnostic to show regions of the flow where more grid points are required. If a problem is detected, the calculation can be rerun using a finer grid. When this is not possible as a result of limited computer resources, the algorithm can be turned off to produce smoother results, although the user should still be suspicious of their accuracy.

The algorithm that determines if a given zone falls within a contact discontinuity is fairly complex. This is unavoidable if one is to provide accurate criteria to distinguish contact discontinuities from other features in the flow. The code looks for a density profile similar to that shown in Figure 5. A zone is considered to be within a contact discontinuity only if all of the following conditions are met:

1. $\partial^2\rho/\partial\xi^2$ must change sign across the zone.
2. $\partial^3\rho/\partial\xi^3$ must be sufficiently large.
3. $\partial\rho/\partial\xi$ and $\partial^3\rho/\partial\xi^3$ must have opposite signs.
4. The density jump must be sufficiently large.
5. The pressure jump must be sufficiently small.

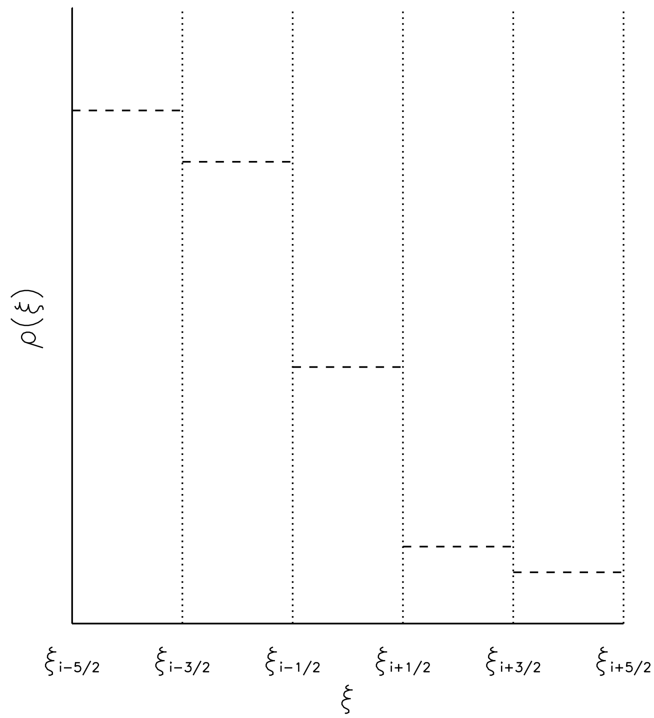

FIG. 5.—Schematic density profile that will trigger the detection of contact discontinuities. The horizontal dashed lines represent the average values of the density in each zone.

If conditions (1)–(3) are not met, the curve has the wrong profile. Condition (4) is included to prevent the algorithm from operating on numerical noise. Finally, condition (5) determines if the zone contains a contact discontinuity or a shock.

The first step in steepening a contact discontinuity in zone i is to construct a linear distribution of the density, for example, in zones $i-1$ and $i+1$ using the monotonized slopes $\delta_m \rho_{i\pm 1}$ as calculated above in equation (26) to monotonize the cubic interpolation polynomials. New left- and right-interface variables in zone i are then calculated from

$$\rho^d_{Li} = \langle \rho \rangle^n_{i-1} + \tfrac{1}{2}\delta_m \rho_{i-1} , \tag{33}$$

$$\rho^d_{Ri} = \langle \rho \rangle^n_{i+1} - \tfrac{1}{2}\delta_m \rho_{i+1} . \tag{34}$$

The next step is to compute finite-difference approximations for the derivatives of the density in order to determine if the above criteria are met. The two quantities that are required, both written for the case of a nonuniform mesh, are the second derivative,

$$\delta^2 \rho_i = \frac{1}{(\Delta\xi_{i-1} + \Delta\xi_i + \Delta\xi_{i+1})}\left(\frac{\langle \rho \rangle^n_{i+1} - \langle \rho \rangle^n_i}{\Delta\xi_{i+1} + \Delta\xi_i} - \frac{\langle \rho \rangle^n_i - \langle \rho \rangle^n_{i-1}}{\Delta\xi_i + \Delta\xi_{i-1}}\right) , \tag{35}$$

and a dimensionless quantity that involves the first and third derivatives,

$$\tilde{\eta}_i = -\left(\frac{\delta^2\rho_{i+1} - \delta^2\rho_{i-1}}{\xi^n_{i+1} - \xi^n_{i-1}}\right)\left[\frac{(\xi^n_i - \xi^n_{i-1})^3 + (\xi^n_{i+1} - \xi^n_i)^3}{\langle \rho \rangle^n_{i+1} - \langle \rho \rangle^n_{i-1}}\right] . \tag{36}$$

The quantity $\tilde{\eta}_i$ is set to 0 in the following situations. Condition (4) is violated if

$$\frac{|\langle \rho \rangle^n_{i+1} - \langle \rho \rangle^n_{i-1}|}{\min\,(\langle \rho \rangle^n_{i+1}, \langle \rho \rangle^n_{i-1})} < 0.01 . \tag{37}$$

In this case, the feature is assumed to be numerical noise and is ignored. The feature has the wrong profile, violating condition (1), if

$$\delta^2\rho_{i-1}\,\delta^2\rho_{i+1} > 0 . \tag{38}$$

The feature is assumed to be a shock instead of a contact discontinuity. Finally, condition (5) is not satisfied if

$$\frac{|\langle P \rangle^n_{i+1} - \langle P \rangle^n_{i-1}|}{\min\,(\langle P \rangle^n_{i+1}, \langle P \rangle^n_{i-1})} > 0.1\gamma\, \frac{|\langle \rho \rangle^n_{i+1} - \langle \rho \rangle^n_{i-1}|}{\min\,(\langle \rho \rangle^n_{i+1}, \langle \rho \rangle^n_{i-1})} . \tag{39}$$

The steepening coefficient η_i is then constructed and constrained to lie between 0 and 1 by

$$\eta_i = \max\,\{0, \min\,[20(\tilde{\eta}_i - 0.05), 1]\} . \tag{40}$$

Finally, the left- and right-interface values of density in zone i are replaced by the values

$$\rho_{Li} \rightarrow \rho_{Li}(1 - \eta_i) + \rho_{Li}^d \eta_i , \tag{41}$$

$$\rho_{Ri} \rightarrow \rho_{Ri}(1 - \eta_i) + \rho_{Ri}^d \eta_i . \tag{42}$$

This procedure should be applied immediately after the interpolation step. The composition variables are treated analogously.

The dimensionless parameters in equations (37)–(40) were determined empirically by performing a large number of calculations for discontinuities with different size jumps, different velocities, etc. There is no physical or mathematical reason why they should have those particular values. However, these values seem to work well under a wide variety of conditions, and any changes should be performed with extreme caution. If the criteria described above erroneously identify a feature as a contact discontinuity, the best approach would be to increase the grid resolution. However, if that is not possible, it would be safer to turn off the steepening entirely for that particular calculation rather than to modify the constants in equations (37)–(40). The steepening procedure is illustrated graphically in Figure 6. The results obtained for the propagation of a simple contact discontinuity both with and without steepening are shown in Figures 7 and 8.

3.1.3. *Flattening*

The next procedure to be applied in the reconstruction step is to check for the presence of shock fronts that are too narrow. Because of the small amount of numerical dissipation in PPM and the self-steepening property of shock waves, the code can produce shocks that are too narrow to be treated accurately. The result is unphysical postshock oscillations that can corrupt the entire solution. To prevent this from happening, additional dissipation must be added to the scheme. However, instead of resorting to the traditional technique of adding an artificial viscosity to the difference equations, another procedure, called flattening, is employed. It is much easier to keep the effects of this dissipation localized to regions near shocks than with artificial viscosity. The method actually measures the number of zones contained within each shock and turns on only for shocks that are too narrow. In this way, it is able to maintain a shock width of one to two zones.

As in the case of contact discontinuity detection, the procedure modifies the interface values for a zone, although in this case, instead of steepening the parabolic distribution within the zone, the effect is to make it flatter. This is accomplished by taking a linear combination of the interface values computed above with those that would have been obtained using a first-order Godunov reconstruction. For the first-order method, which uses a piecewise constant reconstruction, the interface values are just equal to the zone average value. The scheme becomes first-order accurate in regions near shocks. However, since the concept of order applies only to smooth flow, this is not a concern. All shock-capturing schemes are first-order accurate in the vicinity of discontinuities.

The shock width is measured by comparing the pressure gradient taken across two zones with the gradient across four zones. Thus, the shock steepness parameter is defined by

$$S_p = \frac{\langle P \rangle_{i+1}^n - \langle P \rangle_{i-1}^n}{\langle P \rangle_{i+2}^n - \langle P \rangle_{i-2}^n} . \tag{43}$$

In smooth flows,

$$\frac{\langle P \rangle_{i+1}^n - \langle P \rangle_{i-1}^n}{2\,\Delta m_i} \approx \frac{\langle P \rangle_{i+2}^n - \langle P \rangle_{i-2}^n}{4\Delta m_i} , \tag{44}$$

so that $S_p \approx \frac{1}{2}$. Now consider a shock that contains only a single zone, as shown in Figure 9. In this case, $\langle P \rangle_{i+1}^n \approx \langle P \rangle_{i+2}^n$ and $\langle P \rangle_{i-1}^n \approx \langle P \rangle_{i-2}^n$, so that $S_p \approx 1$. Similarly, for a shock that contains two zones (Fig. 10), $S_p \approx \frac{2}{3}$. From experimentation, it was found that shocks that are two zones wide ($S_p \approx \frac{2}{3}$) present no problem in most cases and should not be flattened at all. However, maximum flattening should be applied to shocks that contain only a single zone ($S_p \approx 1$).

A dimensionless number between 0 and 1 is constructed from the steepness parameter according to

$$\tilde{F}_i = \max\,\{0, \min\,[1, 10(S_p - 0.75)]\} . \tag{45}$$

As in the case of contact discontinuity steepening, the constants in equation (45) were determined empirically. $\tilde{F}_i = 0$ for steepness parameters $S_p < 0.75$ and reaches its maximum value of 1 for $S_p > 0.85$.

As with the case of contact discontinuity steepening, it is necessary to apply some additional criteria to ensure that the flattening does not operate in inappropriate regions, i.e., in zones that do not contain shocks. In this case, the two necessary conditions are a significantly large relative pressure jump and a negative velocity divergence. The latter condition ensures that the fluid is being compressed in the relevant region. To ensure proper behavior, $\tilde{F}_i$ is set to 0 if

$$\frac{|\langle P \rangle_{i+1}^n - \langle P \rangle_{i-1}^n|}{\min\,(\langle P \rangle_{i+1}^n, \langle P \rangle_{i-1}^n)} < \frac{1}{3} \tag{46}$$

or if

$$\langle u \rangle_{i+1}^n - \langle u \rangle_{i-1}^n > 0 . \tag{47}$$

For non-Cartesian geometry, the left-hand side of equation (47) should be replaced by the appropriate generalization of the velocity divergence.

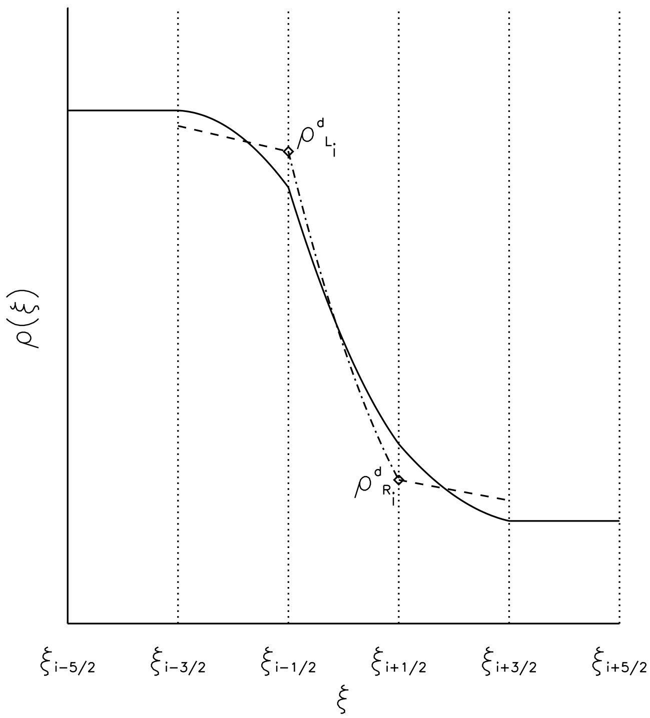

FIG. 6.—Schematic density profile showing the steepening process at a contact discontinuity. The solid line shows the original piecewise parabolic distribution of the density before steepening. The dot-dashed line in zone i shows the new density profile after application of the steepening procedure.

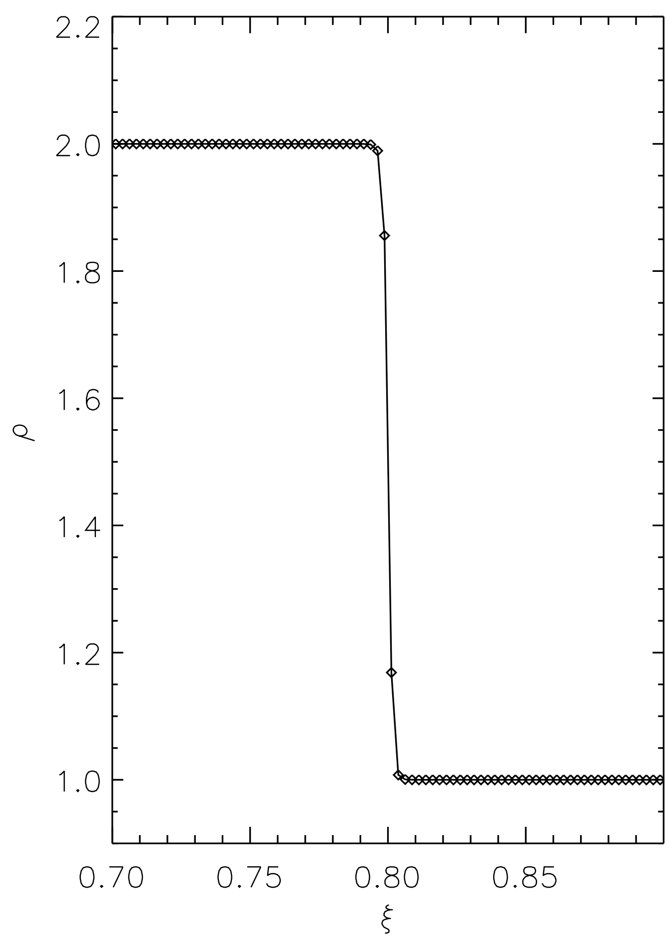

FIG. 7.—Simple contact discontinuity, from a FLASH simulation, with the steepening criteria enabled. Note that there are only two zones defining the discontinuity. At this time, the discontinuity has propagated across 320 zones.

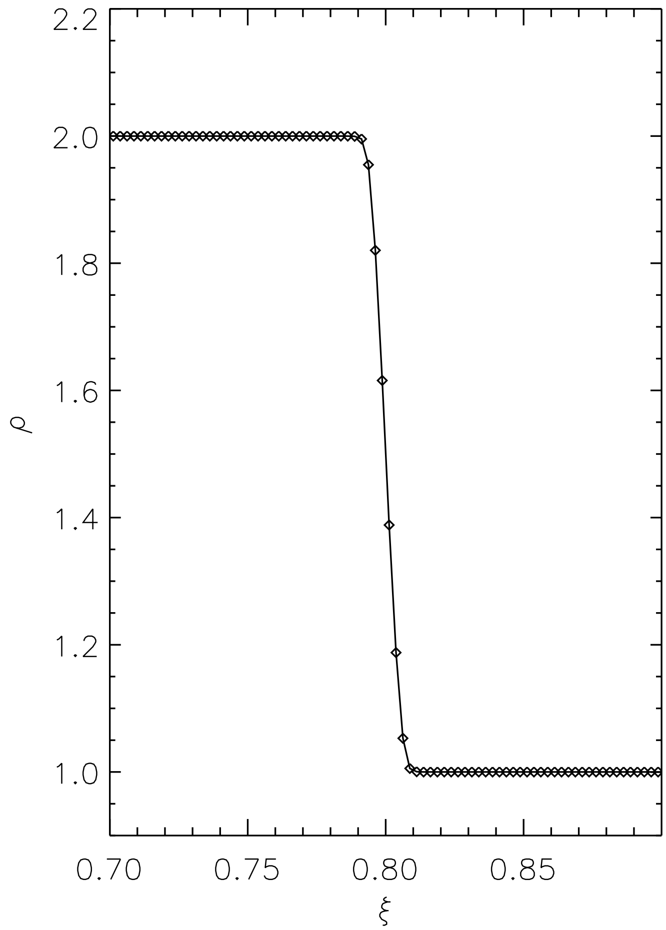

FIG. 8.—Same as Fig. 7, but with the steepening criteria disabled. Note that the discontinuity is now spread over six zones.

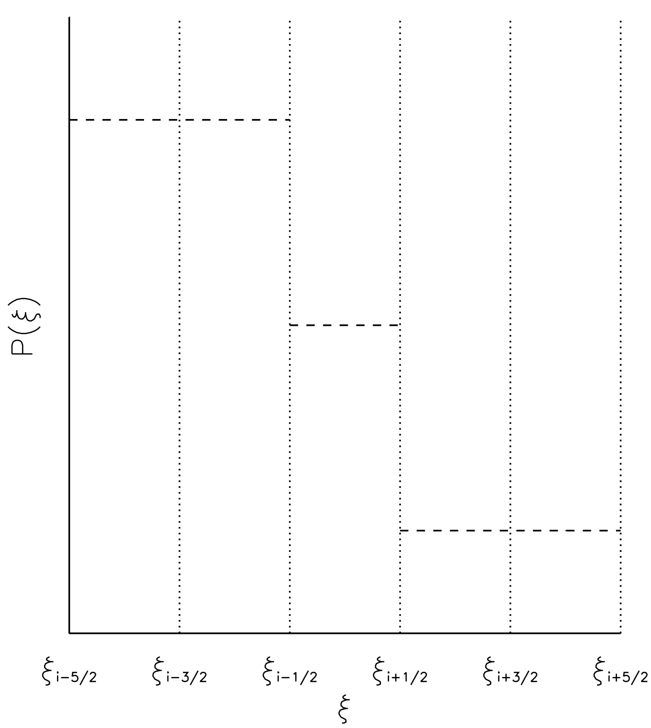

FIG. 9.—Schematic pressure profile showing a shock spread over one zone. The horizontal dashed lines represent the average values of pressure in each zone.

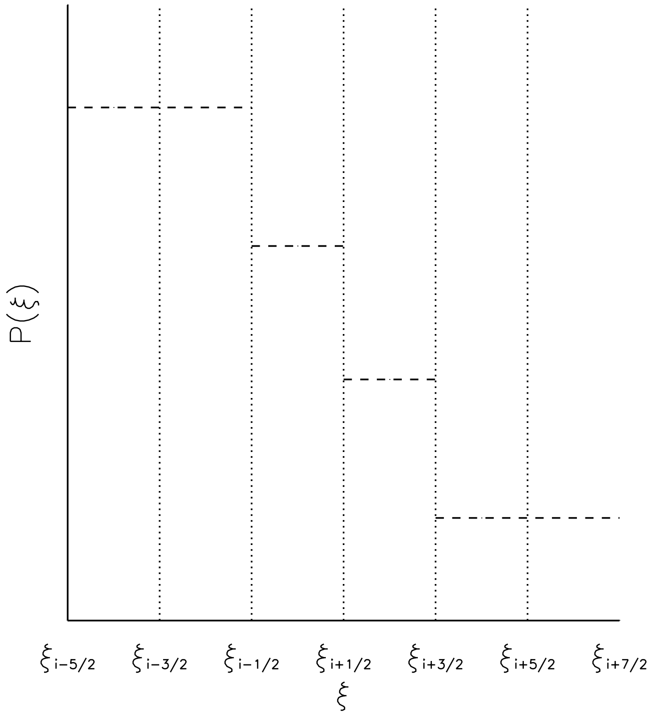

FIG. 10.—Same as Fig. 9, but for a shock spread over two zones

Finally, the flattening coefficient is defined by

$$F_i = \begin{cases} \max(\tilde{F}_i, \tilde{F}_{i+1}) & \text{if } \langle P \rangle_{i+1}^n - \langle P \rangle_{i-1}^n < 0 \,, \\ \max(\tilde{F}_i, \tilde{F}_{i-1}) & \text{otherwise} \,. \end{cases} \tag{48}$$

The interface values in zone i are then modified according to

$$U_{Li} \rightarrow F_i \langle U \rangle_i^n + (1 - F_i) U_{Li} \,, \tag{49}$$

$$U_{Ri} \rightarrow F_i \langle U \rangle_i^n + (1 - F_i) U_{Ri} \,, \tag{50}$$

where U represents any of the primitive variables. If $F_i = 0$, the interface values are unchanged, while if $F_i = 1$, the interface values are set to the zone average values, and the method (in that zone) becomes equivalent to the first-order Godunov method. In fact, for any zone in which $F_i \neq 0$, the method becomes first-order accurate, although, for any zone with $F_i < 1$, there will be less dissipation than for Godunov's method.

The final step in the reconstruction is to ensure that the parabolic distribution of each variable in each zone remains monotonic. Every point on a parabola within a given zone must fall between the two interface values. This requirement can be expressed as

$$\min(U_{Li}, U_{Ri}) < U(\alpha) < \max(U_{Li}, U_{Ri}) \tag{51}$$

for $0 \leq \alpha \leq 1$. Each of the primitive variables must obey equation (51) to prevent severe oscillations from developing in the flow. To prevent this condition from being violated, one or both interface values may need to be modified.

There are two circumstances in which nonmonotonicity might be violated. One is near a local maximum or minimum as shown in Figure 11. The only way to preserve monotonicity in this case is to set both interface values equal to the zone average value. This condition can be expressed as

$$U_{Li} = U_{Ri} = \langle U \rangle_i^n \qquad \text{if } (U_{Ri} - \langle U \rangle_i^n)(\langle U \rangle_i^n - U_{Li}) \leq 0 \,. \tag{52}$$

The other case in which the parabolae may become monotonic is near the top or bottom of a steep gradient (Fig. 12). This situation can be remedied by modifying either the left- or right-interface value according to

$$\begin{aligned} U_{Li} &= 3\langle U \rangle_i^n - 2U_{Ri} && \text{if } (U_{Ri} - U_{Li})[U_{Li} - (3\langle U \rangle_i^n - 2U_{Ri})] < 0 \,, \\ U_{Ri} &= 3\langle U \rangle_i^n - 2U_{Li} && \text{if } (U_{Ri} - U_{Li})[(3\langle U \rangle_i^n - 2U_{Li}) - U_{Ri}] < 0 \,. \end{aligned} \tag{53}$$

This has the effect of setting the slope of the parabola at one interface to 0 and adjusting the value at the opposite interface to maintain the correct average value in the zone. The above procedures introduce discontinuities at zone interfaces. In general, U_{Ri-1} will no longer be equal to U_{Li} at every zone interface. After the final interface values have been obtained, it is necessary to reset the parabolic distributions by recomputing equations (29) and (30).

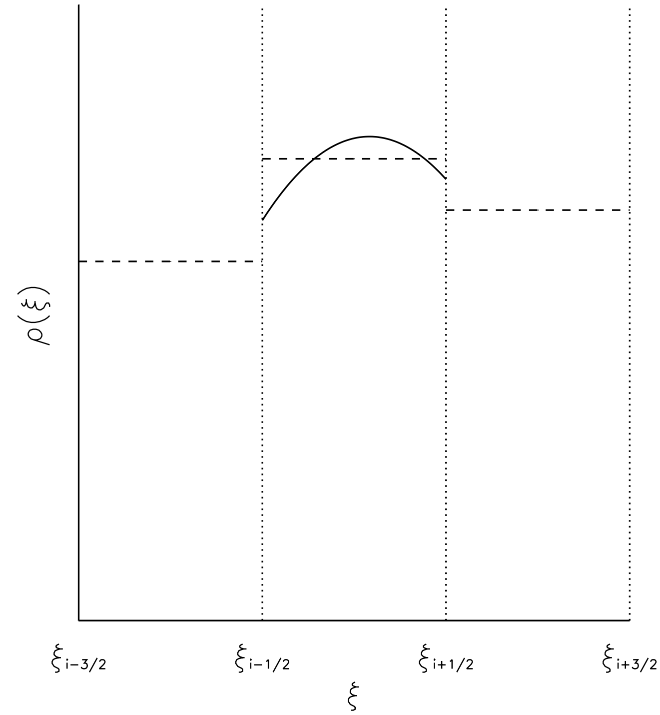

FIG. 11.—Schematic density profile showing how the monotonicity constraint may be violated at a local maximum. The dashed lines represent the zone average values, while the solid line shows the results of the parabolic interpolation. In this case monotonicity is restored by replacing the parabola with the zone average value.

3.1.4. *Solution Step*

Once the reconstruction step has been completed, the solution step can begin. The goal of the solution step is to update the zone average values of the conserved quantities in each zone to the new time step. This is accomplished by solving conservative finite-difference equations. In order to solve these equations, the fluxes of the conserved variables through the zone interfaces must be obtained. The interface values computed above are not suitable for computing the fluxes. First of all, there

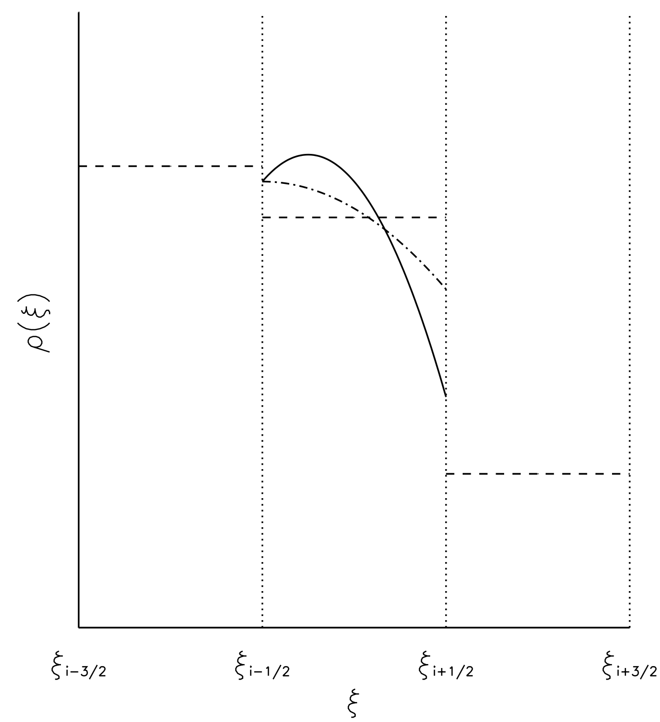

FIG. 12.—Schematic density profile showing how monotonicity may be violated in a steep gradient. The zone average values are indicated by the horizontal dashed lines, and the solid line shows the original parabolic distribution. In order to restore monotonicity, this parabola is replaced by the dot-dashed parabola, which is constructed by keeping the left-interface value fixed. The slope of the parabola at the left interface is set to 0, and the right-interface value is altered in such a way that the average value of the parabola remains unchanged.

may be two values of each variable at each interface. Second, since these values are centered at the old time step, using them would provide forward differencing in time, which is numerically unstable. Instead, the fluxes are computed by solving Riemann's problem at each zone interface.

3.1.5. *Construction of Left and Right States For the Riemann Problem*

Riemann's problem determines the evolution of two constant states that are initially separated by a discontinuity. This is slightly different from the situation here in which the discontinuity at the zone interface separates two states with parabolic distributions. Because of this, it is necessary to construct effective left and right states that will produce a suitable solution to the Riemann problem. Consider the zone interface at $\xi_{i+1/2}$. The simplest choice for the left and right states would be the zone average values in zones i and $i+1$. However, this would give results identical to Godunov's method, and all of the work performed in the reconstruction step described above would have been wasted.

To produce higher order accuracy, a more complicated approach is needed. Because of the hyperbolic nature of the gas dynamics equations, all signals travel with a finite speed. As a result, not all of the material within zones i and $i+1$ can influence the zone interface during the time step (assuming that the CFL number is less than 1, as required for numerical stability). The first step is to determine the domain of dependence of the zone interface during the time step. This is accomplished by tracing characteristics backward in time from the zone interface at time $n+1$ to the ξ axis at time n. Only the material between the point of intersection of the characteristic with the ξ axis and the zone interface can influence the interface during the time step. The amount of material within this region is easily computed by integrating the parabolic distribution.

For the case of Lagrange's equations, this procedure is relatively simple. There are only two characteristics that reach the zone interface from opposite sides (Fig. 13). These characteristics correspond to sound waves traveling to the left and right and are associated with characteristic velocities $\pm c$, where c is the local speed of sound. The left state contains all of the material within the interval $\xi_{i+1/2} - \langle c \rangle_{i-1}^n \Delta t$ to $\xi_{i+1/2}$, while the right state contains the material in the interval $\xi_{i+1/2}$ to $\xi_{i+1/2} + \langle c \rangle_i^n \Delta t$. In this case, the right and left states, denoted by the subscripts r and l, respectively, are obtained by integrating equations (27) and (31) over these intervals. For the right state,

$$U_r = U_{Li+1} + \tfrac{1}{2}\tilde{\alpha}[\Delta U_{i+1} + U_{6i+1}(1 - \tfrac{2}{3}\tilde{\alpha})] , \tag{54}$$

where

$$\tilde{\alpha} = \langle c \rangle_{i+1}^n \frac{\Delta t}{\Delta \xi_{i+1}} . \tag{55}$$

The left state is defined by

$$U_l = U_{Ri} - \tfrac{1}{2}\tilde{\beta}[\Delta U_i - U_{6i}(1 - \tfrac{2}{3}\tilde{\beta})] , \tag{56}$$

where

$$\tilde{\beta} = \langle c \rangle_i^n \frac{\Delta t}{\Delta \xi_i} \tag{57}$$

and $\Delta \xi_i = \xi_{i+1/2} - \xi_{i+1/2}$.

For the Eulerian case, things are considerably more complex. First, there are now three characteristics corresponding to wave speeds u (the entropy wave) and $u \pm c$ (sound waves). Furthermore, each of the characteristics can lie on either side of the zone interface. Thus, the number of characteristics reaching the zone interface from each side can range from zero to three. For supersonic flow to the right (see Fig. 14), all three characteristics reach the zone interface from the left. For supersonic flow to the left, all three characteristics lie to the right of the zone interface. For subsonic flow, there will be one sound wave characteristic on each side, similar to the Lagrangian case. The characteristic associated with the entropy wave could be either on the left or right of the interface, depending on the sign of u (see Fig. 15).

The procedure for the Eulerian case is to compute a first guess at the left and right states by taking the averages under the characteristic on each side that has the largest characteristic speed. If there are no characteristics on one side of the interface, the state is taken to be the zone interface value. These states are still only first-order accurate. To see why these states are not suitable, consider the case of simple advection, in which there are no sound waves present. Using the above states, one assumes that the advected wave travels with a speed of $u \pm c$ rather than the correct speed, which is just u. To achieve higher order accuracy, the above states must be corrected to account for the number of characteristics on each side of the zone interface.

We begin by defining the superscript $\#$, which takes on positive, negative, and zero values corresponding to the three characteristics with characteristic velocities $u+c$, $u-c$, and u, respectively. First, consider the construction of the left state and calculate the amount of material under each of the three characteristics (assuming that they all lie to the left of the interface). If the characteristic speed associated with any of the characteristics is negative, it is set to 0. The three values obtained for the left state are then given by

$$U_l^{\#} = U_{Ri} - \tfrac{1}{2}\tilde{\beta}^{\#}[\Delta U_i - U_{6i}(1 - \tfrac{2}{3}\tilde{\beta}^{\#})] , \tag{58}$$

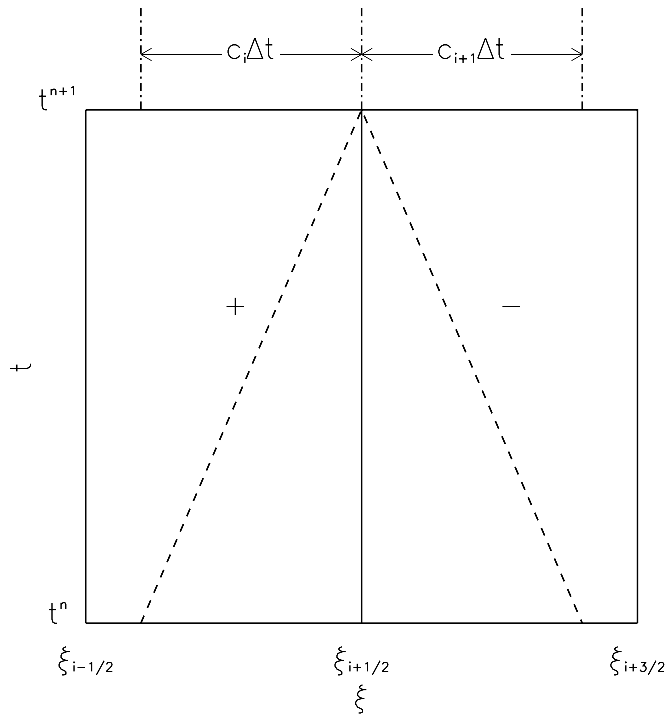

FIG. 13.—Two characteristics in a Lagrangian formulation that reach the zone interface from opposite sides. These characteristics correspond to sound waves traveling to the left and right and are associated with characteristic velocities $\pm c$, where c is the local speed of sound.

with

$$\tilde{\beta}^{\#} = \max\left(\frac{\lambda_i^{\#}\,\Delta t}{\Delta\xi_i}\,,\,0\right), \tag{59}$$

where $\lambda^{\#}$ is the characteristic speed associated with the # characteristic. In calculating $\lambda^{\#}$, the zone average values of u and c are to be used. For the left state, these values are taken from zone i. Similarly, the three average values associated with the

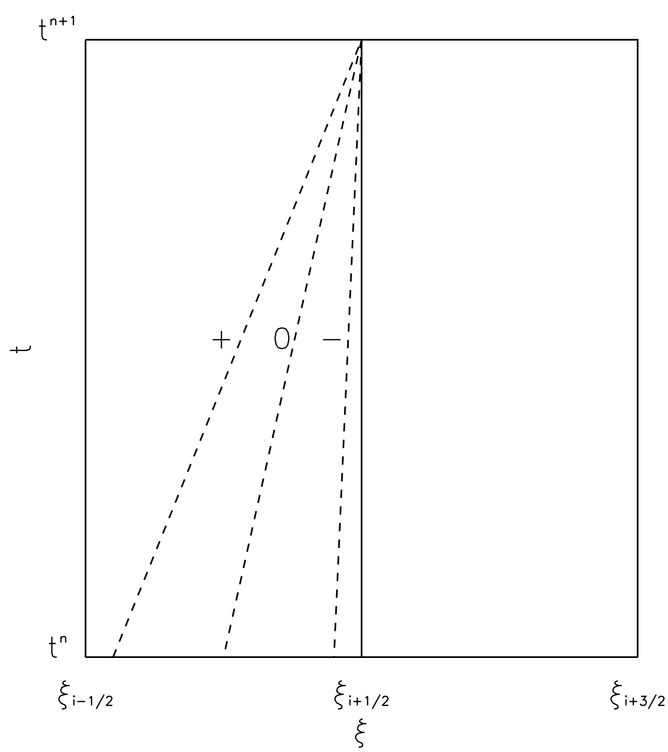

FIG. 14.—Three characteristics in a Eulerian formulation for a supersonic flow propagating to the right. The three characteristics correspond to wave speeds u (the entropy wave, the "0" characteristic) and $u \pm c$ (sound waves, the "+" and "−" characteristics).

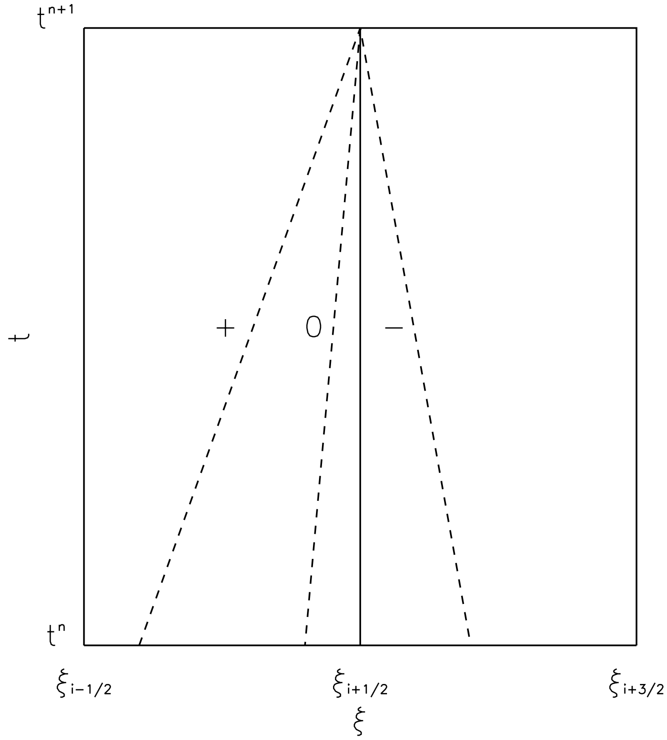

FIG. 15.—Same as Fig. 14, but for a subsonic flow with the fluid moving to the right

characteristics for the right state are

$$U_r^{\#} = U_{Li+1} + \tfrac{1}{2}\tilde{\alpha}^{\#}[\Delta U_{i+1} + U_{6i+1}(1 - \tfrac{2}{3}\tilde{\alpha}^{\#})] , \tag{60}$$

with

$$\tilde{\alpha}^{\#} = \max\left(-\frac{\lambda_{i+1}^{\#}\,\Delta t}{\Delta\xi_{i+1}}, 0\right) . \tag{61}$$

The first guess for the left and right states is just $\tilde{U}_l = U_l^+$ and $\tilde{U}_r = U_r^-$. For the positive and negative characteristics, only the pressure, normal velocity, and density need to be computed. For the zero characteristic, the transverse velocities, mass fractions, and adiabatic indices must also be included.

In order to maintain high-order accuracy, these initial guesses must now be corrected to take into account the number of characteristics on each side of the interface. This involves projecting out of the states defined above the amount of each wave that will not reach the interface during the time step because of the presence of the other waves. The complete derivation is given in Colella & Woodward (1984). Here we present only the final results. We begin by defining the quantities

$$\chi_s^{\pm} = \mp \frac{1}{2\tilde{\rho}_s \tilde{c}_s}\left(\tilde{u}_s - u_s^{\pm} \pm \frac{\tilde{P}_s - P_s^{\pm}}{\tilde{\rho}_s \tilde{c}_s}\right) , \tag{62}$$

$$\chi_s^0 = \frac{\tilde{P}_s - P_s^0}{(\tilde{\rho}_s \tilde{c}_s)^2} + \frac{1}{\tilde{\rho}_s} - \frac{1}{\rho_s^0} , \tag{63}$$

where $s = l, r$. $\chi^{\#}$ is set to 0 if the characteristic velocity has the wrong sign. The condition can be expressed as

$$\begin{aligned} \chi_l^{\#} &= 0 \qquad \text{if } \lambda_i^{\#} \le 0 , \\ \chi_r^{\#} &= 0 \qquad \text{if } \lambda_{i+1}^{\#} \ge 0 . \end{aligned} \tag{64}$$

The quantities $\chi_s^{\#}$ are then used to correct the initial guess for the left and right states according to

$$P_s = \tilde{P}_s + (\tilde{\rho}_s \tilde{c}_s)^2(\chi_s^+ + \chi_s^-) , \tag{65}$$

$$u_s = \tilde{u}_s + \tilde{\rho}_s \tilde{c}_s(\chi_s^+ - \chi_s^-) , \tag{66}$$

$$\rho_s = \left(\frac{1}{\tilde{\rho}_s} - \sum_{\# = +,-,0} \chi_s^{\#}\right)^{-1} . \tag{67}$$

The left and right states for the transverse velocities, mass fractions, and adiabatic indices are taken to be the state obtained from averaging under the zero characteristic.

3.1.6. *Solution to Riemann*'s Problem for Gamma-*Law Gases*

The next step is to solve Riemann's problem at each zone interface using these effective left and right states. The general solution consists of one wave (shock or rarefaction) moving to the left, a second wave traveling to the right, and a contact discontinuity in between. A typical example of Riemann's problem is shown in Figure 16. We make one approximation in constructing the solution, namely, that the two waves propagating to the left and right are both shocks. This approximation, which is accurate to third order in the size of the entropy jump across the wave, will produce a small error at rarefactions. However, rarefactions quickly spread over many zones, so the entropy jump from one zone to the next is usually quite small. This approach leads to a much more efficient solution, since it avoids the need for logic to determine which formulae to use for a given wave and also avoids the use of the rarefaction formulae, which contain fractional powers and are therefore expensive to compute.

We begin by describing the solution to Riemann's problem for the simple case of a gamma-law gas. In the following section we include a description of a method that can be used to treat a general EOS. The following shock jump equations can be derived from the integral form of the gas dynamics equations. For a shock moving to the left,

$$P^* - P_l + W_l(u^* - u_l) = 0 \; . \tag{68}$$

For the wave moving to the right,

$$P^* - P_r - W_r(u^* - u_r) = 0 \; . \tag{69}$$

The superscript * denotes the postshock state, while the subscripts l and r correspond to the preshock states of the two waves. The nonlinear Lagrangian wave speeds W_s are given by

$$W_s = C_s\left[1 + \frac{\gamma+1}{2\gamma}\left(\frac{P^* - P_s}{P_s}\right)\right]^{1/2} , \tag{70}$$

where C_s is the so-called Lagrangian sound speed given by

$$C_s = \rho_s c_s = (\gamma P_s \rho_s)^{1/2} \; . \tag{71}$$

The values of P^* and u^* must be identical for both waves. Any difference in the two values would lead to additional waves being generated. Therefore, equations (68) and (69) represent two nonlinear equations with two unknowns (P^* and u^*), which can be solved by standard numerical techniques.

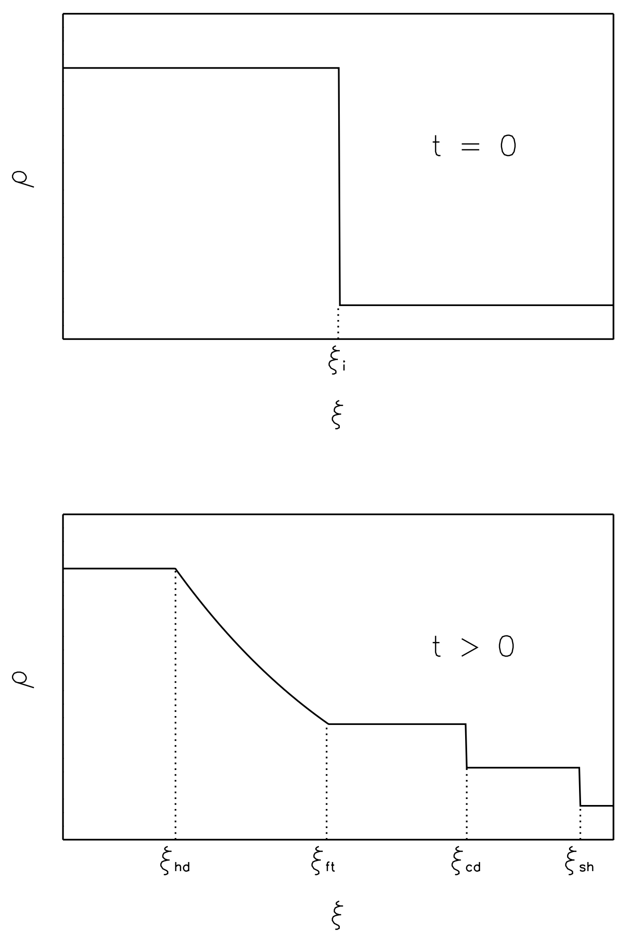

FIG. 16.—Sample Riemann problem starting from shock tube data. *Top:* Initial discontinuity at $t = 0$ located at ξ_i. *Bottom:* Self-similar evolution of the discontinuity for $t > 0$, showing the locations of the head of the rarefaction $\xi_{\rm hd}$, the foot of the rarefaction $\xi_{\rm ft}$, the contact discontinuity $\xi_{\rm cd}$, and the shock $\xi_{\rm sh}$.

The first step is to eliminate u^* from equations (68) and (69), which results in a nonlinear equation for P^*,

$$P^* = P_l + \frac{W_l}{W_l + W_r}\,[P_r - P_l - W_r(u_r - u_l)]\ . \tag{72}$$

The easiest way to solve this equation is to use a simple linear iteration. A first guess can be obtained by assuming that $W_s = C_s$. This results in a linear equation for P^* that can be solved directly. The resulting value of P^* can be used to compute new wave speeds W_s. Equation (72) can then be solved again for P^*. This process is repeated until a converged value for P^* is obtained. For weak shocks, the first guess may give sufficient accuracy. However, very strong shocks may require 10 or more iterations to converge.

Although this process is simple to program, it is not very efficient. A much faster approach uses a Newton-Raphson iteration. The first guess for P^* is the same as for the linear iteration. Subsequent iterations are performed using the formula

$$P^{*(m+1)} = P^{*(m)} - \frac{Q_r Q_l}{Q_r + Q_l}\,[u_r^{*(m)} - u_l^{*(m)}]\ , \tag{73}$$

where m is the iteration number and

$$Q_s = \left|\frac{dP^*}{du^*}\right|_s = \frac{2[W_s^{(m)}]^3}{[W_s^{(m)}]^2 + C_s^2}\ , \tag{74}$$

with

$$u_l^{*(m)} = u_l - \frac{P^{*(m)} - P_l}{W_l^{(m)}}\ , \tag{75}$$

$$u_r^{*(m)} = u_r + \frac{P^{*(m)} - P_r}{W_r^{(m)}}\ , \tag{76}$$

$$W_s^{(m)} = C_s\left[1 + \frac{\gamma+1}{2\gamma}\frac{P^{*(m)} - P_s}{P_s}\right]^{1/2}\ . \tag{77}$$

Although each iteration costs more to compute, many fewer iterations are required for convergence. Reasonably strong shocks can now be handled adequately with two to three iterations. Finally, the value of u^* can be calculated using either equation (68) or equation (69).

The values of P^* and u^* calculated above represent the pressure and fluid velocity at the location of the contact discontinuity. In a Lagrangian calculation, the contact discontinuity remains at the zone interface, and therefore these are the values that are needed to compute the fluxes. However, once again, the Eulerian case is somewhat more complicated. The location in Riemann's problem that lies on the zone interface can now be the left input state, the right input state, the states just to the left or right of the contact discontinuity, or somewhere in the middle of the rarefaction. Which state to use in calculating the fluxes is determined by looking at the speeds of the various waves.

First define

$$\sigma = \mathrm{sgn}\,(u^*) \tag{78}$$

and use this to determine on which side of the contact discontinuity the zone interface lies. The undisturbed state on the correct side is then

$$U_s = \begin{cases} U_l & \text{if } \sigma = 1\ , \\ U_r & \text{if } \sigma = -1\ . \end{cases} \tag{79}$$

The value of ρ^* can then be determined using the shock jump relations according to

$$\rho^* = \left(\rho_s^{-1} - \frac{P^* - P_s}{W_s^2}\right)^{-1}\ . \tag{80}$$

Note that this equation is used even if the wave is a rarefaction.

The next step is to define

$$\lambda_s = \begin{cases} c_s - \sigma u_s & \text{if } P^* < P_s\ , \\ \dfrac{W_s}{\rho_s} - \sigma u_s & \text{otherwise}\ , \end{cases} \tag{81}$$

$$\lambda^* = \begin{cases} c^* - \sigma u^* & \text{if } P^* < P_s\ , \\ \dfrac{W_s}{\rho_s} - \sigma u_s & \text{otherwise}\ . \end{cases} \tag{82}$$

If the zone interface lies within the rarefaction, the rarefaction equations could be integrated to produce the correct values for the fluxes. However, this procedure is rather expensive. A simpler method that provides acceptable accuracy is to perform a linear interpolation between the head and the foot of the rarefaction. Using this procedure, the rarefaction values are given by

$$U_{\rm rare} = \zeta U^* + (1 - \zeta)U_s \,, \tag{83}$$

where

$$\zeta = \frac{1}{2}\left[1 + \frac{\lambda_s + \lambda^*}{\max\,(\lambda_s - \lambda^*,\, \lambda_s + \lambda^*)}\right] . \tag{84}$$

The values to be used for computing the fluxes $\bar{U}$ are then obtained from

$$\bar{U} = \begin{cases} U^* & \text{if } \lambda^* > 0 \,, \\ U_s & \text{if } \lambda_s < 0 \,, \\ U_{\rm rare} & \text{otherwise} \,, \end{cases} \tag{85}$$

where in this case $\bar{U} = (\bar{\rho}, \bar{u}, \bar{P})$. For the passively advected quantities, such as the mass fractions and transverse velocities, $\bar{U} = U_s$.

3.1.7. *Solution to Riemann's* Problem for a General EOS

For the case of a non–gamma-law gas, the method described above for solving Riemann's problem must be generalized to allow for the possibility of different values of γ and Γ for the left and right input states. The procedure used in the FLASH code is based on the method of Colella & Glaz (1985). The iteration to find P^* now uses the secant method instead of Newton's method to allow for the possibility that the appropriate derivatives of the EOS may not be readily available.

The first guess is performed in exactly the same way as described above, assuming that the nonlinear wave speed is equal to the sound speed. For the second guess, the jump in γ across the left and right waves is computed by the approximate equation

$$\gamma_s^* = \gamma_s + 2\left(1 - \frac{\hat{\gamma}}{\hat{\Gamma}}\right)(\hat{\gamma} - 1)\,\frac{P^* - P_s}{P^* + P_s} \,, \tag{86}$$

where

$$\hat{\gamma} = \tfrac{1}{2}(\gamma_l + \gamma_r) \tag{87}$$

and

$$\hat{\Gamma} = \tfrac{1}{2}(\Gamma_l + \Gamma_r) \,. \tag{88}$$

In order to prevent unphysical undershoots and overshoots, the value of γ_s^* should be constrained to fall between the values of $\langle\gamma\rangle$ in the neighboring zones. The values of γ_l^* and γ_r^* represent the states just to the left and right of the contact discontinuity.

Approximate nonlinear wave speeds for the left- and right-moving waves can then be determined from the equation

$$W_s^2 = \rho_s(P^* - P_s)(\gamma_s - 1)\,\frac{P^* + (1/2)(\gamma_s^* - 1)(P^* + P_s)}{(\gamma_s - 1)P^* - (\gamma_s^* - 1)P_s} \,. \tag{89}$$

The second guess for P^* can now be obtained using these approximate wave speeds.

Once the first two guesses have been computed, the secant iteration can begin. This is carried out to convergence using the formula

$$P^{*(m+1)} = P^{*(m)} - [u_R^{*(m)} - u_L^{*(m)}]\left[\frac{|P^{*(m)} - P^{*(m-1)}|}{|u_L^{*(m)} - u_L^{*(m-1)}| + |u_R^{*(m)} - u_R^{*(m-1)}|}\right] , \tag{90}$$

where

$$u_s^{*(m)} = u_s \pm \frac{P^{*(m)} - P_s}{W_s^{(m)}} \,. \tag{91}$$

The remainder of the Riemann solution proceeds exactly as described above for the constant γ case.

3.1.8. *Updating the Conserved Quantities*

The final procedure in the solution step is to update the conserved quantities using centered finite-difference equations. The Riemann problem has the effect of converting the left and right states, which in the above procedure represent spatial averages, into values that represent averages in time over the interval t to $t + \Delta t$. Thus, using these values to evaluate the fluxes automatically guarantees that the difference equations are centered and second-order accurate in time. The resulting difference equations take the form

$$\langle\rho\rangle_i^{n+1} = \langle\rho\rangle_i^n - \frac{\Delta t}{\Delta V_i}\,(A_{i+1/2}\,\bar{\rho}_{i+1/2}\,\bar{u}_{i+1/2} - A_{i-1/2}\,\bar{\rho}_{i-1/2}\,\bar{u}_{i-1/2}) \,, \tag{92}$$

$$\langle\rho\rangle_i^{n+1}\langle u\rangle_i^{n+1} = \langle\rho\rangle_i^n\langle u\rangle_i^n - \frac{\Delta t}{\Delta V_i}(A_{i+1/2}\bar{\rho}_{i+1/2}\bar{u}_{i+1/2}^2 - A_{i-1/2}\bar{\rho}_{i-1/2}\bar{u}_{i-1/2}^2) - \frac{\Delta t}{\Delta\xi}(\bar{P}_{i+1/2} - \bar{P}_{i-1/2}) + \frac{\Delta t}{2}(\langle\rho\rangle_i^n\langle g\rangle_i^n + \langle\rho\rangle_i^{n+1}\langle g\rangle_i^{n+1}) , \quad (93)$$

$$\langle\rho\rangle_i^{n+1}\langle v\rangle_i^{n+1} = \langle\rho\rangle_i^n\langle v\rangle_i^n - \frac{\Delta t}{\Delta V_i}(A_{i+1/2}\bar{\rho}_{i+1/2}\bar{v}_{i+1/2}\bar{u}_{i+1/2} - A_{i-1/2}\bar{\rho}_{i-1/2}\bar{v}_{i-1/2}\bar{u}_{i-1/2}) , \quad (94)$$

$$\langle\rho\rangle_i^{n+1}\langle w\rangle_i^{n+1} = \langle\rho\rangle_i^n\langle w\rangle_i^n - \frac{\Delta t}{\Delta V_i}(A_{i+1/2}\bar{\rho}_{i+1/2}\bar{w}_{i+1/2}\bar{u}_{i+1/2} - A_{i-1/2}\bar{\rho}_{i-1/2}\bar{w}_{i-1/2}\bar{u}_{i-1/2}) , \quad (95)$$

$$\langle\rho\rangle_i^{n+1}\langle E\rangle_i^{n+1} = \langle\rho\rangle_i^n\langle E\rangle_i^n - \frac{\Delta t}{\Delta V_i}[A_{i+1/2}(\bar{\rho}_{i+1/2}\bar{E}_{i+1/2} + \bar{P}_{i+1/2})\bar{u}_{i+1/2} - A_{i-1/2}(\bar{\rho}_{i-1/2}\bar{E}_{i-1/2} + \bar{P}_{i-1/2})\bar{u}_{i-1/2}] + \frac{\Delta t}{2}\left(\langle\rho\rangle_i^n\langle u\rangle_i^n\langle g\rangle_i^n + \langle\rho\rangle_i^{n+1}\langle u\rangle_i^{n+1}\langle g\rangle_i^{n+1}\right) , \quad (96)$$

$$\langle\rho\rangle_i^{n+1}\langle X_\ell\rangle_i^{n+1} = \langle\rho\rangle_i^n\langle X_\ell\rangle_i^n - \frac{\Delta t}{\Delta V_i}(A_{i+1/2}\bar{\rho}_{i+1/2}\bar{X}_{\ell\, i+1/2}\bar{u}_{i+1/2} - A_{i-1/2}\bar{\rho}_{i-1/2}\bar{X}_{\ell\, i-1/2}\bar{u}_{i-1/2}) . \quad (97)$$

In these equations, $A_{i+1/2}$ is the cross-sectional area of the zone interface and ΔV_i is the zone volume. The remaining thermodynamic variables, such as the pressure, temperature, and adiabatic indices at the new time, are obtained from the conserved quantities using the EOS.

One additional modification to the basic PPM scheme is required when performing calculations with more than one species (i.e., $\ell > 1$). Since the difference equations are solved in conservative form, it is guaranteed that the total mass of each species in the computational grid will remain constant (assuming no flow in or out of the boundaries). However, there is no guarantee that equation (10) will remain satisfied. Because of the nonlinear filters used during the reconstruction step, such as contact discontinuity steepening and monotonicity, which act differently on each individual species, the sum of the abundances can frequently differ significantly from 1. This generally happens in the vicinity of discontinuities or at local extrema, which are the only places where these nonlinear filters act.

Fortunately, this error is easy to fix. The modification is to be performed on the values of the mass fractions $\bar{X}_\ell$ computed in the Riemann problem. First, each mass fraction is constrained to lie between 0 and 1. The mass fractions are then normalized by dividing each one by the sum of the mass fractions at that point. It is easy to demonstrate using equation (97) that with the fluxes normalized in this way, the sum of the mass fractions in each zone will not change during the time step. Although it might seem that this procedure will change the hydrodynamic behavior of the system, the only points that are affected are those that lie within a discontinuity. These points are numerical artifacts anyway that result from the numerical diffusion of the scheme. As long as the profile of each mass fraction changes smoothly from one side of the discontinuity to the other, the overall accuracy of the solution will be maintained. Not applying this correction, however, can have disastrous effects, especially for reactive flows.

3.2. *Multidimensional Techniques*

The above description discusses how to solve the gas dynamic equations only for the case of one-dimensional flow. However, these procedures can also be used for the basis of a multidimensional algorithm using a method known as dimensional splitting or Strang splitting. The following discussion will describe only the two-dimensional Cartesian case. Generalization to three dimensions and curvilinear coordinates is straightforward.

Consider Euler's equations, without source terms, written in flux form

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} + \frac{\partial G(U)}{\partial y} = 0 . \quad (98)$$

For dimensional splitting, these equations are solved using the following steps:

$$\begin{aligned}
U_{i,j}^{n+1'} &= U_{i,j}^n - \Delta t\left[\frac{\partial F(U^n)}{\partial x}\right]_{i,j} ,\\
U_{i,j}^{n+1} &= U_{i,j}^{n+1'} - \Delta t\left[\frac{\partial G(U^{n+1'})}{\partial y}\right]_{i,j} ,\\
U_{i,j}^{n+2'} &= U_{i,j}^{n+1} - \Delta t\left[\frac{\partial G(U^{n+1})}{\partial y}\right]_{i,j} ,\\
U_{i,j}^{n+2} &= U_{i,j}^{n+2'} - \Delta t\left[\frac{\partial F(U^{n+2'})}{\partial x}\right]_{i,j} .
\end{aligned} \quad (99)$$

In the above four equations, the value of Δt is to remain fixed. These equations advance the solution two full time steps. Reversing the order for the second time step provides second-order accuracy in time.

For each one-dimensional sweep through the computational grid, we solve equations (11)–(16) with ξ set to the appropriate spatial coordinate for the current sweep direction, u set to the velocity component in the ξ direction, and v and w set to the other two velocity components. For the three-dimensional case, the one-dimensional sweeps are performed in the order $xyz - zyx$. Hydrodynamics is incorporated in the FLASH code by providing a module that solves the above equations on a single block. The algorithm described above uses a nine-point stencil in each dimension, so that four guard cells are required on each side of the block.

A small multidimensional artificial viscosity term is added using the form proposed by Lapidus (1967). This viscosity is not required to stabilize shocks with PPM algorithms but provides a weak coupling between adjacent rows and columns for directionally split schemes. This generally has no noticeable effect on one-dimensional models but reduces stair-casing in shocks orientated diagonally to a rectangular mesh. Typically a coefficient of 0.1 is used.

4. THERMODYNAMICS

Models of stellar events usually require the relationship between various thermodynamic properties over a large span of temperatures, densities, and compositions. With well over 10^9 calls to an EOS being common in two- and three-dimensional hydrodynamic models of stellar phenomena, it is very desirable to have an EOS that is as efficient as possible and yet accurately represents the relevant physics. The reason for the large number of EOS calls is to ensure compatibility of the hydrodynamics and thermodynamics. While FLASH is readily capable of including any EOS, we discuss the salient features of three EOS routines that are supplied with the default FLASH distribution. These three routines are a gamma-law EOS, the Nadyozhin EOS (Nadyozhin 1974; Blinnikov, Dunina-Barkovskaya, & Nadyozhin 1996), and the Helmholtz EOS (Timmes & Swesty 2000).

4.1. *Generalities*

Let isotope i have Z_i protons, A_i nucleons (protons + neutrons), and a binding energy of B_i (in ergs). Let the aggregate total of isotope i have a mass density ρ_i (in g cm^{-3}) and a number density n_i (in cm^{-3}) in a material with a temperature T (in K) and a total mass density ρ (in g cm^{-3}). Define the dimensionless mass fraction of isotope i as $X_i = \rho_i/\rho = n_i A_i/\rho N_{\mathrm{A}}$ and the molar abundance of isotope i as $Y_i = X_i/A_i = n_i/\rho N_{\mathrm{A}}$, where N_{A} is Avogadro's number.

The mean number of nucleons per isotope is defined as $\bar{A} = (\sum X_i/A_i)^{-1}$, the mean charge per isotope as $\bar{Z} = \bar{A} \sum Z_i X_i/A_i$, and the number of electrons per baryon as $Y_e = \bar{Z}/\bar{A}$. Under these conditions, let the material have a total scalar pressure P_{tot} (in ergs cm^{-3}), a total specific internal energy ϵ_{tot} (in ergs g^{-1}), and a total specific entropy S_{tot} (in ergs g^{-1} K^{-1}).

Each EOS routine in FLASH takes as input the temperature T, density ρ, the mean number of nucleons per isotope $\bar{A}$, and the mean charge per isotope $\bar{Z}$. Each EOS routine then returns as output the total pressure (P_{tot}), total specific internal energy (ϵ_{tot}), and total specific entropy (S_{tot}).

Quantities such as the specific heats or adiabatic indices can be determined once the partial derivatives of the pressure and specific internal energy with respect to the density and temperature are known. The adiabatic indices, for example, form vital input for the hydrodynamic algorithms used in FLASH. In cases in which an energy equation instead of a temperature equation is solved (as in FLASH and other conservative hydrodynamic algorithms), the temperature is usually obtained by iteration and the thermodynamic derivatives must be available in order to implement efficient iteration schemes. Confirming that an EOS routine numerically satisfies thermodynamic consistency also demands that the derivatives be available. For these reasons, the EOS routines return the partial derivatives of the pressure, specific internal energy, and entropy with respect to the density and temperature:

$$\left.\frac{\partial P_{\mathrm{tot}}}{\partial T}\right|_\rho \left.\frac{\partial P_{\mathrm{tot}}}{\partial \rho}\right|_T \left.\frac{\partial \epsilon_{\mathrm{tot}}}{\partial T}\right|_\rho \left.\frac{\partial \epsilon_{\mathrm{tot}}}{\partial \rho}\right|_T \left.\frac{\partial S_{\mathrm{tot}}}{\partial T}\right|_\rho \left.\frac{\partial S_{\mathrm{tot}}}{\partial \rho}\right|_T . \tag{100}$$

4.2. *The Gamma-Law EOS*

The gamma-law EOS provided with the FLASH code is a simple ideal gas with a constant adiabatic index γ:

$$P_{\mathrm{tot}} = \frac{N_{\mathrm{A}} k}{\bar{A}} \rho T , \tag{101}$$

$$\epsilon_{\mathrm{tot}} = \frac{1}{(\gamma - 1)} \frac{P_{\mathrm{tot}}}{\rho} , \tag{102}$$

$$S_{\mathrm{tot}} = \frac{(P/\rho + \epsilon)}{T} , \tag{103}$$

where k is Boltzmann's constant. When the assumptions inherent in this EOS permit its use in FLASH, calculation of the thermodynamic state of the material consumes a negligible fraction of the total computational load required to solve a given hydrodynamical problem. The reduced work load is due in part to the fact that it is trivial to obtain the temperature from a given internal energy density with this EOS; no iteration on a nonlinear function is required.

4.3. *Stellar EOS Routines*

More general EOS routines are necessary for realistic models of X-ray bursts, Type Ia supernovae, classical novae, and other stellar phenomena, where the electrons and positrons may have a speed arbitrarily close to the causal limits and an arbitrary degree of degeneracy and where radiation significantly contributes to the thermodynamic state.

The accuracy, thermodynamic consistency, and execution speed of five different EOS routines that are used in modeling stellar events are analyzed by Timmes & Arnett (1999). The EOS routines examined in their survey encompass one that is exact (for the assumptions imposed) in IEEE 64 bit arithmetic and served as the reference point for the comparisons. The other four EOS routines analyzed were one written by Iben (Iben, Fujimoto, & MacDonald 1992), which was designed primarily for evolving models of intermediate- and low-mass stars; one composed by Weaver, Zimmerman, & Woosley (1978), which aims chiefly for evolving models of massive stars; one summarized by Nadyozhin (1974) and explained in detail by Blinnikov et al. (1996); and a table lookup scheme developed by Arnett (1996). The analysis performed in the Timmes & Arnett (1999) survey permitted a complete assessment of these five EOS routines. They suggested that of these five EOS routines, the best balance between accuracy, thermodynamic consistency, and speed is attained with the Nadyozhin EOS. The analytical Nadyozhin EOS is included with the FLASH code and is summarized below.

An electron-positron EOS based on table interpolation of the Helmholtz free energy $F(\rho, T)$ is developed and analyzed in Timmes & Swesty (2000). The interpolation scheme guarantees thermodynamic consistency to the limiting precision of the arithmetic, independent of the chosen interpolating function. The choice of a biquintic Hermite polynomial as the interpolating function results in accurately reproducing the underlying Helmholtz free energy data in the table and yields derivatives of the pressure, specific entropy, and specific internal energy, which are smooth and continuous. In addition, the interpolation in just two dimensions (temperature and density) is valid for any composition characterized by Y_e. Separate planes for each Y_e composition are neither necessary nor desirable, since simple multiplication or division by the composition variables in the appropriate places gives the desired composition scaling. The analysis performed in Timmes & Swesty (2000) suggests that the Helmholtz EOS is about twice as fast as the Nadyozhin EOS, suffers a maximum error of 1 part in 10^5 over the entire ρ-T table (which is about a factor of 100 smaller than the maximum error incurred by the Nadyozhin EOS), and is thermodynamically consistent to the limits of IEEE 64 bit arithmetic everywhere. As such, the tabular Helmholtz EOS is included with the FLASH code.

Both of the stellar EOS routines in FLASH are based on the "physical picture," in which only fundamental species (photons, nuclei, electrons, and positrons) appear. Each EOS returns the scalar pressure, specific internal energy, and specific entropy as a sum over the fundamental species:

$$P_{\rm tot} = P_{\rm rad} + P_{\rm ion} + P_{\rm ele} + P_{\rm pos} , \tag{104}$$

$$\epsilon_{\rm tot} = \epsilon_{\rm rad} + \epsilon_{\rm ion} + \epsilon_{\rm ele} + \epsilon_{\rm pos} , \tag{105}$$

$$S_{\rm tot} = S_{\rm rad} + S_{\rm ion} + S_{\rm ele} + S_{\rm pos} , \tag{106}$$

where the subscripts "rad," "ion," "ele," and "pos" represent the contributions from radiation, nuclei, electrons, and positrons, respectively.

The radiation portion of the Nadyozhin EOS and Helmholtz EOS routines is simple, always a blackbody in local thermodynamic equilibrium:

$$P_{\rm rad} = \frac{aT^4}{3} , \tag{107}$$

$$\epsilon_{\rm rad} = \frac{3P_{\rm rad}}{\rho} , \tag{108}$$

$$S_{\rm rad} = \frac{(P/\rho + \epsilon)}{T} , \tag{109}$$

where a is related to the Stephan-Boltzmann constant $\sigma_{\rm B} = ac/4$ and c is the speed of light. The ion portion of each routine is the ideal gas of equations (101) and (102) with $\gamma = 5/3$, with the Sacker-Tetrode equation providing the specific entropy

$$S_{\rm ion} = \frac{P_{\rm ion}/\rho + \epsilon_{\rm ion}}{T} + \frac{N_{\rm A}\,k}{\bar{A}} \log\left[\frac{\bar{A}^{5/2}}{N_{\rm A}\,\rho}\left(\frac{2\pi kT}{m_p h^2}\right)^{3/2}\right] , \tag{110}$$

where m_p is the proton rest mass and h is Planck's constant.

Both stellar EOS routines in FLASH are based on the formalism of a noninteracting Fermi gas for the electrons and positrons. The number density of free electrons $N_{\rm ele}$ and positrons $N_{\rm pos}$ in this formalism is given by

$$N_{\rm ele} = \frac{8\pi\sqrt{2}}{h^3}\, m_e^3\, c^3 \beta^{3/2} [F_{1/2}(\eta, \beta) + F_{3/2}(\eta, \beta)] , \tag{111}$$

$$N_{\rm pos} = \frac{8\pi\sqrt{2}}{h^3}\, m_e^3\, c^3 \beta^{3/2} [F_{1/2}(-\eta - 2/\beta, \beta) + \beta F_{3/2}(-\eta - 2/\beta, \beta)] , \tag{112}$$

where m_e is the electron rest mass, the relativity parameter β is

$$\beta = kT/(m_e c^2) , \tag{113}$$

the normalized chemical potential energy μ of electrons is

$$\eta = \mu/kT , \tag{114}$$

and $F_k(\eta, \beta)$ is the Fermi-Dirac integral

$$F_k(\eta, \beta) = \int_0^\infty \frac{x^k(1 + 0.5\beta x)^{1/2}\,dx}{\exp(x - \eta) + 1} . \tag{115}$$

The normalized chemical potential η in this formalism has the rest mass energy of the electrons subtracted out. This means that the positron chemical potential must have the rest mass terms appear explicitly, $\eta_{\rm pos} = -\eta - 2/\beta$, as it does in equation (112). Note that β as defined here is the multiplicative inverse of the β used in Chandrasekhar (1939) and Arnett (1996) and of the z in Fowler & Hoyle (1964); this definition avoids the clutter of numerous negative exponents.

For complete ionization, the number density of free electrons in the matter is

$$N_{\rm ele,matter} = \frac{\bar{Z}}{\bar{A}} N_a \rho = \bar{Z} N_{\rm ion} , \tag{116}$$

and charge neutrality requires

$$N_{\rm ele,matter} = N_{\rm ele} - N_{\rm pos} . \tag{117}$$

Solving equation (117) determines the normalized chemical potential η, which was the only unknown. Such a solution fulfills the chemical potential's role as the Lagrange multiplier that was originally introduced to constrain the distribution function to have the correct number of particles. Solving equation (117) in practice means using a one-dimensional root find method to obtain the root η.

Once η is known from the solution of equation (117), the pressure, specific internal energy, and entropy due to the free electrons and positrons are

$$P_{\rm ele} = \frac{16\pi\sqrt{2}}{3h^3} m_e^4 c^5 \beta^{5/2}\left[F_{3/2}(\eta, \beta) + \frac{1}{2\beta F_{5/2}(\eta, \beta)}\right] , \tag{118}$$

$$P_{\rm pos} = \frac{16\pi\sqrt{2}}{3h^3} m_e^4 c^5 \beta^{5/2}\left[F_{3/2}\left(-\eta - \frac{2}{\beta}, \beta\right) + \frac{1}{2\beta F_{5/2}(-\eta - 2/\beta, \beta)}\right] , \tag{119}$$

$$\epsilon_{\rm ele} = \frac{8\pi\sqrt{2}}{\rho h^3} m_e^4 c^5 \beta^{5/2}[F_{3/2}(\eta, \beta) + \beta F_{5/2}(\eta, \beta)] , \tag{120}$$

$$\epsilon_{\rm pos} = \frac{8\pi\sqrt{2}}{\rho h^3} m_e^4 c^5 \beta^{5/2}\left[F_{3/2}\left(-\eta - \frac{2}{\beta}, \beta\right) + \beta F_{5/2}\left(-\eta - \frac{2}{\beta}, \beta\right)\right] + \frac{2m_e c^2 N_{\rm pos}}{\rho} , \tag{121}$$

$$S_{\rm ele} = \left(\frac{P_{\rm ele}}{\rho + \epsilon_{\rm ele} - \eta k T N_a N_{\rm ele}}\right) T^{-1} , \tag{122}$$

$$S_{\rm pos} = \left(\frac{P_{\rm pos}}{\rho + \epsilon_{\rm pos} - \eta k T N_a N_{\rm pos}}\right) T^{-1} . \tag{123}$$

Thus, evaluation of the electron-positron portion of the EOS amounts to performing the root find of equation (117) and evaluating the Fermi-Dirac integrals of equation (115).

4.3.1. *The Nadyozhin EOS*

The Nadyozhin EOS routine uses polynomial or rational functions to evaluate the thermodynamic quantities in equations (118)–(121). The temperature-density plane is decomposed into five regions (see Fig. 12 of Blinnikov et al. 1996), with different expansions or fitting functions applied to each region. The methods used in the five regions are (1) a perfect gas approximation with the first-order corrections for degeneracy, (2) expansions of the half-integer Fermi-Dirac functions, (3) Chandrasekhar's (1939) expansion for a degenerate gas, (4) relativistic asymptotics, and (5) Gaussian quadrature for the thermodynamic quantities. The perturbation expansions, asymptotic relations, and fitting functions for the five regions are combined, with extraordinary care given to making sure that transitions between the regions are continuous, smooth, and thermodynamically consistent. All of the partial derivatives (eq. [100]) are obtained analytically. The Nadyozhin EOS gains elegance and execution efficiency by using new, analytical expressions for a partially relativistic electron-positron gas. According to the Timmes & Arnett (1999) survey, the Nadyozhin EOS is the most accurate, thermodynamically consistent, and efficient of the analytical EOS routines that were examined.

4.3.2. *The Helmholtz EOS*

The Helmholtz EOS routine uses a table lookup of a fundamental thermodynamic potential to evaluate the electron-positron plasma. The first law of thermodynamics

$$d\epsilon = T\,dS + \frac{P}{\rho^2}\,d\rho \tag{124}$$

is an exact differential, which requires that the thermodynamic relations

$$P = \rho^2 \left.\frac{\partial \epsilon}{\partial \rho}\right|_T + T \left.\frac{\partial P}{\partial T}\right|_\rho , \tag{125}$$

$$\left.\frac{\partial \epsilon}{\partial T}\right|_\rho = T \left.\frac{\partial S}{\partial T}\right|_\rho , \tag{126}$$

$$-\left.\frac{\partial S}{\partial \rho}\right|_T = \frac{1}{\rho^2}\left.\frac{\partial P}{\partial T}\right|_\rho \tag{127}$$

be satisfied. An EOS is thermodynamically consistent if all three of these identities are true. Thermodynamic inconsistency may manifest itself in the unphysical buildup (or decay) of the entropy (or temperature) during numerical simulations of what should be an adiabatic flow. Models of events that are sensitive to the entropy (e.g., core-collapse supernovae) may suffer inaccuracies if thermodynamic consistency is significantly violated over a sufficient number of time steps.

When the temperature and density are the natural thermodynamic variables to use, the appropriate thermodynamic potential is the Helmholtz free energy

$$F = \epsilon - TS , \tag{128}$$

$$dF = -S\,dT + \frac{P}{\rho^2}\,d\rho . \tag{129}$$

With the pressure defined as

$$P = \rho^2 \left.\frac{\partial F}{\partial \rho}\right|_T , \tag{130}$$

the first of the Maxwell relations (eq. [125]) is satisfied, as substitution of equation (128) into equation (130) demonstrates. With the entropy defined as

$$S = -\left.\frac{\partial F}{\partial T}\right|_\rho , \tag{131}$$

the second of the Maxwell relations (eq. [132]) is automatically satisfied, as substitution of equation (128) into equation (131) demonstrates. The requirement that the mixed partial derivatives commute,

$$\frac{\partial^2 F}{\partial T\,\partial \rho} = \frac{\partial^2 F}{\partial \rho\,\partial T} , \tag{132}$$

ensures that the third thermodynamic identity (eq. [130]) is satisfied, as substitution of equation (128) into equation (132) shows.

Consider *any* interpolating function for the Helmholtz free energy $F(\rho, T)$ that satisfies equation (132). Thermodynamic consistency is guaranteed as long as equation (130) is used first to evaluate the pressure, equation (131) is used second to evaluate the entropy, and finally equation (128) is used to evaluate the internal energy (Swesty 1996). In fact, this procedure is almost too robust! The interpolated values may be horribly inaccurate, but they will be thermodynamically consistent. Having shown that this evaluation procedure guarantees thermodynamic consistency, we next consider the construction of an interpolating function that retains fidelity to the underlying data.

Given an interpolating function for the Helmholtz free energy, the pressure, entropy, and internal energy are given by derivatives of the interpolating function. The derivatives of the pressure, entropy, and internal energy are in turn given by the second derivatives of the interpolating function. One wants these derivatives of the pressure, entropy, and internal energy to be continuous across the table grid points, not for any thermodynamic reasons but for convergence of the Newton-Raphson iterative schemes that are invariably present in explicit or implicit time integrations of the fluid equations. From these general considerations, the minimum order of an interpolating polynomial that will suffice is a quartic. For the reasons given below, the minimum order of the interpolating polynomial is actually a quintic. Thus, the Helmholtz free energy will be given by a quintic polynomial in both the density and temperature table directions. The pressure, entropy, and internal energy are given by biquartic polynomials, and the derivatives of the pressure, entropy, and internal energy are given by bicubic polynomials.

Suppose one defines a function on the interval (x_i, x_{i+1}) with the following properties:

$$f(x_i) = C_1 \; f(x_{i+1}) = C_2 , \tag{133}$$

$$f'(x_i) = C_3 \; f'(x_{i+1}) = C_4 , \tag{134}$$

$$f''(x_i) = C_5 \; f''(x_{i+1}) = C_6 , \tag{135}$$

where C_i are arbitrary constants and primes denote derivatives. The lowest order polynomial that could satisfy these six conditions is a quintic:

$$f(x) = A + Bx + Cx^2 + Dx^3 + Ex^4 + Fx^5 \; . \tag{136}$$

The conditions of equation (133) determine the coefficients A, B, C, D, E, and F in terms of C_i. The three polynomials multiplying the resultant C_i are the quintic Hermite basis functions (e.g., Davis 1963):

$$\psi_z^0 = -6z^5 + 15z^4 - 10z^3 + 1 \; , \tag{137}$$

$$\psi_z^1 = -3z^5 + 8z^4 - 6z^3 + z \; , \tag{138}$$

$$\psi_z^2 = \tfrac{1}{2}(-z^5 + 3z^4 - 3z^3 + z^2) \; , \tag{139}$$

where

$$z = \frac{x - x_i}{x_{i+1} - x_i} \tag{140}$$

and the interpolating quintic Hermite polynomial is

$$\begin{aligned} H_5(z) = {} & f_i \psi_0(z) + f_{i+1} \psi^0(1 - z) + \frac{\partial f}{\partial x}\Big|_i (x_{i+1} - x_i)\psi_z^1 - \frac{\partial f}{\partial x}\Big|_{i+1} (x_{i+1} - x_i)\psi_{1-z}^1 \\ & + \frac{\partial^2 f}{\partial x^2}\Big|_i (x_{i+1} - x_i)^2 \psi_z^2 + \frac{\partial^2 f}{\partial x^2}\Big|_{i+1} (x_{i+1} - x_i)^2 \psi_{1-z}^2 \; . \end{aligned} \tag{141}$$

The one-dimensional case is extended to two dimensions by interpolating each of the basis functions in the second dimension. The resulting biquintic interpolation function for the density and temperature rectangle bounded by $\rho_i \le \rho < \rho_{i+1}$ and $T_i \le T < T_{i+1}$ is

$$\begin{aligned} H_5(\rho, T) = {} & F^{i,j}\psi_x^0\psi_y^0 + F^{i+1,j}\psi_{1-x}^0\psi_y^0 + F^{i,j+1}\psi_x^0\psi_{1-y}^0 + F^{i+1,j+1}\psi_{1-x}^0\psi_{1-y}^0 + F_T^{i,j}\psi_x^1\psi_y^0 + F_T^{i+1,j}\psi_{1-x}^1\psi_y^0 \\ & + F_T^{i,j+1}\psi_x^1\psi_{1-y}^0 + F_T^{i+1,j+1}\psi_{1-x}^1\psi_{1-y}^0 + F_{TT}^{i,j}\psi_x^2\psi_y^0 + F_{TT}^{i+1,j}\psi_{1-x}^2\psi_y^0 + F_{TT}^{i,j+1}\psi_x^2\psi_{1-y}^0 \\ & + F_{TT}^{i+1,j+1}\psi_{1-x}^2\psi_{1-y}^0 + F_\rho^{i,j}\psi_x^0\psi_y^1 + F_\rho^{i+1,j}\psi_{1-x}^0\psi_y^1 + F_\rho^{i,j+1}\psi_x^0\psi_{1-y}^1 + F_\rho^{i+1,j+1}\psi_{1-x}^0\psi_{1-y}^1 + F_{\rho\rho}^{i,j}\psi_x^0\psi_y^2 \\ & + F_{\rho\rho}^{i+1,j}\psi_{1-x}^0\psi_y^2 + F_{\rho\rho}^{i,j+1}\psi_x^0\psi_{1-y}^2 + F_{\rho\rho}^{i+1,j+1}\psi_{1-x}^0\psi_{1-y}^2 + F_{T\rho}^{i,j}\psi_x^1\psi_y^1 + F_{T\rho}^{i+1,j}\psi_{1-x}^1\psi_y^1 + F_{T\rho}^{i,j+1}\psi_x^1\psi_{1-y}^1 \\ & + F_{T\rho}^{i+1,j+1}\psi_{1-x}^1\psi_{1-y}^1 + F_{TT\rho}^{i,j}\psi_x^2\psi_y^1 + F_{TT\rho}^{i+1,j}\psi_{1-x}^2\psi_y^1 + F_{TT\rho}^{i,j+1}\psi_x^2\psi_{1-y}^1 + F_{TT\rho}^{i+1,j+1}\psi_{1-x}^2\psi_{1-y}^1 \\ & + F_{\rho\rho T}^{i,j}\psi_x^1\psi_y^2 + F_{\rho\rho T}^{i+1,j}\psi_{1-x}^1\psi_y^2 + F_{\rho\rho T}^{i,j+1}\psi_x^1\psi_{1-y}^2 + F_{\rho\rho T}^{i+1,j+1}\psi_{1-x}^1\psi_{1-y}^2 + F_{TT\rho\rho}^{i,j}\psi_x^2\psi_y^2 + F_{TT\rho\rho}^{i+1,j}\psi_{1-x}^2\psi_y^2 \\ & + F_{TT\rho\rho}^{i,j+1}\psi_x^2\psi_{1-y}^2 + F_{TT\rho\rho}^{i+1,j+1}\psi_{1-x}^2\psi_{1-y}^2 \; , \end{aligned} \tag{142}$$

where, in analogy with equation (140),

$$x = \frac{\rho - \rho_i}{\rho_{i+1} - \rho_i} \, , \qquad y = \frac{T - T_j}{T_{i+1} - T_i} \, , \tag{143}$$

and the coefficients are defined by

$$F_T^{lk} = \frac{\partial F}{\partial T}\Big|_{l,k} (T_{j+1} - T_j) \; , \tag{144}$$

$$F_{TT}^{lk} = \frac{\partial^2 F}{\partial T^2}\Big|_{l,k} (T_{j+1} - T_j)^2 \; , \tag{145}$$

$$F_\rho^{lk} = \frac{\partial F}{\partial \rho}\Big|_{l,k} (\rho_{i+1} - \rho_i) \; , \tag{146}$$

$$F_{\rho\rho}^{lk} = \frac{\partial^2 F}{\partial \rho^2}\Big|_{l,k} (\rho_{i+1} - \rho_i)^2 \; , \tag{147}$$

$$F_{T\rho}^{lk} = \frac{\partial^2 F}{\partial T \partial \rho}\Big|_{l,k} (T_{j+1} - T_j)(\rho_{i+1} - \rho_i) \; , \tag{148}$$

$$F^{lk}_{TT\rho} = \frac{\partial^3 F}{\partial T^2 \partial \rho}\bigg|_{i,j} (T_{j+1} - T_j)^2 (\rho_{i+1} - \rho_i) \,, \tag{149}$$

$$F^{lk}_{\rho\rho T} = \frac{\partial^3 F}{\partial \rho^2 \partial T}\bigg|_{i,j} (T_{j+1} - T_j) (\rho_{i+1} - \rho_i)^2 \,, \tag{150}$$

$$F^{lk}_{TT\rho\rho} = \frac{\partial^4 F}{\partial T^2 \partial \rho^2}\bigg|_{i,j} (T_{j+1} - T_j)^2 (\rho_{i+1} - \rho_i)^2 \,. \tag{151}$$

Despite the unwieldy appearance of the 36 terms in equation (142), the repetitive patterns in the structure of the equation allow for a concise evaluation.

To use this biquintic Hermite interpolant for a Helmholtz free energy based EOS, one must tabulate the Helmholtz free energy F and the eight partial derivatives $\partial F/\partial T$, $\partial F/\partial \rho$, $\partial^2 F/\partial T^2$, $\partial^2 F/\partial \rho^2$, $\partial^2 F/\partial T\, \partial \rho$, $\partial^3 F/\partial T^2\, \partial \rho$, $\partial^3 F/\partial \rho^2\, \partial T$, and $\partial^4 F/\partial T^2\, \partial \rho^2$ as a function of density and temperature. In return for this nontrivial investment, the values of the function, first partial derivatives, and second partial derivatives are reproduced exactly at the table grid points. The values of the function and its first and second partial derivatives change continuously as the interpolating point moves from one grid cell to the next. With equation (142) as the interpolating function, the Helmholtz free energy is given by a biquintic polynomial. The pressure, entropy, and internal energy are given by biquartic polynomials, and the derivatives of the pressure, entropy, and internal energy are given by bicubic polynomials. Note that the partial derivatives of the biquintic interpolant are determined by the derivatives of the three basis functions in equations (137)–(139).

Five of the eight partial derivatives needed to use the biquintic interpolant can be formed from the EOS routine that is used to generate the table:

$$F = \epsilon - TS \,, \tag{152}$$

$$\frac{\partial F}{\partial T}\bigg|_{\rho} = -S \,, \tag{153}$$

$$\frac{\partial F}{\partial \rho}\bigg|_{T} = \frac{P}{\rho^2} \,, \tag{154}$$

$$\frac{\partial^2 F}{\partial T^2}\bigg|_{\rho} = -\frac{\partial S}{\partial T}\bigg|_{\rho} \,, \tag{155}$$

$$\frac{\partial^2 F}{\partial \rho^2}\bigg|_{T} = \frac{1}{\rho^2}\frac{\partial P}{\partial \rho}\bigg|_{T} - \frac{2P}{\rho^3} \,, \tag{156}$$

$$\frac{\partial^2 F}{\partial T\, \partial \rho} = -\frac{\partial S}{\partial \rho}\bigg|_{T} = \frac{\partial^2 F}{\partial T\, \partial \rho} = \frac{1}{\rho^2}\frac{\partial P}{\partial T}\bigg|_{\rho} \,. \tag{157}$$

The third partial derivatives ($\partial^3 F/\partial T^2\, \partial \rho$, $\partial^3 F/\partial \rho^2\, \partial T$) and the fourth partial derivative ($\partial^4 F/\partial \rho^2\, \partial T^2$) are rarely available directly from the EOS routine. However, these third and fourth partial derivatives may be obtained from techniques that produce accurate numerical derivatives (e.g., Press et al. 1996). The third- and fourth-order partial derivatives are not necessary to insure that the interpolant and its partial derivatives obtain the proper values at the table grid points or to insure smoothness across cell boundaries. What the third and fourth partial derivatives do insure is that the values of $\partial^2 F/\partial T^2$ and $\partial^2 F/\partial \rho^2$ (both of which contain valuable thermodynamic data) remain well behaved in the middle of a cell. Omission of the three "twist" terms, at temperatures where pair production begins to dominate, for example, can allow the second partial derivatives of the interpolant to exhibit undesirable oscillations as one moves through the center of a cell.

Given the table that lists Helmholtz free energy and eight of its partial derivatives, use of equations (128)–(132) and equations (152)–(157) supplies the thermodynamically consistent interpolated values:

$$P = \rho^2 \frac{\partial F}{\partial \rho} \,, \tag{158}$$

$$\frac{\partial P}{\partial T}\bigg|_{\rho} = \rho^2 \frac{\partial^2 F}{\partial \rho\, \partial T} \,, \tag{159}$$

$$\frac{\partial P}{\partial \rho}\bigg|_{T} = \rho^2 \frac{\partial^2 F}{\partial \rho^2} + 2\rho \frac{\partial F}{\partial \rho} \,, \tag{160}$$

$$S = -\frac{\partial F}{\partial T} \,, \tag{161}$$

$$\frac{\partial S}{\partial T}\bigg|_{\rho} = -\frac{\partial^2 F}{\partial T^2} \,, \tag{162}$$

$$\left.\frac{\partial S}{\partial \rho}\right|_T = -\frac{\partial^2 F}{\partial \rho\, \partial T}\,, \tag{163}$$

$$\epsilon = F + TS\,, \tag{164}$$

$$\left.\frac{\partial E}{\partial T}\right|_\rho = T\,\frac{\partial S}{\partial T}\,, \tag{165}$$

$$\left.\frac{\partial E}{\partial \rho}\right|_T = \frac{\partial F}{\partial \rho} + T\,\frac{\partial S}{\partial \rho}\,. \tag{166}$$

We now have a method that guarantees thermodynamic consistency and a suitable interpolating polynomial. The final task is to construct an accurate electron-positron EOS table. In FLASH, the electron-positron Helmholtz free energy table is constructed with the Timmes EOS, which was designed for maximum accuracy and thermodynamic consistency at the expense of speed (see Timmes & Arnett 1999 for a complete description). Evaluation of the requisite Fermi-Dirac integrals (eq. [115]), along with their partial derivatives, is calculated to machine precision with the efficient quadrature schemes of Aparicio (1998). That is, the Fermi-Dirac integrals and their derivatives are exact in IEEE 64 bit arithmetic (16 significant figures). Newton-Raphson iteration is used to obtain the chemical potential of equation (117) to at least 15 significant figures. All the partial derivatives of the pressure, entropy, and internal energy are formed analytically. Finally, the 1986 recommended values of the fundamental physical constants (Cohen & Taylor 1987) are entered to their published precision.

It is vital to note that the Helmholtz free energy table is constructed only for the electron-positron plasma, two-dimensional $F(\rho, T)$, and made with $\bar{A} = \bar{Z} = 1$ (pure hydrogen). One reason for not including contributions from photons and ions in the table is that these components of the Helmholtz EOS are very simple (eqs. [107]–[109]), and one does not need fancy table lookup schemes to evaluate simple analytical functions. A more important reason for only constructing an electron-positron EOS table with $Y_e = 1$ is that the two-dimensional table is valid for *any* composition. Separate planes for each Y_e are not necessary (or desirable), since simple multiplication by Y_e in the appropriate places gives the desired composition scaling. If photons and ions were included in the table, then this valuable composition independence would be lost, and a three-dimensional table would be necessary.

The table generated from the Timmes EOS stores the electron-positron Helmholtz free energy and the requisite eight partial derivatives to 16 significant figures. The limits of the table were chosen to be $10^{-10} < \rho < 10^{11}$ g cm^{-3} and $10^4 < T < 10^{11}$ K. This range of 21 orders of magnitude in density and 7 orders of magnitude in temperature is large enough to alleviate concerns about exceeding the limits of the table with canonical models of stellar phenomena, except for the interiors of neutron stars. This table of the Helmholtz free energy and eight of its partial derivatives is stored in the data file "helm_table.dat" in the FLASH code. For a grid density of 10 points per decade, there are 211 density points and 71 temperature points in the $\sim$1 MB table provided with the FLASH code. These table increments were chosen to provide a maximum error of about 10^{-5} in the thermodynamic variables (see Timmes & Swesty 2000 for the results of other grid densities).

Several improvements to the Helmholtz EOS routine listed in Appendix A of Timmes & Swesty (2000) are implemented in FLASH. Searches through the helm_table.dat table are avoided by computing the table indices directly from the values of any given (T, ρY_e) pair, i.e., the table is hashed. There are few computationally expensive divisions; all of the repetitive divisions are computed and stored the first time the Helmholtz EOS routine is called. Pipeline mode, which is cache memory efficient and operates on entire temperature, density, and composition arrays, is also used. Memory access to the helm_table.dat table is done only once instead of the 6 times implied by equation (142); this further improves the cache hit rate.

4.3.3. *Accuracy, Consistency, and Performance of the Stellar EOS Routines*

The pressure relative to the (exact) Timmes EOS is shown in Figure 17. The upper panel is for a temperature of 10^8 K, the middle panel for 10^9 K, and the lower panel for 10^{10} K. The y-axis in each panel gives the absolute value of the deviation from the correct answer, while the x-axis gives the mass density. The triangles and squares represent the relative pressure differences of the Nadyozhin and Helmholtz EOS routines, respectively. Note that the temperature and density range of the comparison in Figure 17 is much smaller than the range of validity of each EOS. Typical errors in the pressure (and other thermodynamic variables) made by the Nadyozhin EOS are about 0.01% or less, with the maximum error of about 0.1% located in regions where transitions between different expansions or fitting formulae are used. The typical error made by the Helmholtz EOS is about 1 part in 10^7 with maximum errors of about 1 part in 10^5 located halfway between table grid points. On table grid points, the Helmholtz EOS is exact to machine precision. Periodic structures in the Helmholtz EOS error distribution are due to structure of the interpolating biquintic polynomial.

The locations of the sharp minima made by the Helmholtz EOS in Figure 17 correspond to points that happen to be near zeros of the difference from the exact EOS. Thus, the locations of the minima (there would be more of them) and the amplitudes of minima (they would be deeper) depend on the step size used in making the plots (not the step size used in constructing the table). It is the envelope of the maximum error curves that limits the accuracy, since the accuracy is perfect at the grid points. How the error changes between points of maximum error, which is almost always at half-grid points, indicates the distribution of the errors.

The Helmholtz EOS curve at a temperature of 10^8 decreases at the smaller densities in the upper panel because radiation begins to dominate contributions to the total pressure, and the radiation thermodynamics is identical in the Timmes and Helmholtz EOS routines. A factor that limits the level of agreement with the Nadyozhin EOS in this region is the precision with which the fundamental physical constants are entered and manipulated. In regions where radiation terms dominate the

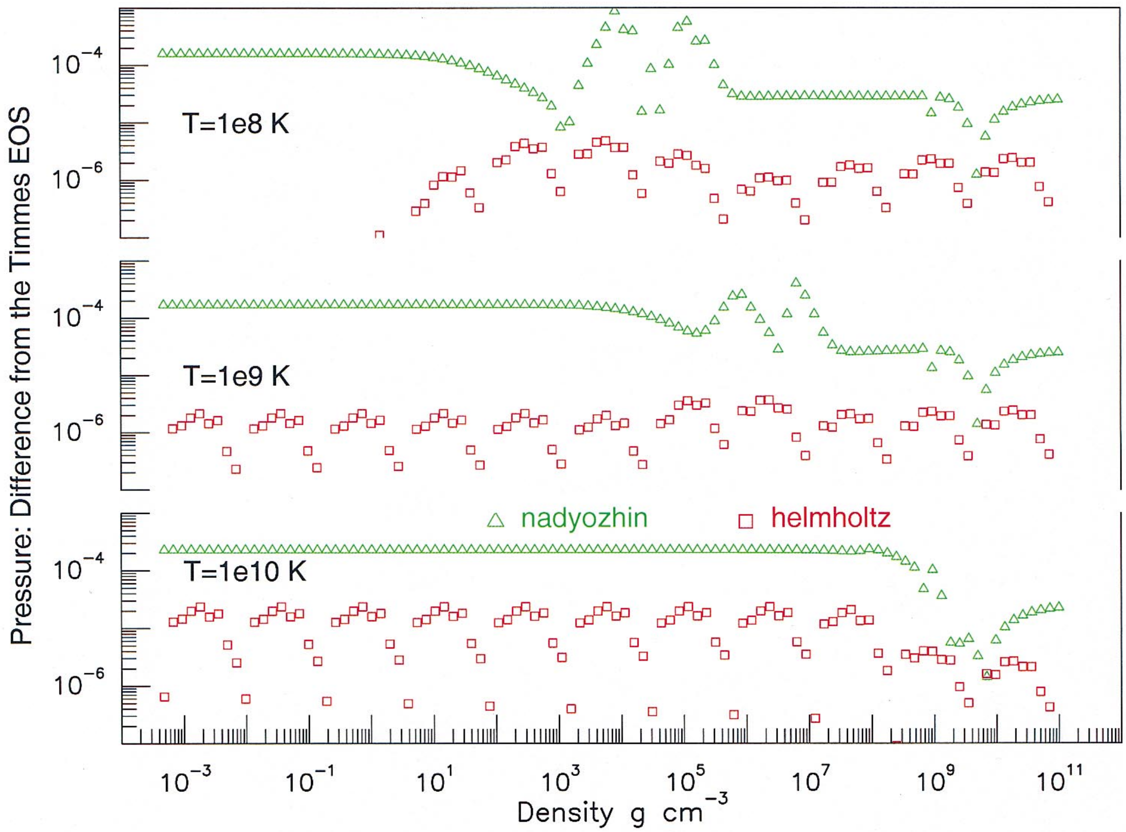

FIG. 17.—Deviation of the pressure given by the Helmholtz and the Nadyozhin EOS from the exact Timmes EOS. The top panel corresponds to a temperature of 10^8 K, middle panel to 10^9 K, and the lower panel to 10^{10} K. The triangles and squares represent the Nadyozhin and the Helmholtz EOS, respectively. The typical error made by the Helmholtz EOS is about 10^{-7} with maximum errors of about 10^{-5} located halfway between table entries. Periodic structures in the Helmholtz EOS error distribution are due to the structure of the interpolating biquintic polynomial.

pressure, values of the constant a (eqs. [107]–[109]) that differ in the fourth significant figure can only result in a relative agreement of 0.01%. As noted above, the Timmes and Helmholtz EOS routines use the 1986 recommended values of the fundamental physical constants to their published precision. The Nadyozhin EOS routine often enters the fundamental physical constants, or various combinations of them, to three to five significant figures. The relative difference in the fundamental constants accounts for some, but not all, of the errors shown in Figure 17. Substantial effort would be required to make all the Nadyozhin equations of state use the same values of the physical constants, and this was not attempted.

Figure 18 shows the deviation that each EOS routine makes in satisfying the second Maxwell relation of equation (125). The format of the plot is the same as for Figure 17. The smaller the deviation, with zero deviation being the perfect case, the closer the EOS comes to satisfying this thermodynamic consistency relation. As asserted above, the Helmholtz EOS satisfies thermodynamic consistency to the limiting precision of IEEE 488 64 bit arithmetic over the entire temperature-density plane under consideration. The consistency of the Timmes EOS is quite good but not perfect at the largest densities because of delicate cancellations that occur in very degenerate material. The Nadyozhin EOS generally satisfies the consistency constraint to a high degree of precision. Abrupt changes in the deviation made by the Nadyozhin EOS are due to traversing the various regions into which the Nadyozhin EOS decomposes the temperature-density plane (see Fig. 12 of Blinnikov et al. 1996).

The execution speeds of the Nadyozhin and Helmholtz EOS routines are shown in Figure 19. The y-axis gives the average number of seconds per EOS evaluation, while the x-axis gives the number of array elements in the pipeline. In pipeline mode each EOS routine operates on entire temperature, density, and composition arrays of size n. Each EOS routine was called 10^6 times in an ordered sweep, with pipeline sizes of $n = 1$, 10, 100, 1000, and 10,000. Figure 19 indicates that the Helmholtz EOS is about twice as fast as the Nadyozhin EOS when the pipeline size is unity and gets relatively more efficient as the pipeline size is made larger. These results are for one Silicon Graphics 250 MHz processor with an R10010 floating point chip, 4 MB data cache, 200 MHz s^{-1} bus, and a compile line of "f90-O2-IPA." The absolute speed of each EOS routine depends, obviously, on the machine architecture and compiler options employed. The relative ratio of each EOS routine's speed, however, is fairly robust with respect to variations in the type of computer and compiler options used.

Figures 17–19 suggest that the two general stellar EOS routines supplied with the FLASH code are reasonably accurate, thermodynamically consistent to a high degree of precision, and fast. A complete analysis of all the relevant thermodynamic variables, consistency relations, and efficiency is described in Timmes & Arnett (1999) and Timmes & Swesty (2000).

In general, we recommend using the Helmholtz EOS with the FLASH code. The Nadyozhin EOS is provided with the FLASH code because it is the best of the analytic routines that were tested and provides an alternative EOS for checking the

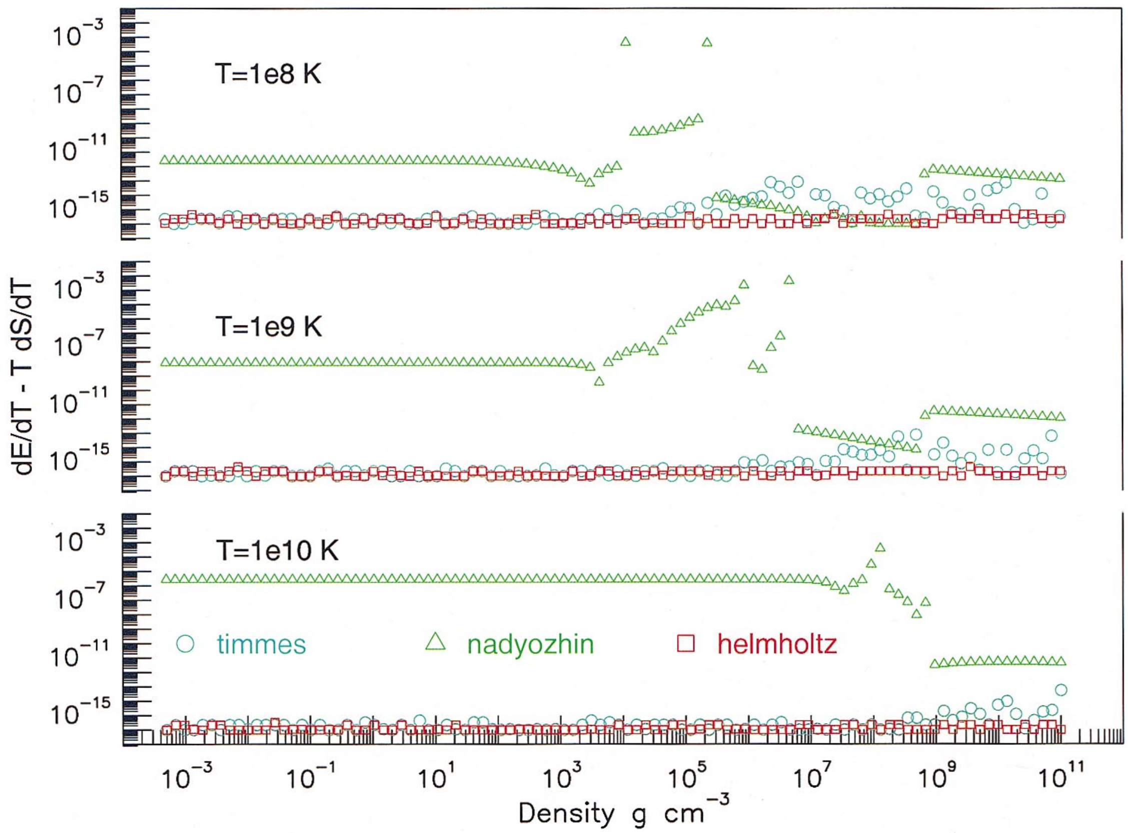

FIG. 18.—Deviation from the thermodynamic relation $(\partial E/\partial T)|_\rho = T(\partial S/\partial T)|_\rho$. The Timmes and Helmholtz EOS routines satisfy this thermodynamic relation, along with the other two relations, to the limiting precision of IEEE 64 bit arithmetic over the entire temperature-density plane.

sensitivity of the model to the EOS. In addition, which physical processes are controlling the thermodynamic state can be gleaned from the analytic Nadyozhin EOS, something that is difficult with the Helmholtz free energy table of numbers.

We address two related points concerning equations of state before moving on to thermonuclear burning in the next section. The first point concerns choices of thermodynamic variables in a hydrodynamics code. When an energy equation instead of a temperature equation is evolved (as in the FLASH code), the temperature must be obtained by iteration. It is our collective experience with the FLASH code that a Newton-Raphson scheme will typically converge to the correct temperature in two to four iterations. The second point is that since a thermodynamic potential can be formulated for any pair of thermodynamic variables, why not create a table with the specific internal energy ϵ_{tot} and density as the entry points and avoid any Newton-Raphson iteration? There are at least three reasons why this idea fails in practice. In several cases, such as the

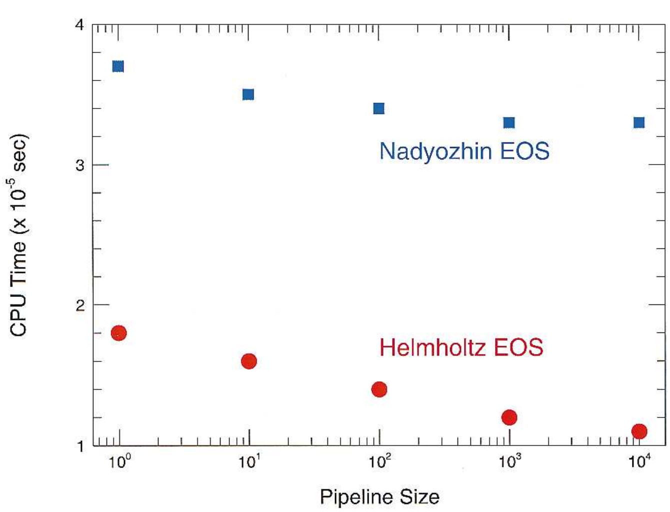

FIG. 19.—Speed of the Nadyozhin (*squares*) and Helmholtz (*circles*) EOS routines. The y-axis gives the average number of CPU seconds per EOS evaluation, while the x-axis gives the number of array elements in the pipeline. The Helmholtz EOS is about twice as fast as the Nadyozhin EOS when the pipeline size is unity and gets relatively more efficient as the pipeline size is made larger. These results are for one particular machine architecture and set of compiler options, but the relative speed ratios are fairly robust with respect to variations in the type of computer and compiler options.

interiors of white dwarfs or the surface layers of neutron stars, the material is degenerate and lacks a vigorous temperature sensitivity. Degeneracy means the table would be nonrectangular, since there would be values of $\epsilon_{\rm tot}$ that are not physical. The proposed table must then be searched for the correct entry points, which loses the speed of a hashed rectangular table. The proposed table would also have to be constructed very carefully in order to ensure the proper temperature sensitivity (recall that the energy generated as a result of nuclear reactions goes as the temperature to a high power, so any error in the temperature may be magnified by nuclear burning processes). Finally, the valuable composition scaling of the electron-positron portion of the EOS is lost, since $\epsilon_{\rm tot}$ is the sum of several pieces (ideal gas, electron-positron plasma, and blackbody radiation). The proposed table would thus have to be three-dimensional, instead of two-dimensional, with separate planes for each composition. It is not very likely that searching a very carefully constructed nonrectangular three-dimensional table will yield a significant speed increase over the present two-dimensional table and root find.

4.3.4. *Correction Terms*

In the preceding sections we compared different implementations of the same basic physics behind an ideal electron-positron liquid. There are several corrections that can change the thermodynamic properties of an ideal electron-positron liquid by a maximum of about 1% in particular regions. These corrections include partial ionization of the ions due to the temperature ($5.0 \times 10^4 \lesssim T \lesssim 10^6$ K; $\rho \lesssim 100$ g cm^{-3}), partial ionization of the ions due to the pressure ($T \lesssim 10^6$ K; $0.01 \lesssim \rho \lesssim 100$ g cm^{-3}), Coulomb interactions of the ionized nuclei with the surrounding electron gas ($T \gtrsim 10^4$ K), polarization of the electrons by the zero-point motions of the ions ($T \gtrsim 10^4$ K), exchange probabilities of the electrons ($T \gtrsim 10^4$ K), correlated statistical fluctuations of the electron number density ($T \gtrsim 10^4$ K), electron capture ($\rho \gtrsim 10^9$ g cm^{-3}), and the onset of ion degeneracy ($\rho \gtrsim 10^{12}$ g cm^{-3}).

At present the stellar EOS routines in FLASH only take into account Coulomb interactions of the ionized nuclei with the surrounding electron gas. This correction is probably the largest of the correction terms in stellar interiors. Future versions of FLASH may take into account some of the other correction terms. Results from Monte Carlo simulations of an idealized one-component plasma in the uniform electron background approximation (Hansen, Torrie, & Vieillefosse 1977; Slattery, Doolen, & DeWitt 1982; Ogata & Ichimaru 1987; Yakovlev & Shalybkov 1989) are adopted. In this approach, the distance between ions a and the dimensionless plasma parameter Γ, which measures the ratio of the electrostatic energy to the thermal energy and is not the same Γ used in the hydrodynamic section, are given by

$$a = \left[\frac{4}{3}\,\pi\,\frac{N_{\rm A}}{\bar{A}}\,\rho\right]^{1/3}, \qquad \Gamma = \frac{\bar{Z}e^2}{kTa}\,. \tag{167}$$

Corrections to specific internal energy, pressure, and entropy are then given by

$$\epsilon_{\rm ub} = 3P_{\rm ub} = \frac{N_{\rm A}\,kT}{\bar{A}}\,f(\Gamma)\,, \qquad S_{\rm ub} = \frac{N_{\rm A}\,kT}{\bar{A}}\,\Gamma^2\,\frac{dg(\Gamma)}{d\Gamma}\,. \tag{168}$$

These (negative) corrections are added to the total specific internal energy, pressure, and entropy of equations (104)–(106). The curve fitting functions $f(\Gamma)$ and $g(\Gamma)$ are given by

$$f(\Gamma) = \begin{cases} a\Gamma + b\Gamma^{1/4} + c\Gamma^{-1/4} + d & \text{if } \Gamma > 1\,, \\ -\dfrac{3^{1/2}}{2}\,\Gamma^{3/2} + \beta\Gamma^{\alpha} & \text{if } \Gamma \le 1\,, \end{cases} \tag{169}$$

$$g(\Gamma) = \begin{cases} a\Gamma + 4b\Gamma^{1/4} - 4c\Gamma^{-1/4} + d\ln\Gamma + e & \text{if } \Gamma > 1\,, \\ -\dfrac{1}{3^{1/2}}\,\Gamma^{3/2} + \dfrac{\beta}{\alpha}\,\Gamma^{\alpha} & \text{if } \Gamma \le 1\,, \end{cases} \tag{170}$$

where the constants are $a = -0.897744$, $b = 0.95043$, $c = 0.18956$, $d = -0.81487$, $e = -2.5820$, $\beta = 0.29341$, and $\alpha = 0.92424$.

At large densities and low temperatures ($\Gamma \sim 100$) these corrections change the thermodynamic quantities of an ideal electron-positron plasma by about 1%, while for low densities and high temperatures ($\Gamma \sim 100$) the corrections are orders of magnitude smaller than the thermodynamic quantities of ideal electron-positron plasma.

4.4. *Thermodynamics and Hydrodynamics Error Considerations*

From the results and discussion above one may get the mistaken impression that one really obtains an absolute accuracy of at least 10 digits when the EOS routines operate with a hydrodynamics algorithm. This is not true.

There are two sources of error in models for the thermodynamic and hydrodynamic behavior of stellar events. Errors are introduced by using approximations to known (but perhaps complicated) physics or by using estimates for incompletely known physics. An example of this type of error committed by the stellar EOS routines in FLASH is the approximation that the sum of the partial pressures equals the total pressure, that is, $P_{\rm tot} = \sum P_i$. Presumably, the global error made in a stellar model that uses this local approximation is quite small. Another example of this type of error are the Coulomb corrections applied to each of the stellar EOS routines. The correction terms for the pressure, internal energy, and entropy can only be calculated to a relatively low precision in the intermediate and strong coupling regimes for multicomponent plasmas. In these regimes, the total accuracy of the stellar EOS routines is about 1 part in 10^3.

Numerical errors are the second source of error in hydrodynamic models of stellar events. Introducing an error by approximating a piece of physics is no excuse for sloppy numerics within that approximation. Our goal for the stellar EOS routines in FLASH is to reduce the numerical error below the formal accuracy of the hydrodynamic algorithm, to have the accuracy of the hydrodynamics be the limiting source of numerical error. For fluid flows with very strong shock features, the absolute accuracy of the hydrodynamics algorithm is about two or three digits. For smooth fluid flows the absolute accuracy is better, perhaps to five or six digits, but highly problem dependent. The efficiency of the stellar EOS routines permits a numerical accuracy that is well below the formal accuracy of the hydrodynamic algorithm, even for smooth fluid flows. By calculating the physics approximations as numerically accurate and efficient as possible, we strive to eliminate the EOS as a potential source of systematic numerical error.

5. THERMONUCLEAR BURNING

Integration of the ordinary differential equations that represent the abundance levels of a set of isotopes serves two functions in models of stellar events. The primary function, as far as the hydrodynamics is concerned, is to provide the magnitude and sign of the thermonuclear energy generation rate. This thermonuclear energy release is usually the largest source or sink of energy in regions conducive to nuclear reactions, which makes its determination vital to accurate hydrodynamic modeling. The second function that the integration serves for the hydrodynamics is to describe the evolution of the abundance levels of the nuclear species. These abundance levels are, of course, fundamental to our understanding of the origin and evolution of the elements.

Obtaining accurate values for the thermonuclear energy generation rate is, unfortunately, expensive in terms of computer memory and CPU time. The largest block of memory in a stellar hydrodynamic code is usually reserved for storing the isotopic abundances at every grid point. This memory requirement can be quite restrictive for three-dimensional models on present parallel computer architectures. Even with modern methods for solving the associated stiff system of ordinary differential equations, evolving the isotopic abundances begins to dominate the total cost of a multidimensional model when the number of isotopes evolved is ~ 30. To decrease the computer time and memory it takes to calculate a model means making a choice between having less isotopes in the reaction network or having less spatial resolution. The general response to this trade-off has been to evolve a limited number of isotopes and thus calculate an approximate thermonuclear energy generation rate. The set of 13 nuclei most commonly used for this purpose is ^{4}He, ^{12}C, ^{16}O, ^{20}Ne, ^{24}Mg, ^{28}Si, ^{32}S, ^{36}Ar, ^{40}Ca, ^{44}Ti, ^{48}Cr, ^{52}Fe, and ^{56}Ni. This minimal set of nuclei, usually called an α-chain network, can reasonably track the abundance levels from helium burning through nuclear statistical equilibrium. More importantly from a hydrodynamics standpoint is that an α-chain reaction network gives a thermonuclear energy generation rate that is generally within $\sim 30\%$ of the energy generation rate given by much larger nuclear reaction networks (Timmes, Hoffman, & Woosley 2000b). In essence, one gets most of the energy generated for most thermodynamic conditions at a fraction of the computational cost (memory + CPU). In addition, with over 10^9 calls to the thermonuclear burning modules required for two- and three-dimensional hydrodynamic models of stellar phenomena, it is desirable to have the solution of the stiff ordinary differential equations be as efficient as possible.

Methods for solving the stiff system of ordinary differential equations that constitute nuclear reaction networks are surveyed by Timmes (1999). The scaling properties and behavior of three semi-implicit time integration algorithms (a traditional first-order accurate Euler method, a fourth-order accurate Kaps-Rentrop method, and a variable-order Bader-Deuflhard method) and eight linear algebra packages (LAPACK, LUDCMP, LEQS, GIFT, MA28, UMFPACK, Y12M, and BiCG) were investigated by running each of these 24 combinations on seven different nuclear reaction networks (a hand-coded 13 isotope network, a hand-coded 19 isotope network, and 47 isotope, 76 isotope, 127 isotope, 200 isotope, and 489 isotope reaction networks). The analysis of Timmes (1999) permitted a full assessment of the integration methods and linear algebra packages surveyed and permitted several pragmatic suggestions to be offered. When the best balance between accuracy, overall efficiency, memory footprint, and ease of use is desirable, Timmes (1999) suggested that the variable-order Bader-Deuflhard time integration method coupled with the MA28 sparse matrix package is a good choice.

While FLASH is readily capable of including any nuclear reaction, network time integration algorithm, or linear algebra package, we discuss the salient features of one thermonuclear burning module that is supplied with the FLASH code. These modules seamlessly incorporate a 13 isotope reaction network, the variable-order Bader-Deuflhard time integration method, and the MA28 sparse matrix package.

5.1. *Nuclear Reaction Networks*

Using the definitions for the mass fractions given in the preceding section, conservation of the nuclear species is expressed by

$$\sum_{i=1}^{N} X_i = 1 \ . \tag{171}$$

This expression should be checked at every time step in a numerical simulation of a stellar event. Failure to satisfy equation (171) to the limiting precision of the arithmetic may manifest itself in the unphysical buildup (or decay) of the abundances or energy generation rate. Models of events that are sensitive to the abundance levels of trace species (e.g., classical novae envelopes) may suffer inaccuracies if mass conservation is significantly violated, even during a single time step. The FLASH code incorporates several refinements in the hydrodynamic modules to ensure that equation (171) is satisfied to the limiting precision of the arithmetic being used.

The most general continuity equation for isotope i in a Lagrangian formulation is

$$\frac{dY_i}{dt} + \nabla \cdot (Y_i V_i) = \dot{R}_i \ . \tag{172}$$

In this partial differential equation, $\dot{R}_i$ is the total reaction rate due to all binary reactions of the form $i(j, k)l$

$$\dot{R}_i = \sum_{j,k} Y_l Y_k \lambda_{kj}(l) - Y_i Y_j \lambda_{jk}(i) , \tag{173}$$

where λ_{kj} and λ_{jk} are the reverse (creation) and forward (destruction) nuclear reaction rates, respectively. Contributions from three-body reactions, such as the triple-α reaction, are easy to append to equation (173). The mass diffusion velocities $\boldsymbol{V}_i$ in equation (172) are obtained from the solution of a multicomponent diffusion equation (Chapman & Cowling 1970; Burgers 1969; Williams 1988) and reflect the fact that mass diffusion processes arise from pressure, temperature, and/or abundance gradients as well as external gravitational or electrical forces.

Consider the case in which the mass diffusion velocities in equation (172) are set to 0. There are two reasons for considering this case. The first reason is that the physics of the situation may dictate that mass diffusion processes are small over the timescales of interest when compared to other transport processes such as thermal diffusion or viscous diffusion (i.e., large Lewis numbers and/or small Prandtl numbers). Such a situation applies, for example, to the study of laminar flame fronts propagating through the quiescent interior of a white dwarf. The second reason for considering this case is that one may wish to apply the numerical technique of operator splitting. In operator splitting, the various pieces of the physics are decoupled from one another over a given time step. Each piece is solved for independently of the others, with the various pieces being appropriately summed at the end of the time step. This approach is quite common in spherically symmetric stellar evolution codes and in multidimensional hydrodynamic codes such as FLASH. In FLASH, for example, the properties of the adiabatic fluid flow are determined for a given time step. The energy generated or changes to the composition due to thermonuclear reactions over the time step are then determined. These two pieces are then summed. Contributions from the diffusion of mass may also be added to the sum. Potential disadvantages of the operator splitting technique are (1) that the point at which decoupling becomes a poor approximation is difficult to estimate a priori and (2) that the ordering of the operators may not be commutative. For either reason, because the physics allows it or because operator splitting is being used, setting the mass diffusion velocity to 0 transforms equation (172) into

$$\frac{dY_i}{dt} = \dot{R}_i \ . \tag{174}$$

This set of ordinary differential equations may be written in the more compact and standard form

$$\dot{\boldsymbol{Y}} = \boldsymbol{f}(\boldsymbol{Y}) \ . \tag{175}$$

The set of ordinary differential equations in equation (174) or equation (175) is what constitutes a "reaction network." In the absence of coupling to neighboring regions through mass diffusion or advection processes, any changes to a composition are due to local processes.

The coefficients that appear in equation (175) span many orders of magnitude because the nuclear reaction rates have highly nonlinear dependences on the temperature and because the abundance levels themselves typically range over several orders of magnitude (e.g., the solar isotopic abundances). As a result, the differential equations that comprise a nuclear reaction network are "stiff." From a mathematical point of view, a set of equations is stiff when the ratio of the maximum to the minimum eigenvalue of the Jacobian matrix $\tilde{\boldsymbol{J}} = \partial \boldsymbol{f} / \partial \boldsymbol{Y}$ is large and imaginary. From a physics point of view, stiff means that at least one of the isotopic abundances is changing on a much faster timescale than the abundance level of another isotope. From a practical point of view, stiff generally means that an implicit or semi-implicit time integration is necessary, hence constructing the Jacobian matrix $\tilde{\boldsymbol{J}}$ is necessary. Solving a stiff set of differential equations with implicit methods is always more involved than solving a nonstiff set of differential equations with explicit methods.

It is instructive at this point to look at an example of how equation (174) and the associated Jacobian matrix are formed. Consider the ^{12}C$(\alpha, \gamma)^{16}$O reaction, which competes with the triple-α reaction during helium burning in stars. The rate at which this reaction proceeds, call it R, is critical for evolutionary models of massive stars, since it determines how much of the core is carbon and how much of the core is oxygen after the initial helium fuel is exhausted. This reaction sequence contributes to the right-hand side of equation (174) through the terms

$$\dot{Y}(^4\mathrm{He}) = -Y(^4\mathrm{He})Y(^{12}\mathrm{C})R + \cdots , \tag{176}$$

$$\dot{Y}(^{12}\mathrm{C}) = -Y(^4\mathrm{He})Y(^{12}\mathrm{C})R + \cdots , \tag{177}$$

$$\dot{Y}(^{16}\mathrm{O}) = +Y(^4\mathrm{He})Y(^{12}\mathrm{C})R + \cdots , \tag{178}$$

where the ellipses indicate additional terms coming from other reaction sequences. The minus signs indicate that helium and carbon are being destroyed, and the positive sign indicates that oxygen is being created. Each of the three expressions in equation (176) contributes two terms to the Jacobian matrix $\tilde{\boldsymbol{J}} = \partial \boldsymbol{f} / \partial \boldsymbol{Y}$:

$$J(^4\mathrm{He}, {}^4\mathrm{He}) = -Y(^{12}\mathrm{C})R + \cdots , \qquad J(^4\mathrm{He}, {}^{12}\mathrm{C}) = -Y(^4\mathrm{He})R + \cdots , \tag{179}$$

$$J(^{12}\mathrm{C}, {}^4\mathrm{He}) = -Y(^{12}\mathrm{C})R + \cdots , \qquad J(^{12}\mathrm{C}, {}^{12}\mathrm{C}) = -Y(^4\mathrm{He})R + \cdots , \tag{180}$$

$$J(^{16}\mathrm{O}, {}^4\mathrm{He}) = +Y(^{12}\mathrm{C})R + \cdots , \qquad J(^{16}\mathrm{O}, {}^{12}\mathrm{C}) = +Y(^4\mathrm{He})R + \cdots . \tag{181}$$

Entries in the Jacobian matrix represent the flow, in number of nuclei s^{-1}, into (positive) or out of (negative) an isotope. All of the temperature and the density dependences are included in the reaction rate R. The Jacobian matrices that arise from nuclear reaction networks are not positive-definite nor symmetric since the forward and reverse reaction rates are generally not equal. While the nonzero pattern of the Jacobian matrix does not change with time, the magnitude of the matrix entries changes as either the abundances, temperature, or density change with time.

5.2. *Reaction Networks Supplied with FLASH*

FLASH presently includes four nuclear reaction networks: a seven-isotope α-chain plus silicon burning reaction network; a 13 isotope α-chain plus heavy-ion reaction network; a 19 isotope that has the same α-chain and heavy-ion reactions as the 13 isotope reaction network, plus additional isotopes to accommodate some types of hydrogen burning (PP and CNO chains), along with some aspects of the photodisintegration of ^{56}N into ^{54}Fe; and a 17 isotope hydrogen burning network. In the future we anticipate adding a flexible, user-defined reaction network.

The 13 isotope α-chain plus heavy-ion reaction network is suitable for most multidimensional simulations of stellar phenomena where having a reasonably accurate energy generation rate is of primary concern (Timmes et al. 2000b). A definition of what we mean by an α-chain reaction network is prudent. A strict α-chain reaction network is only composed of (α, γ) and (γ, α) links among the 13 isotopes ^{4}He, ^{12}C, ^{16}O, ^{20}Ne, ^{24}Mg, ^{28}Si, ^{32}S, ^{36}Ar, ^{40}Ca, ^{44}Ti, ^{48}Cr, ^{52}Fe, and ^{56}Ni. It is essential, however, to include $(\alpha, p)(p, \gamma)$ and $(\gamma, p)(p, \alpha)$ links in order to obtain reasonably accurate energy generation rates and abundance levels when the temperature exceeds $\sim 2.5 \times 10^9$ K. At these elevated temperatures, the flows through the (α, p) (p, γ) sequences are faster than the flows through the (α, γ) channels. An $(\alpha, p)(p, \gamma)$ sequence is, effectively, an (α, γ) reaction through an intermediate isotope. In our α-chain reaction network, we include eight $(\alpha, p)(p, \gamma)$ sequences plus the corresponding inverse sequences through the intermediate isotopes ^{27}Al, ^{31}P, ^{35}Cl, ^{39}K, ^{43}Sc, ^{47}V, ^{51}Mn, and ^{55}Co by assuming steady state proton flows. This strategy permits inclusion of $(\alpha, p)(p, \gamma)$ sequences without explicitly evolving the proton or intermediate isotope abundances. Thus, the α-chain reaction network in FLASH includes not just (α, γ) and (γ, α) links but also links through the $(\alpha, p)(p, \gamma)$ and $(\gamma, p)(p, \alpha)$ sequences.

The Jacobian matrix for the 13 isotope α-chain network during the beginning of a vigorous helium burning episode is shown in Figure 20. Nonzero entries in the matrix are color-coded according to the magnitude of the flow rate into (*red tones*) or out of (*green-blue tones*) an isotope. Matrix elements that are zero are colored white. The matrix is organized and oriented so that the $J(1, 1)$ entry corresponds to $J(^4\text{He}, ^4\text{He})$ in the upper left corner, as indicated by the row and column labels. The number of reaction rates in the network, along with the percentage of the matrix that is zero, is shown to the left of the Jacobian matrix. This shows that the Jacobian matrices of nuclear reaction networks are sparse (a large fraction of zero

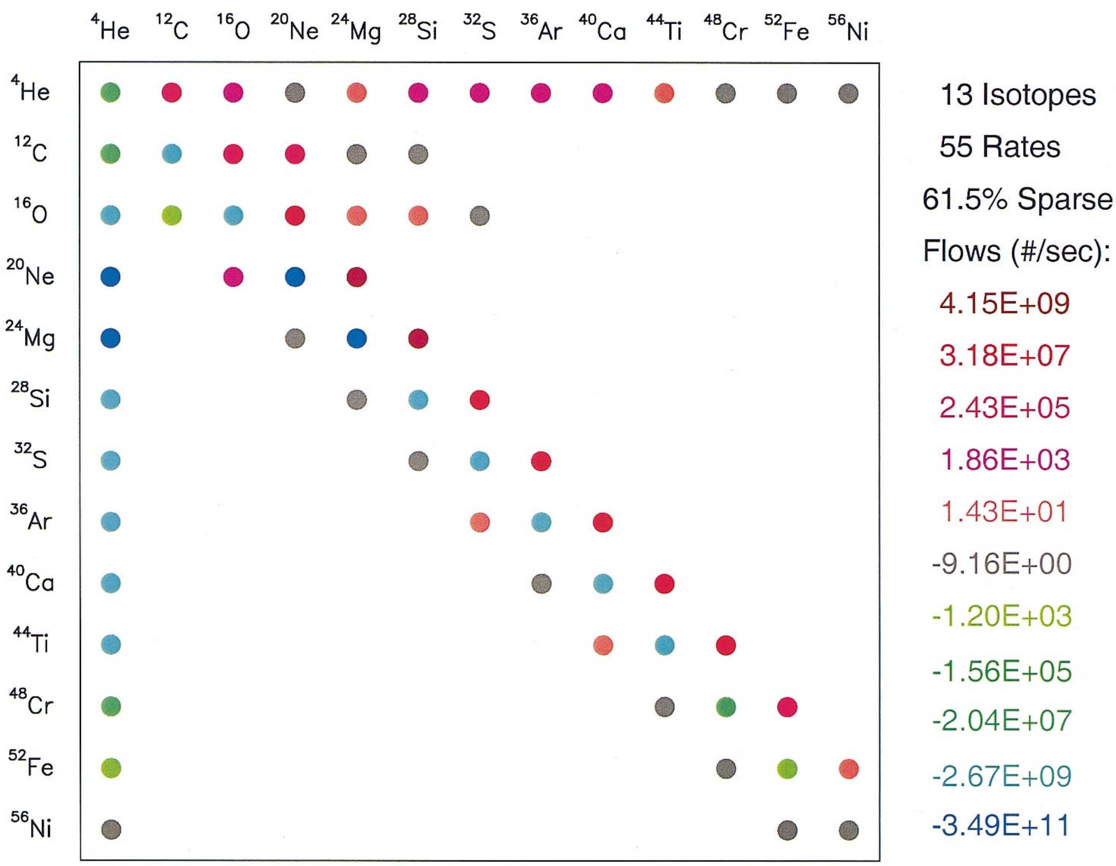

FIG. 20.—Jacobian matrix for the 13 isotope reaction network at a time shortly after the onset of hydrostatic helium burning. Labels on the top and left axes give the names of the isotopes in the reaction network, and the matrix is oriented so that the (1, 1) entry corresponds to (^{4}He, ^{4}He) in the upper left corner. Nonzero entries in the matrix are color-coded according to the magnitude of the flow rate into (*red tones*) or out of (*blue tones*) an isotope. The nonzero pattern of the matrix does not change with time, but the magnitude of the matrix entries changes as either the abundances, temperature, or density change in time.

matrix elements) and become more sparse as the number of isotopes in a reaction network is increased. It makes little sense either to store all those zeros in memory or to perform arithmetic on them during a calculation.

The seven-isotope α-chain plus silicon burning reaction network (Timmes et al. 2000b) is suitable for parameter space surveys of multidimensional stellar models. This reaction network, minimal as it is, reproduces the nuclear energy generation rate of our 13 isotope α-chain reaction network to within 20% during hydrostatic and explosive helium, carbon, and oxygen burning (Timmes et al. 2000a). Depending on the thermodynamic conditions, this reaction network also gives nuclear energy generation rates that are within 30% of the 13 isotope reaction network during hydrostatic and explosive silicon burning. This relative accuracy, combined with a significant reduction in the computational cost, suggests that the seven-isotope reaction network is suitable for most multidimensional simulations of stellar phenomena where having a reasonably accurate energy generation rate is desirable.

The 19 isotope reaction network is described in Weaver et al. (1978). This network includes ^{1}H, ^{3}He, ^{4}He, ^{12}C, ^{14}N, ^{16}O, ^{20}Ne, ^{24}Mg, ^{28}Si, ^{32}S, ^{36}Ar, ^{40}Ca, ^{44}Ti, ^{48}Cr, ^{52}Fe, ^{54}Fe, ^{56}Ni, protons from photodisintegration, and neutrons from photodisintegration. This 19 isotope reaction network is adequate for accurately estimating the energy generation rate from the CNO cycles, even though the network does not explicitly contain the isotopes through which the weak reactions proceed. Like the way the flows through the $(\alpha, p)(p, \gamma)$ sequences are handled, the weak reactions' pathways are assumed to be in steady state with their neighbors. This is often an excellent approximation.

The 17 isotope hydrogen burning reaction network includes ^{1}H, ^{2}H, ^{3}He, ^{7}Li, ^{7}Be, ^{8}B, ^{12}C, ^{13}C, ^{13}N, ^{14}N, ^{15}N, ^{15}O, ^{16}O, ^{17}O, ^{17}F, ^{18}F, and ^{18}F. This reaction network is suitable for problems that require all three proton-proton chains and all four CNO cycles, such as the beginning of a classical nova runaway.

All of the reaction networks described above are "hardwired" reaction networks, where each of the reaction sequences is carefully entered by hand (e.g., the system of equations given by eq. [174]). This hardwired approach is suitable for small networks when minimizing the CPU time required to run the reaction network is a primary concern. A disadvantage of hardwired reaction networks is their lack of flexibility. In the future, the addition of a general "softwired" reaction network to FLASH will permit an arbitrary reaction network to be specified at run time. The cost of this great flexibility is that the softwired reaction network will execute slightly slower than an its hardwired equivalent.

5.3. *Solution of Reaction Networks*

There are two methods for solving sparse linear systems of equations: direct and iterative. Direct methods typically obtain the solution to $\tilde{\boldsymbol{A}} \cdot \boldsymbol{x} = \boldsymbol{b}$ by lower-upper (LU) decomposition or Gaussian elimination and back substitution. Direct methods obtain the solution to a set of linear equations in a finite, well-determined number of operations. Iterative (or "matrix-free") methods seek to minimize a function whose gradient is typically $\tilde{\boldsymbol{A}} \cdot \boldsymbol{x} - \boldsymbol{b}$ and equal to zero when the function is minimized. These methods are attractive because they tend to require only matrix-times-vector operations and usually have smaller storage requirements than direct methods. However, the number of iterations required to converge to a solution is not known a priori, and the iteration count generally increases with the number of unknowns. The total number of iterations, and hence the overall speed, depends crucially on the initial guess and on the stringency of the convergence criteria. Based on the Timmes (1999) survey, the FLASH code has concentrated on direct methods.

Direct methods for sparse matrices typically divide the solution of $\tilde{\boldsymbol{A}} \cdot \boldsymbol{x} = \boldsymbol{b}$ into symbolic and numerical LU decomposition phases and a back-substitution phase. In the symbolic LU decomposition phase, the pivot order of a matrix is determined, and a sequence of decomposition operations that minimize the amount of fill-in is recorded. Fill-in refers to zero matrix elements that become nonzero (e.g., a sparse matrix times a sparse matrix is generally a denser matrix). The matrix is not (usually) decomposed during the symbolic decomposition stage. Only the steps to decompose the matrix are stored. Since the nonzero pattern of a chosen nuclear reaction network does not change, the symbolic LU decomposition is a one-time initialization cost for reaction networks. In the numerical LU decomposition phase, a matrix with the same pivot order and nonzero pattern as the symbolically factorized matrix is numerically decomposed into its LU form. This phase must be done only once for each staged set of linear equations. In the back-substitution phase, a set of linear equations is solved with the factors calculated from a previous numerical decomposition. The back-substitution phase may be performed with as many right-hand sides as needed, and not all of the right-hand sides need to be known in advance.

The MA28 linear algebra package that is used in the FLASH code is described by Duff, Erisman, & Reid (1986). It uses a combination of strategies (nested dissection, frontal envelope decomposition, approximate degree updates) to minimize fill-in during the LU factorization stage. One continuous real parameter sets the amount of searching done to locate the pivot element. When this parameter is set to 0, no searching is done and the diagonal element is the pivot. When set to unity, partial pivoting is done. Since the matrices generated by reaction networks are usually diagonally dominant, the routine is set in FLASH to use the diagonal as the pivot element. Several test cases showed that using partial pivoting did not make a significant accuracy difference and was less efficient, since a search for an appropriate pivot element had to be performed. MA28 accepts the nonzero entries of the matrix in the $(i, j, a_{i,j})$ coordinate system and typically uses 70%–90% less storage than storing the full dense matrix.

The time integration method used in FLASH for evolving the reaction networks in equation (175) is the variable-order Bader-Deuflhard method (e.g., Bader & Deuflhard 1983; Press et al. 1996). The system of stiff ordinary differential equations is advanced over a large time step H from $\boldsymbol{Y}_n$ to $\boldsymbol{Y}_{n+1}$ by the following sequence of matrix equations. First,

$$h = H/m \, , \tag{182}$$

$$(\tilde{1} - \tilde{\boldsymbol{J}}) \cdot \Delta_0 = h\boldsymbol{f}(\boldsymbol{Y}_n) \, , \tag{183}$$

$$\boldsymbol{Y}_1 = \boldsymbol{Y}_n + \Delta_0 \, . \tag{184}$$

Then, from $k = 1, 2, \ldots, m-1$,

$$(\tilde{1} - \tilde{\boldsymbol{J}}) \cdot \boldsymbol{x} = h\boldsymbol{f}(\boldsymbol{Y}_k) - \Delta_{k-1} , \tag{185}$$

$$\Delta_k = \Delta_{k-1} + 2\boldsymbol{x} , \tag{186}$$

$$\boldsymbol{Y}_{k+1} = \boldsymbol{Y}_k + \Delta_k , \tag{187}$$

and closure is obtained by the last stage,

$$(\tilde{1} - \tilde{\boldsymbol{J}}) \cdot \Delta_m = h[\boldsymbol{f}(\boldsymbol{Y}_m) - \Delta_{m-1}] , \tag{188}$$

$$\boldsymbol{Y}_{n+1} = \boldsymbol{Y}_m + \Delta_m . \tag{189}$$

This staged sequence of matrix equations is executed at least twice with $m = 2$ and $m = 6$, which yields a fifth-order method at minimum. The sequence may be executed a maximum of 7 times, which yields a fifteenth-order method. The exact number of times the staged sequence is executed depends on the accuracy requirements (set to 1 part in 10^6 in FLASH) and the smoothness of the solution. Estimates of the accuracy of an integration step are made by comparing the solutions derived from different orders. The minimum cost of this method (which applies for a single time step that met or exceeded the specified integration accuracy) is one Jacobian evaluation, eight evaluations of the right-hand side, two matrix decompositions, and 10 back substitutions. This minimum cost increases at a rate of one decomposition (the expensive part) and m back substitutions (the inexpensive part) for every increase in the order $2k + 1$. The cost of increasing the order is compensated for by taking a correspondingly larger (but accurate) time step. The controls for order versus step size are a built-in part of the Bader-Deuflhard method. The cost per step of this integration method is at least twice as large as the cost per step of either a traditional first-order accurate Euler method or a fourth-order accurate Kaps-Rentrop (essentially an implicit Runge-Kutta) method. However, if the Bader-Deuflhard method can take accurate time steps that are at least twice as large, then this method will be more efficient globally. Timmes (1999) shows that this is typically the case. Note that not all of the right-hand sides are known in advance in equations (182)–(188), since the sequence of linear equations is staged. This staging feature of the integration method will make some matrix packages, such as MA28, a more efficient choice.

The suggestions of the Bader-Deuflhard time integration method and the MA28 sparse matrix package are enthusiastically endorsed, whether the nuclear reaction networks are run in stand-alone mode, in stellar evolution codes, in postprocessing the thermodynamic histories of stellar models, or in multidimensional hydrodynamic codes such as FLASH. In particular, by invoking a full ordinary differential equation integrator inside of FLASH, one is assured of the numerical accuracy (set to 1 part in 10^6 by default) of the reaction network integrations. Any explicit subcycling of the reaction networks with the hydrodynamics is avoided since the integrators will automatically "subcycle" to integrate over the hydrodynamic time step with the requested accuracy.

The instantaneous energy generated or lost is given by the sum

$$\dot{\epsilon}_{\rm nuc} = N_{\rm A} \sum_i B_i \frac{dY_i}{dt} . \tag{190}$$

A nuclear reaction network does not need to be evolved in order to obtain the instantaneous energy generation, since only the right-hand sides of the ordinary differential equations need to be evaluated. It is more appropriate in the FLASH code to use the average nuclear energy generated over a time step

$$\dot{\epsilon}_{\rm nuc} = N_{\rm A} \sum_i B_i \frac{\Delta Y_i}{\Delta t} . \tag{191}$$

In this case, a nuclear reaction network does need to be evolved. The energy generation rate from equation (191), after subtracting any neutrino losses, is returned to the FLASH code for use in the hydrodynamics.

Finally, the tabulation of Caughlan & Fowler (1988) is used in FLASH for most of the strong and electromagnetic nuclear reaction rates. Modern values for some of these reaction rates were taken from the reaction rate library of Hoffman (1999, private communication). Nuclear reaction rate screening effects as implemented by Wallace, Woosley, & Weaver (1982) and decreases in the energy generation rate due to neutrino losses as given by Itoh et al. (1996) are used during the thermonuclear burning calculations within the FLASH code.

5.4. *Nuclear Burning and Hydrodynamics Error Considerations*

Physical and numerical errors are unavoidable in modeling thermonuclear reactive flows. Physical errors are introduced by using approximations to known pieces of physics, or by using estimates for incompletely known physics. An example of this type of error committed by the reaction networks in FLASH are the approximations used for the thermonuclear reaction rates. Many key nuclear reaction rates are not known experimentally to within a factor of 2 [e.g., ^{12}C(α, γ)^{16}O] and sometimes to within a factor of 10. How large an error is made in a stellar model that uses these imprecisely known reaction rates is highly problem dependent. Another example of a physical error made by the small reaction networks in FLASH is that they are all approximations to using a large and complete reaction network. As noted above, a carefully designed α-chain and heavy-ion reaction network can usually reproduce the energy generation rate of a large nuclear reaction network to within $\sim$30% (Timmes et al. 2000b). For thermodynamic conditions where Y_e remains near 0.5 the accuracy is much better than $\sim$30%, but in regimes where Y_e differs from 0.5 the accuracy is significantly reduced.

Numerical errors are introduced by the algorithms and methods employed in a calculation. Our goal for the reaction networks in FLASH is to reduce the numerical integration error below the formal accuracy of the hydrodynamic algorithm, to have the accuracy of the hydrodynamics be the limiting source of numerical error. For fluid flows with very strong shock features, the absolute accuracy of the hydrodynamics algorithm is about two or three digits. For smooth fluid flows the absolute accuracy is better, perhaps to five or six digits, but highly problem dependent. The efficiency of the integration routines permits an integration accuracy that is well below the formal accuracy of the hydrodynamic algorithm, even for smooth fluid flows. By calculating the approximate reaction networks as accurately as possible, we strive to eliminate the integration as a potential source of systematic numerical error.

6. FLASH CODE ARCHITECTURE

In the previous sections we have described in detail the algorithms used in the FLASH code: parallel adaptive mesh refinement with PARAMESH, compressible hydrodynamics with PROMETHEUS, a stellar EOS, and nuclear burning. In this section we discuss the organization and implementation of these algorithms into a modular, extensible simulation code.

Astrophysical phenomena typically involve several different types of physics, each of which is best handled in simulations with different types of numerical methods. Hydrodynamics, nuclear reactions, radiation, gravity, and magnetic fields all pose complex problems for the computational astrophysicist. Developing, testing, and maintaining state-of-the-art codes to implement each type of physics often require the expertise of several individuals. Combining such codes into an integrated code capable of reliably solving interesting problems, while maintaining extensibility, accuracy, and performance, requires the adoption of some type of flexible computational framework, sometimes referred to as a code architecture.

Code architecture is a current research field in computer science, and no "ultimate" method for stitching together codes has yet been (or may ever be) identified. In designing FLASH, therefore, we have been driven less by theory and more by the availability of tools (the modules comprising FLASH) and the need to implement an integrated code that immediately addresses our astrophysics needs while providing a development path toward increasing flexibility. FLASH thus serves two purposes: it is a practical tool for addressing astrophysics problems, and it is an application test bed for advanced code architecture ideas.

Theory is, of course, unavoidable. We can identify in advance several features of a code architecture that are desirable as seen from the astrophysicist's point of view. Because code development is expensive (and usually not the physicist's main interest), it is important to be able to take modules that may have been developed independently and make them work together without requiring extensive rewriting. Some potential applications cannot be anticipated when the code is first written, and often new algorithms are developed later to solve problems more efficiently or accurately. For these reasons a modular code design is to be preferred over a monolithic design. A modular design treats the component solvers as separate units that communicate with each other and the controlling program through interfaces, a collection of subroutines that translate the internal data structures of the components into a common set of data structures (and vice versa). A monolithic design, by contrast, tightly couples its components, making changes difficult.

Performance is another important issue for astrophysicists. Rather than raw floating-point calculation rates, this means, for a fixed wall-clock time (typically days to weeks), the largest, most complex calculation that can be performed. This is because virtually all astrophysically relevant calculations are significantly underresolved at present (and will be for the foreseeable future). The largest calculation, therefore, represents the best we can do for a given problem at any given time. In order to satisfy this definition of performance, it is important that the interfaces mentioned above not significantly degrade the performance of the components. To the extent that interfaces and modularity always degrade performance somewhat, this implies a trade-off between extensibility and performance. Generally we will want to strike a balance somewhere in the middle: a totally flexible, high-level architecture usually performs poorly, while a fully in-lined monolithic code can often achieve very good performance but is very difficult to maintain. Neither extreme is desirable.

Finally, it is important to be able to change initial and boundary conditions readily and to vary the parameters that control a simulation. In addition to influencing a code's versatility (and hence longevity), this criterion affects the ease with which a code's behavior can be verified against standard test problems. Verification is as critical to establishing the scientific credibility of results obtained with a code as controlled experiments and component tests are with a piece of experimental apparatus. It lies at the heart of the scientific method. Ideally, verification should be so easy that it can be performed routinely whenever changes are made to a code, so that errors are caught early. Verification benefits from modularity because a modular design isolates components (so they can be tested independently) and makes it easy to run different types of problems with the same code. With these considerations in mind, we have developed the following architecture for the current version of FLASH.

FLASH consists of several modules that perform various high-level tasks, such as driving the code, solving the Euler equations, and solving the nuclear network equations. Figure 21 schematically represents the relationships between these modules. In this figure, a line connecting a higher block to a lower block indicates that the routines in the higher block call routines in the lower block. While most of the modules are represented by a single block, the three major functions of the driver module (initialization, evolution, and input/output) are broken out into separate blocks. This reflects the fact that the driver module included with FLASH is oriented toward hyperbolic systems of equations.

While modules in FLASH carry out generic high-level functions, we wish to be able to carry out these functions with different types of algorithms or choices of physics. FLASH uses submodules to implement this capability. A submodule is an optional or alternative collection of routines that represents a specific instance of the functionality represented by its parent module. Modules can have multiple submodules, which may or may not be mutually exclusive. Submodules may themselves have submodules. Figure 22 shows two examples of module/submodule relationships in FLASH; one (the EOS) uses submodules to implement different physics, while the other (HYDRO) uses submodules to implement different types of hydrodynamic methods. (The directionally split flux-corrected transport method and unsplit PPM submodules are shown for illustrative purposes only; they are not present in the current version of FLASH.) Source code associated with a module is inherited by its submodules, so modules can serve as an interface layer between their submodules and the rest of the code. This

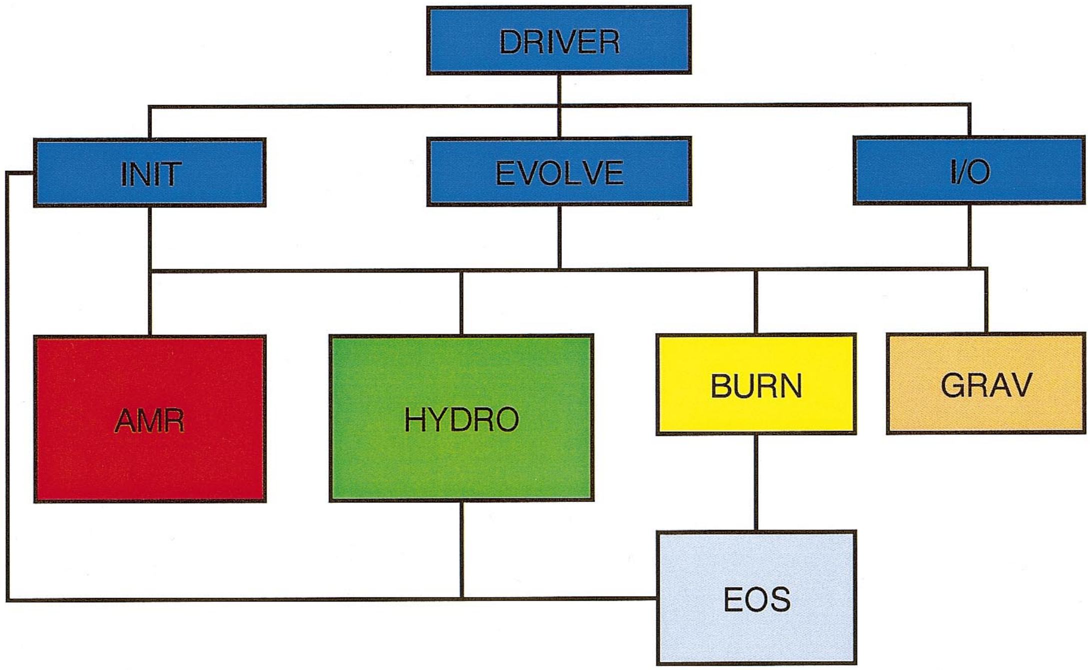

FIG. 21.—Module interrelationships in FLASH

allows different types of numerical methods to be plugged readily into FLASH, and when no module for a given piece of physics is needed, the interface layer can provide empty stub routines to other parts of the code that reference the module. The following section provides more detailed information on the practical organization of the FLASH source into modules and submodules.

Currently, data are shared among the different modules in FLASH by means of included common blocks that are part of each module. The driver module supplies the AMR data structure and all run-time parameters globally to all other modules. However, variables used only within a module are declared in that module's include files, which are not included by other modules. While this mechanism does enable module data to be kept " private " if module programmers are disciplined, the use of global variables leaves open the possibility that new modules will attempt to redefine variables provided by the driver. In the current development version of FLASH we are experimenting with ways to overcome this difficulty without hurting the code's performance. We expect the next version of FLASH to incorporate the use of FORTRAN 90 derived data types passed as arguments to modules by the driver, with no common blocks shared between modules.

6.1. *Code Organization*

The structure of the FLASH source tree is used to organize the source code into (more or less) independent code modules, as shown in Figure 23. The general plan is that source code is organized into one set of directories, while the code is built in a

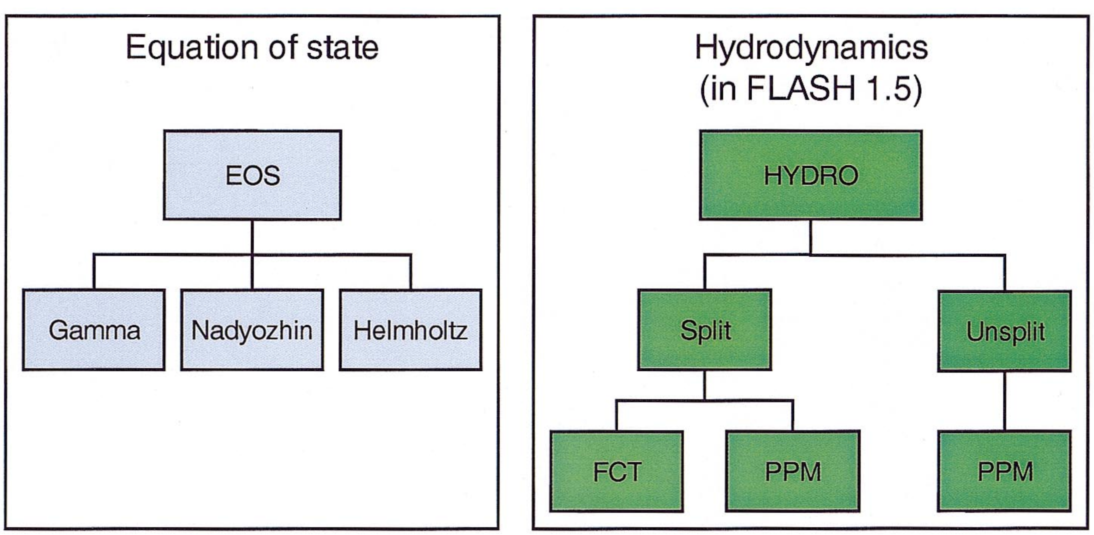

FIG. 22.—Examples of two uses of submodules in FLASH. FLASH 1.5 is the current development version

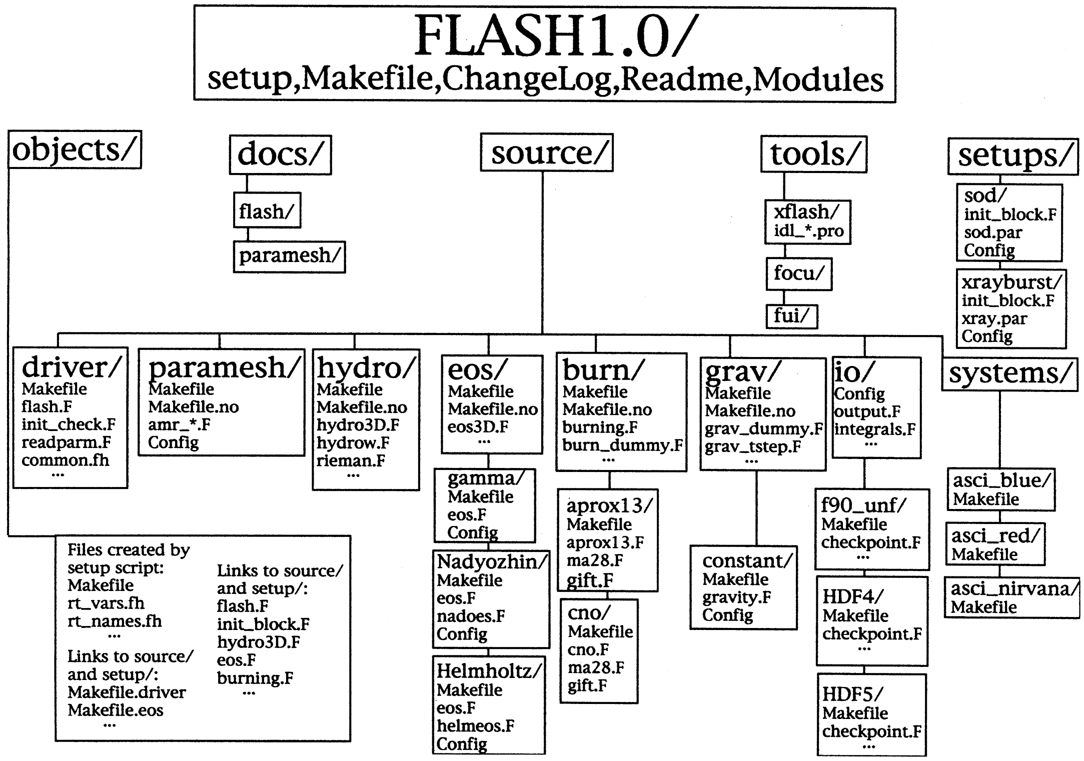

FIG. 23.—FLASH directory structure

separate directory using links to the appropriate source files. The links are created by a source configuration script called "setup," which makes the links using options selected by the user and then creates an appropriate makefile in the build directory. The user then builds the executable with a single invocation of "make."

Source code for each of the different code modules is stored in subdirectories under source/. The code modules implement different physics, such as hydrodynamics, nuclear burning, and gravity, or different major program components, such as the main driver module and the input/output routines. Each module directory contains source files, makefile fragments indicating how to build the module, and a plain text configuration file that contains information on module interrelationships. For each module, the makefile fragment Makefile is used when the module is included in the code, while Makefile.no is used when the module is not included (generally it builds a dummy source file containing subroutine stubs).

Each module subdirectory may have additional subdirectories beneath it corresponding to each of the module's submodules. These directories contain source files, makefiles, and configuration files specific to different variants of the module. For example, the hydro/ module directory can contain files that are generic to hydrodynamical methods, while its ppm/ subdirectory contains files specific to the PPM and its fct/ subdirectory contains files specific to the flux-corrected transport method. Configuration files for other modules that need hydrodynamics can specify hydro as a requirement without mentioning a specific solver; the user can then choose one solver or the other when building the code by means of a plain text module control file, which is used as input by the setup script. When setup configures the source tree, it treats each submodule as inheriting all of the source code and the configuration file in its parent module's directory, so generic code does not have to be duplicated. However, makefiles in the parent directory are not inherited. Submodules can themselves have submodules, so, for example, one might have hydro/split/ppm and hydro/unsplit/ppm.

An additional subdirectory of source/ called systems/ contains code specific to different architectures (e.g., irix and t3e), each of which has its own directory under systems/. Each of these directories contains a makefile fragment, which is used to set machine-dependent compilation flags and so forth. In addition, special versions of any routines that are needed for each architecture are included in these directories.

The setups/ directory has a structure similar to that of source/. In this case, however, each of the "modules" represents a different initial model or problem, and the problems are mutually exclusive; only one is included in the code at a time. Also, the problem directories have no equivalent of submodules. Makefile fragments specific to each problem may also be included.

The setup script creates a directory called object/ in which the executable is built. In this directory, setup creates links to all of the source code files found in the specified module and submodule directories as well as the specified problem directory. Because the problem setup directory and the machine-dependent directory are scanned last, links to files in these directories override the "defaults" taken from the source/ tree. Hence, special variants of routines needed for a particular problem can be used in place of the standard versions by simply giving the files containing them the same names.

Using information from the configuration files in the specified module and problem directories, setup creates FORTRAN include files needed to parse the run-time parameter file. The latter is fed to the FLASH executable at run time and is a plain text file consisting of statements of the form *variable* = *value*. Users can retain parameter files and use them to run the same simulation over and over again (for verification purposes) or modify them (to vary parameters in a set of production runs).

The setup routine also creates links in object/ to all of the appropriate makefile fragments (i.e., Makefile for included modules, Makefile.no for nonincluded modules, and the makefile fragments in the system- and problem-dependent directories). The link names in object/, Makefile.*module*, are associated with the appropriate fragments in the source directories. The setup script creates a master makefile (Makefile) in object/ that includes all of these fragments.

7. TEST CASES

To verify that FLASH works as expected and to debug changes in the code, we have created a suite of standard test problems. Most of these problems have analytical solutions that can be used to test the accuracy of the code. The remaining problems do not have analytical solutions, but they produce well-defined flow features that have been verified by experiments and are stringent tests of the code. These test problems are not as trivial as they might appear since we have added adaptive mesh refinement. The refinement algorithm employed in FLASH is designed to detect changes in the second derivative of a flow's features (see § 2) and to refine the mesh in advance of any of these features. These test problems verify that the hydrodynamics module and refinement/derefinement algorithm are working properly. Since these test calculations are very familiar to fluid dynamicists, they give added confidence that FLASH is working properly. The test suite configuration code is included within the FLASH source tree, so it is easy to configure and run FLASH with any of these problems "out of the box." The test problems described below exercise only the hydrodynamic and dimensionless gamma-law EOS modules of FLASH. Additional tests for reactive flows and more general equations of state will be described in a subsequent paper.

7.1. *The Advection Problem*

In this problem we create a planar density pulse in a region of uniform, dimensionless pressure P_0 and dimensionless velocity $\boldsymbol{v}_0$, with the velocity normal to the pulse plane. The density pulse is defined via

$$\rho(s) = \rho_1 \phi\left(\frac{s}{w}\right) + \rho_0\left[1 - \phi\left(\frac{s}{w}\right)\right], \tag{192}$$

where s is the distance of a point from the pulse midplane and w is the characteristic width of the pulse. The pulse shape function ϕ for a square pulse is

$$\phi_{\mathrm{SP}}(\xi) = \begin{cases} 1 & \text{if} \, |\xi| < 1 \,, \\ 0 & \text{otherwise} \,, \end{cases} \tag{193}$$

while for a Gaussian pulse,

$$\phi_{\mathrm{GP}}(\xi) = e^{-\xi^2} \,. \tag{194}$$

For these initial conditions the Euler equations reduce to a single advection equation with wave speed u_0; hence, the density pulse should move across the computational volume at this speed without changing shape. Advection problems similar to this were first proposed by Boris & Book (1973) and Forester (1977).

The advection problem tests the ability of the code to handle planar geometry. It also tests the code's treatment of flow features that move at one of the characteristic speeds of the hydrodynamical equations. This is difficult because noise generated at a feature tends to move with it, accumulating there as the calculation advances. The square pulse problem compares the code's treatment of leading and trailing contact discontinuities. The Gaussian pulse problem tests the treatment of narrow flow features. Many hydrodynamical methods have a tendency to clip narrow features or to distort pulse shapes by introducing artificial dispersion and dissipation (Zalesak 1987).

To demonstrate the performance of FLASH on the advection problem, we have performed tests of both the square and Gaussian pulse profiles with the pulse normal parallel to the x-axis ($\theta = 0°$) and at an angle to the x-axis ($\theta = 45°$) in two dimensions. The square pulse used $\rho_1 = 1$, $\rho_0 = 10^{-3}$, $P_0 = 10^{-6}$, $v_0 = 1$, and $w = 0.1$. With six levels of refinement in the domain $(0, 1) \times (0, 1)$, this value of w corresponds to having about 52 zones across the pulse width. The Gaussian pulse tests used the same values of ρ_1, ρ_0, P_0, and v_0, but with $w = 0.015625$. This value of w corresponds to about eight zones across the pulse width at six levels of refinement. For each test we performed runs at two, four, and six levels of refinement to examine the quality of the numerical solution as the resolution of the advected pulse improves. The runs with $\theta = 0°$ used zero-gradient (outflow) boundary conditions, while the runs performed at an angle to the x-axis used periodic boundaries. Periodic boundaries in the $\theta = 45°$ case are used because the pulse profile is not normal to the boundary, so simply continuing the solution into the boundary zones as is done for $\theta = 0°$ would produce a reflection.

Figure 24 shows, for each test, the advected density profile at $t = 0.4$ in comparison with the analytical solution. The upper two frames of this figure depict the square pulse with $\theta = 0°$ and $\theta = 45°$, while the lower two frames depict the Gaussian pulse results. In each case the analytical density pulse has been advected a distance $v_0 t = 0.4$, and in the figure the axis parallel to the pulse normal has been translated by this amount, permitting comparison of the pulse displacement in the numerical solutions with that of the analytical solution.

The advection results show the expected improvement with increasing AMR level N_{ref}. Inaccuracies appear as diffusive spreading, rounding of sharp corners, and clipping. In both the square pulse and Gaussian pulse tests, diffusive spreading is limited to about one zone on either side of the pulse. For $N_{\mathrm{ref}} = 2$ the rounding of the square pulse and the clipping of the Gaussian pulse are quite severe; in the latter case, the pulse itself spans about two zones, which is the approximate smoothing

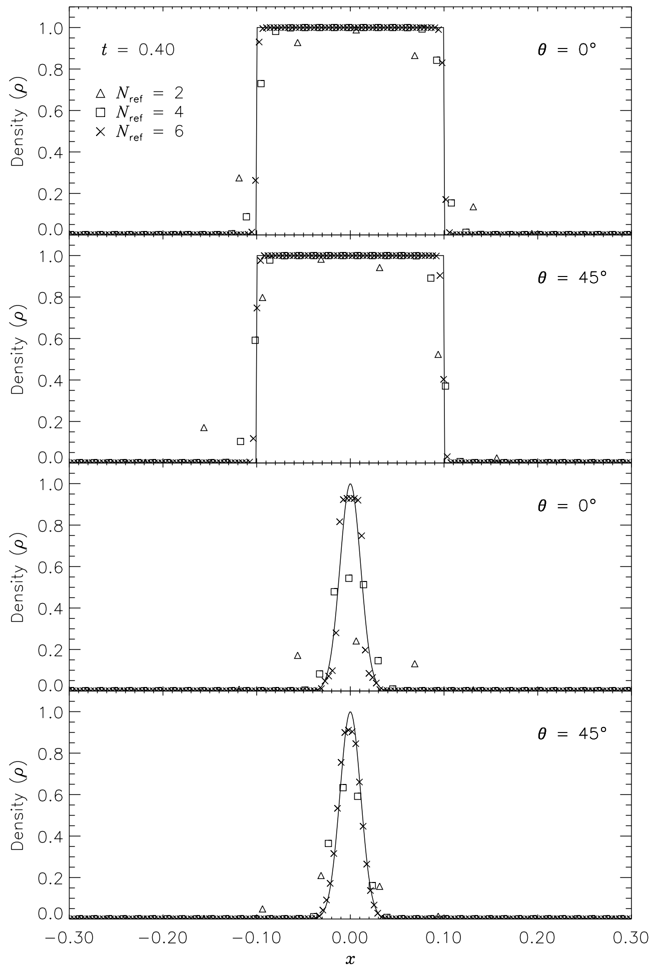

FIG. 24.—Density pulse in the advection tests for two-dimensional grids at $t = 0.4$. Symbols represent numerical results using grids with different levels of refinement $N_{\rm ref}$ (2, 4, and 6).

length in PPM for a single discontinuity. For $N_{\rm ref} = 4$ the treatment of the square pulse is significantly better, but the amplitude of the Gaussian is still reduced by about 50%. In this case the square pulse discontinuities are still being resolved with two to three zones, but the zones are now a factor of 25 smaller than the pulse width. With six levels of refinement, the same behavior is observed for the square pulse, while the amplitude of the Gaussian pulse is now 93% of its initial value. The absence of dispersive effects (i.e., oscillations) despite the high order of the PPM interpolants is due to the enforcement of monotonicity in the PPM algorithm.

The diagonal runs are consistent with the runs that were parallel to the x-axis, with the possibility of a slight amount of extra spreading behind the pulse. However, note that we have determined density values for the diagonal runs by interpolation along the grid diagonal. The interpolation points are not centered on the pulses, so the density does not always take on its maximum value (particularly in the lowest resolution case).

These results are consistent with earlier studies of linear advection with PPM (e.g., Zalesak 1987). They suggest that, in order to preserve narrow flow features in FLASH, the maximum AMR level should be chosen so that zones in refined regions are at least a factor of 5–10 smaller than the narrowest features of interest. In cases in which the features are generated by shocks (rather than moving with the fluid), the resolution requirement is not as severe, as errors generated in the preshock region are driven into the shock rather than accumulating as it propagates.

7.2. *Sod's Problem*

The Sod problem (Sod 1978) is a one-dimensional flow discontinuity problem that provides a good test of a compressible code's ability to capture shocks and contact discontinuities within a small number of zones and to produce the correct profile in a rarefaction. It also tests a code's ability to satisfy correctly the Rankine-Hugoniot shock jump conditions. When implemented at an angle to a multidimensional grid, it can also be used to detect irregularities in planar discontinuities produced by grid geometry or operator splitting effects.

We construct the initial conditions for the Sod problem by establishing a planar interface at some angle to the x- and y-axes. The fluid is initially at rest on either side of the interface, and the density and pressure jumps are chosen so that all three types of flow discontinuities (shock, contact, and rarefaction) develop. To the "left" and "right" of the interface we have the dimensionless densities and pressures

$$\rho_l = 1 \; , \tag{195}$$

$$\rho_r = 0.125 \; , \tag{196}$$

$$P_l = 1 \; , \tag{197}$$

$$P_r = 0.1 \; . \tag{198}$$

The ratio of specific heats γ is chosen to be 1.4 on both sides of the interface.

Figure 25 shows the result of running the Sod problem with FLASH on a two-dimensional grid, with the analytical solution shown for comparison. The hydrodynamical algorithm used here is the directionally split PPM included with FLASH. In this run the shock normal is chosen to be parallel to the x-axis. With six levels of refinement, the effective grid size at the finest level is 256^2, so the finest zones have width 0.00390625. At $t = 0.2$ three different nonlinear waves are present: a rarefaction between $x \approx 0.25$ and $x \approx 0.5$, a contact discontinuity at $x \approx 0.68$, and a shock at $x \approx 0.85$. The two sharp discontinuities are each resolved with approximately two to three zones at the highest level of refinement, demonstrating the ability of PPM to handle sharp flow features well. Near the contact discontinuity and in the rarefaction, we find small errors of about 1%–2% in the density and specific internal energy, with similar errors in the velocity inside the rarefaction. Elsewhere the numerical solution is nearly exact; no oscillation is present.

Figure 26 shows the result of running the Sod problem on the same two-dimensional grid with different shock normals: parallel to the x-axis ($\theta = 0°$) and along the box diagonal ($\theta = 45°$). For the diagonal solution we have interpolated values of density, specific internal energy, and velocity to a set of 256 points spaced exactly as in the x-axis solution. This comparison shows the effects of the second-order directional splitting used with FLASH on the resolution of shocks. At the right side of the rarefaction and at the contact discontinuity the diagonal solution undergoes slightly larger oscillations (on the order of a few percent) than the x-axis solution. The location and thickness of the discontinuities is the same between the two solutions. In general, shocks at an angle to the grid are resolved with approximately the same number of zones as shocks parallel to a coordinate axis.

Figure 27 presents a color map of the density at $t = 0.2$ in the diagonal solution together with the block structure of the AMR grid in this case. Note that regions surrounding the discontinuities are maximally refined, while behind the shock and

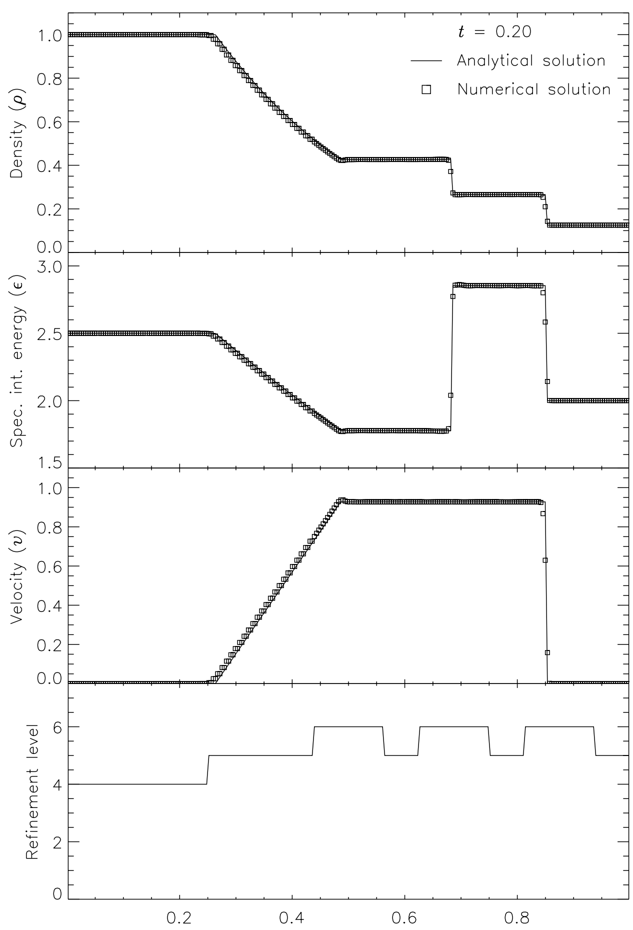

FIG. 25.—Comparison of numerical and analytical solutions to the Sod problem. A two-dimensional grid with six levels of refinement is used. The shock normal is parallel to the x-axis.

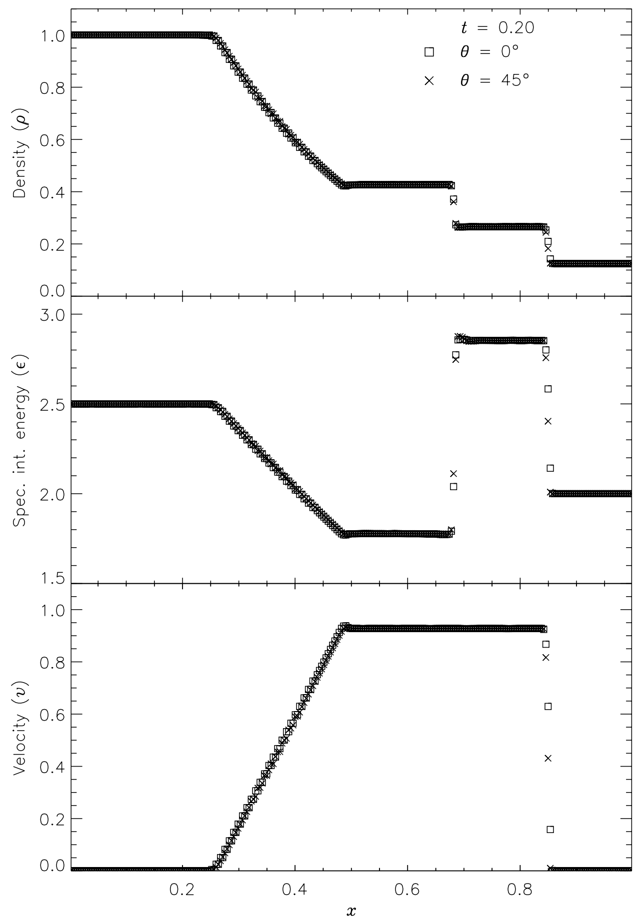

FIG. 26.—Comparison of numerical solutions to the Sod problem for two different angles (θ) of the shock normal relative to the x-axis. A two-dimensional grid with six levels of refinement is used.

contact discontinuity, the grid has derefined as the second derivative of the density has decreased in magnitude. Because zero-gradient outflow boundaries were used for this test, some reflections are present at the upper left and lower right corners, but at $t = 0.2$ these have not yet propagated to the center of the grid.

7.3. *Strong Shock Tube Problem*

We included the Sod problem because it has become a standard test case for hydrodynamics codes. However, it is not a very discriminating test for modern codes. The strong shock tube problem (Zalesak 2000, private communication) is a more demanding test problem because of the stronger discontinuities across the shock interface and the narrow density peak that forms behind the shock. We set the density and pressure to be

$$\rho_l = 10 \; , \tag{199}$$

$$\rho_r = 1 \; , \tag{200}$$

$$P_l = 100 \; , \tag{201}$$

$$P_r = 1 \; , \tag{202}$$

and the material is initially at rest everywhere. The ratio of specific heats γ is taken to be 1.4 on both sides of the interface.

Figure 28 shows the result of running the strong shock tube problem with FLASH on a one-dimensional grid with eight levels of refinement. The analytical solution for the density at 0.4 s is shown for comparison. The numerical solution is virtually indistinguishable from the analytical solution.

7.4. *Sedov Explosion*

The Sedov explosion problem (Sedov 1959) is another purely hydrodynamical test in which we check the code's ability to deal with strong shocks and nonplanar symmetry. The problem involves the self-similar evolution of a cylindrical or spherical blast wave from a delta-function initial pressure perturbation in an otherwise homogeneous medium. To initialize the code,

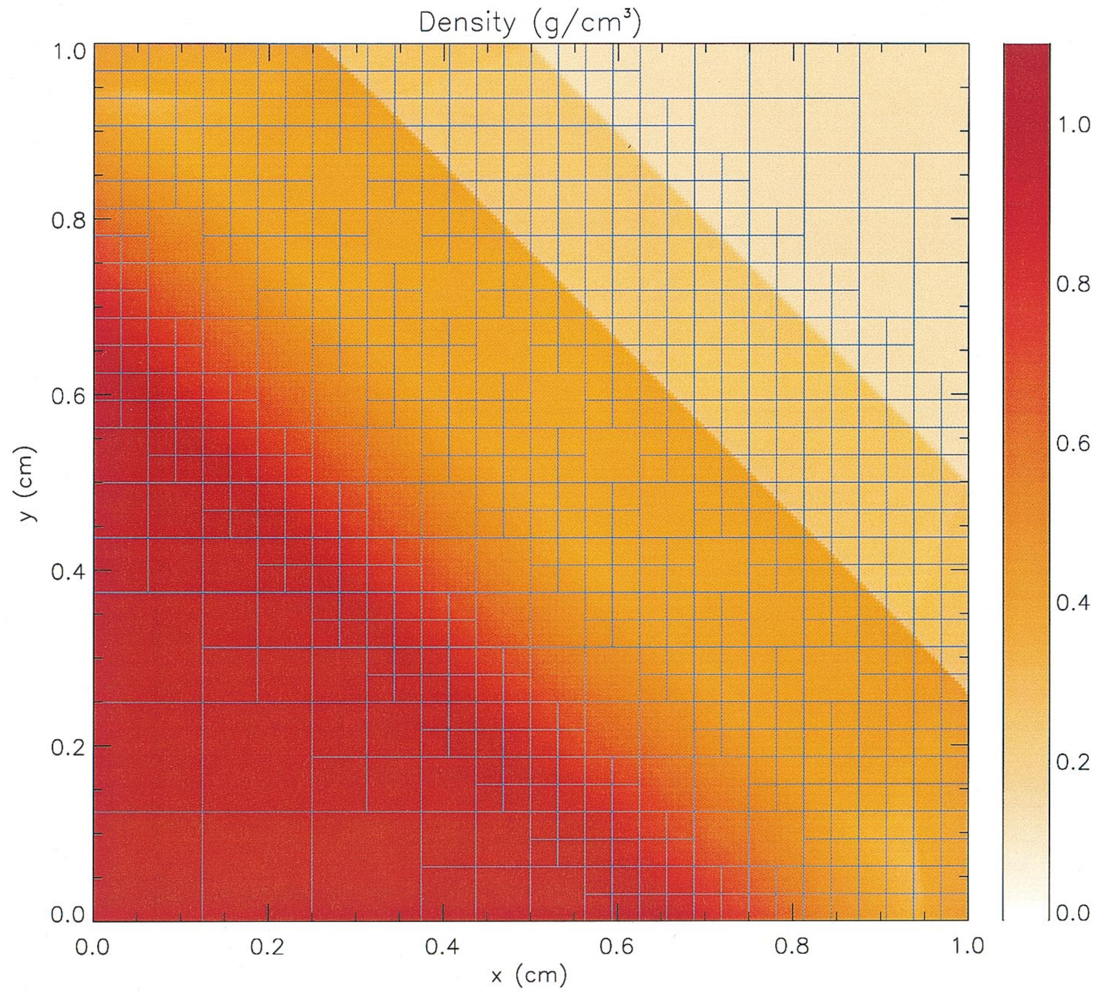

FIG. 27.—Density in the diagonal two-dimensional Sod problem with six levels of refinement at $t = 0.2$. The outlines of AMR blocks are shown (each block contains 8×8 zones).

we deposit a quantity of dimensionless energy $\epsilon = 1$ into a small region of radius δr at the center of the grid. The dimensionless pressure inside this volume, P'_0, is given by

$$P'_0 = \frac{3(\gamma - 1)\epsilon}{(\nu + 1)\pi\,\delta r^{\nu}}\,, \tag{203}$$

where $\nu = 2$ for cylindrical geometry and $\nu = 3$ for spherical geometry. In running this problem we choose δr to be 3.5 times as large as the finest adaptive mesh resolution in order to minimize effects due to the Cartesian geometry of our grid. The density

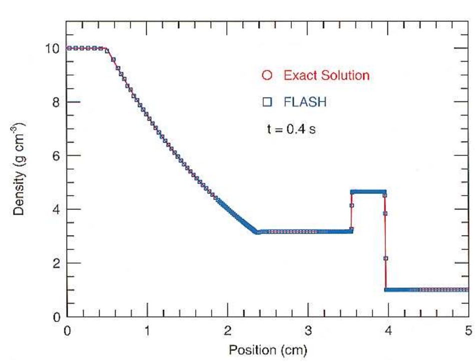

FIG. 28.—Comparison of numerical and analytical solutions for the density to the strong shock tube problem. A one-dimensional grid with eight levels of refinement is used.

is set equal to $\rho_0 = 1$ throughout the grid, and the pressure is set to a small value $P_0 = 10^{-5}$ except in the explosion region. The fluid is initially at rest. In the self-similar blast wave that develops for $t > 0$, the density, pressure, and radial velocity are all functions of $\xi \equiv r/R(t)$, where

$$R(t) = C_\nu(\gamma)\left(\frac{\epsilon t^2}{\rho_0}\right)^{1/(\nu+2)} . \tag{204}$$

Here C_ν is a dimensionless constant depending only on ν and γ; for $\gamma = 1.4$, $C_2 \approx C_3 \approx 1$ to within a few percent. Just behind the shock front at $\xi = 1$ we have

$$\rho = \rho_1 \equiv \frac{\gamma+1}{\gamma-1}\,\rho_0 , \tag{205}$$

$$P = P_1 \equiv \frac{2}{\gamma+1}\,\rho_0 u^2 , \tag{206}$$

$$v = v_1 \equiv \frac{2}{\gamma+1}\,w , \tag{207}$$

where $w \equiv dR/dt$ is the speed of the shock wave. Near the center of the grid,

$$\frac{\rho(\xi)}{\rho_1} \propto \xi^{\nu/(\gamma-1)} , \tag{208}$$

$$\frac{P(\xi)}{P_1} = \text{constant} , \tag{209}$$

$$\frac{v(\xi)}{v_1} \propto \xi . \tag{210}$$

Figure 29 shows density, pressure, and velocity profiles in the two-dimensional Sedov problem at $t = 0.05$. Solutions obtained with FLASH on grids with two, four, six, and eight levels of refinement are shown in comparison with the analytical solution. In this figure we have computed average radial profiles in the following way. We interpolated solution values from the adaptively gridded mesh used by FLASH onto a uniform mesh having the same resolution as the finest AMR blocks in each run. Then, using radial bins with the same width as the zones in the uniform mesh, we binned the interpolated solution values, computing the average value in each bin. At low resolutions, errors show up as density and velocity overestimates behind the shock, underestimates of each variable within the shock, and a numerical precursor spanning one to two zones in front of the shock. However, the central pressure is accurately determined, even for two levels of refinement; because the density goes to a finite value rather than its correct limit of zero, this corresponds to a finite truncation of the temperature (which should go to infinity as $r \to 0$). As resolution improves, the artificial finite density limit decreases; by $N_{\rm ref} = 6$ it is less than 0.2% of the peak density. Except for the $N_{\rm ref} = 2$ case, which does not show a well-defined peak in any variable, the shock itself is always captured with about two zones. The region behind the shock containing 90% of the swept-up material is represented by four zones in the $N_{\rm ref} = 4$ case, 17 zones in the $N_{\rm ref} = 6$ case, and 69 zones for $N_{\rm ref} = 8$. However, because the solution is self-similar, for any given maximum refinement level this region will be four zones wide at a sufficiently early time. The behavior when the shock is underresolved is to underestimate the peak value of each variable, particularly the density and pressure.

Figure 30 shows the pressure field in the eight-level calculation at $t = 0.05$ together with the block refinement pattern. Note that a relatively small fraction of the grid is maximally refined in this problem. Although the pressure gradient at the center of the grid is small, this region is refined because of the large temperature gradient there. This illustrates the ability of PARAMESH to use refinement criteria based on several different variables at once.

We have also run FLASH on the spherically symmetric Sedov problem in order to verify the code's performance in three dimensions. The results at $t = 0.05$ using five levels of grid refinement are shown in Figure 31. In this figure we have plotted the rms numerical solution values in addition to the average values. As in the two-dimensional runs, the shock is spread over about two zones at the finest AMR resolution in this run. The width of the pressure peak in the analytical solution is about one and a half zones at this time, so the maximum pressure is not captured in the numerical solution. Behind the shock the numerical solution average tracks the analytical solution quite well, although the Cartesian grid geometry produces rms deviations of up to 40% in the density and velocity in the derefined region well behind the shock. This behavior is similar to that exhibited in the two-dimensional problem at comparable resolution.

7.5. *The Interacting Blast-Wave Problem*

This problem was originally used by Woodward & Colella (1984) to compare the performance of several different hydrodynamical methods on problems involving strong, thin shock structures. It has no analytical solution, but since it is one-dimensional, it is easy to produce a converged solution by running the code with a very large number of zones, permitting an estimate of the self-convergence rate when narrow, interacting discontinuities are present. For FLASH it also provides a good test of the AMR scheme.

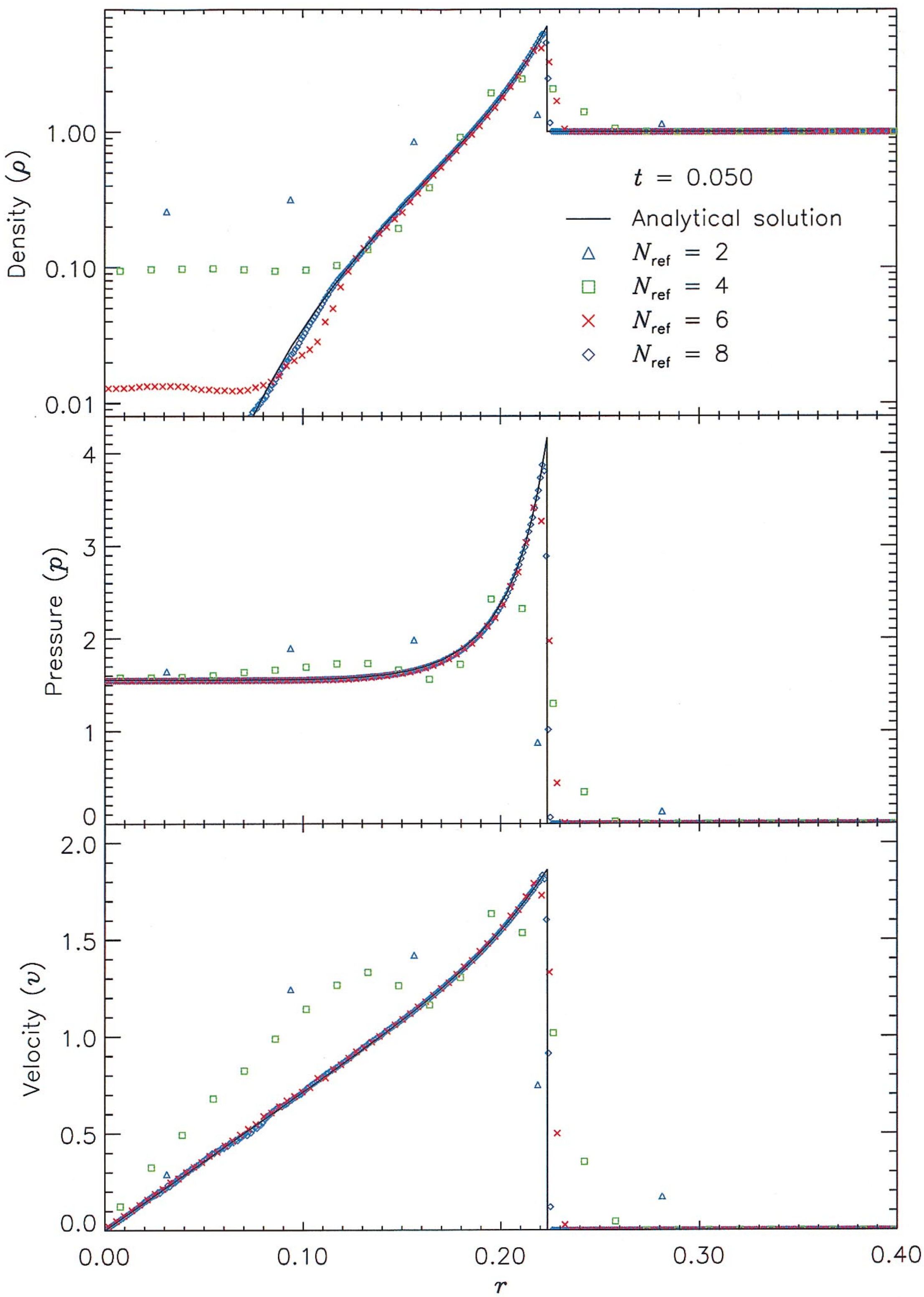

FIG. 29.—Comparison of numerical and analytical solutions to the Sedov problem in two dimensions. Numerical solution values are averages in radial bins at the finest AMR grid resolution in each run.

The initial conditions consist of two parallel, planar flow discontinuities. Reflecting boundary conditions are used. The dimensionless density in the left, middle, and right portions of the grid (ρ_l, ρ_m, and ρ_r, respectively) is unity; everywhere the velocity is zero. The dimensionless pressure is large to the left and right and small in the center:

$$P_l = 1000 , \tag{211}$$

$$P_m = 0.01 , \tag{212}$$

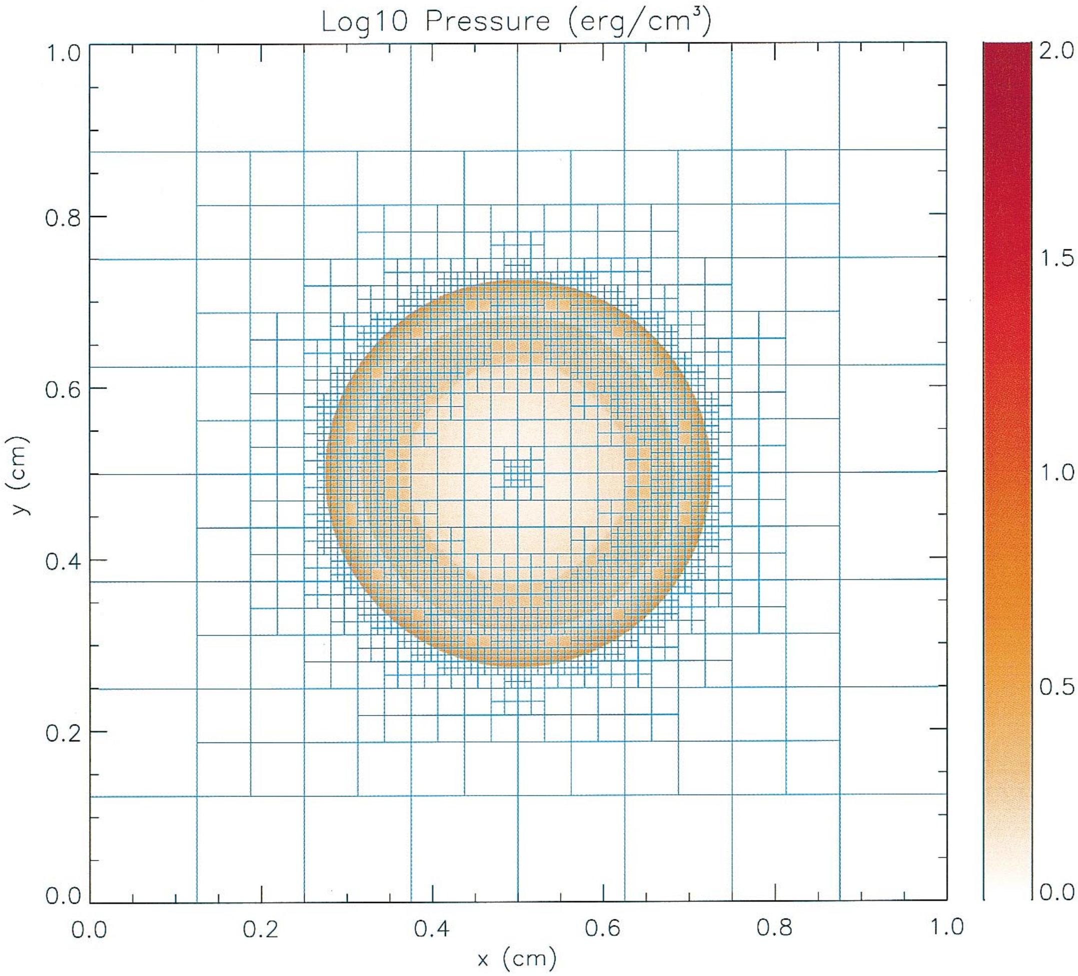

FIG. 30.—Pressure field in the two-dimensional Sedov explosion problem with eight levels of refinement at $t = 0.05$. Overlaid on the pressure color map are the outlines of the AMR blocks.

$$P_r = 100 \ . \tag{213}$$

The EOS is a dimensionless gamma law with $\gamma = 1.4$.

Figure 32 shows the density and velocity profiles at several different times in the converged solution, demonstrating the complexity inherent in this problem. The initial pressure discontinuities drive shocks into the middle part of the grid; behind them, rarefactions form and propagate toward the outer boundaries, where they are reflected back onto the grid and interact with themselves. By the time the shocks collide at $t = 0.028$, the reflected rarefactions have caught up to them, weakening them and making their postshock structure more complex. Because the right-hand shock is initially weaker, the rarefaction on that side reflects from the wall later, so the resulting shock structures going into the collision from the left and right are quite different. Behind each shock is a contact discontinuity left over from the initial conditions (at $x \approx 0.50$ and 0.73). The shock collision produces an extremely high and narrow density peak; the peak value we get with FLASH is consistent with the peak value in the Woodward & Colella (1984) calculation. Reflected shocks travel back into the colliding material, leaving a complex series of contact discontinuities and rarefactions between them. A new contact discontinuity has formed at the point of the collision ($x \approx 0.69$). By $t = 0.032$ the right-hand reflected shock has met the original right-hand contact discontinuity, producing a strong rarefaction that meets the central contact discontinuity at $t = 0.034$. Between $t = 0.034$ and $t = 0.038$ the slope of the density behind the left-hand shock changes as the shock moves into a region of constant entropy near the left-hand contact discontinuity.

Figure 33 shows the self-convergence of density and pressure when FLASH is run on this problem. For several runs with different maximum refinement levels, we compare the density, pressure, and total specific energy at $t = 0.038$ to the solution obtained using FLASH with 10 levels of refinement. This figure plots the L1 error norm for each variable U, defined using

$$\mathscr{E}(N_{\mathrm{ref}}; U) \equiv \frac{1}{N(N_{\mathrm{ref}})} \sum_{i=1}^{N(N_{\mathrm{ref}})} \left| \frac{U_i(N_{\mathrm{ref}}) - U_i(10)}{U_i(10)} \right| , \tag{214}$$

against the effective number of zones $[N(N_{\mathrm{ref}})]$ at the highest level of refinement N_{ref}. In computing this norm, both the "converged" solution $U(10)$ and the test solution $U(N_{\mathrm{ref}})$ are interpolated onto a uniform mesh having $N(N_{\mathrm{ref}})$ zones. Values

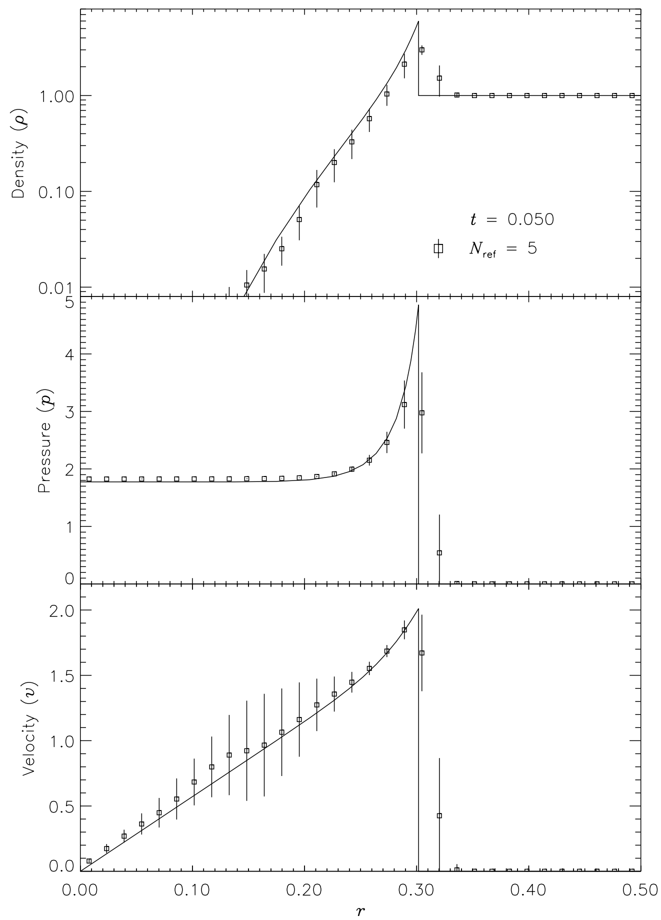

FIG. 31.—Comparison of numerical and analytical solutions to the spherically symmetric Sedov problem. A three-dimensional grid with five levels of refinement is used.

of $N_{\rm ref}$ between 2 (corresponding to cell size $\Delta x = 1/16$) and 9 ($\Delta x = 1/2048$) are shown. Although PPM is formally a second-order method, one sees from this plot that, for the interacting blast-wave problem, the convergence rate is only linear. Indeed, in their comparison of the performance of seven nominally second-order hydrodynamic methods on this problem, Woodward & Colella (1984) found that only PPM achieved even linear convergence; the other methods were worse. The L1 error norm is very sensitive to the correct position and shape of the strong, narrow shocks generated in this problem.

7.6. *Wind Tunnel with a Step*

The problem of a wind tunnel containing a step was first described by Emery (1968), who used it to compare several hydrodynamical methods that are only of historical interest now. Woodward & Colella (1984) later used it to compare several more advanced methods, including PPM. Although it has no analytical solution, this problem is useful because it exercises a code's ability to handle unsteady shock interactions in multiple dimensions. It also provides an example of the use of FLASH to solve problems with irregular boundaries.

The problem uses a two-dimensional rectangular domain three units wide and one unit high. Between $x = 0.6$ and $x = 3$ along the x-axis is a step 0.2 units high. The step is treated as a reflecting boundary, as are the lower and upper boundaries in the y-direction. For the right-hand x boundary we use an outflow (zero gradient) boundary condition, while on the left-hand side we use an inflow boundary. In the inflow boundary zones we set the dimensionless density to ρ_0, the dimensionless pressure to P_0, and the dimensionless velocity to u_0, with the latter directed parallel to the x-axis. The domain itself is also initialized with these values. For the Emery problem we use

$$\rho_0 = 1.4 \, , \tag{215}$$

$$P_0 = 1 \, , \tag{216}$$

$$\gamma = 1.4 \, , \tag{217}$$

$$v_0 = 3 \, , \tag{218}$$

which corresponds to a Mach 3 flow. Because the outflow is supersonic throughout the calculation, we do not expect reflections from the right-hand boundary.

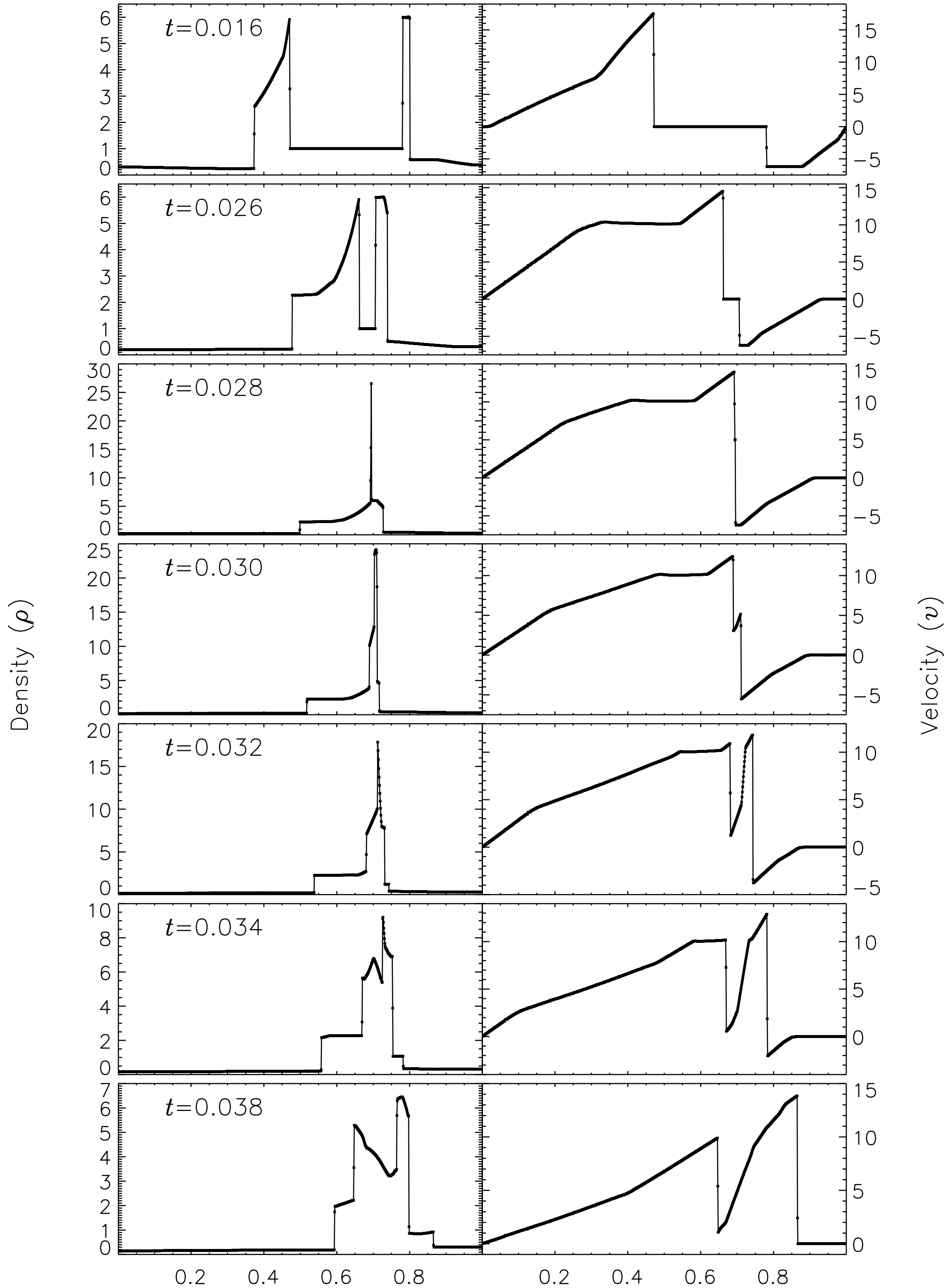

FIG. 32.—Density and velocity profiles in the Woodward-Colella (1984) interacting blast-wave problem, as computed by FLASH using 10 levels of refinement.

Until $t = 12$ the flow is unsteady, exhibiting multiple shock reflections and interactions between different types of discontinuities. Figures 34 and 35 show the evolution of density and velocity between $t = 0$ and $t = 4$ (the period considered by Woodward & Colella 1984). Immediately, a shock forms directly in front of the step and begins to move slowly away from it. Simultaneously, the shock curves around the corner of the step, extending farther downstream, growing in size until it strikes the upper boundary just after $t = 0.5$. The corner of the step becomes a singular point, with a rarefaction fan connecting the still gas just above the step to the shocked gas in front of it. Entropy errors generated in the vicinity of this singular point produce a numerical boundary layer about one zone thick along the surface of the step. Woodward & Colella (1984) reduce this effect by resetting the zones immediately behind the corner to conserve entropy and the sum of enthalpy and specific

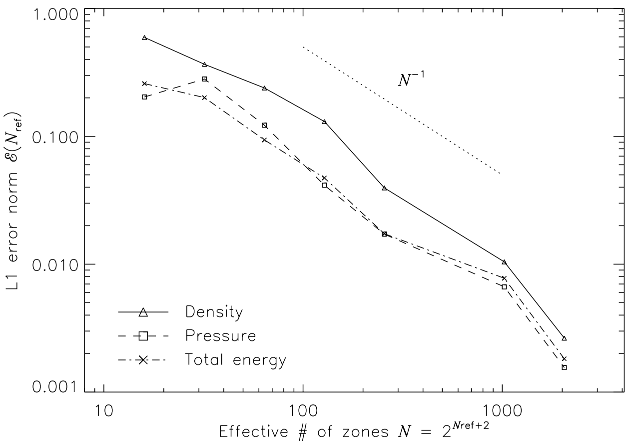

FIG. 33.—Self-convergence of the density, pressure, and total specific energy in the 2blast test problem

kinetic energy through the rarefaction. However, we are less interested here in reproducing the exact solution than in verifying the code and examining the behavior of such numerical effects as resolution is increased, so we do not apply this additional boundary condition. The errors near the corner result in a slight overexpansion of the gas there and a weak oblique shock where this gas flows back toward the step. At all resolutions we also see interactions between the numerical boundary layer and the reflected shocks that appear later in the calculation.

By $t = 1$ the shock reflected from the upper wall has moved downward and has almost struck the top of the step. The intersection between the primary and reflected shocks begins at $x \approx 1.45$, when the reflection first forms at $t \approx 0.65$, and then moves to the left, reaching $x \approx 0.95$ at $t = 1$. As it moves, the angle between the incident shock and the wall increases until $t = 1.5$, at which point it exceeds the maximum angle for regular reflection ($40°$ for $\gamma = 1.4$) and begins to form a Mach stem. Meanwhile the reflected shock has itself reflected from the top of the step, and here too the point of intersection moves to the left, reaching $x \approx 1.65$ by $t = 2$. The second reflection propagates back toward the top of the grid, reaching it at $t = 2.5$ and forming a third reflection. By this time in low-resolution runs, we see a second Mach stem forming at the shock reflection from the top of the step; this results from the interaction of the shock with the numerical boundary layer, which causes the angle of incidence to increase faster than in the converged solution. Figure 36 compares the density field at $t = 4$ as computed by FLASH using several different maximum levels of refinement. Note that the size of the artificial Mach reflection diminishes as resolution improves.

The shear zone behind the first (" real ") Mach stem produces another interesting numerical effect, visible at $t = 3$ and $t = 4$: Kelvin-Helmholtz amplification of numerical errors generated at the shock intersection. The wave thus generated propagates downstream and is refracted by the second and third reflected shocks. This effect is also seen in the calculations of Woodward & Colella (1984), although their resolution was too low to capture the detailed eddy structure we see. Figure 37 shows the detail of this structure at $t = 4$ on grids with several different levels of refinement. The effect does not disappear with increasing resolution, for two reasons. First, the instability amplifies numerical errors generated at the shock intersection, no matter how small. Second, PPM captures the slowly moving, nearly vertical Mach stem with only one to two zones on any grid, so as it moves from one column of zones to the next, artificial kinks form near the intersection, providing the seed perturbation for the instability. This effect can be reduced by using a small amount of extra dissipation to smear out the shock, as discussed by Colella & Woodward (1984). This tendency of physical instabilities to amplify numerical noise vividly demonstrates the need to exercise caution when interpreting features in supposedly converged calculations.

The Kelvin-Helmholtz instability illustrates the importance of using care in selecting grid refinement criteria. With the refinement and derefinement thresholds set to 0.8 and 0.2, respectively (see the discussion of $CTORE$ and $CTODE$ in § 2), the five-level run did not refine the slip line immediately behind the Mach stem, although it did refine farther downstream. As a result, the instability fully manifested itself only for $x \gtrsim 1.7$. Reducing the refinement threshold to 0.6 caused the slip line and the instability to be maximally refined everywhere. We interpret this effect as being due to interaction between the converging strength of the contact discontinuity and the refinement threshold. By coincidence the strength of the contact discontinuity in this problem produces a second derivative that lies close to the default refinement threshold, and as the value of this derivative converges with increasing refinement level, it drops below the threshold. As a result, the instability appears coarsely resolved unless the threshold is reduced. The lesson here is that the refinement criterion and thresholds need to be carefully considered for any particular problem. In a full convergence study, any features that are not refined as the maximum refinement level is increased must be tested by varying the refinement criterion.

Finally, we note that in high-resolution runs with FLASH, we also see some Kelvin-Helmholtz roll-up at the numerical boundary layer along the top of the step. This is not present in Woodward & Colella's (1984) calculation, presumably because their grid resolution is lower (corresponding to two levels of refinement for us) and because of their special treatment of the singular point.

7.7. *Shock through a Forced Jump in Mesh Refinement*

The mesh refinement algorithm used by FLASH is designed to detect sharp changes in a flow's features and to refine the mesh in advance of any of these features (see § 2). The suite of test problems presented above, particularly the Sod problem

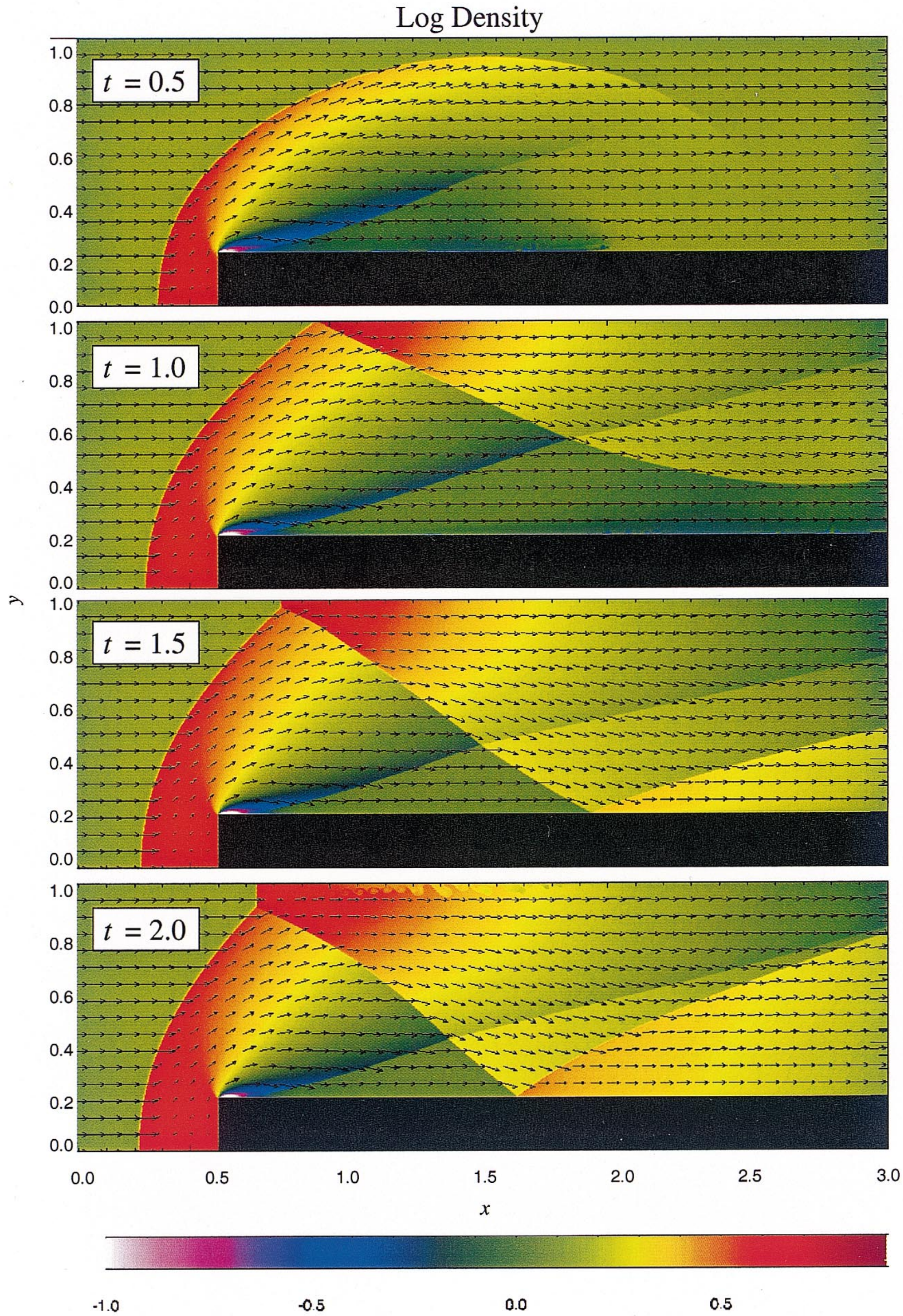

FIG. 34.—Density and velocity in the Emery wind tunnel test problem, as computed with FLASH. A two-dimensional grid with five levels of refinement is used.

with various inclination angles, demonstrates that the mesh refinement procedures function as designed. There may be cases, however, in which it is desirable to force a jump in the mesh refinement. Such cases may arise from limited computational resources, or from carrying regions where a fully refined solution is not necessary. For example, one may wish to place the boundaries of the simulation domain far away from the region where the dynamics of the model occur. Coarsely refining the simulation domain in the "uninteresting" regions around the boundaries can conserve considerable computational resources.

Figure 38 shows the density field at $t = 0.25$ s for a Sedov shock wave (described in § 7.4) propagating through a forced jump in mesh refinement at $x = 0.5$. Seven levels of mesh refinement are used between $x = 0$ and $x = 0.5$, while only six levels of refinement are used for $x > 0.5$. The solution in the $x > 0.5$ region with six levels of refinement is broader and coarser than the corresponding solution in the $y > 0.5$ region that has seven levels of refinement. In addition, Figure 38 shows that a

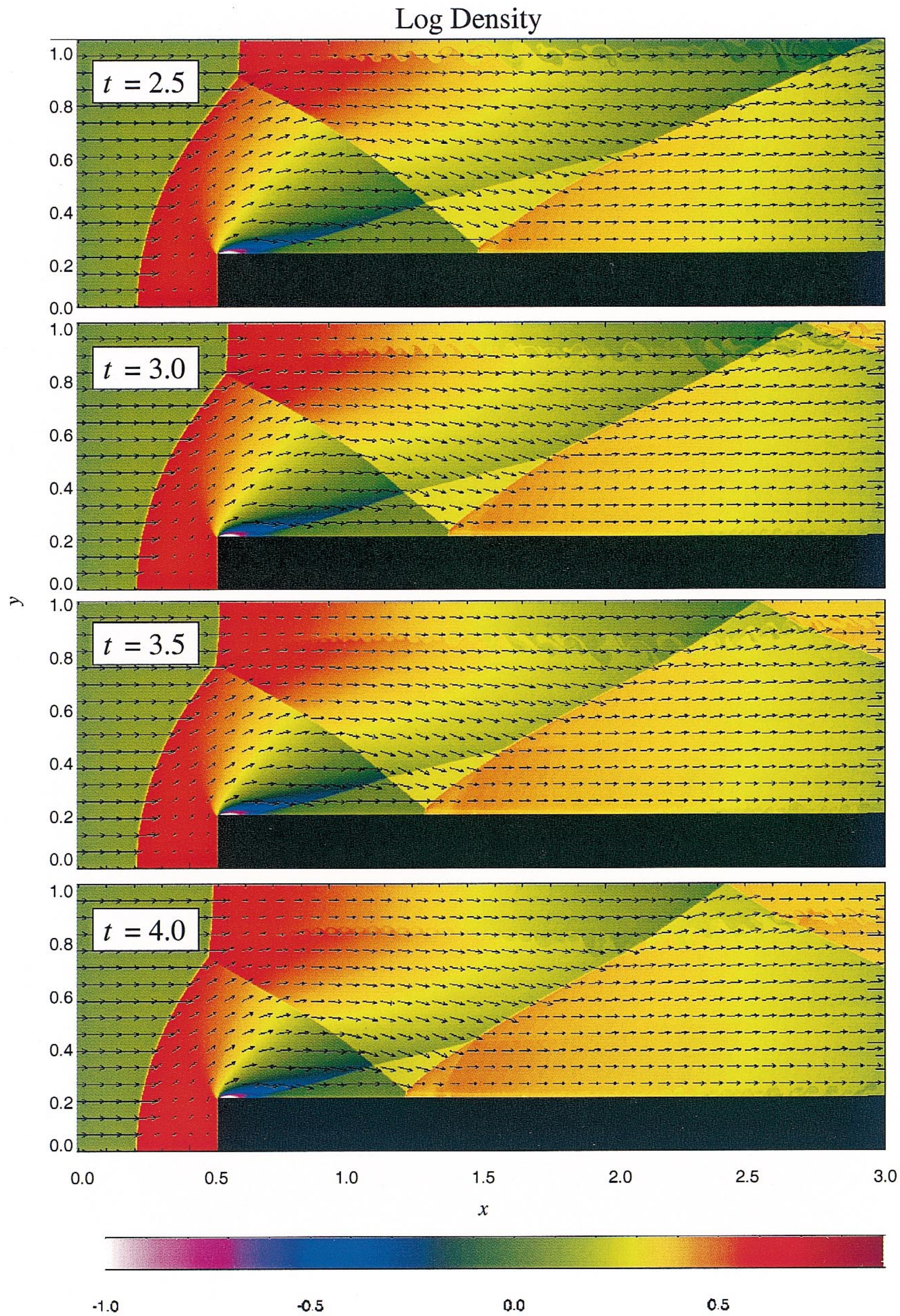

FIG. 35.—Density and velocity in the Emery wind tunnel test problem (continued)

reflected shock wave is produced at the forced jump in refinement at $x = 0.5$. The smaller the angle between the direction of the shock wave and the forced jump in refinement, the stronger the erroneous reflected shock becomes.

Figure 39 compares the density structure along the two coordinate axes of Figure 38. The black curve corresponds to the solution along the x-axis, which has a forced jump in mesh refinement at $x = 0.5$. The blue curve corresponds to the solution along the y-axis, which does not have a forced jump in mesh refinement. Since the shock front has reached $\sim$0.92 cm along the x-axis but has reached $\sim$0.88 cm along the y-axis, the speed of the shock front is slower in the more coarsely refined regions. Figure 39 also shows that the peak density in the more coarsely resolved regions (*black curves*) is smaller than peak density in the more refined regions (*blue curves*).

The results of this test problem suggest that forced jumps in mesh refinement should only be used for regions in the simulation domain that are not critical to the model. In these noncritical regions it is preferable to have jumps in mesh

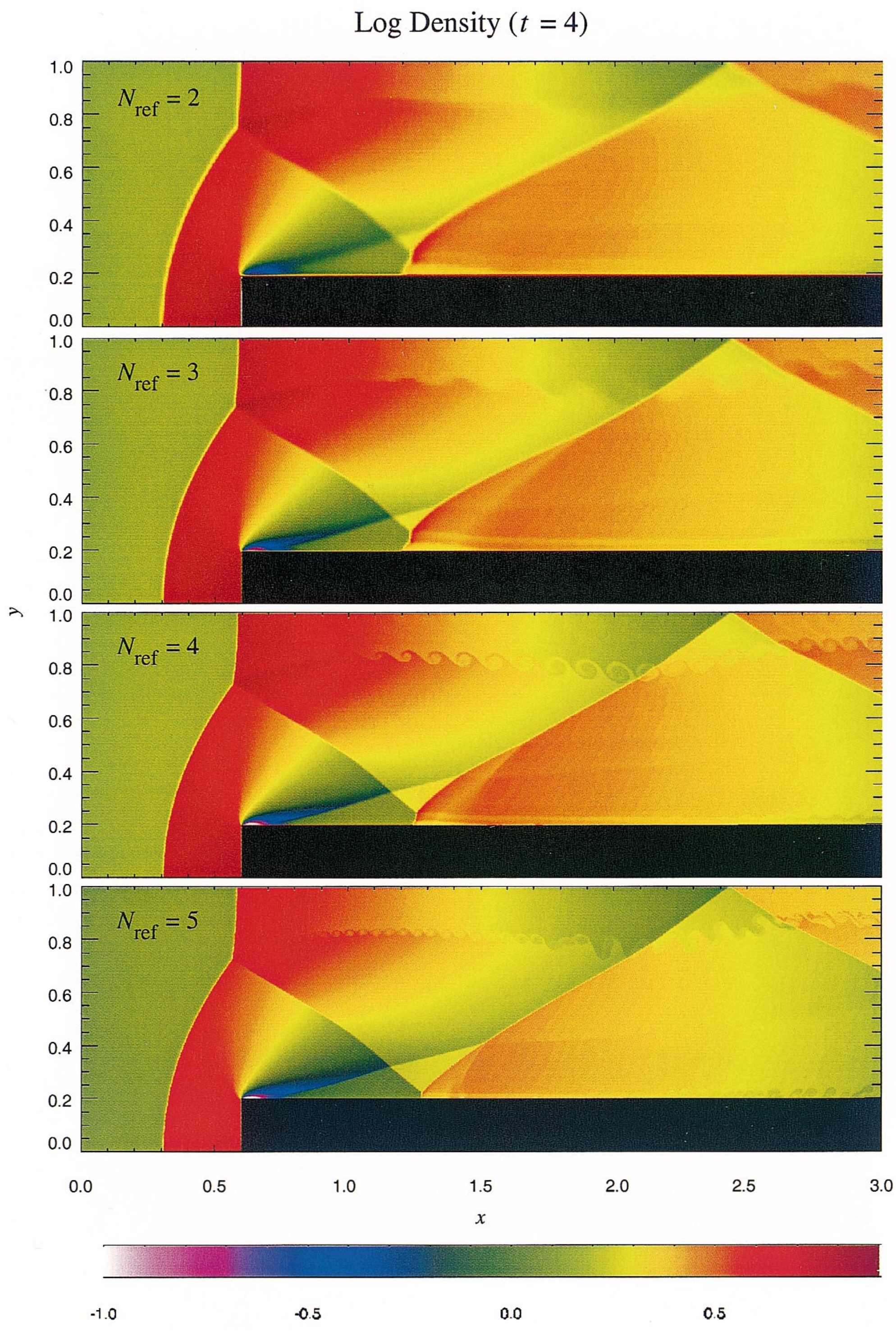

FIG. 36.—Density at $t = 4$ in the Emery wind tunnel test problem, as computed with FLASH using several different levels of refinement

refinement run perpendicular to any shocks that may cross the jumps in refinement. The errors incurred when a shock crosses a jump in refinement head-on appear small and well behaved. Larger errors appear, particularly the generation of reflected shocks, when the shock runs parallel to the refinement interface.

8. SOME PERFORMANCE MEASURES ON ASCI MACHINES

The FLASH code was designed to run on the three ASCI machines, the Cray T3E, Beowulf systems, and common workstations. The ASCI machines include Nirvana at Los Alamos National Laboratory, Red at Sandia National Laboratory, and Blue Pacific at Lawrence Livermore National Laboratory. Nirvana is a cluster of 16 SGI Origin 2000 computers, each of which contains 128 250 MHz R10000 processors and 32 GB of memory. This machine runs the IRIX operating system. Red consists of over 4000 nodes, each of which contains two Intel Pentium II Xeon processors and 256 MB of memory. This

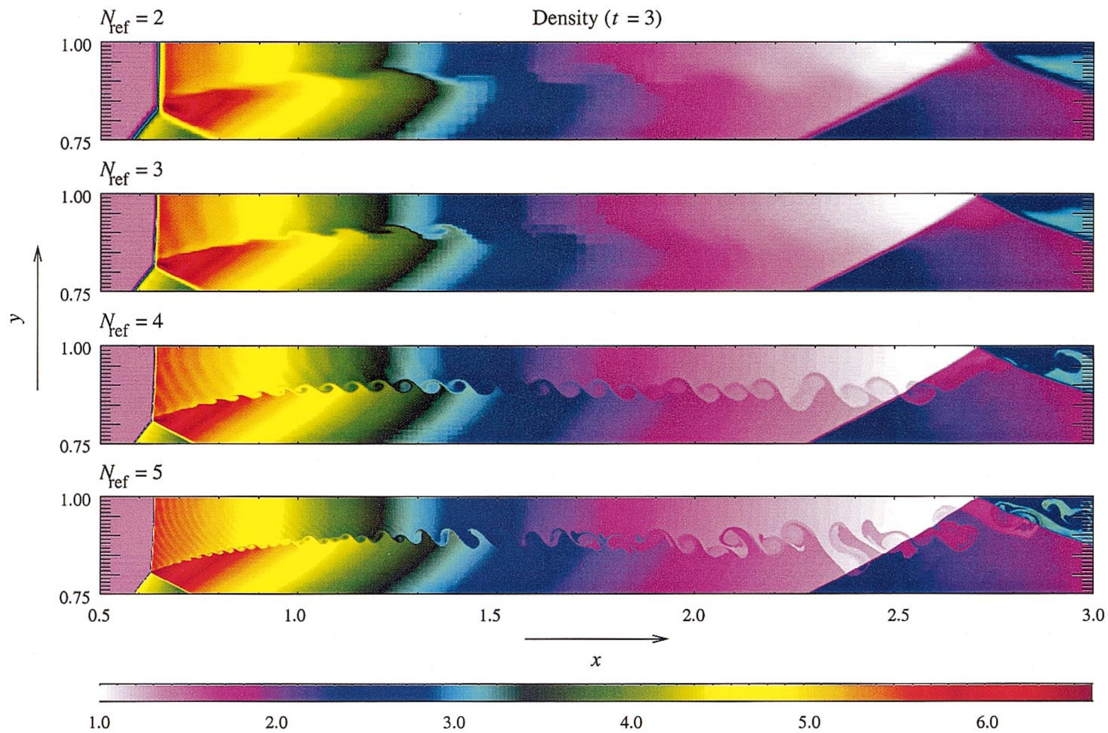

FIG. 37.—Detail of the Kelvin-Helmholtz instability seen at $t = 3$ in the Emery wind tunnel test problem for several different levels of refinement

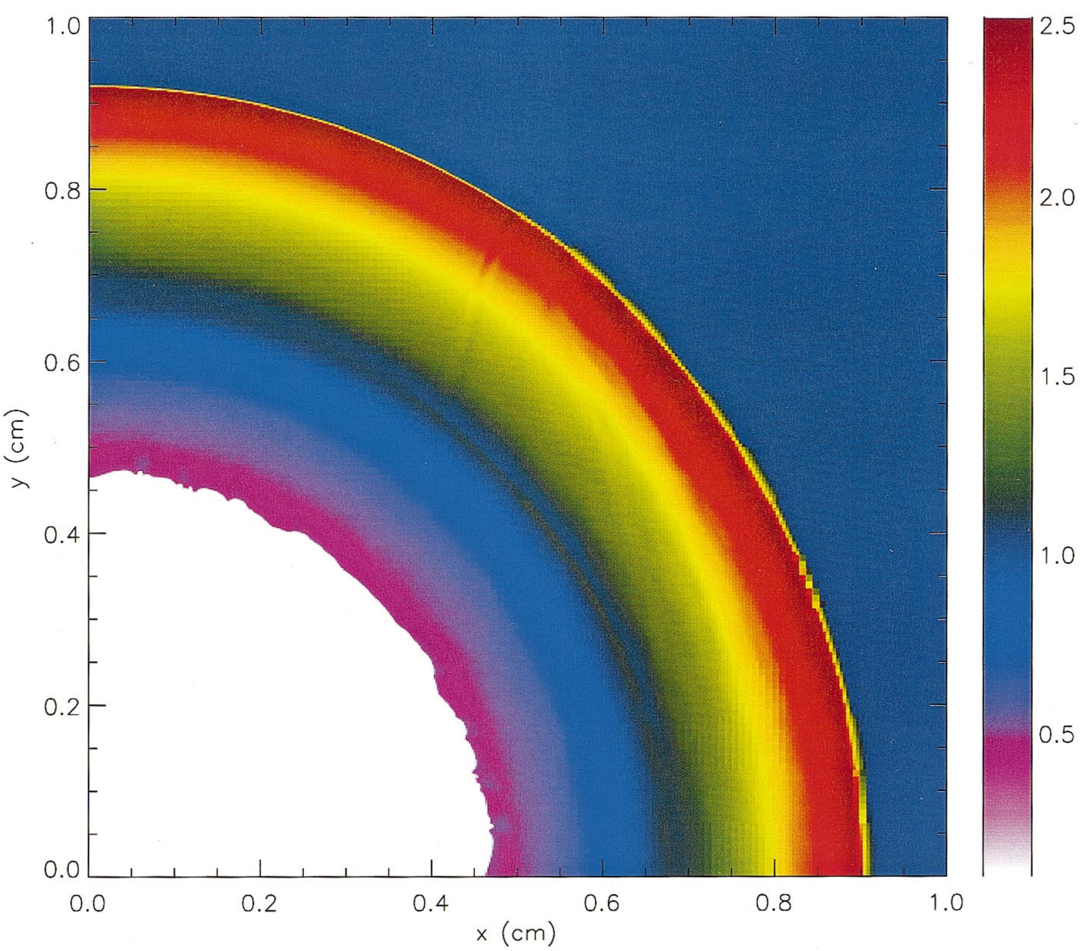

FIG. 38.—Density field at $t = 0.25$ s for a shock propagating through a forced jump in mesh refinement at $x = 0.5$. Seven levels of mesh refinement are used between $x = 0$ and $x = 0.5$, while only six levels of refinement are used for $x > 0.5$. The solution along the x-axis in the region with six levels of refinement is visibly coarser than the corresponding solution along the y-axis that has seven levels of refinement. A reflected shock wave is produced at the forced jump in refinement at $x = 0.5$.

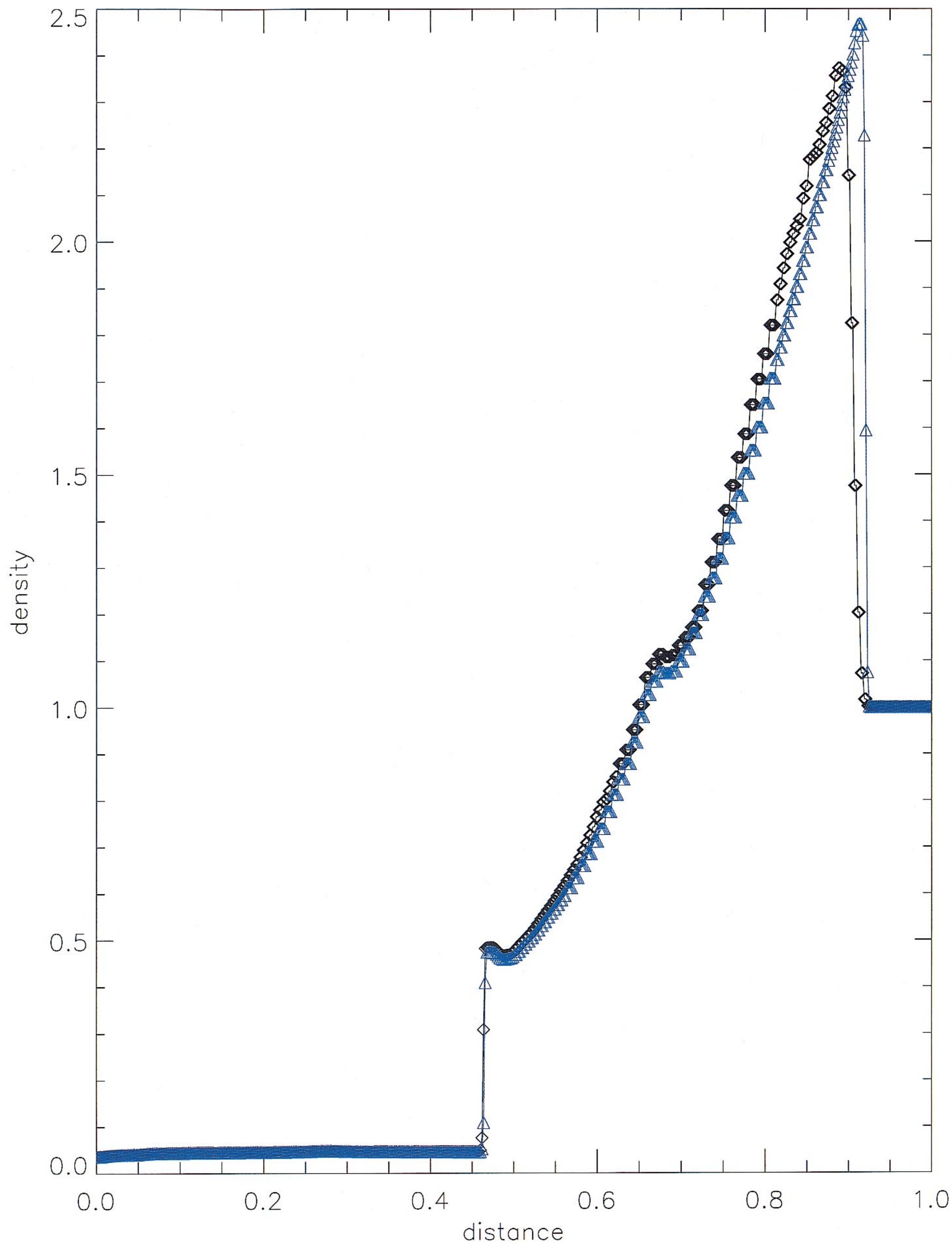

FIG. 39.—Density structure along the two coordinate axes of Fig. 38. The black curve corresponds to the solution along the x-axis, which has a forced jump in mesh refinement at $x = 0.5$. The blue curve corresponds to the solution along the y-axis, which does not have a forced jump in mesh refinement.

machine runs the Cougar operating system, and its compilers are produced by the Portland Group. Blue Pacific, built by IBM, consists of three 448 node sections. Each node of this machine contains four 332 MHz PowerPC 604e processors with 1.5–2.5 GB of memory.

8.1. *Uniform Grid Version of FLASH*

We have developed a uniform mesh version of the FLASH code in order to assess the overhead associated with the AMR algorithms in FLASH. The uniform mesh version achieves parallelism by assigning one block of cells to each processor. The size of each block is then adjusted according to the selected resolution and the number of processors selected. The uniform grid code eliminates many of the overheads associated with the AMR code (i.e., refinement testing, refinement or derefinement, and prolongation) since the mesh is fixed throughout a calculation and the blocks remain associated with the processor to which they are initially assigned.

Figure 40 shows the scaling of the uniform mesh code as a function of the number of processors for the Sedov explosion test problem (Sedov 1959) on the ASCI Red machine. The grid size in this test is fixed. The plot shows the expected scaling for a uniform mesh hydrodynamic code. Deviations from ideal scaling occur only when the communication cost associated with guard cell filling begins to compete with the computation cost associated with the hydrodynamics and EOS modules.

8.2. *Scaling with Constant Total Work*

Figure 41 compares the scaling of the uniform mesh version of the FLASH code with the AMR version on the Sedov test case used in Figure 40. The minimum zone size used in the AMR run was set equal to the zone size used in the uniform mesh run. For this type of test (i.e., constant amount of work), the AMR run does not scale as well as the uniform mesh run because the ratio of guard cell communication cost to computation cost increases as the number of processors increases. Once the number of processors increases beyond the number of blocks, execution time remains constant with increasing number of processors.

8.3. *Scaling with Constant Work per Processor*

A better test of the scaling of the AMR code is to run a case in which the amount of work any one processor has to do is fixed. For this test, we set up a planar shock problem (Sod 1978) in which all waves propagate in the x-direction. For each processor doubling, we increase the problem domain size by a factor of 2 in the y-direction. In Figures 42 and 43 we show the scaling results for this test case in two and three dimensions on ASCI Red. Also shown in these plots are the times required to perform the hydrodynamics calculations, the AMR tree manipulations, and the guard cell communication. In this test the ratio of communication cost to computation cost remains constant, so the AMR code scales well.

Figure 44 shows results of the same test as run on all three ASCI machines. A horizontal line indicates that the work per processor remains constant with increasing number of processors. On ASCI Red we find nearly perfect scaling up to 1024 processors when one processor per node is used. Both Blue Pacific and Nirvana show good scaling up to one-half the number of processors in a single shared-memory node; when attempting to use all of the processors in a node, competition with the

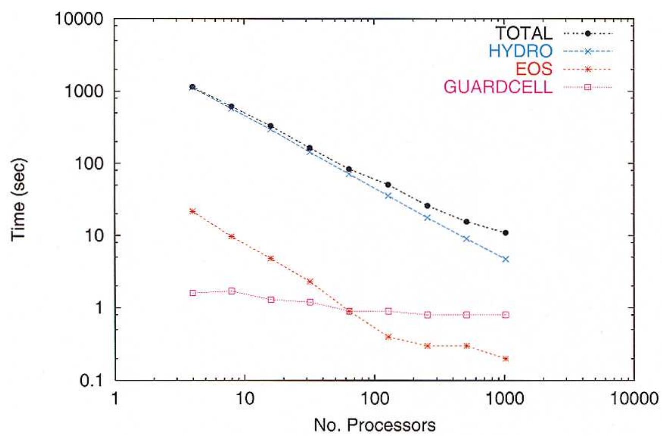

FIG. 40.—Scaling of the uniform mesh version of FLASH for a fixed problem size. The curve labeled " GUARDCELL " gives the number of CPU seconds required to fill the PARAMESH guard cells as a function of the number of processors. Similarly, the curve labeled " EOS " gives the time for the gamma-law EOS, and the curve labeled " HYDRO " is for the PPM hydrodynamic solver. The sum of these three curves is labeled " TOTAL." This Sedov test case was run for 20 time steps on the ASCI Red machine.

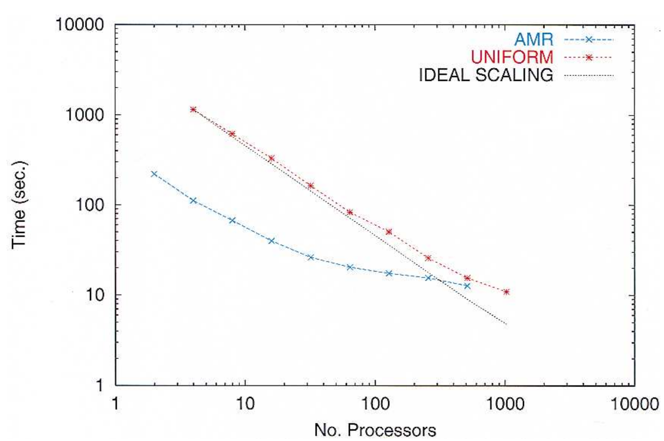

FIG. 41.—Scaling of the uniform mesh code (*red curve*) and the AMR code (*blue curve*) using the same Sedov test problem as in Fig. 40. The ideal scaling law is shown by the green line.

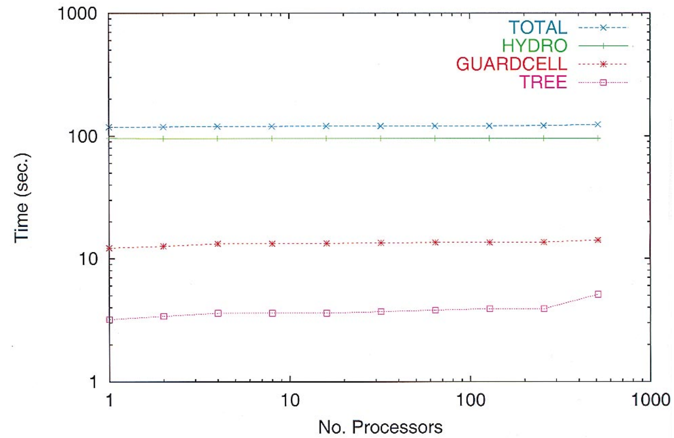

FIG. 42.—Scaling of the AMR code on a two-dimensional problem where the amount of work per processor remains constant. Results were produced using ASCI Red.

operating system begins to affect scaling. For Blue Pacific, we observe a steady loss of parallel efficiency when using more than one node. Nirvana displays a more dramatic loss of efficiency, with execution time increasing by more than a factor of 2 between 64 and 256 processors. Above 256 processors, the code again begins to scale correctly, although the lower bandwidth of the internode network keeps the total execution time much higher than for runs that fit within a node. The single-processor performance of the code varies between machines as a result of differences in processor speed and cache optimization.

8.4. *Overhead Associated with Adaptive Mesh Refinement*

In Figures 45 and 46 we show the execution time for the AMR code and the uniform mesh code on the Sedov problem in two and three dimensions, respectively. In these figures we plot the time required to perform 20 time steps versus the AMR

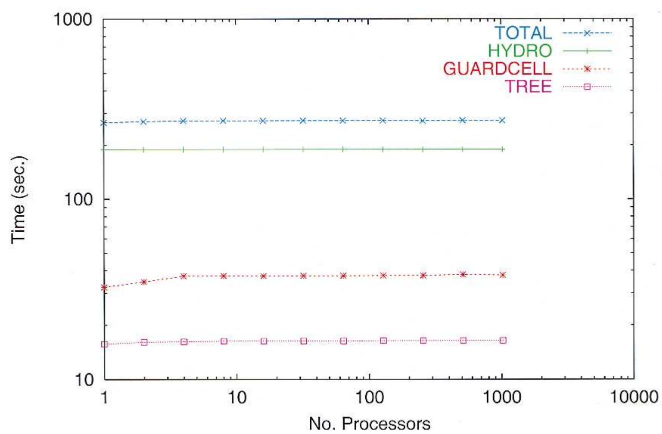

FIG. 43.—Scaling of the AMR code on a three-dimensional problem in which the amount of work per processor remains constant. Results were produced using ASCI Red.

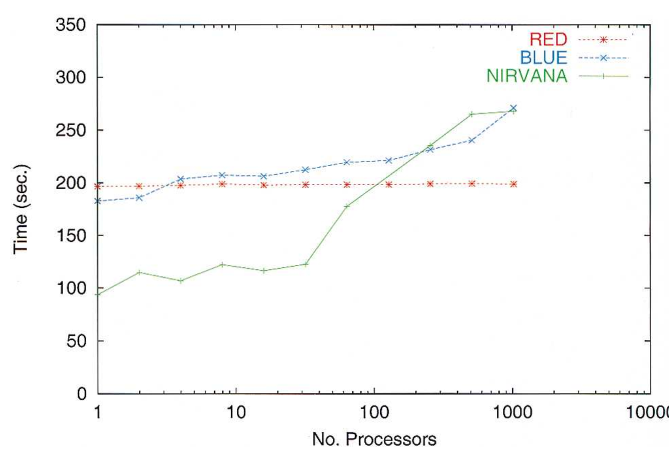

FIG. 44.—Scaling of the AMR code on all three ASCI machines. The problem run was the same as is shown in Fig. 43.

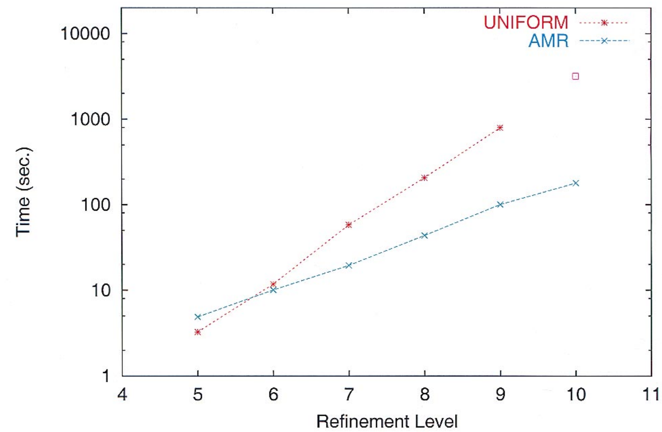

FIG. 45.—Comparison of execution times for the uniform mesh and AMR versions of FLASH as functions of the number of levels of AMR. A two-dimensional Sedov test problem is used on Nirvana.

level. The two-dimensional results were produced using 16 processors on Nirvana, while the three-dimensional results were produced using 1024 processors on ASCI Red. These plots show that beyond a refinement level of six (corresponding to a uniform mesh with 512 cells per side), the AMR code is faster than the uniform mesh code, with the speed difference increasing with increasing mesh resolution. The square symbols in these plots indicate times estimated for the uniform mesh code from its known scaling properties in cases for which available memory was insufficient for the uniform mesh.

We performed another test to determine the crossover point at which the speedup associated with AMR begins to justify the overhead associated with the AMR data structures. In this problem, we ran the Sedov test for a fixed number of time steps,

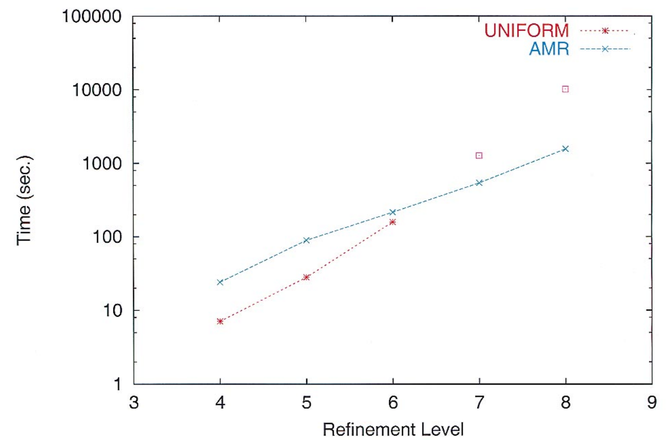

FIG. 46.—Comparison of execution time for the uniform mesh and AMR versions of the FLASH code. A three-dimensional Sedov test problem is used on ASCI Red.

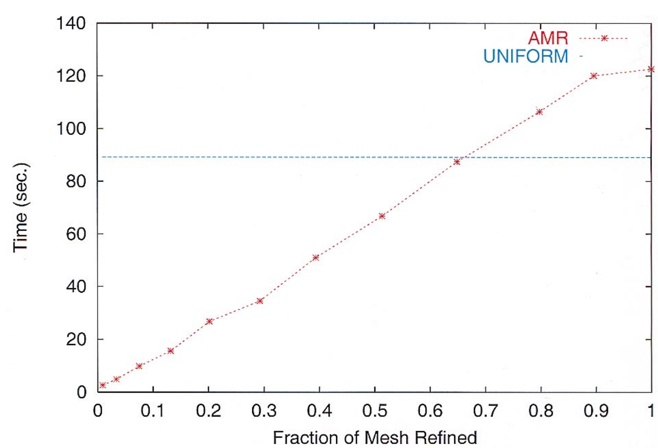

FIG. 47.—Execution time as a function of mesh refinement for the two-dimensional Sedov test problem. The blue dashed line shows the execution time using the uniform-mesh code. The fraction of the mesh refined refers to the fraction of the total domain that is maximally refined.

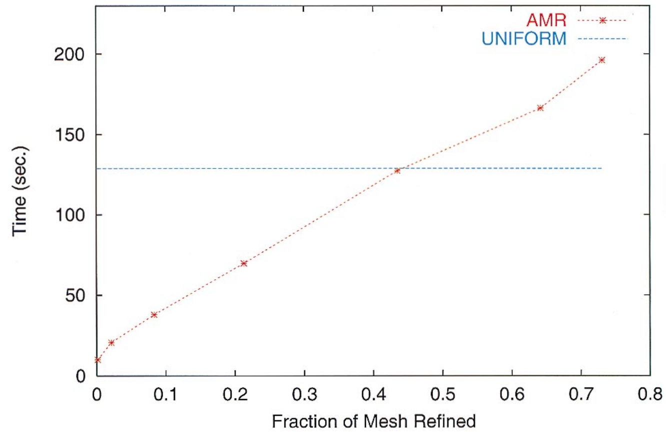

FIG. 48.—Same as Fig. 47, but for three dimensions

varying the fractional refined area or volume between runs. Figures 47 and 48 show the execution time in these runs versus the fractional mesh refinement. These plots suggest that at most 60% of the mesh in two dimensions, or 40% in three dimensions, can be maximally refined before the AMR overhead begins to outweigh the speed advantage of resolving only part of the computational volume.

These experiments define an operating point for FLASH. Refinement criteria for FLASH (and other AMR codes) must avoid excessive grid refinement (while still refining sufficiently for solution accuracy) in order to justify the overhead associated with adaptive mesh data structures. Defining such refinement criteria is likely to be highly problem dependent, and it is a significant challenge when using any adaptive mesh refinement technique.

9. SUMMARY

We have reported on the construction of the FLASH code. This code was designed to attack some of the most challenging outstanding computational astrophysics problems, namely, X-ray bursts, classical novae, and Type Ia supernovae. The code solves the fully compressible, reactive hydrodynamic equations, using a Godunov-type hydrodynamic solver. It uses adaptive mesh refinement and MPI to achieve parallelism. State-of-the-art modules for the equations of state and thermonuclear reaction networks are also included. We have constructed the FLASH code so that changes or replacement of any module is readily accomplished. In this paper we described the details of the FLASH code, including the physics modules, the code architecture, and the run-time environment of FLASH.

The first production version of the FLASH code has now been completed. We have reported the results of a suite of test problems and the performance of the FLASH code on the massively parallel ASCI machines.

Future versions of the FLASH code will include the addition of a variety of physics modules, including self-gravity, radiative transport, thermal conduction, flame front tracking, magnetohydrodynamics, and architectural refinements. The FLASH code, in its current implementation, can be used to solve a very broad class of astrophysics problems. The results of our helium burning on neutron stars, cellular detonations (Timmes et al. 2000a), Rayleigh-Taylor and Richtmyer-Meshkov instabilities, and thermonuclear flame front simulations will be presented in a series of forthcoming papers.

This work is supported by the Department of Energy under grant B341495 to the Center for Astrophysical Thermonuclear Flashes at the University of Chicago. The user's manual for FLASH and requests to obtain the FLASH code are available on—line from the Center's homepage.[11] The authors thank the anonymous referee for his/her comments and suggestions. We also thank Alan Calder, Carrie Clark, Scott Dodelson, Todd Dupont, Jonathan Dursi, Lori Freitag, Rusty Lusk, Mila Kuntu, Greg Miller, Ridgeway Scott, Greg Weirs, and other members of the ASCI FLASH center for useful discussions. We acknowledge the efforts of Ewald Müller in developing the PROMETHEUS code. Finally, we also thank our ASCI FLASH collaborators Dave Arnett, Ron Eastman, Rob Hoffman, Doug Swesty, and Stan Woosley for helpful input.

REFERENCES

Aparicio, J. M. 1998, ApJS, 117, 627
Arnett, D. 1996, Supernovae and Nucleosynthesis: An Investigation of the History of Matter, from the Big Bang to the Present (Princeton: Princeton Univ. Press)
Bader, G., & Deuflhard, P. 1983, Numerische Mathematik, 41, 373
Berger, M. J. 1982, Ph.D. thesis, Stanford Univ.
Berger, M. J., & Collela, P. 1989, J. Comput. Phys., 82, 64
Berger, M. J., & Oliger, J. 1984, J. Comput. Phys., 53, 484
Blinnikov, S. I., Dunina-Barkovskaya, N. V., & Nadyozhin, D. K. 1996, ApJS, 106, 171
Boisseau, J. R., Wheeler, J. C., Oran, E. S., & Khokhlov, A. M. 1996, ApJ, 471, L99
Boris, J. P., & Book, D. L. 1973, J. Comput. Phys., 11, 38
Burgers, J. M. 1969, Flow Equations for Composite Gases (New York: Academic)
Caughlan, G. R., & Fowler, W. A. 1988, At. Data Nucl. Data Tables, 40, 283
Chandrasekhar, S. 1939, An Introduction to the Study of Stellar Structure (Chicago: Univ. Chicago Press)
Chapman, S., & Cowling, T. G. 1970, The Mathematical Theory of Non-uniform Gases (Cambridge: Cambridge Univ. Press)
Cohen, E. R., & Taylor, B. N. 1987, J. Res. NBS, 92, 2
Colella, P., & Glaz, H. M. 1985, J. Comput. Phys., 59, 264
Colella, P., & Woodward, P. 1984, J. Comput. Phys., 54, 174
Davis, P. J. 1963, Interpolation and Approximation (Waltham: Blaisdell)
de Zeeuw, D., & Powell, K. G. 1993, J. Comput. Phys., 104, 56

[11] http://www.flash.uchicago.edu.

Dorband, J. E. 1988, in Proc. Frontiers of Massively Parallel Computation, ed. R. Mills (Washington: IEEE Comput. Soc.), 137
Duff, I. S., Erisman, A. M., & Reid, J. K. 1986, Direct Methods for Sparse Matrices (Oxford: Oxford Univ. Press)
Emery, A. F. 1968, J. Comput. Phys., 2, 306
Forester, C. K. 1977, J. Comput. Phys., 23, 1
Fowler, W. A., & Hoyle, F. 1964, ApJ, 91, 1
Fryxell, B. A., Müller, E., & Arnett, D. 1989, Hydrodynamics and Nuclear Burning (MPI Astrophys. Rep. 449; Garching: MPI Astrophys.)
Fryxell, B. A., & Woosley, S. E. 1982, ApJ, 258, 733
Garcia-Senz, D., Bravo, E., & Woosley, S. E. 1999, A&A, 349, 177
Glasner, S. A., & Livne, E. 1995, ApJ, 445, L151
Glasner, S. A., Livne, E., & Truran, J. W. 1997, ApJ, 475, 754
Godunov, S. K. 1959, Mat. Sbornik, 47, 271
Godunov, S. K., Zabrodin, A. V., & Prokopov, G. P. 1961, U.S.S.R. Comput. Math. & Math. Phys., 1, 1187
Hansen, J.-P., Torrie, G. M., & Vieillefosse, P. 1977, Phys. Rev., A16, 2153
Iben, I., Jr., Fujimoto, M. Y., & MacDonald, J. 1992, ApJ, 388, 521
Itoh, N., Hayashi, H., Nishikawa, A., & Kohyama, Y. 1996, ApJS, 102, 411
Kercek, A., Hillebrandt, W., & Truran, J. W. 1998, A&A, 337, 379
———. 1999, A&A, 345, 831
Khokhlov, A. M. 1995, ApJ, 449, 695
———. 1997, Memo 6406-97-7950 (Naval Res. Lab.)
Kohn, S., Garaizar, X., Hornung, R., & Smith, S. 1999, SAMRAI
Lapidus, A. 1967, J. Comput. Phys., 2, 154
LeVeque, R., & Berger, M. 1999, AMRCLAW
Livne, E. 1993, ApJ, 412, 634
Löhner, R. 1987, Comp. Meth. Appl. Mech. Eng., 61, 323
MacNeice, P., Olson, K. M., Mobarry, C., de Fainchtein, R., & Packer, C. 1999, NASA Tech. Rep. CR-1999-209483
———. 2000, Comput. Phys. Commun., 126, 330
Nadyozhin, D. K. 1974, Nauchnye informatsii Astron., 32, 33
Neeman, H. 1999, HAMR
Ogata, S., & Ichimaru, S. 1987, Phys. Rev., A36, 5451
Parashar, M. 1999, DAGH
Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. 1996, Numerical Recipes in Fortran 90 (Cambridge: Cambridge Univ. Press)
Quinlan, D. 1999, AMR++
Quirk, J. J. 1991, Ph.D. thesis, Cranfield Inst. Tech.
Sedov, L. I. 1959, in Similarity and Dimensional Methods in Mechanics (New York: Academic Press)
Shankar, A., & Arnett, W. D. 1994, ApJ, 433, 216
Shankar, A., Arnett, W. D., & Fryxell, B. 1992, ApJ, 394, L13
Slattery, W. L., Doolen, G. D., & DeWitt, H. E. 1982, Phys. Rev., A26, 2255
Sod, G. A. 1978, J. Comput. Phys., 27, 1
Steinmetz, M., Müller, E., & Hillebrandt, W. 1992, A&A, 254, 177
Swesty, D. 1996, J. Comput. Phys., 127, 118
Timmes, F. X. 1999, ApJS, 124, 241
Timmes, F. X., & Arnett, D. 1999, ApJS, 125, 294
Timmes, F. X., et al. 2000a, ApJ, 543, 938
Timmes, F. X., Hoffman, R. D., & Woosley, S. E. 2000b, ApJS, 129, 377
Timmes, F. X., & Swesty, F. D. 2000, ApJS, 126, 501
van Leer, D. 1979, J. Comput. Phys., 32, 101
Wallace, R. K., Woosley, S. E., & Weaver, T. A. 1982, ApJ, 258, 696
Warren, M. S., & Salmon, J. K. 1993, in Proc. Supercomputing (Washington DC: IEEE Comput. Soc.), 12
Weaver, T. A., Zimmerman, G. B., & Woosley, S. E. 1978, ApJ, 225, 1021
Williams, F. A. 1988, Combustion Theory (Menlo Park: Benjamin-Cummings)
Woodward, P., & Colella, P. 1984, J. Comput. Phys., 54, 115
Yakovlev, D. G., & Shalybkov, D. A. 1989, Soviet Sci. Rev. E. Astrophys. Space Phys., 7, 311
Zalesak, S. T. 1987, in Advances in Computer Methods for Partial Differential Equations VI, ed. R. Vichnevetsky & R. S. Stepleman (IMACS; New Brunswick: Rutgers Univ. Press), 15