PL-TR--91-1051

PL-TR--
91-1051

(2)

THE VAN LEER ADVECTION ALGORITHM IN THE MACH2 COMPUTER CODE

Carl R. Sovinec

August 1991

Final Report

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

PHILLIPS LABORATORY
Directorate of Advanced Weapons and Survivability
AIR FORCE SYSTEMS COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-6008

PL-TR-91-1051

This final report was prepared by the Phillips Laboratory, Kirtland Air Force Base, New Mexico, under Job Order 57971199. The Laboratory Project Officer-in-Charge was Capt Carl R. Sovinec (WSEP).

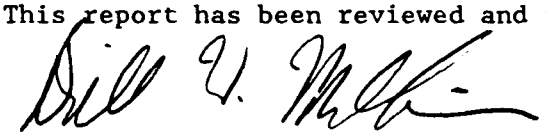
When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been authored by an employee of the United States Government. Accordingly, the United States Government retains a nonexclusive royalty-free license to publish or reproduce the material contained herein, or allow others to do so, for the United States Government purposes.

This report has been reviewed by the Public Affairs Office and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nationals.

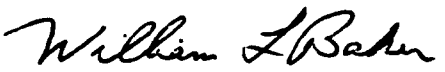
If your address has changed, please notify PL/WSEP, Kirtland AFB, NM 87117-6008 to help us maintain a current mailing list.

This report has been reviewed and is approved for publication.



BILLY W. MULLINS, Maj, USAF
Project Officer

FOR THE COMMANDER



WILLIAM L. BAKER
Chief, Electromagnetics Division

DO NOT RETURN COPIES OF THIS REPORT UNLESS CONTRACTUAL OBLIGATIONS OR NOTICE ON A SPECIFIC DOCUMENT REQUIRES THAT IT BE RETURNED.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1991	Final, 1 Dec 89 - 31 May 91

4. TITLE AND SUBTITLE

THE VAN LEER ADVECTION ALGORITHM IN THE MACH2 COMPUTER CODE

5. FUNDING NUMBERS

PE: 62601F
PR: 5797
TA: 11
WU: 99

6. AUTHOR(S)

Carl R. Sovinec

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Phillips Laboratory
Kirtland Air Force Base, NM 87117-6008

8. PERFORMING ORGANIZATION REPORT NUMBER

PL-TR--91-1051

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The van Leer advection algorithm has been added to the two-dimensional magnetohydrodynamics code, MACH2. The mathematical theory and the implementation for nonmagnetic quantities are discussed. One-dimensional test problems are presented to compare this algorithm with the Godunov advection algorithm, which was previously used in MACH2. The new implementation induces less smoothing on advected discontinuities, while maintaining the monotonic property of cell-centered distributions. A special treatment of the momentum advection is required to maintain a monotonic velocity distribution. The net practical result is a substantial savings on computational time as the van Leer algorithm requires less resolution to achieve the same level of convergence as the Godunov algorithm for dynamic problems.

14. SUBJECT TERMS

Numerical Convection, van Leer, Magnetohydrodynamic Simulation

15. NUMBER OF PAGES
46

16. PRICE CODE

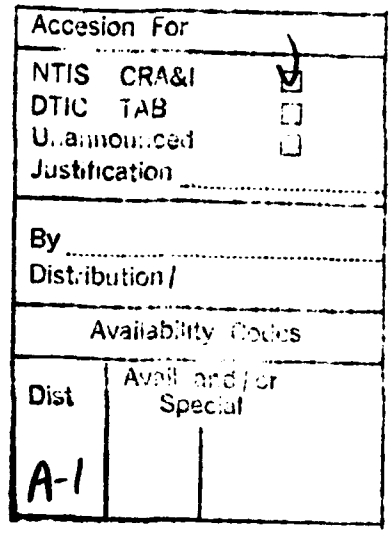
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

PREFACE

The code development and test simulations described in this report were supported by funding for the MARAUDER project in the High Energy Plasma Branch of the Phillips Laboratory. The advancements have been used to help simulate the physical phenomena in this and many other projects.

The author wishes to thank Dr. Jeremiah Brackbill of Los Alamos National Laboratory, Los Alamos, New Mexico, for suggesting this code development and for providing his implementation of it. Many of the details discussed in Section 3.0 were adapted from Dr. Brackbill's implementation. The author also wishes to thank Dr. David Dietz of the Phillips Laboratory for his patient and enthusiastic guidance on the analytical mathematics behind the Transport Theorem. Anthony Giancola of Mission Research Corporation, Albuquerque, New Mexico, modified the original coding to make it more readable and flexible to other changes. Finally, but certainly not least in significance, the author is grateful to the users of this code development, and in particular Dr. Robert Peterkin of Mission Research Corporation, who have played a crucial role of identifying problems with the early versions of the algorithm.

CONTENTS



1.0 INTRODUCTION

The MACH2 (Ref. 1) two-dimensional magnetohydrodynamics (MHD) computer code is an important tool for the research conducted in the High Energy Plasma Branch of the Phillips Laboratory.* Like all non-Lagrangian fluid simulation codes, its accuracy is dependent on the treatment of the advection terms of the difference equations. These terms carry physical properties such as density, internal energy and momentum with the fluid as it moves across the computational grid. MACH2 may be run in a purely Lagrangian mode, where the grid vertices move with the fluid, but this is rarely a good approach. Lagrangian computational grids usually become entangled when the fluid has variations in more than one dimension, and this causes the code to crash. MACH2 has an adaptive grid generator (Refs. 2 and 3) which can be used to create an almost Lagrangian grid that remains smooth. However, any grid that is not purely Lagrangian will depend on the advection algorithm. Further, using a purely Eulerian grid, whose vertices remain at fixed positions, is often the easiest approach for the initial simulations in a study, and it may be the only practical approach for a very complicated geometry. This report discusses an in-house project to replace the first-order accurate Godunov advection algorithm (Ref. 4) in MACH2 with a more accurate van Leer algorithm (Ref. 5).

Fluid simulation codes solve a set of coupled partial differential equations that are conservation statements for mass, momentum and energy. The solution procedure for these equations in MACH2 is an Arbitrary-Lagrangian-Eulerian (ALE) algorithm (Ref. 6). During each time step, it solves for changes in the physical quantities on a Lagrangian grid, creates a new grid with the adaptive grid generator, and then applies the advection terms separately to map the Lagrangian results onto the new grid. Separating the advection terms is not a new idea--see, for example, References 6 and 7, but the adaptive grid generator gives MACH2 much more flexibility than most codes.

*This code was developed under contract for the Air Force Weapons Laboratory--currently Phillips Laboratory--by Mission Research Corporation. It features a computational block structure that allows a user to model complicated geometries with ease. It has been applied to a large number of Air Force projects including plasma flow switches, cylindrical implosions, plasma guns, and plasma toroid experiments, and it has been applied to many other Department of Defense projects.

This report concentrates on the advection terms for mass, internal energy and momentum. Because MACH2 treats conducting fluids, it solves Faraday's law for the evolution of the magnetic field, and the appropriate form of this law has an advection term. However, this term requires special attention and will be addressed in a separate report. Section 2.0 of this report gives the analytical and numerical formulations of the advection terms. Section 3.0 explains how the numerical formulation is implemented in MACH2. Section 4.0 discusses test problems that illustrate the advantages of the algorithm. The final section concludes the report with a discussion of how the algorithm has influenced simulations of real problems.

2.0 MATHEMATICAL DESCRIPTION

2.1 ANALYTIC FORMULATION

The Transport Theorem can be used to find the analytic relation between the time-derivative of Lagrangian volume-integrals and the time-derivative of other volume-integrals. Marsden and Tromba (Ref. 8) state the scalar and vector forms of the theorem. Their definitions may be expanded in two ways without changing the results. They define a time-independent velocity vector field which describes the motion of fluid elements and integration domains that are composed of these elements. First, the velocity vector field may be time-dependent because it is not differentiated with respect to time in the proof. Second, the velocity vector field may describe an imaginary fluid--one that is not moving with the physical fluid, because no physical laws are applied. For a pertinent example of an imaginary fluid, consider the time-dependent vertex positions generated by an adaptive mesh algorithm to be markers on an imaginary fluid. With these definitions, the scalar and vector forms of the theorem are, respectively

$$\frac{d}{dt}\iiint_{\Phi} f\, d^3x\,\Big|_{t=\tau} = \iiint_{\Phi|_{t=\tau}} \left(\frac{\partial}{\partial t} f\,|_{t=\tau} + \bar{W}\cdot\nabla f\,|_{t=\tau} + f\nabla\cdot\bar{W}\,|_{t=\tau}\right) d^3x \qquad (1)$$

$$\frac{d}{dt}\iiint_{\Phi} f\,\bar{G}\, d^3x\,\Big|_{t=\tau} = \iiint_{\Phi|_{t=\tau}} \left[\frac{\partial}{\partial t} f\,\bar{G}\,|_{t=\tau} + (\bar{W}\cdot\nabla)(f\,\bar{G})\,|_{t=\tau} + f\,\bar{G}(\nabla\cdot\bar{W})\,|_{t=\tau}\right] d^3x \qquad (2)$$

where f is a scalar function of position, $\bar{x}$, and time, t, and $\bar{W}$ and $\bar{G}$ are vector functions of position and time. The functions f and $\bar{G}$ are arbitrary, but $\bar{W}$ is the velocity vector field of the real or imaginary fluid that carries the moving region Φ. The second and third terms in the integrand on the right side of Equations 1 and 2 may be combined into the divergence of a vector, $f\,\bar{W}$, for the former and the divergence of a tensor, $\bar{W}\,f\,\bar{G}$, for the latter. The Divergence Theorem may be applied to each equation with the following results

$$\frac{d}{dt}\iiint_{\Phi} f\, d^3x\,\Big|_{t=\tau} = \iiint_{\Phi}\Big|_{t=\tau}\frac{\partial}{\partial t} f\,\Big|_{t=\tau} d^3x + \iint_{\partial\Phi}\Big|_{t=\tau} \bar{n}\cdot f\,\bar{W}\,\Big|_{t=\tau} d^2x \qquad (3)$$

$$\frac{d}{dt}\iiint_{\Phi} f\,\bar{G}\, d^3x\,\Big|_{t=\tau} = \iiint_{\Phi}\Big|_{t=\tau}\frac{\partial}{\partial t} f\,\bar{G}\,\Big|_{t=\tau} d^3x + \iint_{\partial\Phi}\Big|_{t=\tau} f\,\bar{G}(\,\bar{W}\cdot\bar{n}\,)\,\Big|_{t=\tau} d^2x \qquad (4)$$

where $\bar{n}$ is the outward normal on the closed surface $\partial\Phi$.

If $\bar{W}$ is the physical fluid velocity, $\bar{V}$, then each of the above expressions relates the time derivative of a moving Lagrangian volume to the time-derivative of a corresponding stationary volume plus a surface flux term. Consider also the same two equations with $\bar{U}$, an imaginary fluid velocity, substituted for $\bar{W}$. The resulting expressions are related to the same stationary volume integrals as the Lagrangian equations, provided that the physical fluid region and the imaginary fluid region correspond at $t = \tau$. Subtracting the scalar relation (Eq. 3) for the Lagrangian volume from the same relation for the volume of imaginary fluid results in the following

$$\frac{d}{dt}\iiint_{\Psi} f\, d^3x\,\Big|_{t=\tau} = \frac{d}{dt}\iiint_{\Omega} f\, d^3x\,\Big|_{t=\tau} + \iint_{\partial\Psi}\Big|_{t=\tau} \bar{n}\cdot f(\,\bar{U}-\bar{V}\,)\,\Big|_{t=\tau} d^2x \qquad (5)$$

where Ω is the Lagrangian volume moving with the physical fluid, and Ψ is the volume moving with the imaginary fluid. The vector relation obtained from Equation 4 is

$$\frac{d}{dt}\iiint_{\Psi} f\,\bar{G}\, d^3x\,\Big|_{t=\tau} = \frac{d}{dt}\iiint_{\Omega} f\,\bar{G}\, d^3x\,\Big|_{t=\tau} + \iint_{\partial\Psi}\Big|_{t=\tau} (f\,\bar{G})[(\,\bar{U}-\bar{V}\,)\cdot\bar{n}]\,\Big|_{t=\tau} d^2x \qquad (6)$$

2.2 NUMERICAL FORMULATION

The difference forms of Equations 5 and 6 are the equations solved in the advection step in MACH2 using $\bar{U}$ as the velocity field of the grid as suggested in the example above. The

difference form of the time-derivative terms are simple. Each is the difference between a new volume integral and an old volume integral, divided by the time increment, and a volume integral is the average of a quantity in a cell, multiplied by the volume of the cell. Because the Lagrangian cells and the mesh cells are equivalent prior to the Lagrangian step, these "old" volume integrals may be eliminated from the difference equations. The Lagrangian step of the momentum algorithm produces the "new" averages in the Lagrangian cells, so the new Lagrangian volume integrals are known at the start of the advection step. After multiplying by the time increment, Δt, the following difference form of Equation 5 results.

$$(f_\Psi \upsilon_\Psi) = (f_\Omega \upsilon_\Omega) + \Delta t \sum_s \langle a\, \bar{n} \cdot (\bar{U} - \bar{V})\, f \rangle_s \tag{7}$$

where f_Φ is the new average value of f in the cell and υ_Φ is the new cell volume for either the Lagrangian cell, where the index $\Phi = \Omega$, or the grid cell, where $\Phi = \Psi$. The summation on the right side represents the flux of f over the cell surface for one advection step. The sides are indexed by s, and each has a surface area, a. The difference form of Equation 6 is

$$(f_\Psi\, \bar{G}_\Psi \upsilon_\Psi) = (f_\Omega\, \bar{G}_\Omega \upsilon_\Omega) + \Delta t \sum_s \langle a\, \bar{n} \cdot (\bar{U} - \bar{V})\, f\, \bar{G} \rangle_s \tag{8}$$

where $\bar{G}_\Phi$ is the new cell average of $\bar{G}$. After solving for the left side of Equation 7 or 8, one may divide by the new grid cell volume to obtain the desired new grid cell average. As van Leer points out, these equations are exact (Ref. 5), and the accuracy of an algorithm depends on the average flux terms inside the summation. Note that when the grid moves with the fluid, $\bar{U} = \bar{V}$, and the Lagrangian values remain unchanged. The grid is stationary when $\bar{U} = 0$, and in this case, the two equations form an Eulerian representation.

To understand the van Leer approach, it is easiest to start with the Godunov or 'donor cell' method. For the latter, the average flux terms in Equations 7 and 8 are rather simple. Many fluid codes have velocities centered between cells, so Newton's Second Law may be written in a centered difference form with pressures being cell-centered (along with mass and internal energy). Thus, the velocities and areas in the flux terms are located at the interfaces. To construct the entire flux term, Godunov considers the cell-centered quantities to be constant within each cell. In one-dimension the representation is a set of slabs, and one example is illustrated in Figure 1. The flux at each interface becomes the interface velocity multiplied by the magnitude of the slab from which the flow comes, hence the name 'donor cell.' The method is explicit because the velocities and slab magnitudes used in the flux terms are those at the

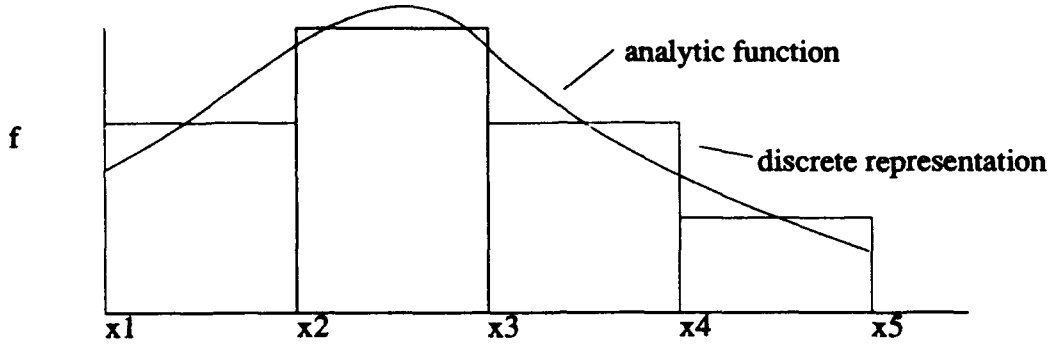


Figure 1. Godunov representation of a cell-centered quantity on a on-dimensional grid.

beginning of the advection step. It is also conservative because the flux that comes out of the donor cell goes into the adjacent cell.

When describing the explicit nature of a method, one is tempted to refer to the beginning of the time step, but this would be misleading. Simulation codes often treat physical processes independently during each time step. This suggests a linearization in time, which is reasonable as long as the time step is small. In MACH2 the changes from each physical process are added sequentially. Furthermore, for any ALE code, the advection step is a mapping from the Lagrangian grid, so the Lagrangian cell-centered values are the explicit values during the advection step.

To obtain more accuracy than the Godunov method, the van Leer technique replaces the slab approximation of the distribution with a better approximation (Ref. 5). It uses derivatives to make the distribution a set of trapezoids instead of slabs (Fig. 2). This representation is piecewise continuous, like the slab representation, and the correct cell averages are preserved.

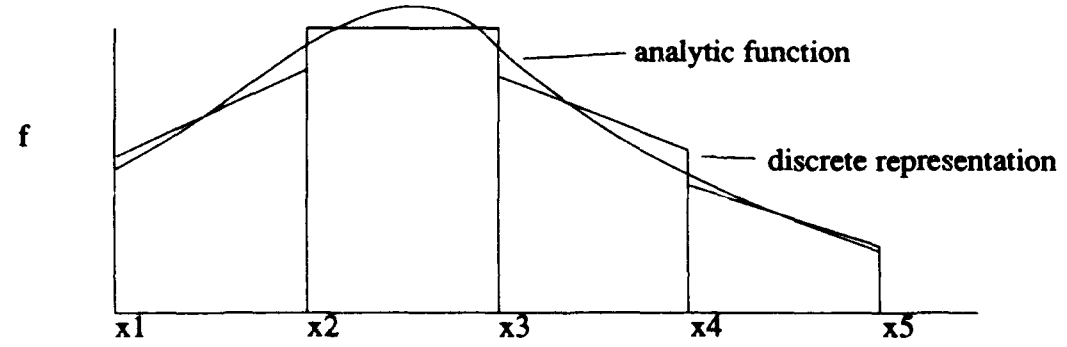


Figure 2. Van Leer representation of a cell-centered quantity on a one-dimensional grid.

Van Leer proposes several possible formulations for the derivatives. In the simplest scheme, the upwind cell-centered quantity and its corresponding centered difference are used to construct the flux terms. Any subsequent mention of the "centered-difference scheme" refers to this approach. The scalar flux term from Equation 7 with this scheme is

$$\langle a\,\bar{n}\cdot(\bar{U}-\bar{V})f\rangle_s = a\zeta_s\left[f_\Omega \pm \frac{1}{2}\left(1-\frac{\zeta_s\Delta t}{\Delta x}\right)\Delta f_\Omega\right] \tag{9}$$

where f_Ω is the upwind cell-centered quantity, Δf_Ω is its centered difference with respect to the advection direction, Δx is the cell dimension in the advection direction, and

$$\zeta_s = |\,\bar{n}\cdot(\bar{U}-\bar{V})\,|_s$$

The sign of the Δf_Ω term in Equation 9 is positive when the s-interface is the upper bound of the upwind cell and negative when it is the lower bound. When the time step is limited by the Courant-Friedrichs-Lewy (CFL) condition, $|\zeta_s(\Delta t/\Delta x)| \le 1$, the term in the parentheses on the right side is bounded by zero and one, so this scheme limits to the donor cell method as $|\zeta_s(\Delta t/\Delta x)|$ approaches unity. The vector equation corresponding to Equation 9 has $f\,\bar{G}$ in place of f. For multiple dimensions, there will be multiple centered differences for each quantity.

Van Leer derives the amplification factor for this scheme with uniform grid spacing and velocities,

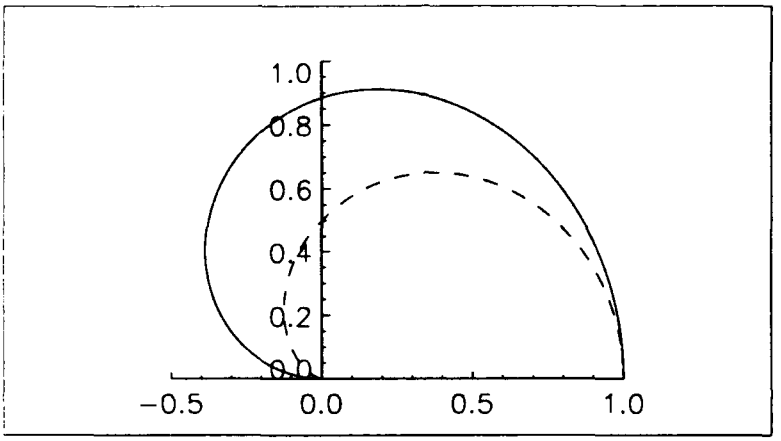
$$g_1 = 1-\sigma\left(\frac{1+\sigma}{2}-\frac{1-\sigma}{2}\cos\alpha\right)(1-\cos\alpha)$$

$$-\,i\sigma\left(\frac{3-\sigma}{2}-\frac{1-\sigma}{2}\cos\alpha\right)\sin\alpha \tag{10}$$

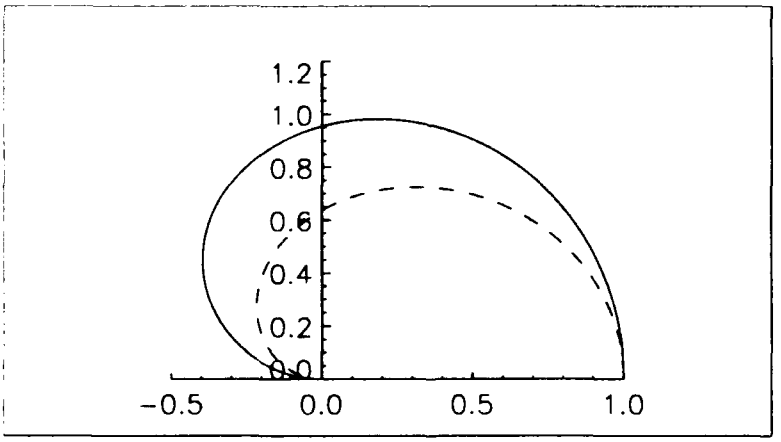
where $\sigma = \zeta(\Delta t/\Delta x)$, and the index is omitted to indicate the uniformity (Ref. 5). The angle $\alpha = 2\pi\Delta x/l$, and l is the length of a wave moving across the grid. The dissipation error per time step is one minus the magnitude of the amplification factor, and it has a maximum at $\sigma = 1/2$ (Ref. 5). Thus, for $\sigma = 1/2$ and $\alpha = \pi/2$, the centered-difference scheme has a dissipation error of 0.12 per time step. The amplification factor for the Godunov method is

$$g_{DC} = 1 - \sigma(1 - \cos\alpha) - i\,\sigma \sin\alpha \qquad (11)$$

Its dissipation error for $\sigma = 1/2$ and $\alpha = \pi/2$ is 0.5 per time step, so even the simplest van Leer scheme is a considerable improvement for intermediate length waves. The dispersion error is measured by the ratio of the numerical advection speed to the true advection speed, $\omega = \arg(g) / (-\sigma\alpha)$. Polar plots of $|g|$ at $\sigma = 1/2$ and ω in the limit of vanishing σ for both the centered-difference scheme and the Godunov method are shown in Figure 3. A sufficient



(a) Magnitude of amplification factors at $\sigma = 1/2$.



(b) Ratio of numerical advection speed to true advection speed in the limit of vanishing σ.

Figure 3. Performance of the advection methods. The solid line illustrates the centered difference scheme, and the dashed line illustrates the Godunov method. The angle from the positive horizontal axis is $\alpha = 2\pi\Delta x/1$.

condition for stability is that $|g| \leq 1$, for if this were not satisfied, repeated applications of the advection step would force wave amplitudes to grow geometrically. Both the Godunov and centered-difference scheme meet this criterion when the CFL condition is satisfied. Although it is not accurate to make generalizations about the stability of difference equations, the upwind nature of an advection scheme tends to add stability, and both of these schemes have an upwind nature.

Besides accuracy and stability, the monotonicity of a distribution should be respected by the advection algorithm. To quote van Leer (Ref. 5), "The monotonicity condition says that, when a monotonic initial value distribution is numerically convected, the resulting distribution must be monotonic again." This is enforced by placing limits on the centered difference, Δf_{Ω}, in Equation 9. The limits prevent the linear distribution of a quantity within a cell from exceeding the cell-centered average of that quantity in the adjacent cells. Also, if a cell-centered average is not between those of the adjacent cells, the slab representation is used. This prevents the development of new extrema. The representation in Figure 2 is properly limited.

The centered-difference scheme requires only a small amount of additional computation time over the Godunov method, most of which is spent on finding the centered differences. These differences are calculated during the advection algorithm and do not require permanent storage. The other second-order schemes proposed by van Leer are based on derivatives that are computed separately from the cell-centered quantities. These derivatives require separate storage, and they must be updated during all of the other algorithms that change the corresponding cell-centered quantities. The accuracy analysis in Reference 5 shows that some of the more complicated schemes can track waves as short as two cell lengths with very little dissipation and dispersion, whereas the scheme with the centered differences loses accuracy for wavelengths less than four cell lengths. However, for enhancing an existing fluid code, it is far easier to add the centered-difference scheme to the advection algorithm than it is to rewrite the entire code to track derivatives. Therefore, the centered-difference scheme has been added to MACH2.

3.0 IMPLEMENTATION

3.1 PRELIMINARIES

For a simple one-dimensional advection problem with uniform velocities and surface areas between cells, the terms of Equation 9 have been sufficiently defined. For anything more complicated, they are ambiguous. The resolution of the ambiguities is a code-dependent issue. This section will address this issue for version v9101 of MACH2, and it will provide a guided tour through the subroutines for those who use and modify the code.

The subroutine ARUN contains the main loop of MACH2 that calls separate routines for each of the physical processes. The subroutine HYDRO, which is called from ARUN, calculates the Lagrangian stage of the ALE algorithm. Following HYDRO, ARUN calls REMESH. The first part of this subroutine calls the adaptive mesh generator, and the second part calls TRNSPT, the advection algorithm, to complete the ALE algorithm.

When MACH2 is used for planar geometries, the x-y plane is the computational plane, and no variations are allowed in the perpendicular direction. For axisymmetric geometries, the r-z plane forms the computational domain. For either case, TRNSPT calculates the advection that results from velocity components in the computational plane. Other advection terms that result from the θ-component of velocity are also nonzero in axisymmetric geometries. These terms are treated at the end of HYDRO and are not considered in TRNSPT.

To simulate complicated geometries with MACH2, the spatial domain is decomposed into four-sided blocks--the reader is encouraged to see Reference 1 for more information on allowed domains. The blocks are divided into quadrilateral cells which form the computational grid. Cells have a horizontal index, i, and a vertical index, j. Each physical algorithm solves or iterates its process on one block at a time, and boundary conditions couple adjacent blocks. The spatially-dependent physical quantities are stored in "pointered" memory location; i. e., the POINTER extension of standard FORTRAN is used to set the two-dimensional arrays for the physical quantities to the appropriate set of memory locations for each block. This conserves memory because the dimension of pointered arrays can vary from block to block. However, the pointers and dimensions must be set before the arrays can be correctly accessed. Therefore, all of the physical algorithms will contain "do-loops" over the blocks, and the first call is always to the subroutine SETBLK which sets the pointers and dimensions. These loops will be mentioned

frequently in the description of the advection algorithm below.

3.2 CELL-CENTERED QUANTITIES

The advection of the cell-centered quantities is not complicated. The first block loop of TRNSPT calls the subroutine TRNSINIT--see the Appendix for a listing of the nonmagnetic advection subroutines. The first loop of TRNSINIT creates the Lagrangian-cell volume integrals, each of which forms the first term on the right side of Equation 7 or 8 for the volume of one cell. For example, when mass is advected, this integral is simply the total mass of the Lagrangian cell. This is also the same as the mass of the cell prior to the Lagrangian phase, which is the old mass density, stored in the "ro" array, multiplied by the old cell volume, "oldvol." For the internal energy, the integral is the total internal energy in the cell. This is the Lagrangian cell mass just computed, "mp," multiplied by the specific internal energy after the Lagrangian stage, "sel." This loop also initializes the Lagrangian densities where necessary. For internal energy, this quantity is the internal energy per unit volume. It is found by multiplying the Lagrangian mass density, "rol," by "sel," and is stored back in "sel" array.

This TRNSINIT loop also defines the Lagrangian cell volumes, the "lagvol" array, and the volumes exchanged between these cells during the mapping to the new grid. The exchange volumes are determined with the cross product of two vectors, the grid velocity relative to the fluid and the displacement vector from one vertex of the new-grid cell to its next vertex. The relative velocities define the $\overline{U} - \overline{V}$ vector in Equation 9 and are the vertex-centered ("url","vrl") array pair. Figure 4 illustrates these vectors for the bottom exchange volume, "dxbott." The dimensions on the cross products are area per unit time. They are multiplied by the time increment "dt" and an appropriate perpendicular dimension to form a volume. For planar geometries, the radius array, "r," is set to unity, so the volume is per unit depth perpendicular to the computational plane. For axisymmetric geometries, the radius is factored into the relative velocities to create a volume per unit angle--see the TRNSINIT listing in the Appendix for the formulation. These exchange volumes take the place of the $a\zeta_s$ factor on the right side of Equation 9.

The "200" loop of TRNSINIT separates the "con2" fraction of marker material to advect its mass separately from the rest of the mass. The "300" and "400" loops compare the size of the exchange volumes to the Lagrangian and new-grid volumes and saves the largest ratio for the time step control.

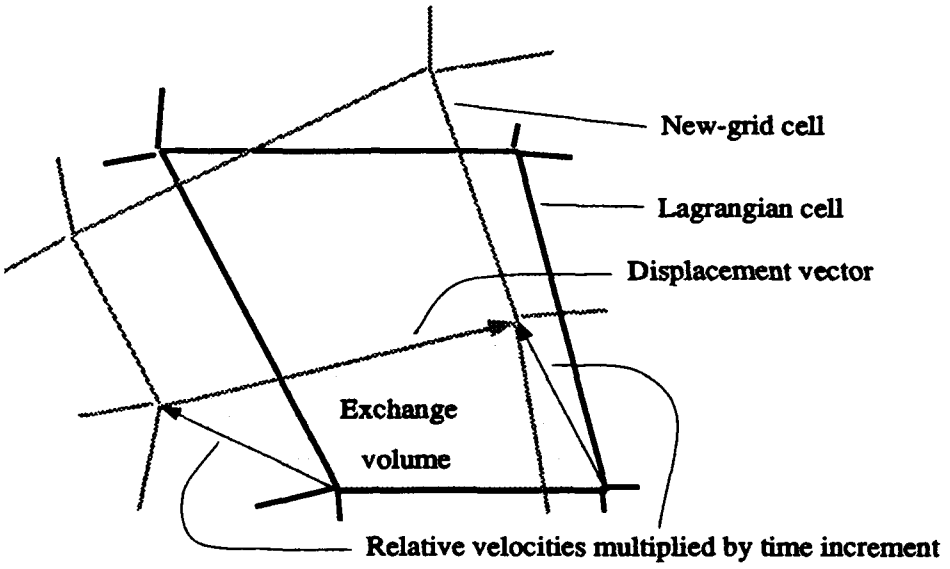


Figure 4. Illustration of the "dxbott" exchange volume between a Lagrangian cell and a new grid cell.

Once the TRNSINIT loop of TRNSPT is complete, the cell-centered quantities are separately passed into the subroutine TRNSLP. The first block loop in TRNSLP calls TRNSGR which finds the centered differences of the quantity passed into the routine. Note that these differences are differences of the Lagrangian densities. The "200" loop of TRNSGR finds the centered difference in the j-index direction, and the "300" loop finds the difference in the i-index direction. Each are limited to twice the corresponding backward and forward differences for monotonicity, and if the signs of those differences do not agree, the centered difference is reduced to zero. This implementation is the monotonicity algorithm of Equation 66 in Reference 5. The more conservative algorithm of Equation 67 in Reference 5 is in the current version of MACH2, v9101.

The second block loop of TRNSLP calls TRNSDQBC, which communicates the differences along the boundaries of adjacent blocks, and subsequently calls TRNSADV, which performs the advection. The "100" loop of TRNSADV advects the quantity in the j-index direction, and the "200" loop advects it in the i-index direction. They create the flux of the quantity on the bottom side and left side of each cell, respectively. This is the application of Equation 9 for scalar quantities. For vector quantities, each component is separately passed into TRNSLP. The ratio of the volume flux to the donor-cell Lagrangian volume in TRNSADV replaces the $\zeta_s(\Delta t/\Delta x)$ in

Equation 9. Both ratios represent the fraction of the cell advected, but the volume ratio automatically accounts for arbitrary cell shapes. After finding the flux, these loops remove it from the donor-cell's Lagrangian volume integral and add it to the adjacent cell's integral. When the TRNSADV loop is complete, the integral array holds the new-grid integrals, each of which is the left side of Equation 7 or 8 for the new-grid cell.

3.3 MOMENTUM

The advection of momentum is more difficult than the advection of cell-centered quantities. The velocities are centered at the grid vertices and not the cell centers, and to use the cell-centered scheme, one must first create cell-centered momenta. Margolin and Beason have suggested creating cell-centered quantities that are the average and derivatives of the surrounding vertex quantities (Ref. 9). The current implementation in MACH2 is similar to this approach. For each velocity component, it creates four cell-centered momentum densities which are the products of each of the four vertex velocities and the cell-centered mass density. Following the advection, the four resulting cell integrals are distributed into vertex momenta. To create the new vertex velocity, each vertex momentum is divided by the vertex mass, which is the average of the four cell masses surrounding the vertex. This is similar to the scheme in Reference 9, for if one used the average and three possible differences instead of the average and three possible derivatives, the advection is algebraically equivalent to the MACH2 scheme.

Unlike the cell-centered quantities, TRNSLP is not called directly from TRNSPT. Instead, TRNSPT calls the subroutine TRNSMM. This subroutine first calls TRNSMMIN to create the four momentum densities and Lagrangian momentum integrals for each component. TRNSMM then calls TRNSLP for each of the four. Before returning to TRNSPT, the subroutine TRNSMMF is called to distribute the new-grid momentum integrals among the vertices. The new velocities are calculated after TRNSPT during the REMESH call to RMSHVEL.

The monotonicity of the momentum advection also needs special attention. Although the monotonicity algorithm in TRNSGR will not create new extrema in the resulting momentum distribution, it does not guarantee the same for the resulting velocity distribution. Consider a situation where the mass density distribution is monotonic, and the gradient is in the direction of a uniform flow in the computational plane. In addition, there is momentum density perpendicular to the computational plane, and its distribution is not monotonic. The centered difference will be used for the advection of the mass and the fraction of mass removed from the

donor cell will be larger than the fraction of volume removed. However, the fraction of perpendicular momentum removed from the donor cell will be equivalent to the volume fraction, and the resulting velocity distribution will show a new maxima. The results of a simulation with these conditions is presented in Section 4.0. It has been found that when new maxima develop in the velocity components that are in the computational plane, a numerical instability may result.

One prescription to avoid the instability is to discard the difference of the momentum density and use the difference of the mass density, multiplied by the vertex velocity, instead. With this prescription, the amount of momentum advected is proportional to the amount of mass advected. Mathematically, this uses the product rule on the derivative of the momentum,

$$\frac{\partial (\rho v)}{\partial x} = \rho \frac{\partial v}{\partial x} + v \frac{\partial \rho}{\partial x} \tag{12}$$

where ρ is the mass density and v is a velocity component, and throws away the first term on the right before converting to a difference form. This product is formed during the call to TRNSMMGR in the second block loop of TRNSLP.

Another possible prescription to avoid the creation of new velocity extrema is to difference both terms on the right of Equation 12 and apply the monotonicity algorithm to each term separately. This scheme has also been attempted with MACH2, and although it advects velocity gradients with less diffusion, it seems to be noisy. One-dimensional advection problems develop enough noise to have noticeable two-dimensional variations.

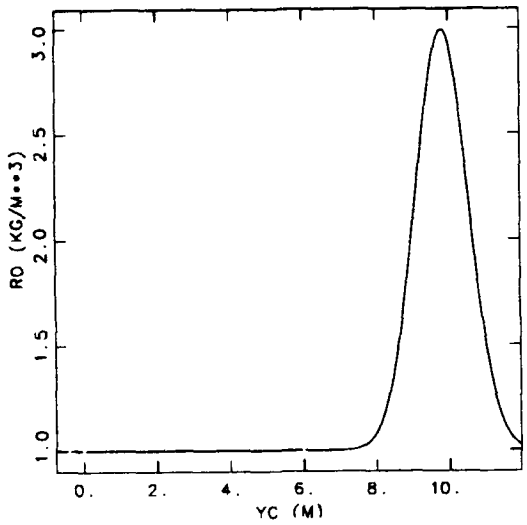
4.0 RESULTS

There are two one-dimensional test problems that are often used to evaluate the performance of an advection algorithm. The first is the uniform advection of a square pulse of some quantity, and the second is the shock tube. The Fourier Transform of a square pulse is an oscillating, continuous function, and the magnitude of the oscillations is inversely proportional to the wave number and does not diminish abruptly. For the test problem, the initial pulse is formed with an integral number of cells. Thus, when the initial pulse has relatively few cells, the problem will exercise an algorithm's ability to advect high frequency waves in a manner that is relevant to simulations of actual experiments and physical phenomena.

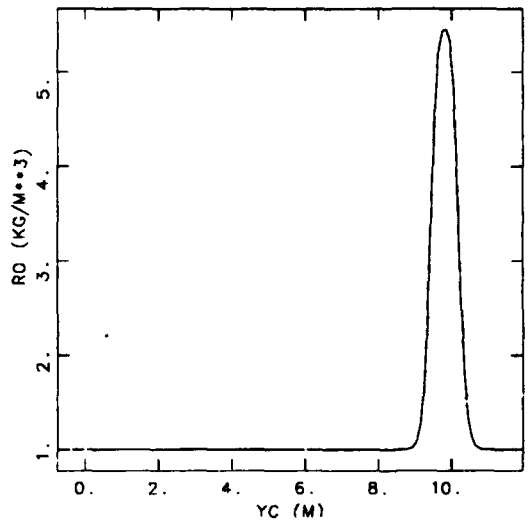
The pulse test problem described here is initiated in the following manner. The domain is a long rectangular chamber, 0.4 m in width and 12.8 m in length, which is divided into 512 square cells. The fluid is given a uniform velocity component of 1 m/s in the long dimension. The pulse is positioned from 0.4 m to 0.8 m from the left side of the chamber, so it is initially four cells long. It is composed of mass that is 10 kg/m^3 in density, which is a factor of 10 larger than the density in the rest of the chamber. It is given a temperature of 10^{-15} eV, also a factor of 10 above that outside the pulse. The extreme temperatures are chosen to make the sound speed very small compared with the advection velocity. Finally, the time step is limited so that $\sigma \leq 1/2$.

Figures 5 and 6 show the mass density and temperature distributions, respectively, for the centered-difference scheme with monotonicity and the Godunov method, after the pulses travel across 100 cells. Although both algorithms diffuse the small pulse, the peak mass with the centered-difference scheme is twice that obtained with the Godunov method, and the pulse width is much less. The temperature pulses have a different shape from the density pulses and maintain relatively larger peaks. This occurs because the temperature is the quotient of two advected quantities, internal energy and mass. In this case, the initial internal energy pulse is two orders of magnitude larger than the background. Note that the algorithm will not create new extrema in the temperature distribution because the monotonicity algorithm prevents the creation of new extrema in both the internal energy and mass distributions.

A variation of the pulse problem can illustrate the difficulty with momentum. Consider, again, a rectangular chamber, with a mass density of 1 kg/m^3 in half of the chamber and 10^{-3} kg/m^3 in the other half. The velocity in the computational plane is a uniform 1 m/s towards the side with the greater density, and the less dense side has a perpendicular velocity component of 1 m/s. Thus,

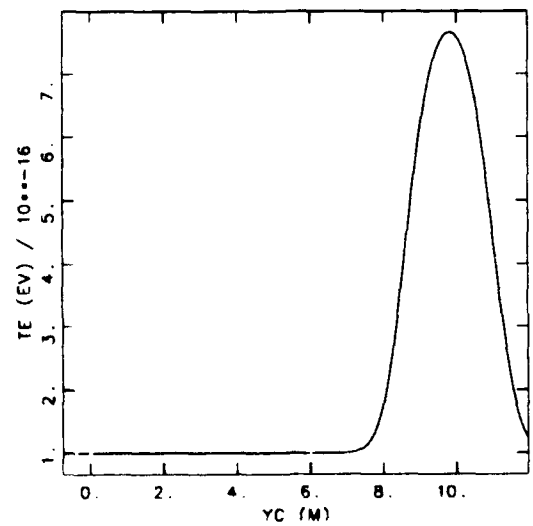


(a) Godunov method.

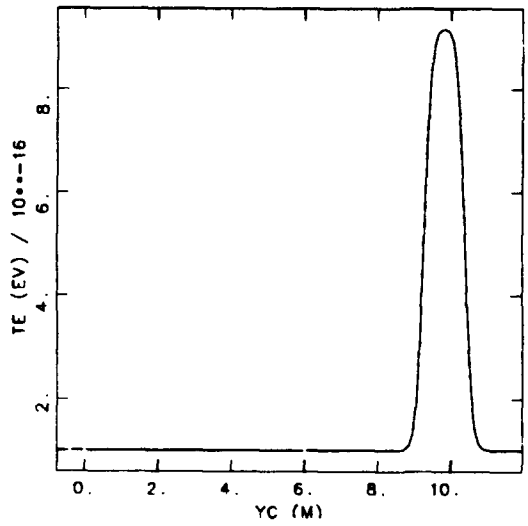


(b) Centered-difference scheme.

Figure 5. Mass density distributions for the square pulse advection.



(a) Godunov method.

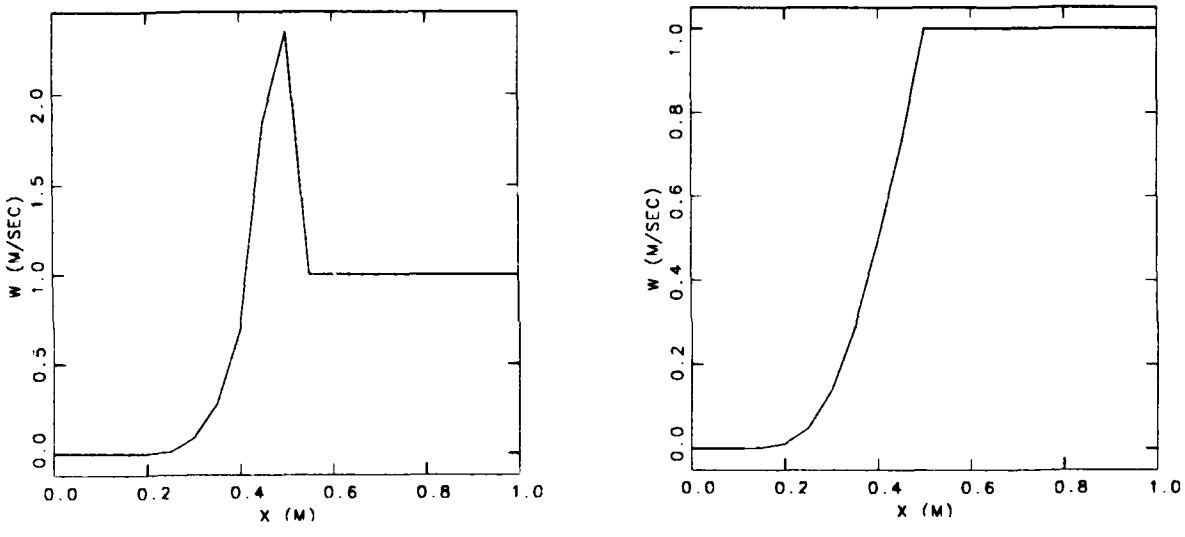


(b) Centered-difference scheme.

Figure 6. Temperature distributions for the square pulse advection.

the density and perpendicular velocity distributions are both step functions, but the change of one is opposite the other. This is the problem described in Subsection 3.3. Figure 7 shows two distributions of perpendicular velocity--the distribution on the left results when the momentum difference is created from the product of the cell-centered mass and vertex-centered velocity, and the distribution on the right results when one uses the density difference multiplied by the velocity. The new maximum in the former is obvious. Note that the distribution of vertex-centered perpendicular momentum, defined by the product of the vertex-centered velocity component and an average mass of the adjacent cells, has a maximum at the beginning of this problem. Therefore, one may consider this velocity maximum to be rather construed. However, for simulations of physical phenomena, the development of new velocity maxima may be misleading or even catastrophic.

The shock tube is also a simple problem, but it exhibits some important fluid phenomena. The domain is also a long straight tube, or chamber, which is divided by a diaphragm. The initial mass density on one side of the diaphragm is larger than what is on the other side, but both sides have the same initial temperature. For an ideal gas equation of state, the initial pressure is directly proportional to the initial density. When the diaphragm is released, the gas with the



(a) Monotonic momentum difference.

(b) Altered momentum difference.

Figure 7. Advection of the step function of perpendicular velocity.

greater density will expand into the lower density gas, launching a shock wave ahead of the diaphragm and creating a rarefaction wave behind it. In so doing, the pressure equilibrates across the diaphragm. An analytic solution may be found for this problem using the method of characteristics for the rarefaction wave and the Rankine-Hugoniot relations for the shock (Ref. 10).

Figure 8 shows the analytic solution and three solutions calculated with MACH2 at 30 μs for a shock tube with a $\gamma = 5/3$ gas and an initial density ratio of four across the diaphragm. The diaphragm is initially located at the 0.8-m position, and the initial sound speed is 12.4 km/s. Note that the results with the van Leer algorithm improve the performance for the advection of the contact surface, which is the discontinuity at the released diaphragm, compared with the results from the Godunov method. The third MACH2 curve is a Lagrangian version of the same shock tube. It maintains a perfect contact surface, but performs only slightly better than the Eulerian simulations for the shock and rarefaction waves. Thus, the error in modeling the two

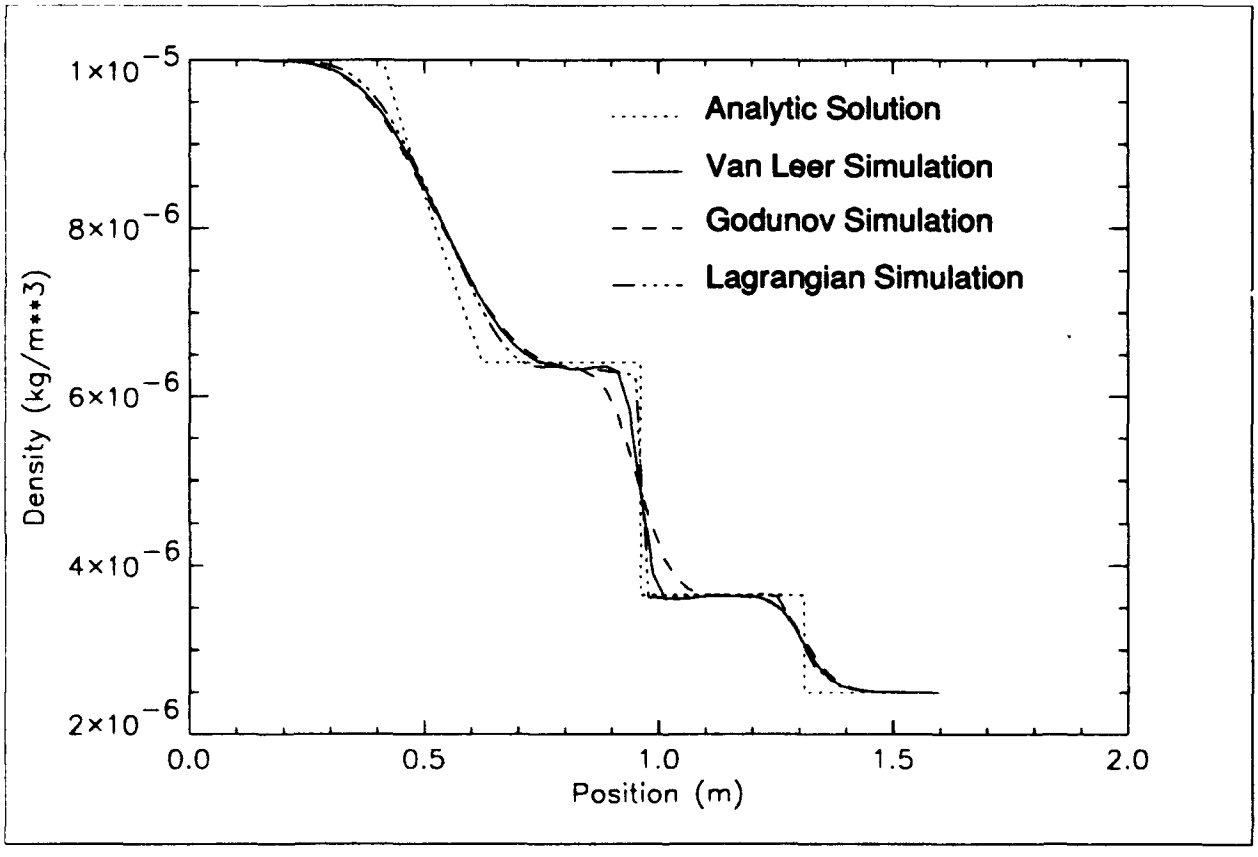


Figure 8. Shock tube results.

waves may be attributed to the Lagrangian phase and not the advection phase. All three simulations were run with fully-advanced time centering for the implicit Lagrangian algorithm. When the Lagrangian simulation is repeated with half-advanced time centering, the waves are sharper, but the solution is also oscillatory.

5.0 DISCUSSION

The van Leer advection algorithm currently in MACH2 has been used and upgraded over the past 2 1/2 years. It has been used for many complicated simulations of hydrodynamic and MHD phenomena. The algorithm seems to be rather robust and does not require special attention in most cases. In a practical sense it provides efficiency. When simulating complicated phenomena, one typically uses only enough grid resolution to provide reasonable convergence towards a solution--hopefully the correct solution, to save on computation and personal time. For a rough estimate, one needs about half as many cells in each dimension with the van Leer algorithm, in comparison with what is needed with the Godunov method, to achieve the same level of convergence for dynamic simulations. When one considers that an increased cell size also increases the allowed time increment, the savings in computation time can be close to an order of magnitude.

The algorithm presented here should not be considered a final state. If the user has ideas for improvements or has special needs for a particular problem, the author encourages him to pursue them. Writing code for MACH2 is fairly easy after one learns the block structure and the tool routines for setting boundary conditions.

REFERENCES

1. Peterkin, R. E., Jr., Giancola, A. J., Frese, M. H., and Buff, J., "MACH2: A Reference Manual--Fourth Edition," MRC/ABQ-R-1207, Mission Research Corporation, Albuquerque, NM, November 1989.

2. Frese, M. H., "A Two-Dimensional Complex Mesh Generator," AMRC-R-687, Mission Research Corporation, Albuquerque, NM, November 1989.

3. Brackbill, J. U., "Coordinate System Control: Adaptive Meshes," Numerical Grid Generation, J. F. Thompson, ed., Elsevier Science, 1982.

4. Godunov, S. K., Mat. Sb. 4, p271, 1959.

5. Van Leer, B., "Towards the Ultimate Conservative Difference Scheme. IV. A New Approach to Numerical Convection," Journal of Computational Physics, Vol. 23, p. 276, 1977.

6. Hirt, C. W., Amsden, A. A., and Cook, J. L., "An Arbitrary Lagrangian-Eulerian Computing Method for All Flow Speeds," Journal of Computational Physics, Vol. 14, p. 227, 1974.

7. Trulio, J. G., "Theory and Structure of AFTON Codes," AFWL-TR-66-19, Air Force Weapons Laboratory, Kirtland AFB, NM, June 1966.

8. Marsden, J. E. and Tromba, A. J., Vector Calculus, 2nd ed., W. H. Freeman and Co., New York, NY, p. 450, 1981.

9. Margolin, L. G. and Beason, C. W., "Remapping on the Staggered Mesh," UCRL-99682, Lawrence Livermore National Laboratory, Livermore, CA, September 1988.

10. Harlow, F. W. and Amsden, A. A., "Fluid Dynamics," LA-4700, Los Alamos Scientific Laboratory, Los Alamos, NM, June 1971.

APPENDIX

```
*dk trnspt
      subroutine tr   pt(dlogmmx)

c-----invoke the van Leer transport loop for mass, energy,
c-----magnetic field and momenta.

cdir$ nolist
      include 'common.h'
      include 'inputcom.h'
      include 'pointer.h'
      include 'mgcom.h'
cdir$ list

      pointer( kp006, grxrol( 0:ip2, 0:jp2) )
      pointer( kp010, gryrol( 0:ip2, 0:jp2) )
c-----initialize the volume integrated quantities.
      do 100 lblk = 1,nblk
        call setblk
        call trnsinit(dlogmmx)
  100 continue

c-----first transport mass:

      lblk = 1
      call setblk
      call trnslp('other',rol,mp)
c-----save the mass gradients for momentum
      do 110 lblk=1,nblk
        call setblk
        call bkpntrs(lblk,lblk,all,c  ll,all,cell)
        call bkcpyvf(dquanx,dquany,grxrol,gryrol)
  110 continue

c-----mass of material 2:

      if (con2on) then
        lblk = 1
        call setblk
        call trnslp('other',con2,mp2)
c-------if con2on these gradients are needed
        do 120 lblk=1,nblk
          call setblk
          call bkpntrs(lblk,lblk,all,cell,all,cell)
          call bkaddar(dquanx,grxrol,grxrol)
```

```
        call bkaddar(dquany,gryrol,gryrol)
  120   continue
      endif

c-----energy:

      lblk = 1
      call setblk
      call trnslp('other',sel,ep)

c-----ion energy

      if ( tsplit .ne. 0 ) then
       lblk = 1
       call setblk
       call trnslp('other',sieion,eip)
      endif

      if (strength) then
       lblk = 1
       call setblk
       call trnslp('other',sigdxxl,sigdxx)
       lblk = 1
       call setblk
       call trnslp('other',sigdxyl,sigdxy)
       lblk = 1
       call setblk
       call trnslp('other',sigdxzl,sigdxz)
       lblk = 1
       call setblk
       call trnslp('other',sigdyyl,sigdyy)
       lblk = 1
       call setblk
       call trnslp('other',sigdyzl,sigdyz)
      endif

c-----vertex centered momenta get special attention for
c-----boundary conditions and fluxing.
c-----three momentum components:

      lblk = 1
      call setblk
      call trnsmm( ul, up )
      lblk = 1
      call setblk
      call trnsmm( vl, vp )
```

```
      lblk = 1
      call setblk
      call tmsmm( wl, wp )

c-----magnetic fields
      if (magon) then

c-------find the gradients of (bxl,byl,bzl)
        call tmsbgr(brbzon)

        if (brbzon) then
          do 160 lblk=1,nblk
            call setblk
c-----------find the poloidal fields for fluxing
            call tmsinib
  160     continue
        endif

        do 170 lblk =1,nblk
          call setblk
          if (brbzon) then
c-----------find E dot dl for poloidal fluxes
            call tmsedl(dt)
c-----------transport the poloidal flux
            call tmsflux
c-----------compute new poloidal field
            call tmsbxby
          endif
c---------transport out-of-plane magnetic flux
          call tmsbz
  170   continue
      endif

c-----divide returned quantities by new cell mass and update density.
      do 200 lblk = 1,nblk
        call setblk
        call tmsfin
  200 continue

      return
      end
```

```
*dk trnsinit
      subroutine trnsinit(dlogmmx)

c-----initialize the cell integrals for advection

cdir$ nolist
      include 'common.h'
      include 'inputcom.h'
      include 'pointer.h'
cdir$ list

      dimension dlogm(mxij)
      dimension con2t( 0:mxij , 0:mxij )

      common /flxpnt/ istart(mxblks),iend(mxblks),
     %               jstart(mxblks),jend(mxblks)

      ifstart = istart(lblk)
      ifend = iend(lblk)
      jfstart = jstart(lblk)
      jfend = jend(lblk)

      t3 = 1./3.
      do 100 j=0,jp1
       do 100 i=0,ip1
         mp(i,j)     = ro(i,j)  * oldvol(i,j)
         mp2(i,j)    = zero
         con2t(i,j)  = con2(i,j)
         ep(i,j)     = mp(i,j)  * sel(i,j)
         eip(i,j)    = mp(i,j)  * sieion(i,j)
         sigdxx(i,j) = mp(i,j)  * sigdxxl(i,j)
         sigdxy(i,j) = mp(i,j)  * sigdxyl(i,j)
         sigdxz(i,j) = mp(i,j)  * sigdxzl(i,j)
         sigdyy(i,j) = mp(i,j)  * sigdyyl(i,j)
         sigdyz(i,j) = mp(i,j)  * sigdyzl(i,j)
         sel(i,j)    = rol(i,j) * sel(i,j)
         sieion(i,j) = rol(i,j) * sieion(i,j)
         sigdxxl(i,j) = rol(i,j) * sigdxxl(i,j)
         sigdxyl(i,j) = rol(i,j) * sigdxyl(i,j)
         sigdxzl(i,j) = rol(i,j) * sigdxzl(i,j)
         sigdyyl(i,j) = rol(i,j) * sigdyyl(i,j)
         sigdyzl(i,j) = rol(i,j) * sigdyzl(i,j)
         up(i,j) = 0.
         vp(i,j) = 0.
         wp(i,j) = 0.
c--------define a lagrangian volume
```

```
      lagvol(i,j) = mp(i,j) / ( rol(i,j) + tiny )
c---------volume flux out of the bottom of this cell.
      rub = ( 2.*r(i,j) + r(i+1,j) ) * url(i,j) +
     %      ( r(i,j) + 2.*r(i+1,j) ) * url(i+1,j)
      rvb = ( 2.*r(i,j) + r(i+1,j) ) * vrl(i,j) +
     %      ( r(i,j) + 2.*r(i+1,j) ) * vrl(i+1,j)
      dxbott(i,j) = -0.5 * dt * t3 *
     %      ( rub*(y(i+1,j) - y(i,j)) - rvb*(x(i+1,j) - x(i,j)) )
c---------volume flux out of the left of this cell.
      rul = ( 2.*r(i,j) + r(i,j+1) ) * url(i,j) +
     %      ( r(i,j) + 2.*r(i,j+1) ) * url(i,j+1)
      rvl = ( 2.*r(i,j) + r(i,j+1) ) * vrl(i,j) +
     %      ( r(i,j) + 2.*r(i,j+1) ) * vrl(i,j+1)
      dxleft(i,j) = -0.5 * dt * t3 *
     %      ( rul*(y(i,j) - y(i,j+1)) - rvl*(x(i,j) - x(i,j+1)) )
  100 continue

c-----separate con2 from con1 for fluxing to keep con2 < 1.
    if (con2on) then
     do 200 i=0,ip1
      do 200 j=0,jp1
       mp2(i,j) = con2t(i,j) * mp(i,j)
       mp(i,j) = ( 1. - con2t(i,j) ) * mp(i,j)
       con2(i,j) = con2t(i,j) * rol(i,j)
       rol(i,j) = ( 1. - con2t(i,j) ) * rol(i,j)
  200  continue
    endif

c-----time step control based on volume flux to cell volume ratio--compare
c-----with both new volumes and lagrangian volumes.
    do 300 j=jfstart,jfend
     do 350 i=1,icels
      volmin = min( lagvol(i,j), lagvol(i,j-1),
     %              one / rvol(i,j), 1. / rvol(i,j-1) )
      dlogm(i) = dxbott(i,j) / volmin
  350  continue

     if (j .ne. jp1) then
      idlogm = isamax(icels,dlogm(1),1)
      if ( abs( dlogm(idlogm) ) .gt. dlogmmx ) then
       idtc = idlogm
       jdtc = -j
       ldtc = lblk
       dlogmmx = abs( dlogm(idlogm) )
      endif
     endif
```

```
300 continue

   do 400 i=ifstart,ifend
    do 450 j=1,jcels
     volmin = min( lagvol(i,j), lagvol(i-1,j),
  %               one / rvol(i,j), one / rvol(i-1,j) )
     dlogm(j) = dxleft(i,j) / volmin
450  continue

    if (i .ne. ip1) then
     jdlogm = isamax(jcels,dlogm(1),1)
     if ( abs( dlogm(jdlogm) ) .gt. dlogmmx ) then
      jdtc = jdlogm
      idtc = -i
      ldtc = lblk
      dlogmmx = abs( dlogm(jdlogm) )
     endif
    endif

400 continue

   return
   end
```

```
*dk trnslp
      subroutine trnslp(quantype,trnsfrom,trnsto)

c-----transport the quantity trnsfrom
c-----using the van Leer transport scheme where the flux
c-----across boundary i+1 for a positive velocity, v, is
c-----v( q(i) + .5(1 - v dt/dx) X dq(i) ) , and
c-----dq(i) = ( q(i+1) - q(i-1) ) / 2 .
c-----this first loops over the blocks to find the dq's,
c-----then loops over the blocks to transport q.

cdir$ nolist
      include 'paramcom.h'
      include 'inputcom.h'
      include 'pointer.h'
cdir$ list

      character*(*) quantype
      dimension trnsfrom(0:ip2,0:jp2), trnsto(0:ip2,0:jp2)
      pointer ( kpptl, qul(0:ip2, 0:jp2) )
      pointer ( kpptn, qun(0:ip2, 0:jp2) )

c-----get pointer numbers for input arrays
      ikpptl = lindex (trnsfrom)
      ikpptn = lindex (trnsto)

c-----gradient loop

      if (quantype .ne. 'momentum') then

       do 100 lblk=1,nblk
        call setblk
        kpptl = lpoint( ikpptl, lblk)
        call trnsgr(quantype,qul)
  100   continue

c-------transport loop

       do 200 lblk=1,nblk
        call setblk
        kpptl = lpoint( ikpptl, lblk)
        kpptn = lpoint( ikpptn, lblk)
        call trnsdqbc
        call trnsadv(qul,qun)
  200   continue
```

```
      else

        do 300 lblk=1,nblk
          call setblk
          kpptl = lpoint( ikpptl, lblk)
          kpptn = lpoint( ikpptn, lblk)
          call tmsmmgr(qul)
          call tmsadv(qul,qun)
  300   continue
      endif

      return
      end
```

```
*dk trnsgr
      subroutine trnsgr(quantype,quan)

c-----find the limited, centered differences for use in the
c-----van Leer transport scheme.

cdir$ nolist
      include 'common.h'
      include 'inputcom.h'
      include 'pointer.h'
cdir$ list

      character*(*) quantype
      common /flxpnt/ istart(mxblks),iend(mxblks),
     %                jstart(mxblks),jend(mxblks)

      dimension quan(0:ip2,0:jp2)

      do 100 j=0,jp2
        do 100 i=0,ip2
          dquanx(i,j) = 0.
          dquany(i,j) = 0.
  100 continue

c-----set gradient ranges; limit it along walls to
c-----prevent confusion (taken care of in rmshbcs),
c-----forcing donor cell there except poloidal B.

      if ( .not. donor(lblk) ) then
        if (quantype .eq. 'polbfld') then
          igrxst = 1
          igrxend = icels
          jgrxst = 0
          jgrxend = jp1
          igryst = 0
          igryend = ip1
          jgryst = 1
          jgryend = jcels
        else
          igrxst = istart(lblk)
          igrxend = iend(lblk) - 1
          jgrxst = 1
          jgrxend = jcels
          igryst = 1
          igryend = icels
          jgryst = jstart(lblk)
```

```
      jgryend = jend(lblk) - 1
    endif

    do 200 j=jgryst,jgryend
     do 200 i=igryst,igryend
      diffb = quan(i,j) - quan(i,j-1)
      difft = quan(i,j+1) - quan(i,j)
      diffc = ( quan(i,j+1) - quan(i,j-1) ) / 2.d0
      sdiffb = sign( one, diffb )
      sdifft = sign( one, difft )
      sdiffc = sign( one, diffc )
      dlimb = diffb * 2.d0
      dlimt = difft * 2.d0
      sdqy = max( zero, sdiffb * sdifft ) / sdiffc
      dquany(i,j) = sdqy * min(abs(dlimb),abs(dlimt),abs(diffc))
      srmrvl = sign( one , ( ro(i,j) - rofvl ) )
      grmlt = max( zero , srmrvl )
      dquany(i,j) = grmlt * dquany(i,j)
200  continue

    do 300 i=igrxst,igrxend
     do 300 j=jgrxst,jgrxend
      diffl = quan(i,j) - quan(i-1,j)
      diffr = quan(i+1,j) - quan(i,j)
      diffc = ( quan(i+1,j) - quan(i-1,j) ) / 2.d0
      sdiffl = sign( one, diffl )
      sdiffr = sign( one, diffr )
      sdiffc = sign( one, diffc )
      dliml = diffl * 2.d0
      dlimr = diffr * 2.d0
      sdqx = max( zero, sdiffl * sdiffr ) / sdiffc
      dquanx(i,j) = sdqx * min(abs(dliml),abs(dlimr),abs(diffc))
      srmrvl = sign( one , ( ro(i,j) - rofvl ) )
      grmlt = max( zero , srmrvl )
      dquanx(i,j) = grmlt * dquanx(i,j)
300  continue
  endif

  return
  end
```

```
*dk trnsdqbc
      subroutine trnsdqbc

c-----communicates the van Leer gradient across
c-----block boundaries.

cdir$ nolist
      include 'common.h'
      include 'pointer.h'
      include 'bccommon.h'
      include 'inputcom.h'
cdir$ list


      do 100 i=1,4
       ibdry = iproseq(i,lblk)
       lnbr = knbr(ibdry,lblk)
       if (lnbr .ne. 0) then
        call setnbrb(lnbr)
c                      from     to     range
        call bcpntrs(ibdry,nebr,edge,this,ghst,edge,cell)
        call bccpyvf(dqxnbr,dqynbr,dqulx,dquly)
       endif
  100 continue

      return
      end
```

```
*dk trnsadv
      subroutine trnsadv(quanf,quant)

c-----transport a quantity with flux based on what is in
c-----quanf into what is in quant with a van Leer scheme.
c-----be aware that what come into this routine is something
c-----per unit volume (something X density) and it returns a
c-----volume integrated quantity (to be divided by new cell mass).

cdir$ nolist
      include 'common.h'
      include 'inputcom.h'
      include 'pointer.h'
cdir$ list

      dimension quanf(0:ip2, 0:jp2), quant(0:ip2, 0:jp2)

      common /flxpnt/ istart(mxblks),iend(mxblks),
     %                jstart(mxblks),jend(mxblks)

      ikpptt = lindex(quant)
      ifstart = istart(lblk)
      ifend = iend(lblk)
      jfstart = jstart(lblk)
      jfend = jend(lblk)

c-----flux in j-direction

      do 100 j=jfstart,jfend

c-------compute the fluxes between this row and the row below.

       do 150 i=1,icels

c---------advection (don't forget dxbott > 0 implies downward flow)
         sdv = sign( one , dxbott(i,j) )
         sigma = 0.5d0 * ( (one + sdv) * dxbott(i,j)/lagvol(i,j)
     %             + (one - sdv) * dxbott(i,j)/lagvol(i,j-1) )
         qubsp = ( quanf(i,j) - 0.5d0 * (one - sigma) * dquany(i,j) )
         qubsm = ( quanf(i,j-1) + 0.5d0 * (one+sigma) * dquany(i,j-1) )
         dqbs = 0.5d0 * dxbott(i,j) *
     %         ( (one +sdv) * qubsp + (one - sdv) * qubsm )
         quant(i,j) = quant(i,j) - dqbs
         quant(i,j-1) = quant(i,j-1) + dqbs

  150  continue
```

```
 100 continue

c-----flux in i-direction

    do 200 i=ifstart,ifend

c-------compute the fluxes between this column and the column to the left.

      do 250 j=1,jcels

c---------advection (don't forget dxleft > 0 implies flow to the left)
        sdv = sign( one , dxleft(i,j) )
        sigma = 0.5d0 * ( (one + sdv) * dxleft(i,j)/lagvol(i,j)
     %            + (one - sdv) * dxleft(i,j)/lagvol(i-1,j) )
        qulsp = ( quanf(i,j) - 0.5d0 * (one - sigma) * dquanx(i,j) )
        qulsm = ( quanf(i-1,j) + 0.5d0 * (one+sigma) * dquanx(i-1,j) )
        dqls = 0.5d0 * dxleft(i,j) *
     %         ( (one + sdv) * qulsp + (one - sdv) * qulsm )
        quant(i,j) = quant(i,j) - dqls
        quant(i-1,j) = quant(i-1,j) + dqls

 250  continue

 200 continue

    return
    end
```

```
*dk trnsmm
      subroutine trnsmm(trnsfrom,trnsto)

c-----Use the mach2 momentum components [cell-ro * vert-v]
c-----and use only v * grad(ro)
c-----to avoid advecting a lot of mass and little momentum.

cdir$ nolist
      include 'paramcom.h'
      include 'pointer.h'
cdir$ list

      dimension trnsfrom(0:ip2,0:jp2), trnsto(0:ip2,0:jp2)
      pointer( kpptf, qul(0:ip2, 0:jp2) )
      pointer( kpptt, qun(0:ip2, 0:jp2) )

c-----get pointer numbers
      ikpptf = lindex(trnsfrom)
      ikpptt = lindex(trnsto)

      do 100 lblk = 1,nblk
        call setblk
        kpptf = lpoint( ikpptf, lblk )
        call trnsmmin(qul)
  100 continue

c-----advect the four 'moments':

      lblk = 1
      call setblk
      call trnslp('momentum',rovv1,rovv1t)
      lblk = 1
      call setblk
      call trnslp('momentum',rovv2,rovv2t)
      lblk = 1
      call setblk
      call trnslp('momentum',rovv3,rovv3t)
      lblk = 1
      call setblk
      call trnslp('momentum',rovv4,rovv4t)

c-----recreate vertex momenta.

      do 200 lblk = 1,nblk
        call setblk
        kpptt = lpoint( ikpptt, lblk )
```

```
      call tmsmmf(qun,ikpptt)
200 continue

    return
    end
```

```
*dk tmsmmin
      subroutine tmsmmin(vel)

c-----initialize the four 'moments' of momentum for one component.

cdir$ nolist
      include 'common.h'
      include 'inputcom.h'
      include 'pointer.h'
cdir$ list

      dimension vel(0:ip2, 0:jp2)

      do 100 j=0,jp1
       do 100 i=0,ip1
         vm1 = vel(i+1,j)
         vm2 = vel(i+1,j+1)
         vm3 = vel(i,j+1)
         vm4 = vel(i,j)
         tmass = ro(i,j) * oldvol(i,j)
         rovv1(i,j) = vm1
         rovv2(i,j) = vm2
         rovv3(i,j) = vm3
         rovv4(i,j) = vm4
         rovv1t(i,j) = tmass * vm1
         rovv2t(i,j) = tmass * vm2
         rovv3t(i,j) = tmass * vm3
         rovv4t(i,j) = tmass * vm4
 100 continue

      return
      end
```

```
*dk trnsmmf
      subroutine trnsmmf(velp,ikpptt)

cdir$ nolist
      include 'common.h'
      include 'inputcom.h'
      include 'pointer.h'
cdir$ list

      dimension velp(0:ip2, 0:jp2)
      pointer(npptt, velpnbr(0:inp2, 0:jnp2) )

c-----acquire the part of the vertex momentum along boundaries
c-----that was created in previous blocks.

      do 100 ibdry=1,4
       lnbr = knbr(ibdry,lblk)
       if (lblk .gt. lnbr .and. lnbr .gt. 0) then
        call setnbrb(lnbr)
        npptt = lpoint( ikpptt, lnbr )
c                       from      to      range
        call bcpntrs(ibdry,nebr,edge,this,edge,edge,vert)
        call bccpysc(velpnbr,velp)
       elseif (lblk .eq. lnbr .and. ibdry .gt. nbrbdy(ibdry,lblk)) then
        call setnbrb(lnbr)
        npptt = lpoint( ikpptt, lnbr )
c                       from      to      range
        call bcpntrs(ibdry,nebr,edge,this,edge,edge,vert)
        call bccpysc(velpnbr,velp)
       endif
  100 continue

      do 200 icrnr=1,4
       ldnbr = ldignbr(icrnr,lblk)
       if (lblk .gt. ldnbr .and. ldnbr .gt. 0) then
        call setnbrb(ldnbr)
        npptt = lpoint( ikpptt, ldnbr )
c                       from      to
        call ccpntrs(icrnr,nebr,edge,0,this,edge,0,vert)
        call cccpysc(velpnbr,velp)
       endif
  200 continue

c-----recombine the vertex momentum in this block.

      do 375 j=1,jcels
```

```
      do 300 i=1,icels
        velp(i+1,j) = velp(i+1,j) + 0.25 * rovv1t(i,j)
  300 continue
      do 325 i=1,icels
        velp(i+1,j+1) = velp(i+1,j+1) + 0.25 * rovv2t(i,j)
  325 continue
      do 350 i=1,icels
        velp(i,j+1) = velp(i,j+1) + 0.25 * rovv3t(i,j)
  350 continue
      do 375 i=1,icels
        velp(i,j) = velp(i,j) + 0.25 * rovv4t(i,j)
  375 continue

c-----put the contribution of this block into the boundaries
c-----of previous blocks.

      do 400 ibdry=1,4
        lnbr = knbr(ibdry,lblk)
        if (lblk .gt. lnbr .and. lnbr .gt. 0) then
          call setnbrb(lnbr)
          npptt = lpoint( ikpptt, lnbr )
c                         from      to      range
          call bcpntrs(ibdry,this,edge,nebr,edge,edge,vert)
          call bccpysc(velp,velpnbr)
        elseif (lblk .eq. lnbr .and. ibdry .gt. nbrbdy(ibdry,lblk)) then
          call setnbrb(lnbr)
          npptt = lpoint( ikpptt, lnbr )
c                         from      to      range
          call bcpntrs(ibdry,this,edge,nebr,edge,edge,vert)
          call bccpysc(velp,velpnbr)
        endif
  400 continue

      do 500 icrnr=1,4
        ldnbr = ldignbr(icrnr,lblk)
        if (lblk .gt. ldnbr .and. ldnbr .gt. 0) then
          call setnbrb(ldnbr)
          npptt = lpoint( ikpptt, ldnbr )
c                         from      to
          call ccpntrs(icrnr,this,edge,0,nebr,edge,0,vert)
          call cccpysc(velp,velpnbr)
        endif
  500 continue

      return
      end
```