Numerical Simulation of a Stratified Gas ICF Cavity

Tim Bartel, R.R. Peterson, G.A. Moses

April 1986

UWFDM-679

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

Numerical Simulation of a Stratified Gas ICF Cavity

Tim Bartel, R.R. Peterson, G.A. Moses

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

April 1986

UWFDM-679

NUMERICAL SIMULATIONS OF A STRATIFIED GAS ICF CAVITY

T. J. Bartel
R. R. Peterson
G. A. Moses

April 1986

Nuclear Engineering Department
University of Wisconsin-Madison
1500 Johnson Drive
Madison, Wisconsin 53706

ABSTRACT

One and two dimensional radiation hydrodynamic simulations of a light ion fusion target generated microfireball in a stratified gas atmosphere have been performed. Three scenarios were investigated: the target in nitrogen gas with a layer of helium above it, the target in helium with a layer of nitrogen below it, and the target in a sphere of helium surrounded by nitrogen. The gases were of equal pressure (15 torr). The distance from the target to the gas interface was varied from 10 to 100 cm. Target micro-explosions of 200, 400, and 800 MJ were investigated. The intent of these configurations was to determine if stratified gases (with different opacities) could be used to reduce the overpressure on the diodes placed at the walls of the target chamber and also the diagnostic equipment placed below the target explosion. Nonspherical fireball propagation caused by "venting" of the fireball once its radiation front reached the gas interface was investigated. The interface was within the distance from the target where the fireball shock breaks away from the radiation diffusion wave.

The configuration with the target in the nitrogen resulted in an overpressure reduction of only 20%. The blast wave was formed very early and the venting process was insufficient to greatly modify its magnitude. The configurations with the target in the helium gas, in either gas layers or spherical shells, resulted in a reduction of at least 50%. This is because a fireball was created at the gas interface and was free to expand in either direction. However, the wall heat flux in the helium gas layer was very high due to the low X-ray stopping power of helium. A target in a central cell of helium, surrounded by nitrogen, is the best compromise configuration for both pressure load and heat flux reduction.

TABLE OF CONTENTS

1. Introduction

The design of a cavity vessel for a light ion beam inertial confinement fusion (ICF) reactor must consider effects from intense pressure loading on the cavity wall[1]. This is because most of the target X-ray and ionic debris energy is absorbed in the cavity gas which is converted to the mechanical and radiant energy of a blast wave. This blast wave exhibits mechanical shock and thermal radiation wave behavior. The vessel must both be able to withstand the high peak overpressure which is experienced from a single event and, since this is a cyclic process, the effect of cavity wall fatigue. Therefore, there is an important advantage to reducing the mechanical impulse on the cavity wall and several approaches have been suggested [2]. The present investigation explores the possibility of reducing the pressure impulse by using multiple layered cavity gases with different abilities to stop both thermal and X-ray photons. Strong shock theory [3,4] states that the overpressure is proportional to the energy behind the shock. Is is hoped that the expanding radiation field can be directed such that the resulting nonspherical fireball expansion would decrease the mechanical impulse on the diodes or on diagnostic equipment placed below the target. Also, the absorption of target generated X-rays can be controlled by using a non-uniform gas which can be optimized to reduce damage of the facility. This approach would help avoid excessive structural material in the instrumentation module that affects the measured X-ray and neutron spectra.

Figure 1 illustrates the geometry under consideration. The diagnostics package is depicted by the module beneath the target. The target chamber was taken as a right circular cylinder for the simulation. This study considered the cavity gases segregated into the two regions as illustrated; the top region would contain an optically transparent gas, helium, and the bottom region contained a gas with a much higher opacity, nitrogen. Three scenarios were investigated. Two were with the geometry of fig 1: the target location was either in the helium or in the nitrogen region. The target in figure 1 is shown for example in the lower or nitrogen gas region. The last scenario was to enclose the target in a spherical region of helium which was surrounded by nitrogen. This configuration is shown in fig. 2.

The hypothesis for the target in the nitrogen region is that once the radiation front of the expanding fireball has reached the gas interface, "vent-

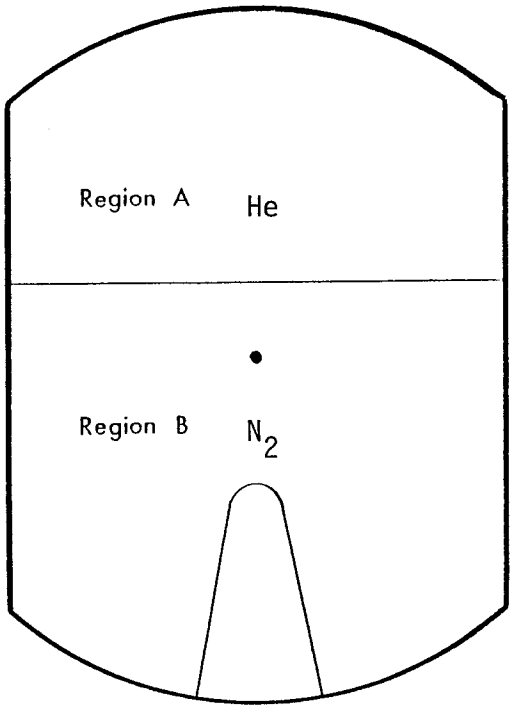


Fig. 1 Stratified Gas Cavity Geometry
(target shown in N_2 region)

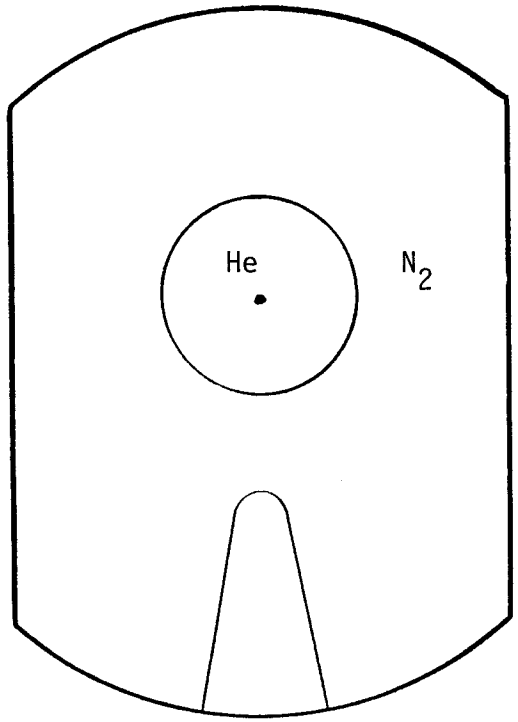


Fig. 2 Target in He 'Bag'

ing" of the radiation upward into the He gas would result in a nonspherical hydrodynamic pressure expansion in the nitrogen gas region and thus reduce the pressure loading in the radial (diodes) and downward axial (instrumentation) directions[5]. These effects are not unlike an explosion of a depth charge near the surface of the water; the opacity differences take the role of the density ratio.

To examine the viability of this approach, a 2-D Eulerian radiation fluid dynamics computer code was written. A short description and a complete listing of the code is presented in Appendix A. The diffusion approximation [6] was used for modelling the radiation field, where the radiation is assumed to act as a fluid with a single well defined temperature. This assumption is valid for the lower cavity gas but is incorrect for the upper gas, where because of its low opacity, the photons are not diffusing but are free-streaming. However, since we were not interested in modelling the behavior of the fireball in this region, the diffusion model was sufficient to obtain realistic boundary conditions for the lower gas region. The ramifications of this approximation will be discussed later. A tabular equation of state was used for the lower gas[7] ; the upper gas was modelled as optically transparent.

A different effect was important for the case of the target in the He region. Since He is not truly optically transparent, some of the initial X-ray energy will be absorbed in He. However, a larger effect is that the X-ray absorption will be very high in the N_2, at the gas interface. Now the fireball will be free to expand back into the helium as well as into the nitrogen. Also, the peak energy density in the nitrogen will be reduced, compared to an explosion in nitrogen, due to the isotropic spreading of the X-rays before they are absorbed. The pressure impulse on the instrumentation package and the diodes would then be reduced. This scenario is predicated on the assumption that plasma channels can be formed in the helium region for the ion beam to propagate through. We do not intend to investigate this, but merely assume it.

A 1-D Lagrangian radiation fluid dynamics computer code [8] was used to simulate this problem. This code has the ability to model radiation, fluid dynamics, and X-ray deposition in multiple materials. It was used to determine if a full multidimensional treatment of this situation was warranted; a spherical coordinate system was used for the present study.

2. Target in N_2 Region

The present analysis used helium as the transparent gas in region A and nitrogen as the target cavity gas in region B as shown in fig. 1. The calculations were done in a cylindrical geometry using 5 cm square computational meshes. The radius was taken as 250 cm with a no-flow right boundary. The axial "top" and "bottom" were modelled as free-flow boundaries. Typically the region below the target was 250 cm and the He region 200 cm. This was done to prevent boundary contamination from phenomena such as artificial shock reflections from affecting the regions of interest. Figure 3 shows the computational domain for the 100 cm calculation.

The initial cavity gas number density was taken as that which would have a pressure at 0^{o} C of 15 torr; both gas regions were at the same initial pressure. The shot energy was either 200 MJ (the standard TDF base case) or 800 MJ (for high yield targets). The present code does not model X-ray attenuation; therefore, MF-FIRE [8] was used to obtain the initial gas temperature profile. Figure 4 shows this profile for the 200 MJ case.

The present investigation was not concerned with detailed modelling of the nitrogen-helium interface. Thus, the computer code considered only a single species; the helium region was just modelled as a nitrogen gas with negligible opacity. Essentially, the helium region served as a pseudo-boundary condition for the nitrogen region. Only the fireball loading in the nitrogen region, both radially outward and axially downward from the target, were of interest. The pure hydrodynamic analogy for this situation is an underwater depth charge blast near the water surface; when the pressure pulse 'breaks' the surface, the explosion energy is directed upward.

2.1 200 MJ Target Yield

Indicated on Fig 4, the initial temperature profile for the 200 MJ case, is the region where the shock is "launched"; that is where the fireball hydrodynamic speed is greater than the diffusion speed. For the present test conditions, this value was found to be approximately 130 cm [9]. The distance between the He region and the target was varied in this study; fig. 4 indicates the three values used: 10, 40, and 100 cm. These were chosen for one to be inside the initial high energy deposition region, just beyond it, and prior to launching the shock.

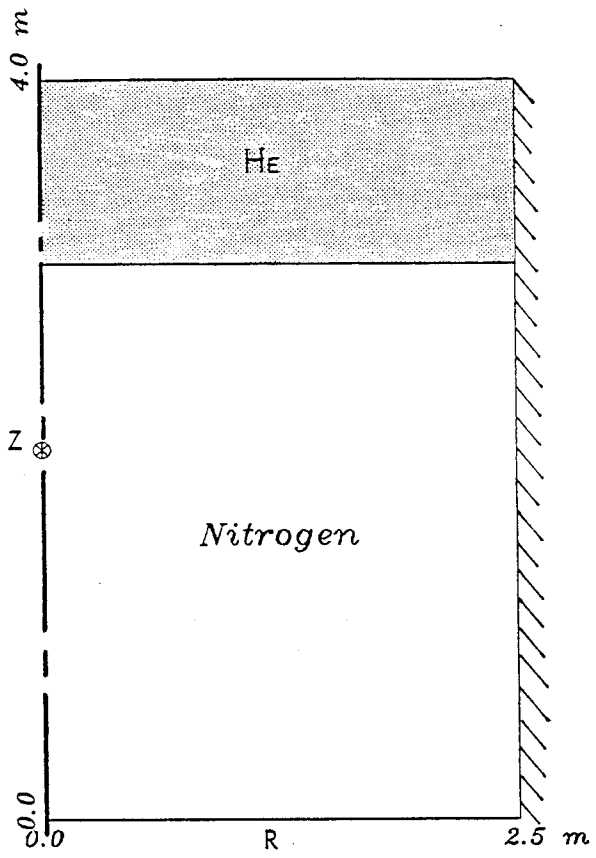


Fig. 3 Typical Computational Domain (100 cm case)

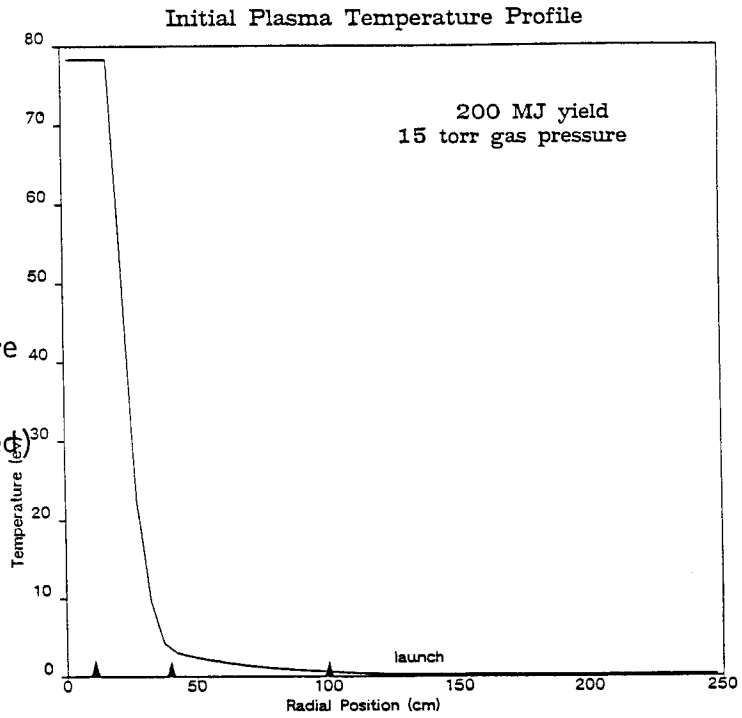


Fig. 4 Initial Gas Temperature Profile (target locations are indicated)

2.1.1 100 cm.

The first calculation positioned the interface 100 cm above the target. This allowed sufficient time for the fireball to develop before it encountered the He region. Figure 5 shows the development of the fireball from contours of gas temperature. One can note that the fireball has just begun to interact with the He at 17 microseconds. Prior to this time, it has essentially undergone a spherical expansion in the nitrogen. At about 32 microseconds, the gas temperature contours have become nonspherical due to the change in gas properties at the interface; the helium was optically transparent to the radiation while the nitrogen was not. Thus, a radiation enhanced thermal wave propagated into the nitrogen but since the radiation free-streamed in the helium, only a thermal conduction wave propagated in this region. The nitrogen thermal wave was enhanced due to the tight coupling of the radiation and gas fields; its propagation speed is dominated by the energy exchange between these fields. The propagation speed of a thermal conduction wave is due solely to its thermal conductivity; therefore, on the time scales under consideration, the thermal wave does not propagate as far into the helium as into the nitrogen region. Figure 6 illustrates an interesting effect due to the opacity difference at the interface: the gas temperature in the first helium zone becomes very high. Compression heating of the helium from the essentially stationary pressure gradient at the interface rapidly increased its temperature. Since the region was modelled as optically transparent, the gas could not lose energy by radiating.

Finally figure 7 illustrates the spatial distribution of the radiation temperature after the fireball has reached the interface. Here we can see that the radiation field had "burst" into the He gas and the fireball vented energy "upward" into the cavity. One will note that the radiation temperature is approximately 2.5 eV at this point. This will be a crucial value in determining the effectiveness of this pressure reduction scheme.

Figure 8 shows the comparison between the vented 100 cm case and a single region nitrogen case. Essentially there are only minor differences. This is due to the relatively low radiation interface temperature when the fireball reached the He. Since the radiation energy density is proportional to the fourth power of temperature, the actual energy flux being "vented" out of the fireball is comparatively small; the overpressure reduction would be negligible.

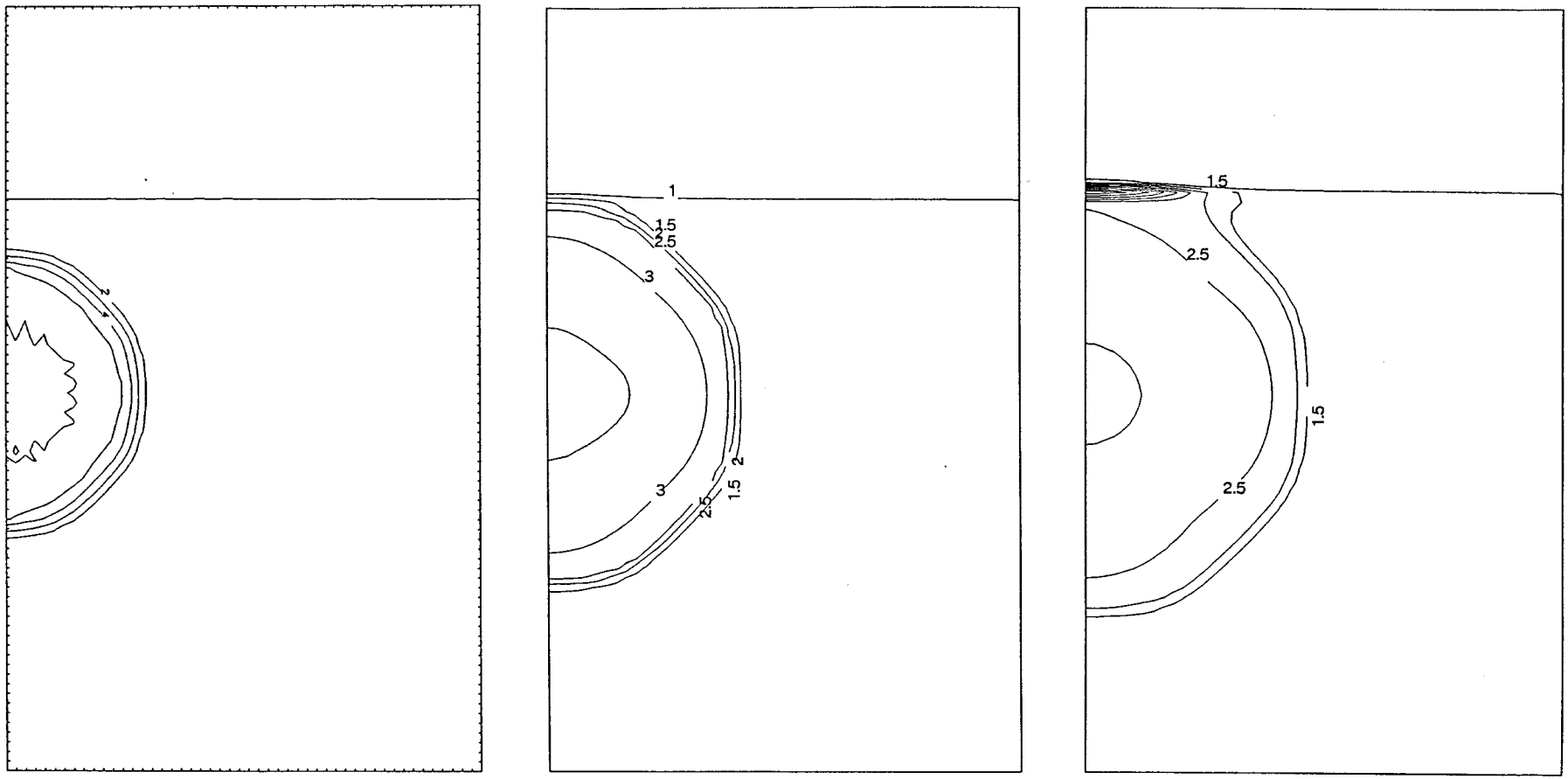
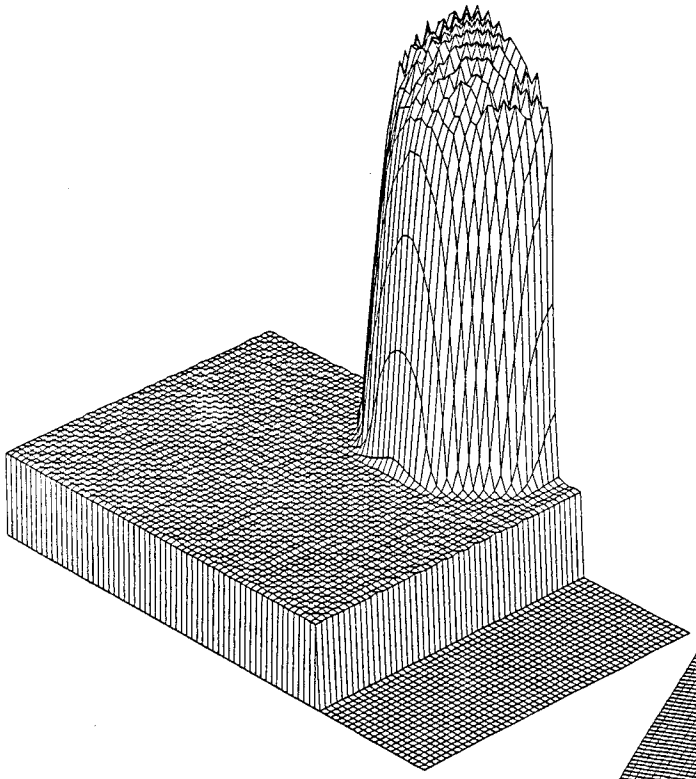
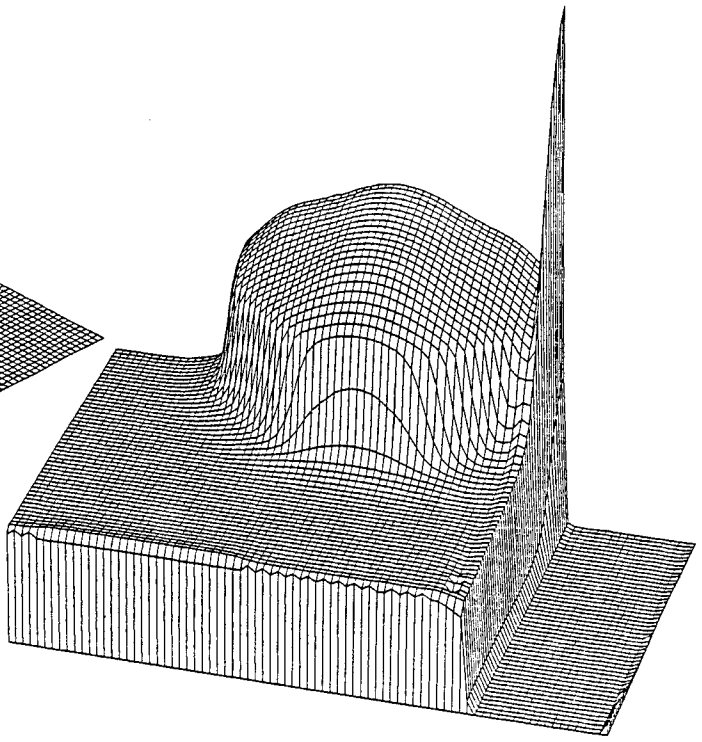
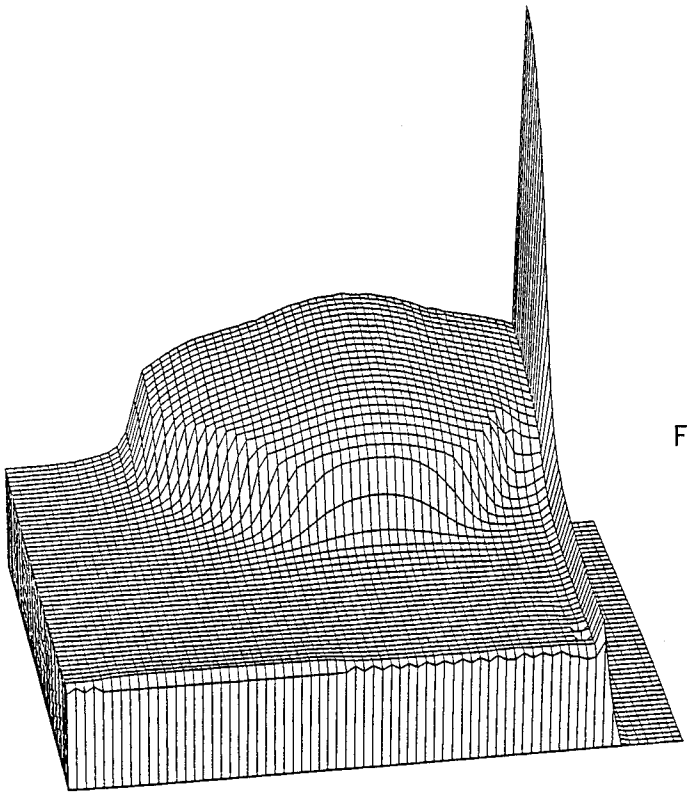


Fig. 5 Gas Temperature Contours for the 100 cm case.



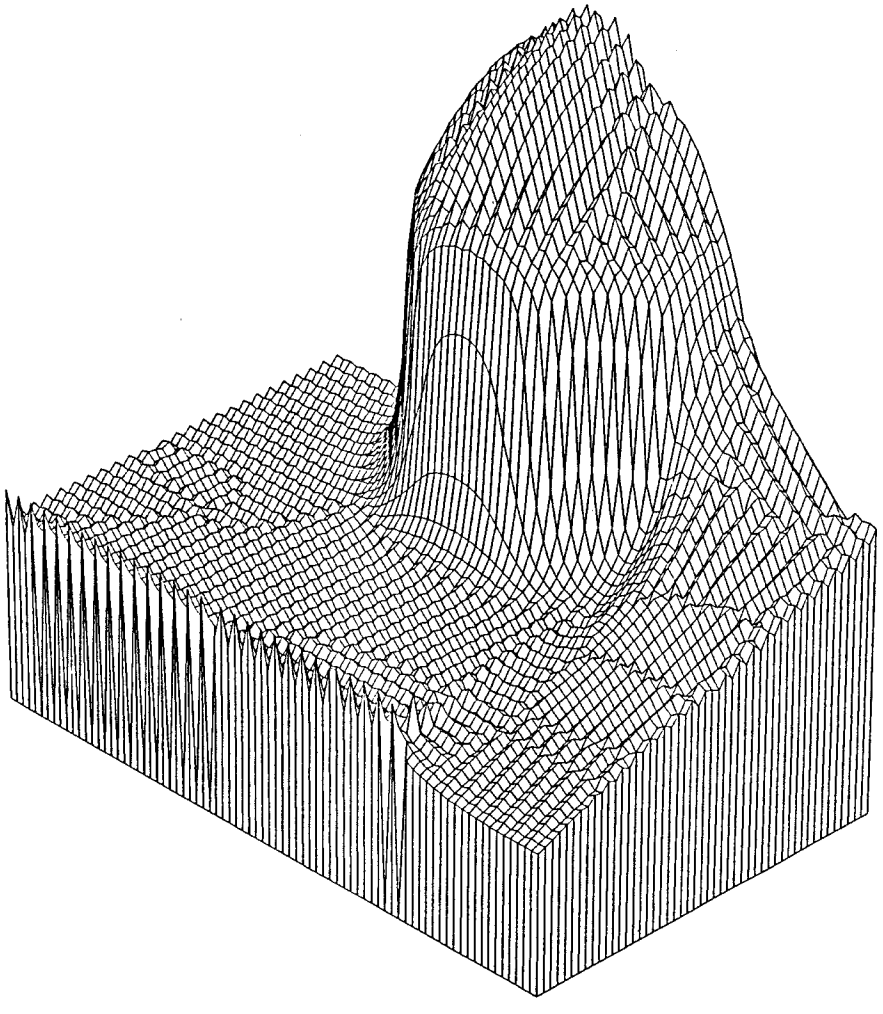


Plasma Temperature 3.17e-5 s

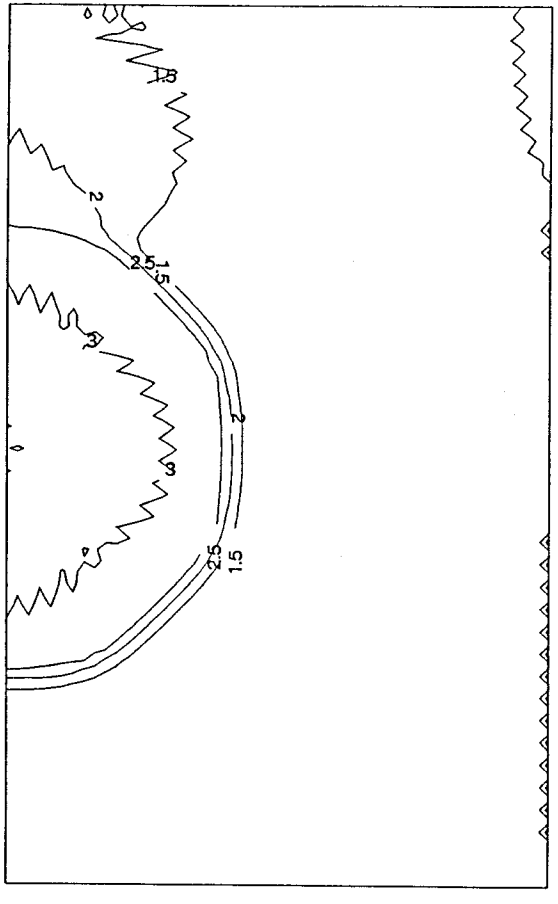


Plasma Temperature (100cm) 44.5e-6s

Fig. 6 Gas Temperature for 100 cm case.



Radiation Temperature 2.16e-5 s



Radiation Temperature 2.16e-5 s

Fig. 7 Radiation Temperature for 100 cm case

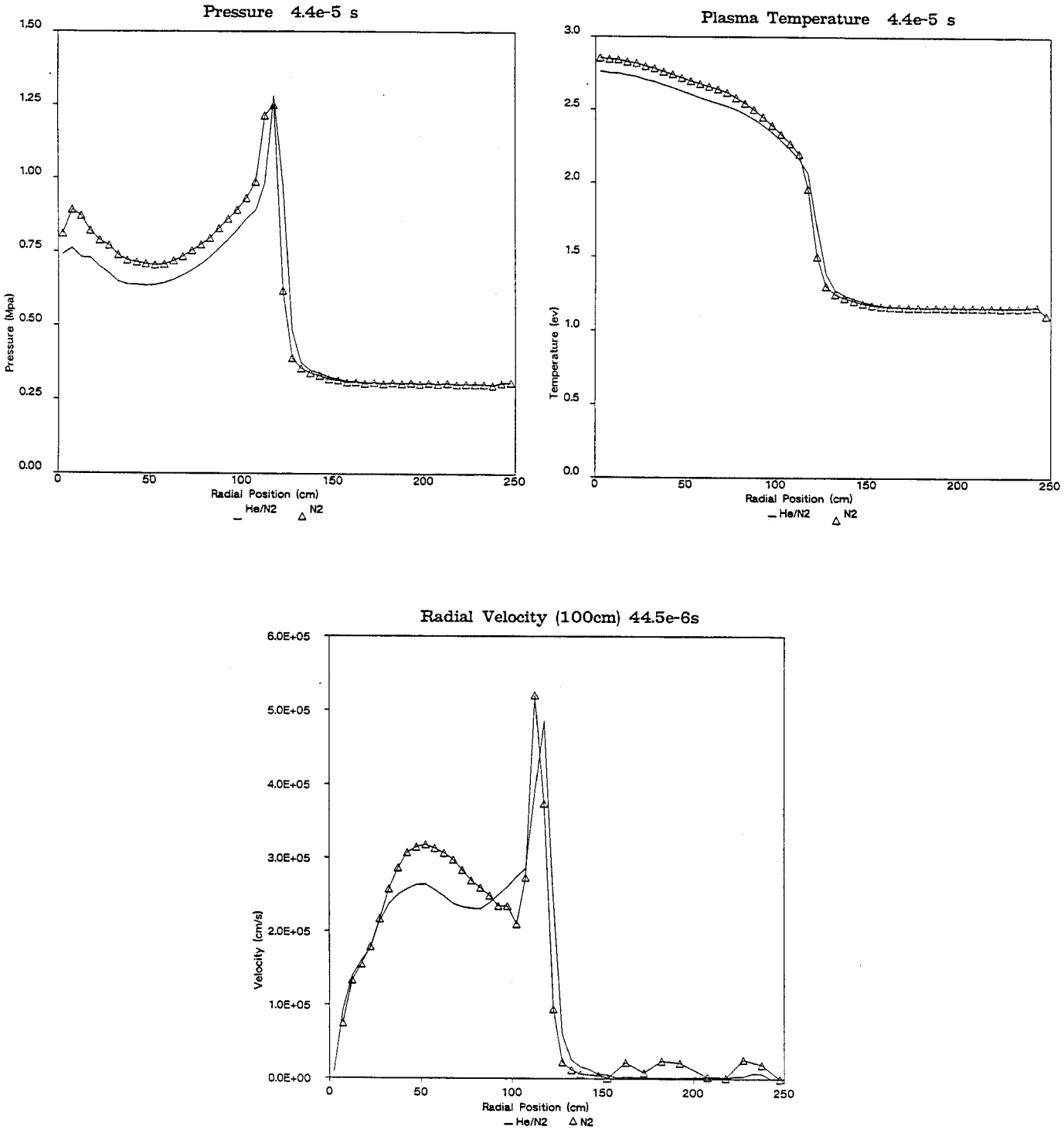


Fig. 8 Comparison of Pressure, Gas Temperature and Velocity between the Vented 100 cm case and Pure N_2 solution (plotted along the radial direction from the target)

2.1.2 40 cm.

In an effort to increase the interface radiation temperature when the fireball reached the He region, the separation distance was reduced from 100 cm to 40 cm. Figure 9 shows the gas and radiation temperatures along the vertical axis for two early times. The target was at 200 cm and the interface located at 240 cm. Here we see that the interface radiation temperature was much higher that the 100 cm case when the fireball reaches the helium. A temperature of 16 eV resulted in approximately 1700 times the vented energy flux than the 100 cm case. The spatial and temporal behavior of the fireball was otherwise similar to the 100 cm case.

Figure 10 shows the comparison between the 40 cm vented case and a one dimensional simulation of a 200 MJ explosion in pure nitrogen. One interesting point is that the location of the fireball edge, using the point of the maximum velocity, is the same for both calculations. This will simplify the later analysis. One can easily see that the vented fireball contains less energy due to the reduced core gas temperature and velocity. However, the peak velocities are similar because they are essentially determined from the pressure gradient at the edge of the fireball, which are also similar for both calculations. It is speculated that the pressure gradient, or equivalently the temperature gradient in the diffusion dominated region, is determined by the temperature dependence of the cavity gas opacity at the thermal front. If true, one would expect the gradients to be similar irrespective of the venting process, as the present calculations show.

2.1.3 10 cm.

The final calculation for the 200 MJ simulations reduced the distance between the target and the He region to 10 cm. This was done to determine the maximum realistic effect of energy venting. Figure 11 shows the gas and radiation temperatures along the vertical axis during the initial stages of the fireball evolution. Here, one can easily see the interaction of the He region with the formation of the fireball. One interesting point is that the radiation temperature quickly reached a steady value of about 7 eV while the gas temperature remained somewhat higher, 12 eV; the energy loss by venting was then balanced at these temperatures by the radiation emission from the gas. These values are determined by the opacity differences for the temperatures and number densities of interest. This equilibrium radiation temperature placed

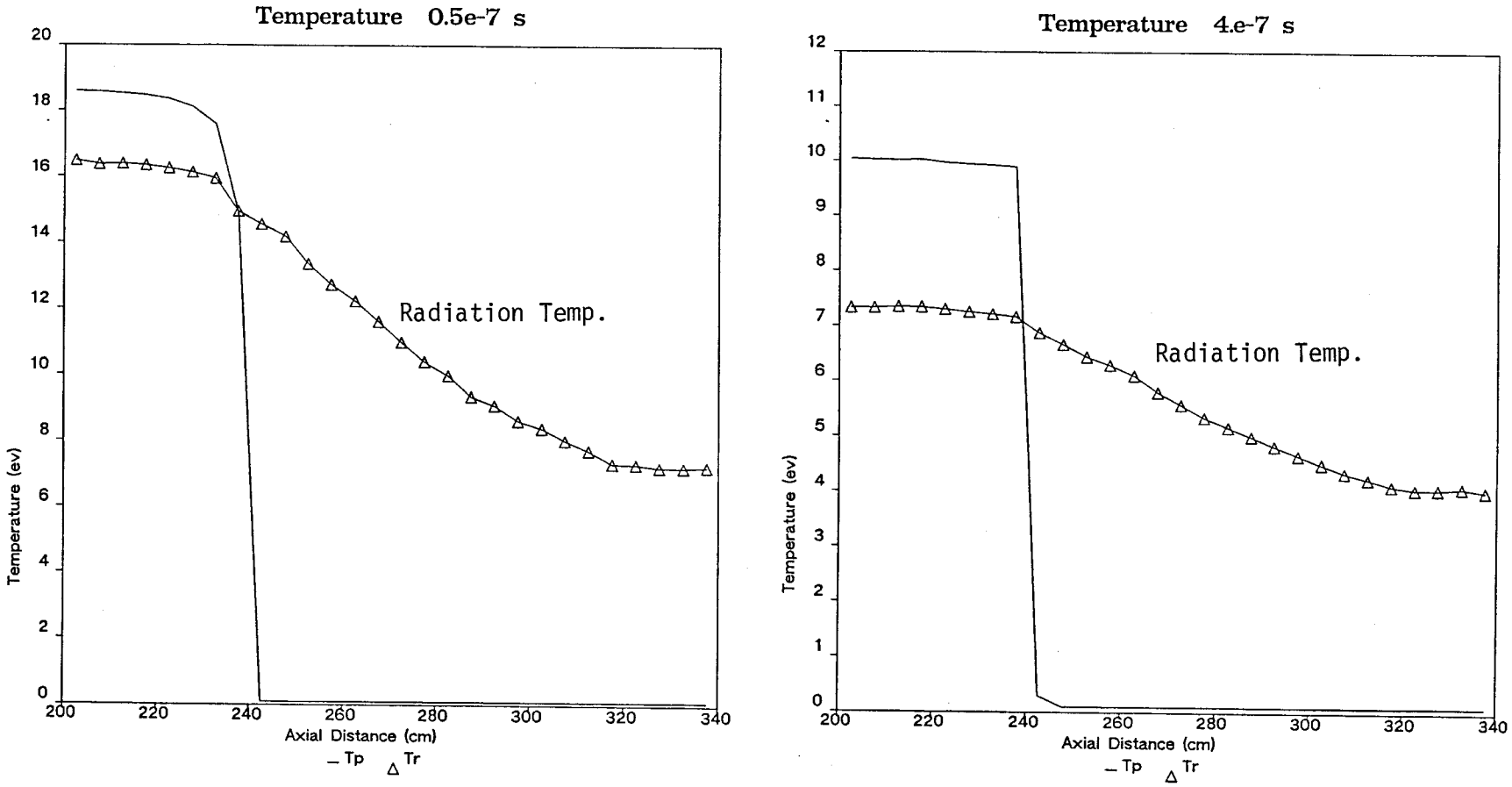


Fig. 9 Gas and Radiation Temperatures in the Axial Direction (upward from the target) for the 40 cm case (target located at 200 cm and gas interface at 240 cm)

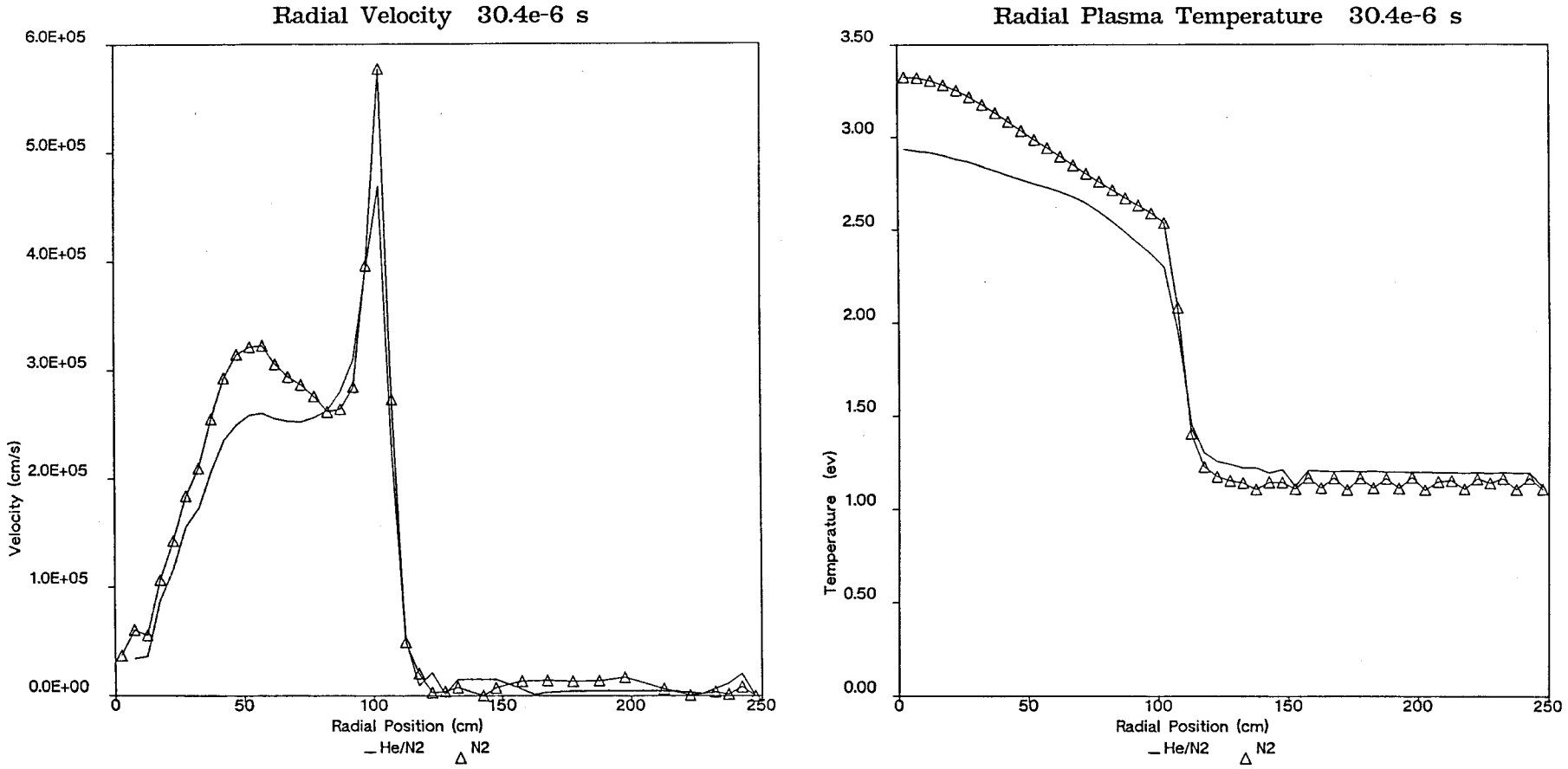


Fig. 10 Comparison of Velocity and Gas Temperatures between the Vented 40 cm case and pure N_2 solution (plotted along the radial direction from the target)

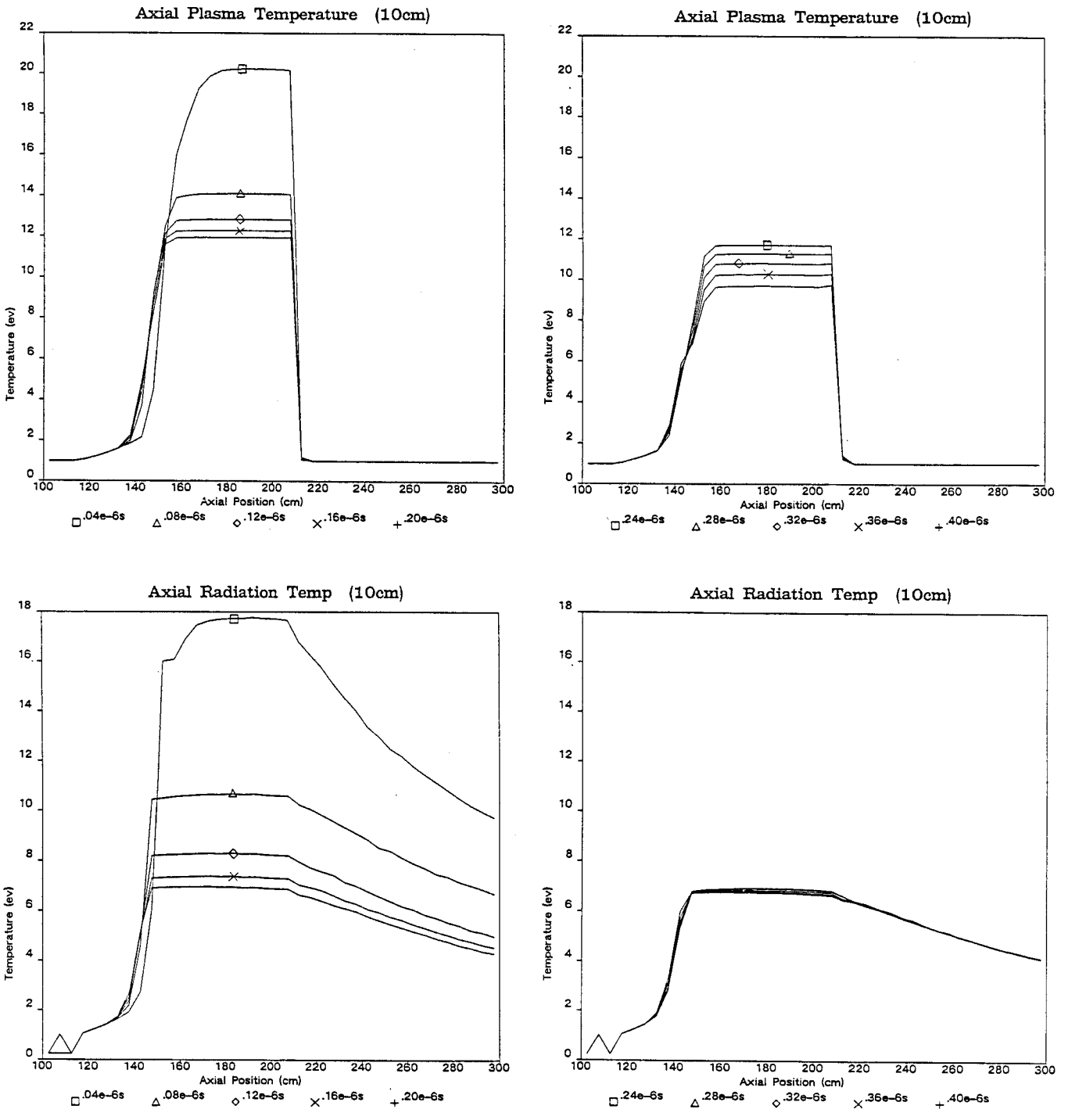


Fig. 11 Axial Gas and Radiation Temperatures for 10 cm case (target located at 200 cm and gas interface at 210 cm).

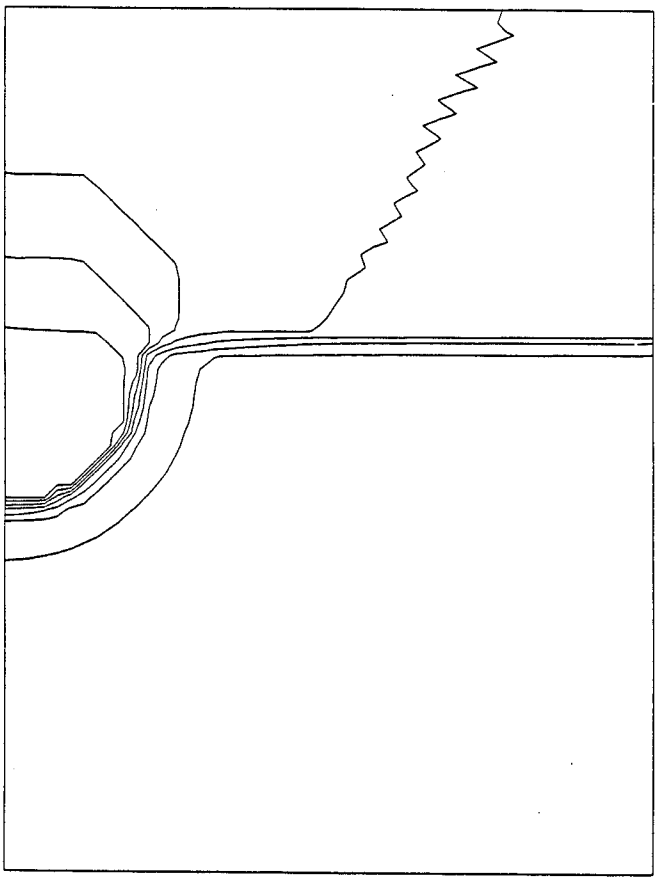
a limit on the vented energy loss. Different gas or fluid conditions might yield a more favorable equilibrium temperature.

As mentioned earlier, the diffusion approximation was used to model the radiation transport. This assumption is not valid in the helium region with the assumption of the optically transparent gas. Figure 13 vividly shows a consequence of this. The contour plot radiation temperature shows the non-physical propagation of the radiation wave in the He region. One would expect little radial diffusion as the radiation energy was transported into the helium from the interface; it would have the characteristics of a columnar beam. However, the diffusion approximation with its scalar effective diffusivity predicted large radial spreading. For this simulation, the radiation energy then reentered the nitrogen region and was subsequently attenuated, increasing its temperature. This resulted in a diffuse region near the interface and is illustrated in fig. 12, the perspective plot of the gas temperature. In this figure, one can contrast the sharp temperature gradient along the downward axial direction with the gentle slope in the radial direction. This in entirely an artifact of the computational models used for radiation diffusion. This effect was only observed for the situations where the target was very close to the gas interface.

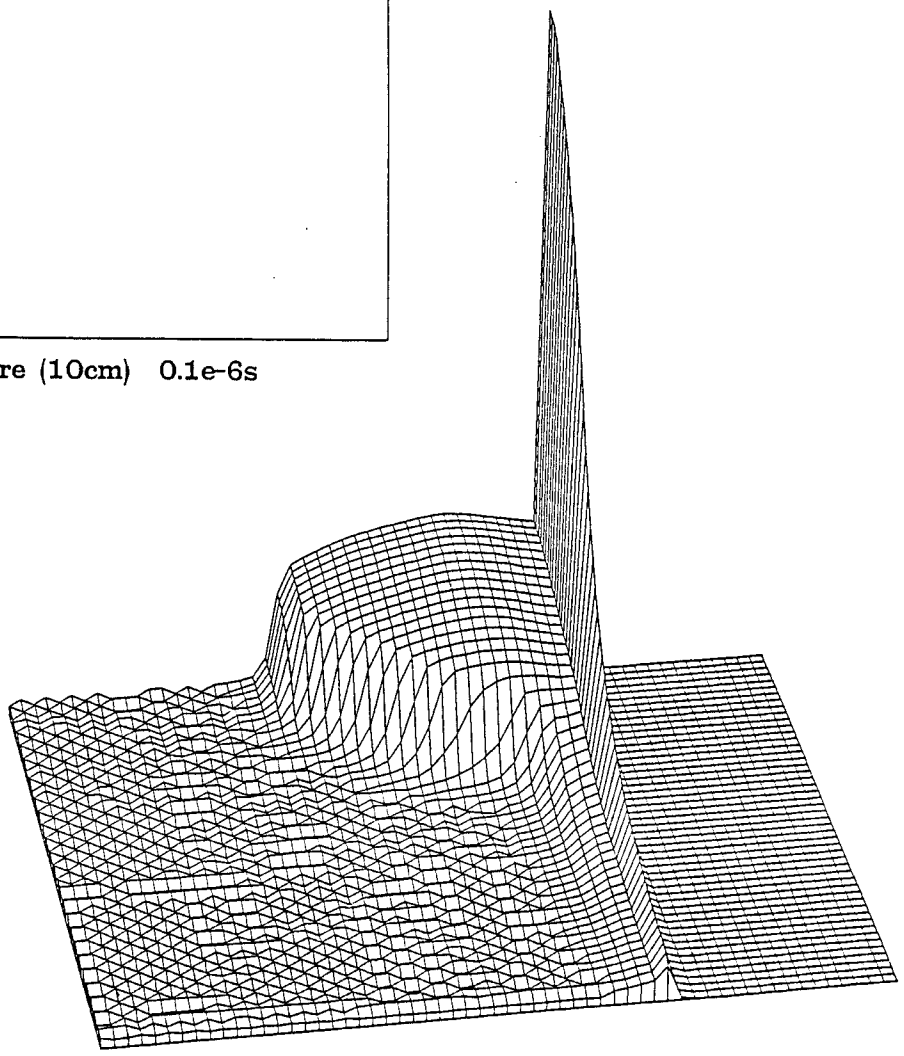
For this reason, comparisons between the pure nitrogen calculation were done using the fluid values along the downward axial direction from the target, that would be unaffected by this interface problem. Figure 13 shows this comparison. One can observe that although the peak stagnation pressure was essentially the same for both cases, the core values were noticeably reduced for the vented case. The same trends are also seen in the plot of total energy density. However, since the total fireball energy is a volume integral of this quantity and the majority of the volume of a sphere is in its outer radius, the differences are not as great as the plots would tend to indicate.

2.2 800 MJ Target Yield

Although a target yield of 200 MJ was the design base value for the TDF cavity, high gain targets will be periodically tested. For this reason, a stratified cavity gas simulation was performed with a target yield of 800 MJ; the separation distance was taken as 20 cm. Figure 14 shows the initial gas temperature profile for the 800 MJ case as calculated by MF-FIRE[8]. Its peak temperature is much higher than that for the 200 MJ case as shown in figure



Radiation Temperature (10cm) 0.1e-6s



Plasma Temperature (10cm) 16.9e-5s

Fig. 12 Gas and Radiation Temperatures for the 10 cm vented case.

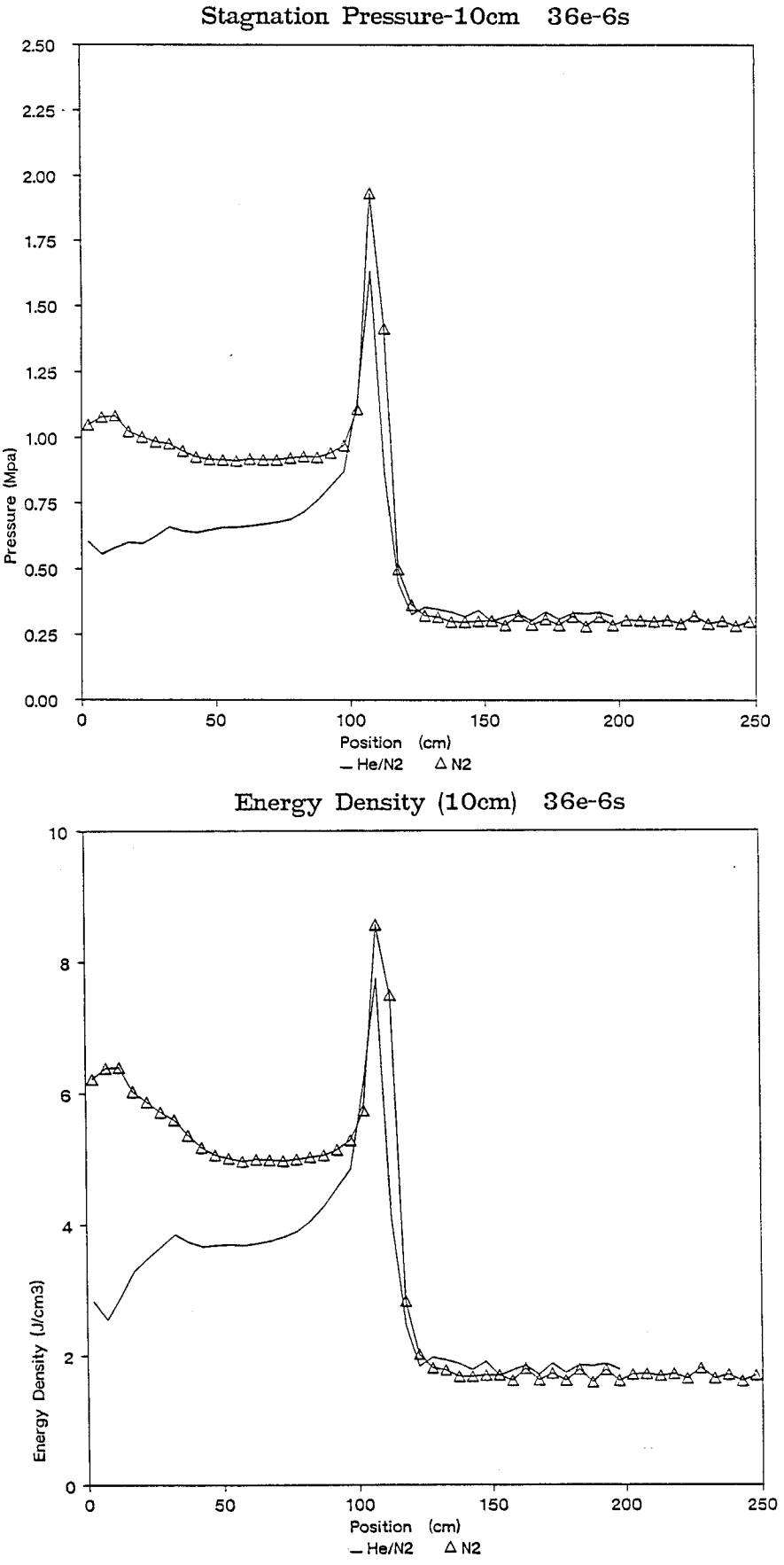


Fig. 13 Comparison between the Stagnation Pressure and Total Energy Density fro the vented 10 cm and pure N_2 cases (target at 0 cm)

3. The MF-FIRE initialization for the 800 MJ case used a finer mesh near the target center to better resolve this high gradient region. One might expect a stronger venting effect for this case because of the much higher temperatures.

2.2.1 20 cm.

Figure 15 shows the initial axial gas temperature distribution for this case. The effects of the He region are clearly shown by the sudden drop in temperature at 220 cm; here, the target location was at 200 cm. This is a result of the low opacity of helium. Figures 16 and 17 show the axial gas and radiation temperatures for four simulation times. The target location and the helium zone for the simulation are indicated. One will note that the gas temperature remained at a high value for at least 1 microsecond. This is unlike the 200 MJ cases, shown in fig 11, where the gas temperature rapidly dropped as the core gas radiated; the gas and radiation temperatures converged for the 200 MJ yield. Comparison of the gas and radiation temperatures for the 800 MJ simulation in figures 16 and 17 show that the radiation temperature did not drastically increase. This is because the 800 MJ target fully ionized the nitrogen gas; the opacity became very small so the gas and radiation became weakly coupled.

Figure 16 also shows the expanding thermal fronts, both in the nitrogen and the helium regions. A thermal wave is evident in the nitrogen, where thermal radiation from the center of the fireball is absorbed and is re-emitted from successive layers of gas. This process occurs because of the high opacity of the nitrogen and is therefore not present in the helium, where the opacity is very low. Also shown is that the radiation temperature at the helium-nitrogen interface remained low; thus, fireball energy loss from radiation "leakage" to the helium region was low. This finding was the same as for the 200 MJ simulations.

The increased energy deposition of the 800 MJ target over the 200 MJ simulation resulted in a very different hydrodynamic behavior at the fireball core. Figures 18 and 19 show the initial and three subsequent axial density and velocity profiles. The very high gas temperatures resulted in a large pressure gradient which lead to rapid hydromotion of the gas out of the core. This is shown by the very low core density in figure 18 and the large, early time, outward velocities in figure 19. When the core gas temperature was reduced to a point where the nitrogen opacity became significant, little mass

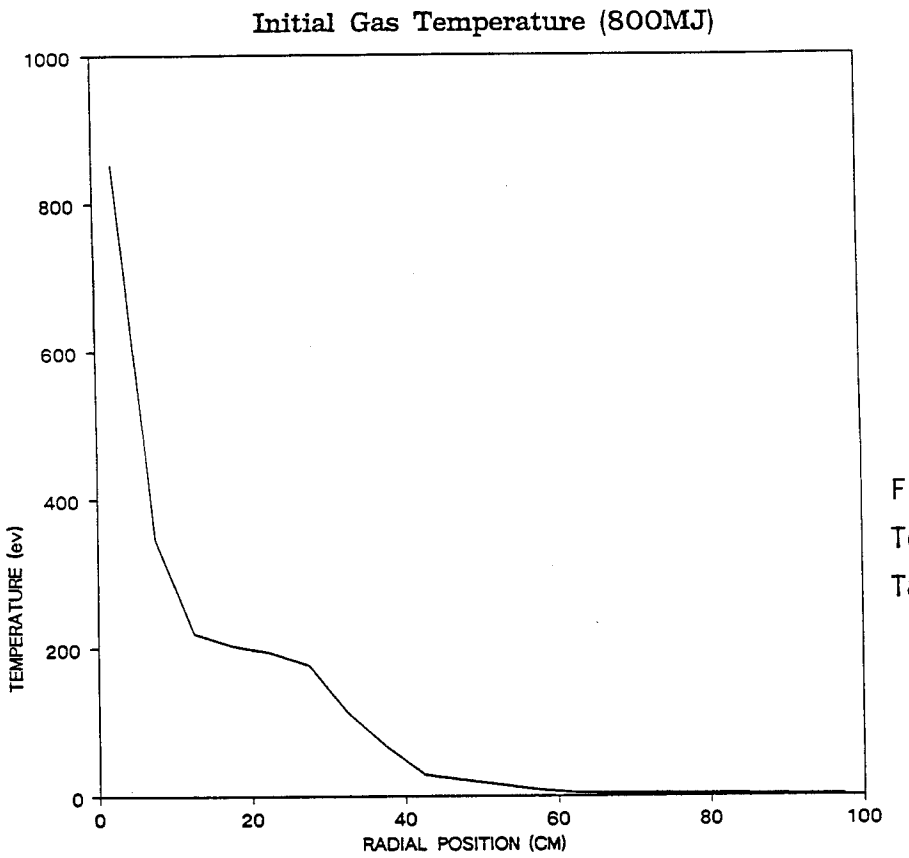


Fig. 14 Initial Radial Gas Temperature for 800 MJ Target in N_2.

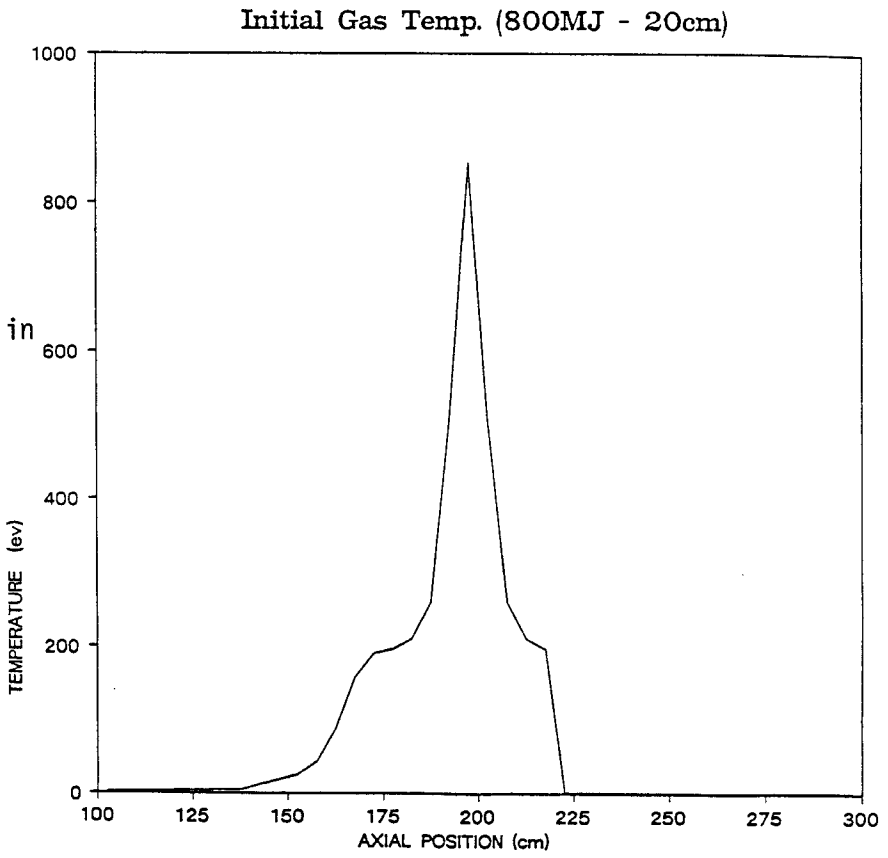


Fig. 15 Initial Axial Gas Temperature for 800 MJ Target in N_2 (Target located at 200 cm, gas interface at 220 cm).

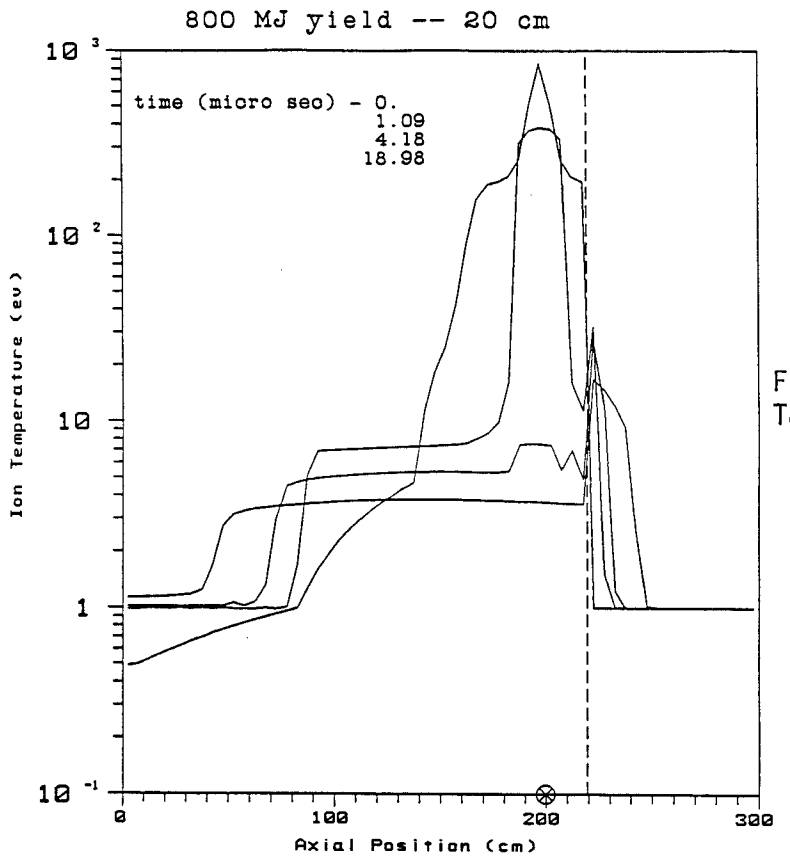


Fig. 16 Axial Gas Temperature for Target in N_2 region.

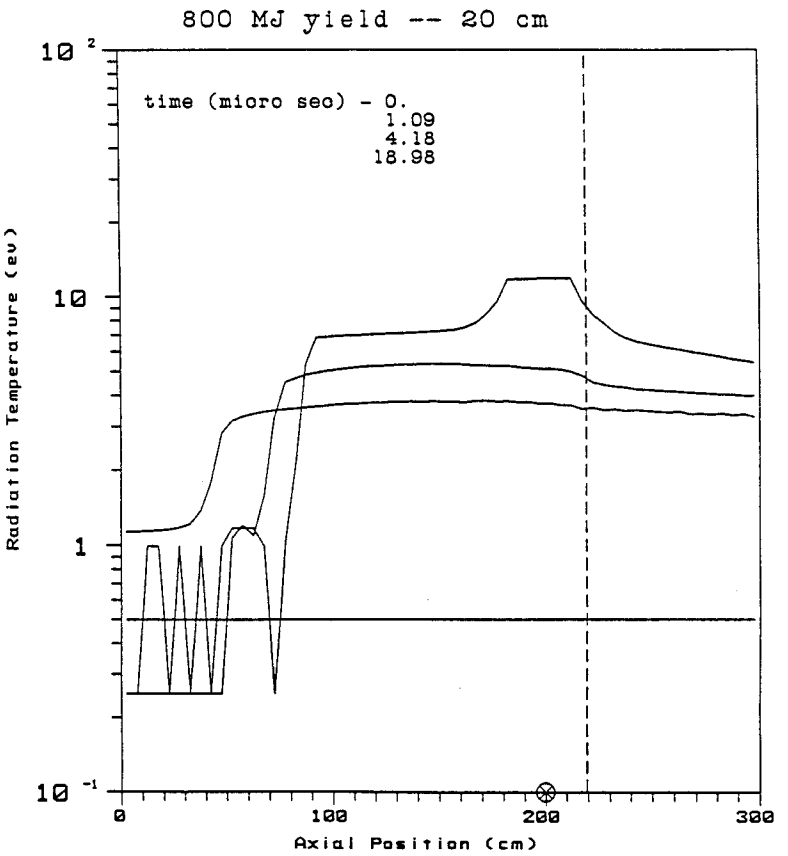


Fig. 17 Axial Radiation Temp. for 800 MJ Target in N_2 region.

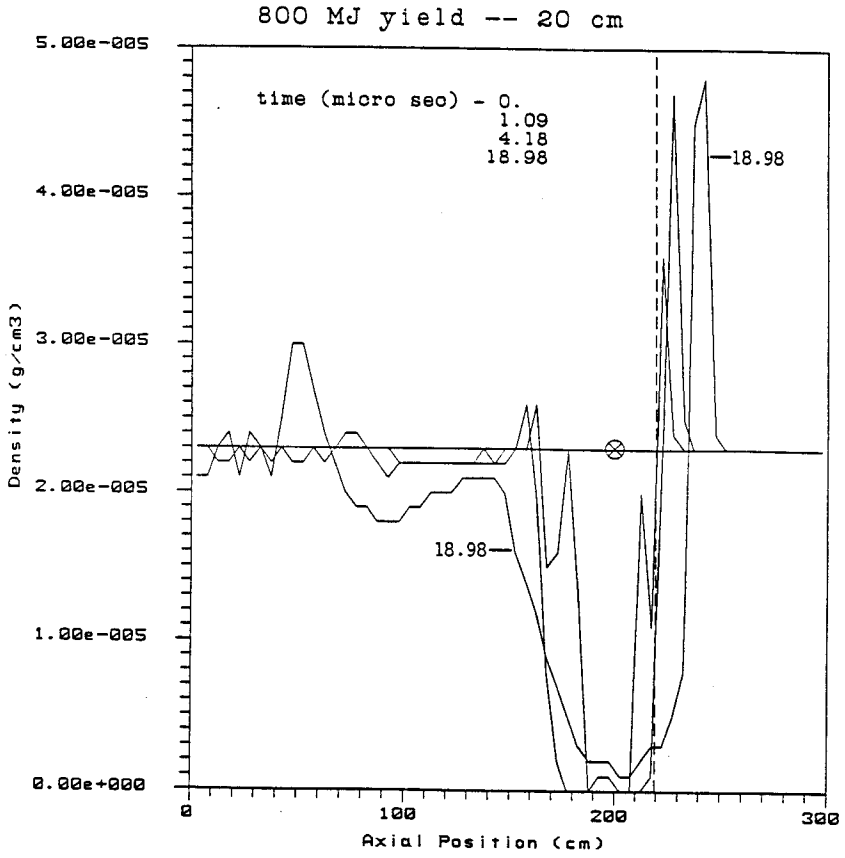


Fig. 18 Axial Fluid Density for 800 MJ Target in N_2 region.

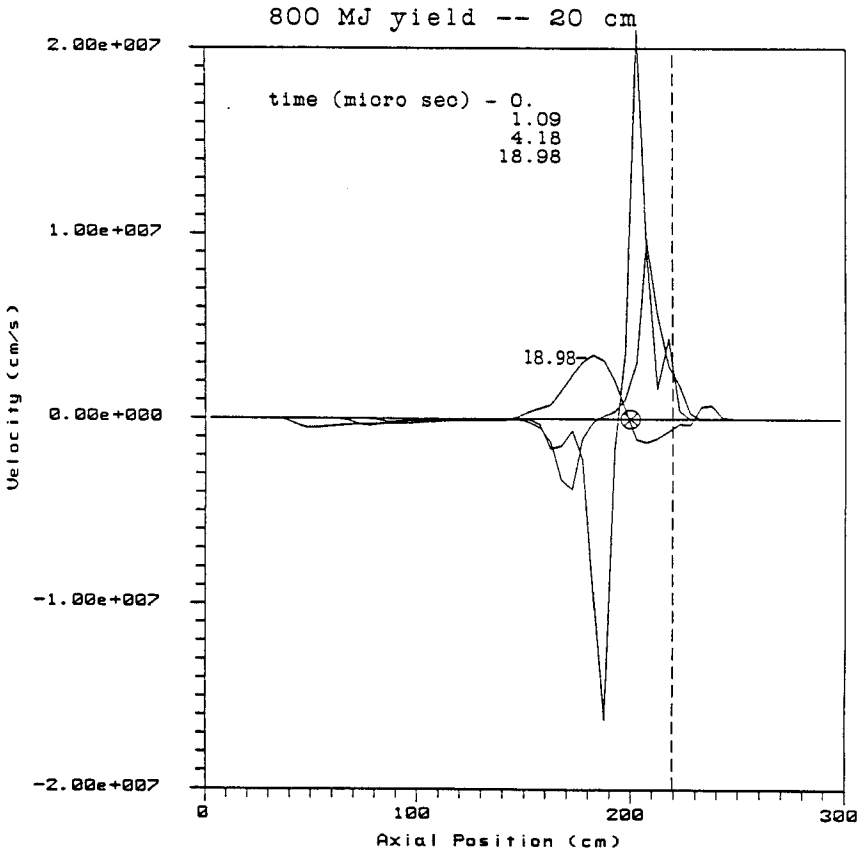


Fig. 19 Axial Fluid Velocity for 800 MJ Target in N_2.

remained in the core. At this point in time, the gas temperature quickly dropped, as is shown in fig. 15, which resulted in an adverse pressure gradient. Therefore the hydromotion in the core was reversed and the core compressed as mass moved back into it. Figure 18 shows the increased density in the core at the last plotted time and figure 19, the change in sign of the velocity front. However, this effect is not important to the overall propagation of the fireball since only a small amount of mass is involved. The majority of the fireball's mass was located at its perimeter, as shown in figure 18. The high density helium region located at the gas interface was due to the same intense pressure gradient as discussed earlier for the 200 MJ case, that is, the result of a conduction and not radiation driven thermal wave in the helium region.

Figures 20 and 21 compare the stratified gas and pure nitrogen calculations where the gas temperature and pressure are shown. The spatial axis has been modified so that the target is at position 0 cm for both conditions. There are no significant differences between the two simulations at the edge of the fireball. The differences in the pressure at the core at 15 microseconds was that the radiation 'venting' allowed the flow reversal to occur earlier than for the pure nitrogen case. Essentially the core gas temperature was reduced faster through the radiation loss. Figure 22 shows pressure profiles for the pure nitrogen case. We can see that at 4.22 microseconds the flow reversal has not occurred, but at 50 microseconds it has.

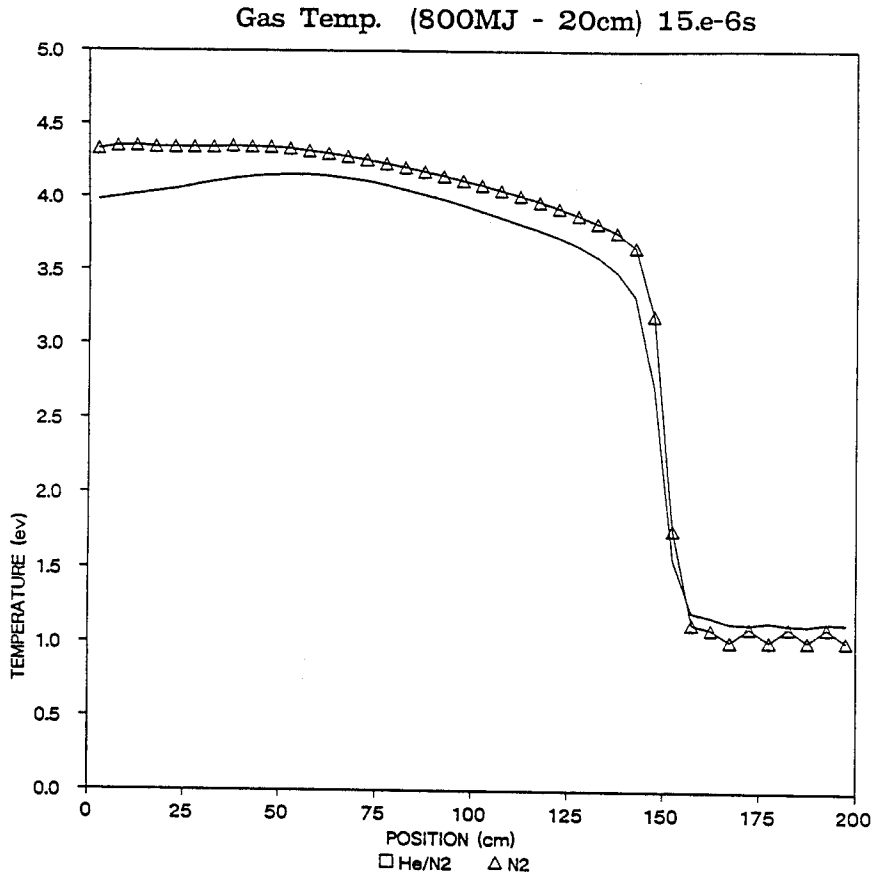


Fig. 20 Local Gas Temperatures for vented and pure N_2 case (target located at 0 cm).

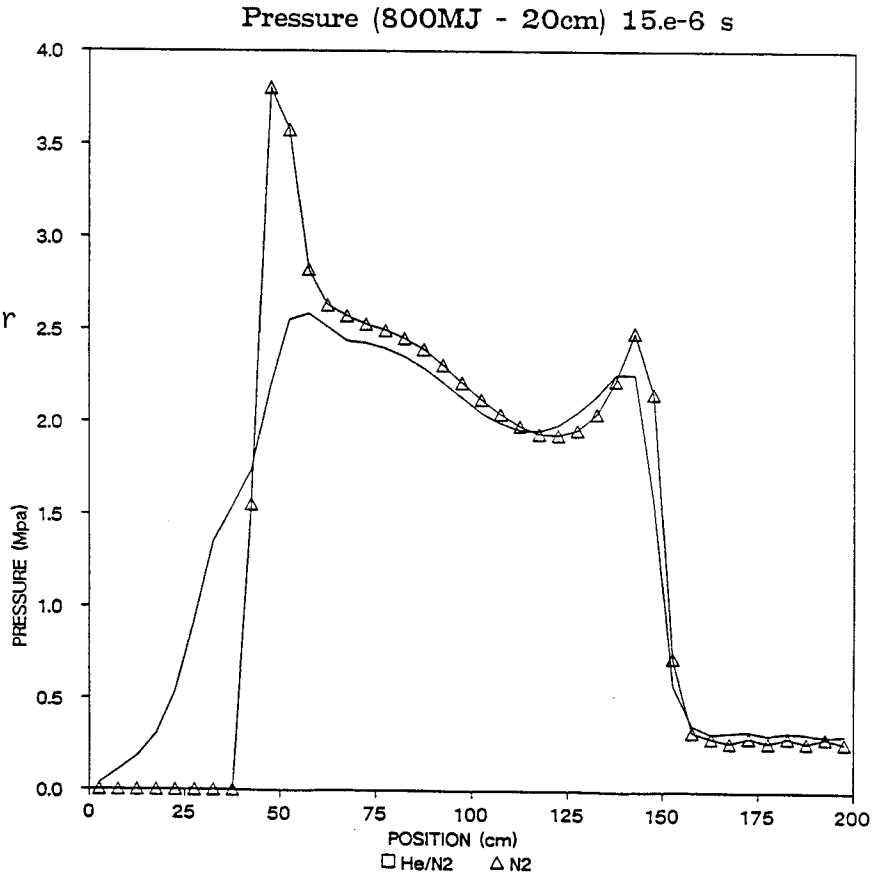


Fig. 21 Local Gas Pressure for vented and pure N_2 case. (target located at 0 cm).

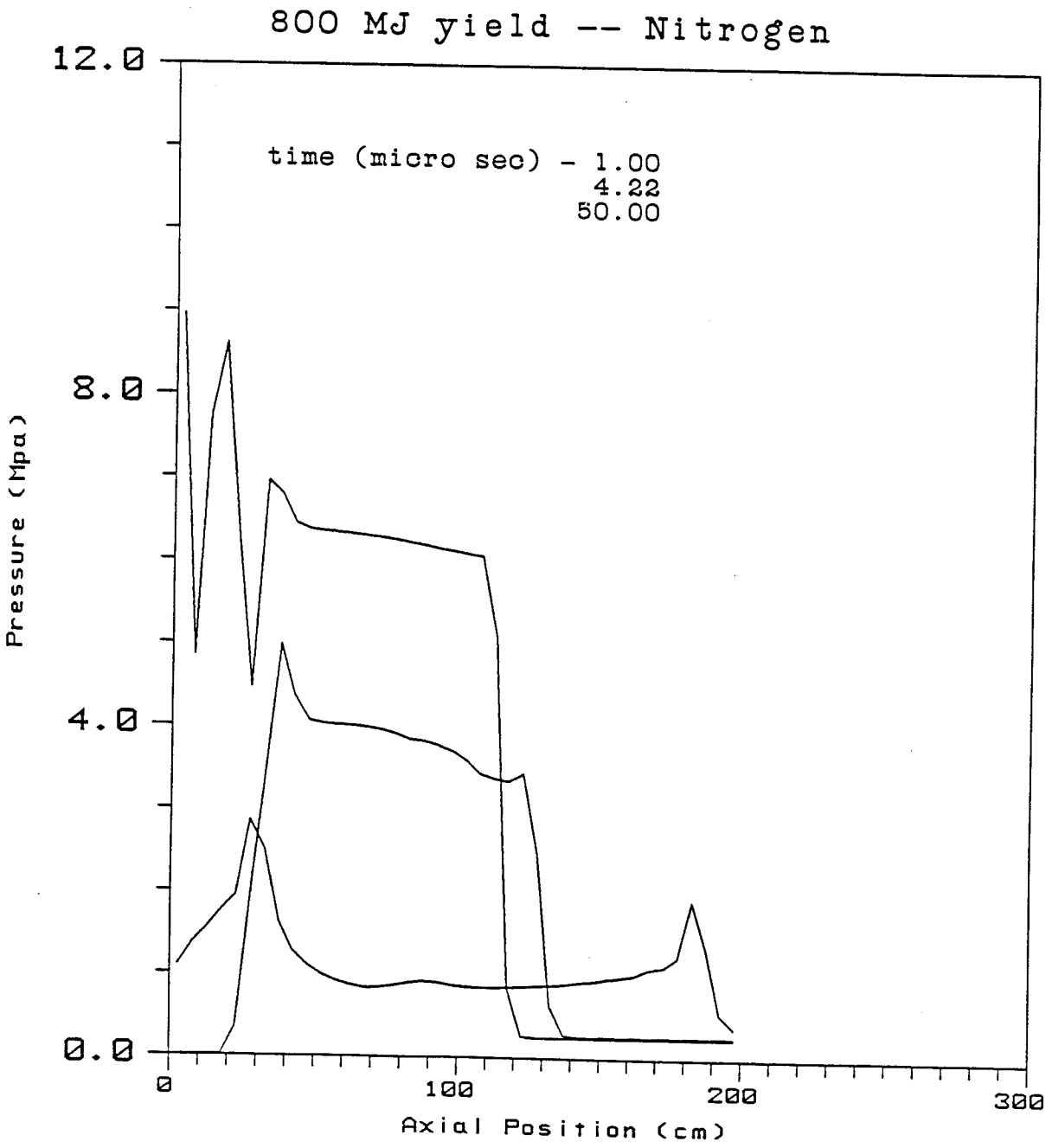


Fig. 22 Local Gas Pressure for Pure N_2 case.

3. Target in He Region

A series of one dimensional calculations were performed when the target was located in the helium region and the instrument package located in the nitrogen. The initial cavity density was taken as that which would have a pressure at 0^{o} C of 15 torr, the same as the earlier calculations. Two target yields were simulated: 400 and 800 MJ. And as before, the distance from the target to the gas interface was varied to determine spatial effects. The distance from the target to the 'wall' was kept a constant for all cases: 300 cm.

MF-FIRE[8] was used to simulate this series of calculations. This code has the X-ray deposition model and multi-material capability which were required. This code is only one-dimensional; a spherical coordinate system was chosen for the present investigation. In essence the geometry simulated was one of concentric spheres: the inner one was composed of helium and the outer spherical shell was nitrogen. A spherical system was used as opposed to a cartesian system to account for the $1/r^{2}$ nature of the X-ray deposition.

Recall that figures 1 and 2 show two cavity configurations for the case where the target is in the helium. From the prior discussion, it is evident that the present investigation identically models the geometry in fig. 2. However, the results can still be used to predict the initial pressure and heat flux loads on the instrument package for the cavity in fig. 1. Since the one dimensional simulation does not simulate the radiation losses or fluid motion to the upper region, only the initial wall responses can be used. The long term pressure impulse and surface heat flux history must be obtained from a full two dimensional simulation.

3.1 400 MJ Target Yield

Four calculations were performed with a target yield of 400 MJ: a target-to-interface distance of 25 and 50 cm and the target in either pure helium or nitrogen.

Figure 23 shows the initial gas temperatures for the 25 and 50 cm simulations. The temperatures are of course identical from 0 to 25 cm in the helium region. The much higher X-ray stopping power of the nitrogen gas caused the gas temperature in the 25 cm case to be very high at the interface; this effect is also seen for the 50 cm case. The pressure distribution

would be very similar to the temperature distribution. Thus, a sharp pressure gradient was created at the gas interface region. The fireball had become essentially a plane pressure source at the interface and could expand in either direction: into the helium or the nitrogen. One would thus expect the stagnation pressure on the wall to be reduced by a factor of 2.

Figure 24 shows the stagnation pressure on the wall, located 300 cm from the explosion, for the four 400 MJ simulations. The peak pressure for the pure nitrogen case is approximately 3 MPa while the peak pressures for either target in helium case is only 1.5 MPa; the maximum pressure point for a pure helium simulation is illustrated for comparison. Essentially the use of two gases with different opacities and X-ray stopping powers changes the fireball expansion from having kinetic energy in one direction to having it in two; the maximum pressures are reduced by 50% simply by momentum and energy conservation arguments.

The limiting case of target to interface distance, the pure helium simulation, had a maximum pressure of approximately 20% of the pure nitrogen case. This helium peak pressure would correspond to a target yield of 200 MJ in a pure nitrogen cavity. The reason for this effect will be described next.

A drawback to using the helium gas, either in conjunction with nitrogen with the geometry of figure 1 or as a single cavity gas, is the resulting increase in surface heat flux. This is simply due to the reduced opacity and X-ray stopping power of the helium. Table 1 shows the maximum surface heat fluxes for the 400 MJ and 800 MJ simulations. One can note the drastic reduction in surface heat loading by using an absorbing gas as opposed to a transparent gas, the helium. Since energy must be conserved, the ratio of pressure loading to surface heat flux can be varied, depending on the particular system limitations. The TDF cavity under consideration is an experimental facility which will explode targets only about 10 times a day; a commercial reactor is expected to have an operation cycle on the order of a fraction of a second. Therefore, the heat flux question is much more important for the reactor, because of wall ablation, than it is for a test facility.

Figure 25 shows velocity profiles for the 25 and 50 cm calculations for the times where the velocity pulse reached the wall. Several features can be observed. First, the pulse arrived at the wall at essentially the same time for both calculations and the peak velocities were also equivalent. The difference

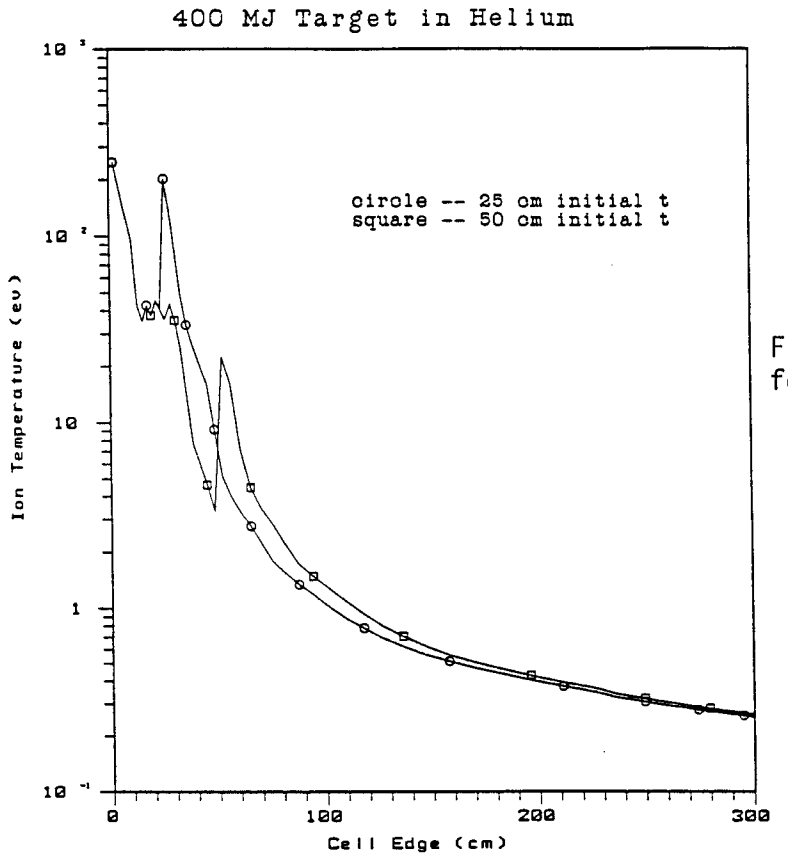


Fig. 23 Initial Gas Temperature for 400 MJ Target in He region.

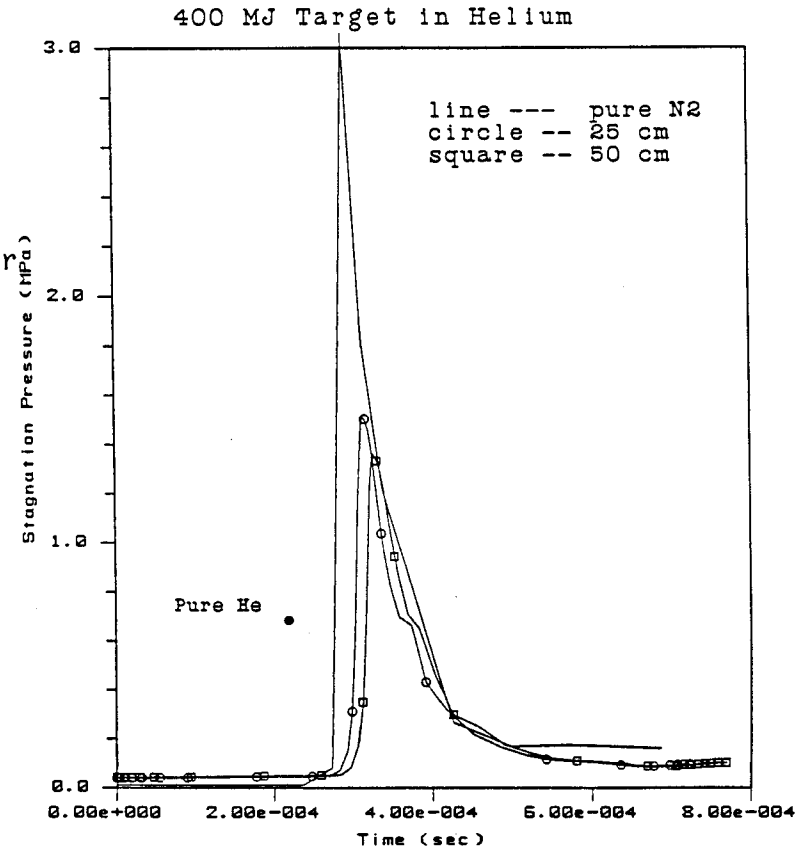


Fig. 24 Stagnation Pressure for 400 MJ Target in He region and pure N_2 case (only max. value for pure He case shown).

in the helium region thickness was not enough change the initial wall loading by the blast wave. Now recall that these calculations were done in a spherical coordinate system. One will note that the fluid in the 25 cm case has reflected from the center and was propagating toward the gas interface; the helium fluid for the 50 cm calculation was still reversing toward the center. An important finding is that the magnitude of the velocity peak in the helium for either calculation is much smaller than the peak in the nitrogen; therefore, wall loading effects from this second pulse would not be important.

TABLE 1

Maximum Surface Stagnation Pressure and Heat Flux

Case	Stagnation Pressure (MPa)	Heat Flux (MW/cm^2)
200 MJ pure Nitrogen	.62	
400 MJ pure Nitrogen	3.06	.03
400 MJ 25 cm	1.51	.18
400 MJ 50 cm	1.36	.13
400 MJ pure He	.68	5.75
800 MJ pure Nitrogen	6.32	.10
800 MJ 25 cm	3.19	.72

3.2 800 MJ Target Yield

A simulation with a single separation distance, 25 cm, was performed for the high yield target case. Figure 26 shows the wall stagnation pressure for both this case and one for a pure nitrogen cavity gas. Again, as was the situation for the 400 MJ targets, the peak pressure has been reduced by 50% with the use of a gas interface region. The maximum surface heat fluxes are given in Table 1. The heat flux follows the same inverse trend with the peak pressure as the 400 MJ simulations.

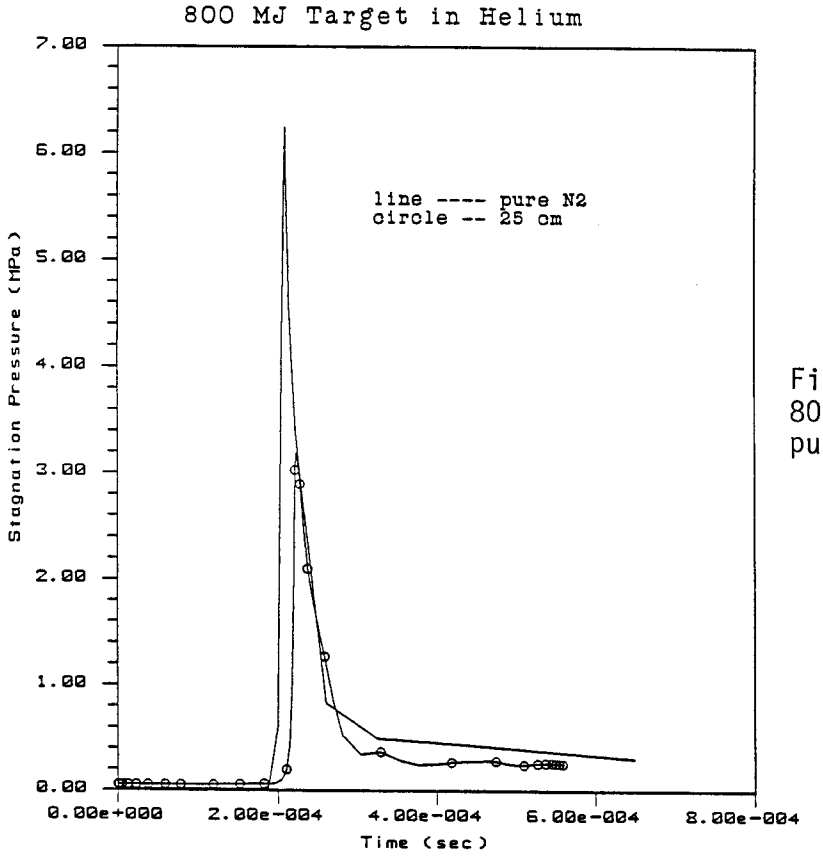


Fig. 25 Stagnation Pressure for 800 MJ Target in He region and pure N_2 case.

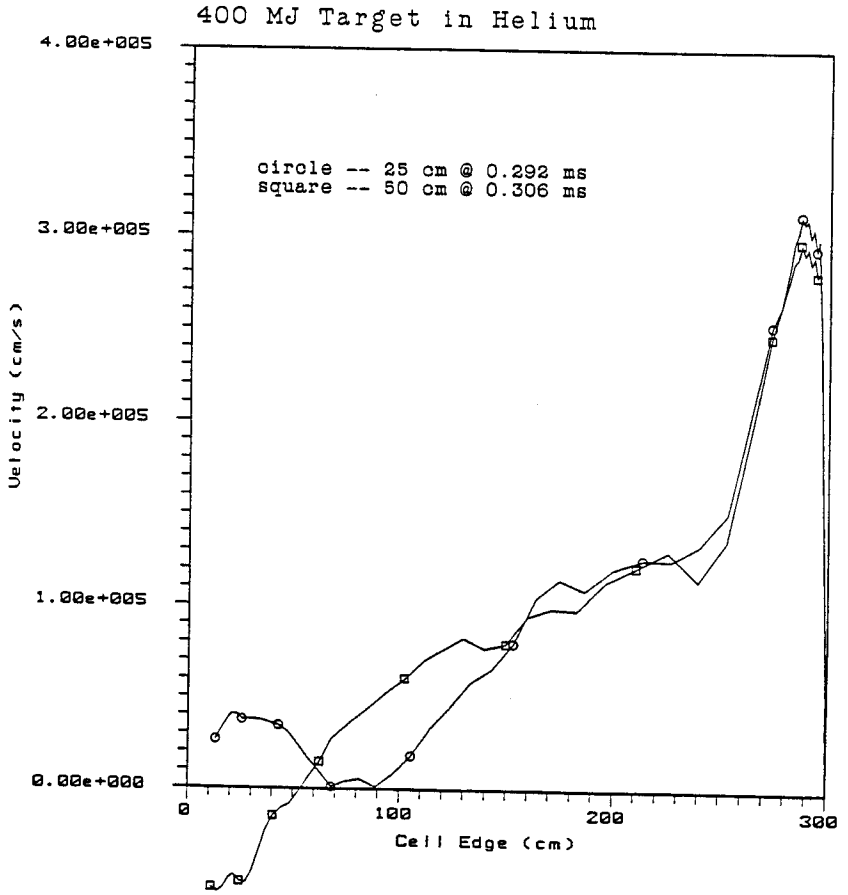


Fig. 26 Velocity Profiles for 400 MJ Target in He. (target at 0 cm).

4. Analysis

4.1 Target in N_2 Region

Figure 27 clearly shows the effects of the venting for the 200 MJ simulations. Here the energy in a pseudo-uniform fireball whose radial profile is taken as the vented case is scaled to the pure nitrogen case where no venting occurs. One can see that the 100 cm separation distance resulted in only a minimal effect while the 10 cm case achieved a reduction of approximately 20%. Since the radial position of the fireball, determined from the locations of the peak velocity, is similar for both the vented and non-vented cases, it is easy to determine the overpressure reduction one would expect. Strong shock theory[3,4] states that the peak stagnation pressure is proportional to the total blast energy and inversely proportional to the radius cubed. Thus for the same radius, the impulse ratio between the vented and non-vented cases simply reduces to the ratio of the fireball energies; therefore, figure 27 gives the pressure reduction directly. An asymptotic value of a 20% pressure reduction was predicted for the 200 MJ simulations.

Figure 28 shows the temporal interface radiation temperature behavior for the three 200 MJ cases; it is equivalently the vented energy flux. This figure helps to interpret the results of the preceding figure. We can see that the temperatures for the 10 and 40 cm cases are essentially the same. Thus the differences between the energy ratios in figure 27 were due to the increased vent area for the 10 cm case. The 100 cm interface was too far from the target and thus its vented energy density was too low to significantly affect the fireball evolution.

4.2 Target in He Region

A significant reduction in the peak wall pressure was calculated for the case where the target is in the helium region. This was due to the conversion of the spherical fireball pressure expansion to a surface source; conservation of momentum and energy lead to a 50% reduction in wall pressure loading. A cavity with a single gas, helium, had an even further stagnation pressure reduction when compared with the case of a pure nitrogen cavity. The penalty for this pressure reduction is the increased surface heat flux as shown in table 1. The use of nitrogen in either a stratified gas mode or as a single cavity gas drastically reduced the wall heat loading. This is just a

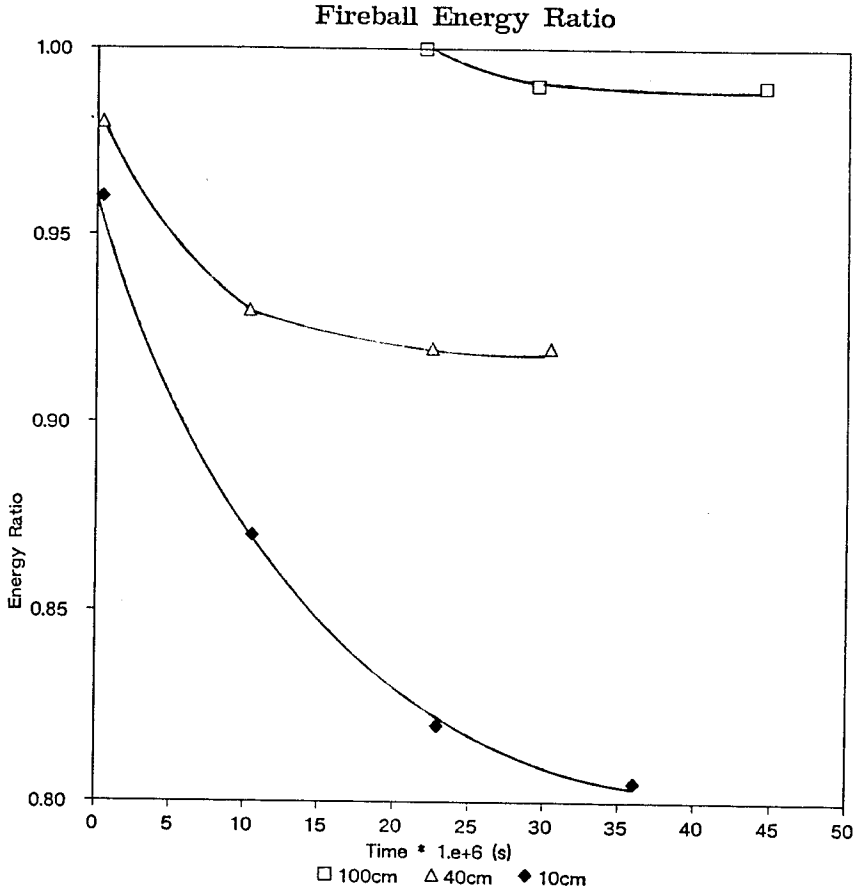


Fig. 27 Ratio of Fireball Total Energy for vented and pure N_2 case (200 MJ)

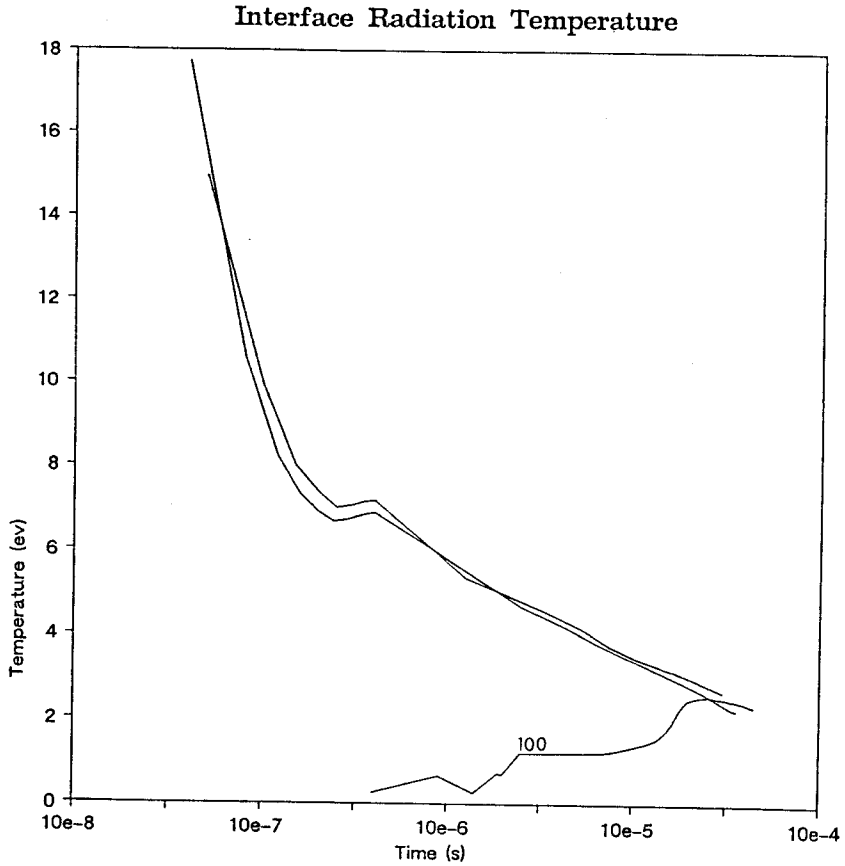


Fig. 28 Radiation Temperature at the gas interface along the z-axis for the 10, 40 and 100 cm cases (200 MJ)

statement about the conservation of energy.

The reduction in surface heat flux from 25 to the 50 cm case for 400 MJ target explosions is an interesting result. Recall that a 1-D simulation in spherical coordinates was used for these simulations. Thus the actual geometry is a sphere of helium in a larger sphere of nitrogen. The reason for the reduction in surface heat flux when the helium sphere was increased from 25 to 50 cm was that due to the low opacity of the He, the ion temperature reached an almost uniform value shortly after X-ray deposition. Now, from figure 23, it is easy to see that this equilibrium value for the 50 cm case will be much lower than for the 25 cm case because roughly the same energy is spread over 8 times the volume. The lower temperature of the 50 cm case simply means a lower source term for the surface heat flux; the initial maximum temperature in the nitrogen region is much lower for the 50 cm than for the 25 cm case. The net result from both of these effects is a reduced heat flux loading on the wall.

5. Conclusions

"Venting" the radiation energy from a target explosion in nitrogen to a helium region, shown in fig. 1, to reduce the surface pressure loading had only a minor effect for the 200 MJ target yields; the 800 MJ targets had little effect. The fireball inertia prevents this loss mechanism to be of significant benefit. The pressure shock is formed at very early times, before any appreciable energy loss can occur. The interesting core flow reversal of the 800 MJ target cases was not seen in the 200 MJ simulations. This could be simply attributed to the finer zoning used to determine the 800 MJ initial gas temperature profile, the X-ray deposition. The core temperature for the 800 MJ case was much higher than the 200 MJ one.

The geometries with the target in the helium region lead to a 50% peak pressure reduction on the instrumentation package due to the conversion of a spherical expansion to a surface source. However, the surface heat flux for this case was higher than for the pure nitrogen cavity. Table 1 summarizes these results for several calculations.

Exploding the target in the helium region will reduce the pressure on the instrumentation package; unfortunately, the diodes will experience a very high heat flux if the geometry of figure 1 is used. The use of a central cell of helium, for example a gas bag as shown in figure 2, might be a compromise between pressure reduction and wall heat flux loading for a test facility. Future calculations will investigate increasing the helium region to determine the minimum nitrogen region required. As the helium region is increased, the pressure loading will be reduced by simple geometric considerations. The target X-rays will expand in a spherical nature; thus when they are stopped in the nitrogen and a surface pressure source created, the energy per unit area will be lower. The surface heat flux will also be reduced by increasing the volume of the helium region which will result in a lower mixed-mean temperature as a radiation source.

ACKNOWLEDGEMENTS

This work has been supported by Sandia National Laboratory and USDOE under contract # DE-AS08-83DP40189.

6. References

1. Moses, G.A., R.R. Peterson,et al., "Preconceptual Design of the Light Ion Beam Fusion Target Development Facility", UWFDM-664, November 1985.

2. Badger, B., et al., "Light Ion Beam Fusion Target Development Facility Studies: Progress Report for the Period Oct. 1, 1984 to Sept. 30, 1985", UWFDM-652, 30 September 1985.

3. Zek'dovich, Y.B., Y.P. Raizer, Physics of Shock Waves and High-Temperature Hydrodynamic Phenomena, Academic Press, 1966.

4. Bethe, H.A., et al., "Blast Wave", LA-2000, August 1947.

5. Moses, G.A., R. Spencer, "Compact-Electron-Beam or Light-Ion-Beam Fusion Reactor Cavity Design Using Non-Spherical Blast Waves", Nuclear Fusion, Vol. 19, No.10 (1979), pp 1386-1388.

6. Pomraning, G.C., "Radiation Hydrodynamics", LA-UR-82-2625.

7. Peterson, R.R., G.A. Moses, "MIXERG-An Equation of State and Opacity Computer Code", UWFDM-464, March 1982.

8. Moses, G.A., T.J. McCarville, R.R. Peterson, "Documentation for MF-FIRE, A Multifrequency Radiative Transfer Version of FIRE", UWFDM-458, March 1982.

9. Uesaka, M., G.A. Moses, "Parametric Survey of Microfireball Calculations for the Light Ion Fusion Target Development Facility Design", UWFDM-533, August 1983.

10. Cloutman, L.D., C.W. Hirt, N.C. Romero, "SOLA-ICE: A Numerical Solution Algorithm for Transient Compressible Fluid Flows", LA-6236, July 1976.

11. Spitzer, L., Physics of Fully Ionized Gases, Interscience, 1962.

12. Collatz, L. The Numerical Treatment of Differential Equations, Springer, 1966.

13. Schlichting, H., Boundary-Layer Theory, McGraw-Hill, 1968.

7. Appendix A

7.1 GAS2DRFD Overview

GAS2DRFD is a 2-D Eulerian radiation fluid dynamics computer code. It can simulate problems in spherical (R), cartesian (X-Y), or cylindrical (R-Z) coordinate systems. The code comprises about 2200 lines of standard Fortran (about 30% comments). It is written in a simple modular fashion which allows easy modification. The code is also highly vectorized; although it is a 2-D code, the arrays are simple vectors. Thus, the typical problem under discussion would have DO loop indices from 1 to 5000 rather than nested loops of from 1 to 70. This allowed the full vector potential of the Cray computer to be realized. The code will be briefly described here; only special, non-standard features will be discussed.

The SOLAICE[10] algorithm was used to solve the standard compressible Navier-Stokes equations. A staggered mesh was used with cell edge velocities. Essentially, first upwind differencing was used on the convective terms and centered differencing on the diffusion terms. The SOLAICE technique is first order accurate in time. The convective terms vary from explicit to implicit based on the local pressure change during the time step; a simple point SOR method was used to iteratively solve the continuity equation and reduced momentum and energy equations. The fluid diffusion terms and convective energy transport were then updated in a explicit manner.

A single temperature diffusion approximation[6] was used to simulate the radiation transport. Local thermodynamic equilibrium was assumed between the electrons and ions; a single temperature was used for the gas. A three parameter, both temperatures and fluid density, equation-of-state table was used to determine the Planck and Rosseland opacities. The coupling coefficients between the radiation and plasma fields are highly nonlinear; a simple explicit or Crank-Nicolson scheme for the diffusion terms would be inadequate. An iterative scheme to update these coefficients was used with an explicit differencing scheme to obtain the updated energy densities. A standard flux limiter model[8] was used to limit the radiation field propagation speed.

7.2 Governing Equations

Let:

ρ -- fluid density

u -- fluid velocity along X or R direction

v -- fluid velocity along Y or Z direction

ξ -- coordinate system controller

- 0 -- cartesian
- 1 -- cylindrical
- 2 -- spherical

E_m -- fluid specific internal energy

E_r -- radiation energy density

T_m -- material (gas) temperature

T_r -- radiation temperature

μ -- fluid viscosity

k_m -- fluid thermal conductivity

k_r -- effective radiation thermal conductivity (includes the flux limiter logic found in see ref. 8)

g_x, g_y -- body accelerations (e.g. gravity)

γ -- ratio of fluid specific heats

ω_a -- radiation absorption coefficient (see ref. 6)

ω_e -- radiation emission coefficient (see ref. 6)

P -- total pressure (material + radiation)

note: - x direction is either X or R depending on ξ

- viscous dissipation terms, Φ, are neglected in the material energy equation
- Stokes Hypothesis[13] used for viscous momentum terms

Continuity Equation:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} + \frac{\partial \rho v}{\partial y} + \frac{\xi \rho u}{x} = 0$$

Momentum Equations: (non-conservative form)

$$\rho\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = \rho g_x + -\frac{\partial P}{\partial x} + \mu\left(\frac{4}{3}\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{1}{3}\frac{\partial^2 v}{\partial x \partial y}\right) + \frac{4}{3}\frac{\xi\mu}{x}\left(\frac{\partial u}{\partial x} - \frac{u}{x}\right)$$

$$\rho\left(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) = g_y + -\frac{\partial P}{\partial y} + \mu\left(\frac{4}{3}\frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial x^2} + \frac{1}{3}\frac{\partial^2 u}{\partial x \partial y}\right) + \frac{\xi\mu}{x}\left(\frac{\partial v}{\partial x} + \frac{1}{3}\frac{\partial u}{\partial x}\right)$$

Energy Equations: (non-conservative form)

$$\rho\left(\frac{\partial E_m}{\partial t} + u\frac{\partial E_m}{\partial x} + v\frac{\partial E_m}{\partial y}\right) = -p\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\xi u}{x}\right) + k_m\left(\frac{\partial^2 T_m}{\partial x^2} + \frac{\partial^2 T_m}{\partial y^2} + \frac{\xi}{x}\frac{\partial T_m}{\partial x}\right) + \omega_a E_r - \omega_e T_m$$

$$\rho\left(\frac{\partial E_r}{\partial t} + u\frac{\partial E_r}{\partial x} + v\frac{\partial E_r}{\partial y}\right) = -p\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\xi u}{x}\right) + k_r\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T_r}{\partial y^2} + \frac{\xi}{x}\frac{\partial T_r}{\partial x}\right) - \omega_a E + \omega_e T_m$$

One will note that since GAS2DRFD was based on a Navier-Stokes solver, the viscous momentum terms have been retained; these terms are not typically included in a radiation fluid dynamics code. For the problem under consideration, the fluid viscosity, μ, and the fluid thermal conductivity, k_m, were very small. The flow was essentially inviscid and the thermal transport dominated by the absorption and reemission of the radiation and fluid fields.

7.3 Equation-of-State

The present problem consisted of two different gases: nitrogen and helium. A three parameter EOS table [7] was used to determine the nitrogen opacities: both Planck and Rosseland. The gas heat capacity, charge state, and specific energy density were also obtained from it. A standard Spitzer model [11] was used for the thermal conductivity. The Navier-Stokes gas dynamics code

was originally written assuming an ideal gas; that is, γ, the ratio of specific heats, was held constant. This limitation was removed by simply calculating a local time dependent γ for each grid point from the EOS tables. The actual gas behavior was then able to be modelled.

The helium region was treated differently. Since GAS2DRFD was used to model the case where the target and the diagnostic module were both in the nitrogen region, the helium region served only as a special boundary condition for the nitrogen field. Therefore its equation-of-state was greatly simplified. First, an ideal gas assumption was used: γ was held constant. Next, the helium was assumed to be optically transparent; the radiation flux propagated through the gas without being absorbed. Finally, the Spitzer thermal conductivity[11] was also used for the helium gas.

7.4 Special Features

The numeric techniques used to solve the hydrodynamic motion and the radiative transport are quite standard. However, several special features were incorporated into GAS2DRFD to better simulate the present problem. These include procedures for the initial conditions, time step control, and special diffusion stencils for the radiation transport.

7.4.1 Ion Deposition.

An X-ray deposition model was not included in GAS2DRFD. Therefore, the following procedure was utilized to obtain the initial energy profile in the cavity. First, the MFFIRE code[8] was used to determine a 1-D (spherical) initial X-ray deposition profile with the target in nitrogen. All of the energy deposition was assumed to be absorbed in the gas; the radiation energy density remained at its initial state. No deposition was assumed in the helium region; it was modelled as optically transparent. After this initial time zero profile was established, the GAS2DRFD code was used to obtain the temporal solution; no further X-ray deposition modelling was used.

7.4.2 Time Step.

This problem has two vastly different time constants: the hydrodynamic motion and the energy exchange between the radiation and gas energy fields. The differences were further amplified by the use of a 2-T radiation transport model; the resulting highly nonlinear coupling coefficients made the equations very stiff. Therefore, for computational expediency, two different time steps were used in GAS2DRFD. The fluid motion was held constant during the

radiation time step; only the pressure work term was updated. At the end of a given number of radiation time steps, the hydrodynamic equations were solved. The radiation time step was typically from 1,000 to 10,000 times smaller than the hydrodynamic time step; the CFL limit for the radiative field was approximately 10^{-11} sec (based on the speed of light and a 5 cm mesh).

7.4.3 Radiation Diffusion Stencil.

A second order centered difference scheme typically uses a 5-point symmetric stencil on a finite difference mesh. This creates a diffusion flux bias in the orthogonal directions (major mesh axes). A cylindrical coordinate system was used for the problem under consideration; the spherical expansion of the fireball caused the dominant fluxes to be along the diagonals of the computational cells. Initial calculations with the 5-pt stencil resulted in a definite cylindrical bias for the spherical expansion along the major mesh axes. Figure 29a shows a typical result for this stencil; the rectangular bias is clearly shown. A 9-pt stencil was developed which used the additional four nearest points, along the cell diagonals. The 8 perimeter points were essentially evenly weighted (adjusted for different physical lengths). This is unlike the 9-pt higher ordered methods of Collatz [12] which just increased the number of points along the major axes; these methods simply more accurately resolved the cylindrical bias. The new 9-pt stencil greatly reduced this bias and resulted in a near-spherical expansion. One can compare fig. 29a for the 5-pt stencil and fig. 29b for the new 9-pt stencil; the improvement is obvious.

An explicit solution scheme was used to advance the solution in time because it could be fully vectorized. There are no advantages to implicit schemes for calculating time accurate solutions. For the present problem, the numerical, flux limited [8] diffusion propagation speed from an explicit scheme was more time accurate than either Crank-Nicolson or fully implicit, with an infinite diffusion speed, methods.

Figure 30 compares the radial velocity and ion temperature for a calculation performed in both a 1-D spherical and 2-D cylindrical coordinate system. The difference equations in GAS2DRFD were analytically manipulated to split them into a cartesian and radial portion; a simple multiplier was then used to indicate the coordinate system. The comparisons are quite good although the equations were not in conservative form.

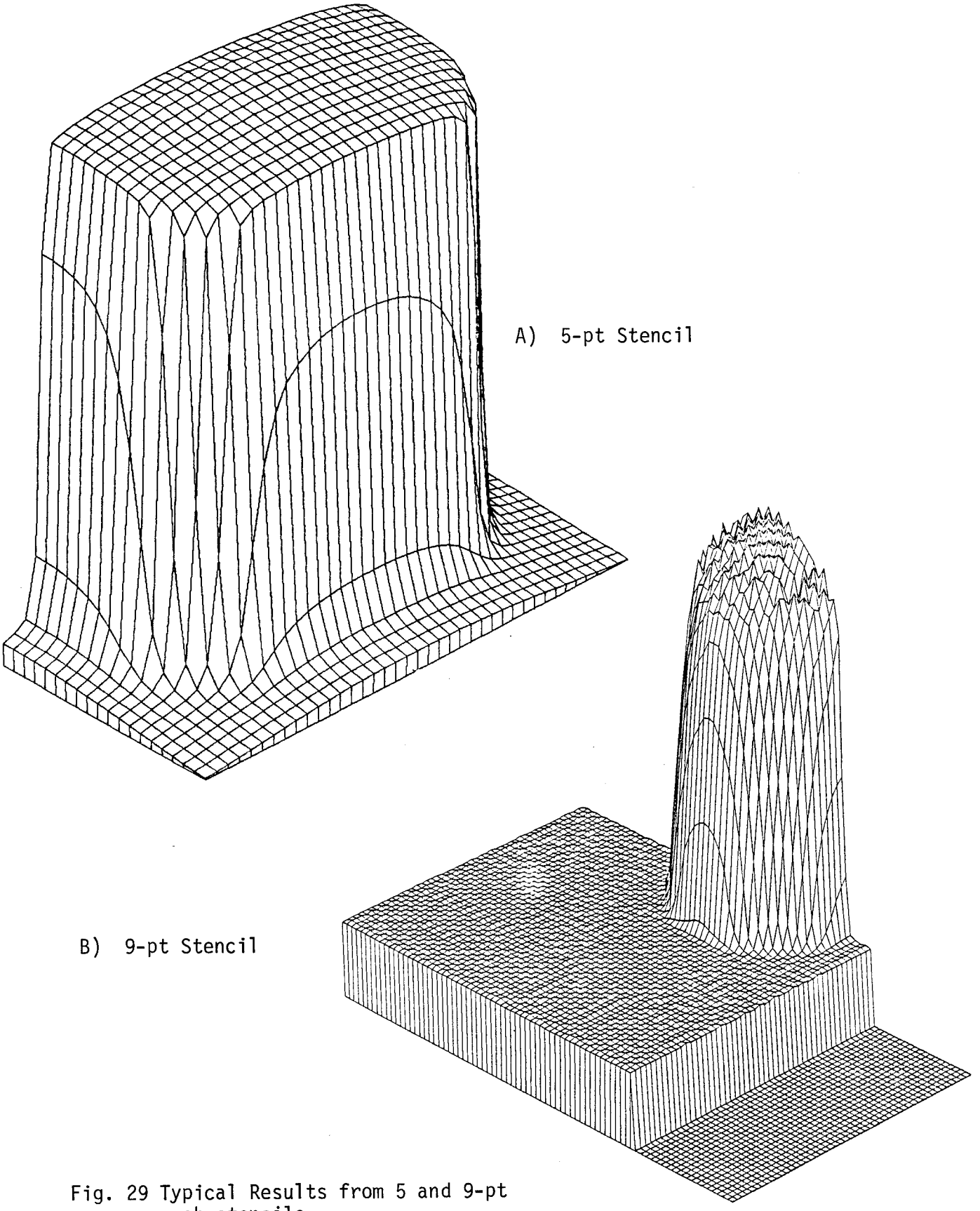
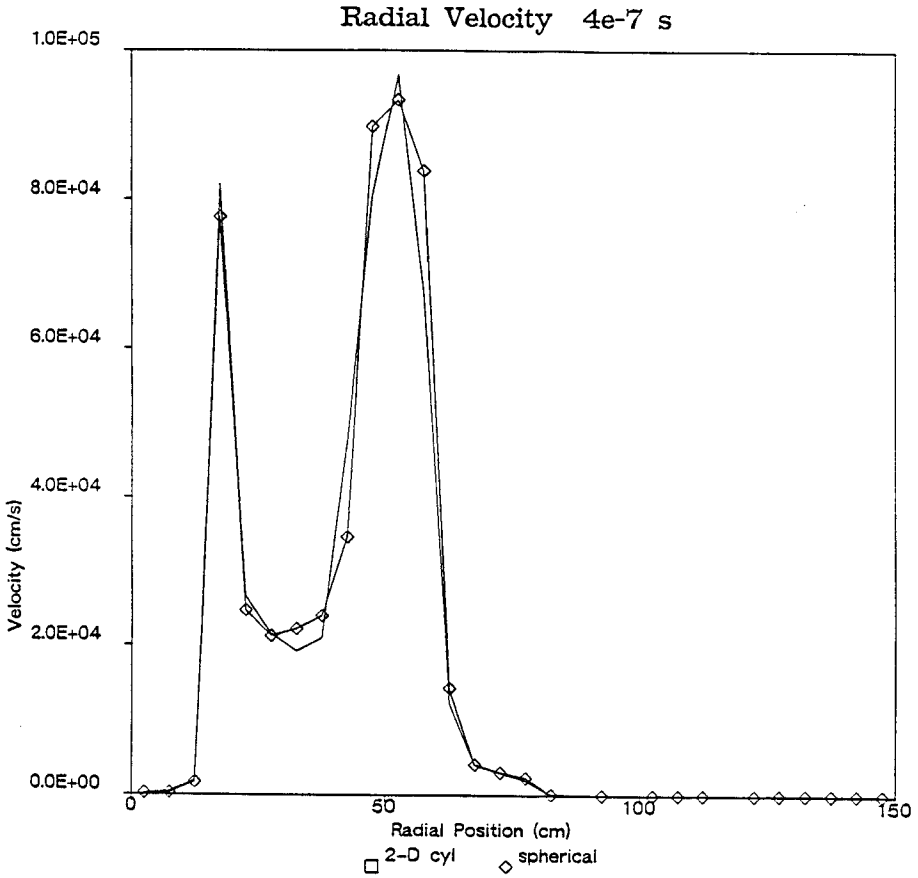


Fig. 29 Typical Results from 5 and 9-pt mesh stencils.



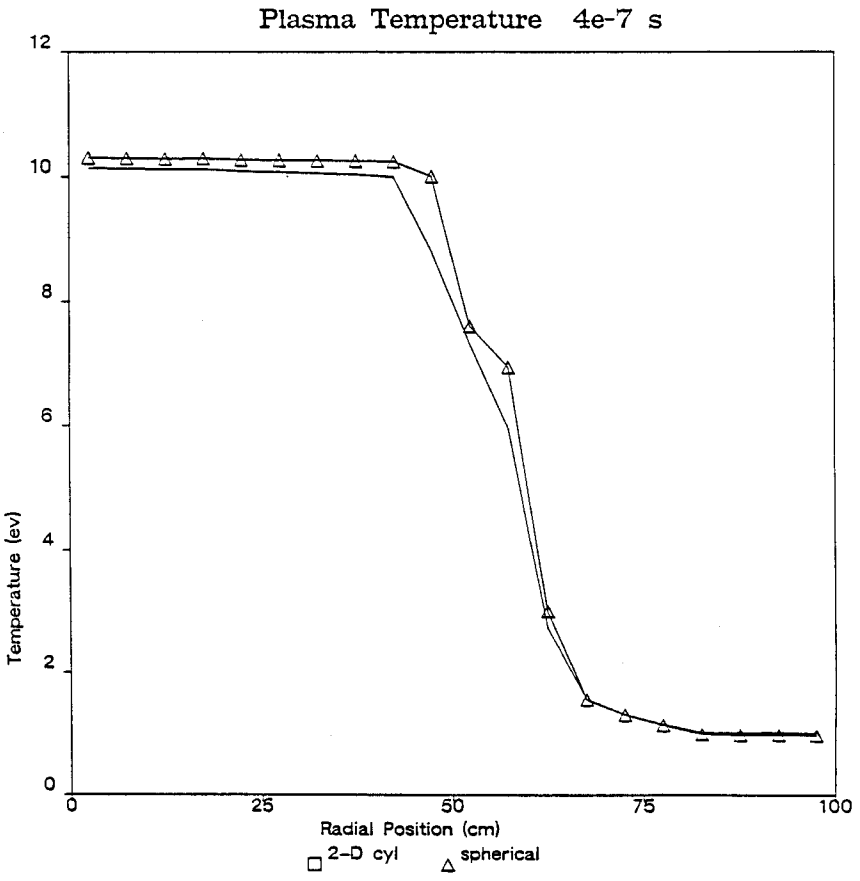


Fig. 30 Comparisons between 1-D spherical and 2-D cylindrical.

```
      program gas2d
c version  4.9  rfd
c
c       this is a special version of gas2d--modified with radiation hydro.
c       2T approx made (single equation for radiation energy density).
c       an equation of state package is also integrated into the code.
c       the plasma ideal gas equation has been modified by using a mesh
c       varying ratio of specific heats .
c
c       caution:  nine pt diffusion stencil currently requires uniform mesh
c                 with aspect ratio of unity
c
c         special modifications for the 2 gas layer blast problem
c         let----- gamma- ratio of specific heats for 'upper' region
c                  xkap2- thermal conductivity for 'upper region'
c                  bvar1- left hand boundary efor fit A
c                  ivar2- location centre of energy deposition
c                  bvar3- offset for the B fit
c                  bvar4- plasma temperature at centerline (ev)
c                  bvar5- epsilon energy for Er diffusion model
c                  bvar6- exponential attenuation factor for B (1/cm)
c                  bvar7- exponential attenuation factor for A (1/cm)
c                  bvar8- right hand boundary of fit A
c                  dmpcd- number of time steps for dump c/d
c                  er1  - initial radiation temp for 'upper' region (ev)
c                  iloche-j location of the He 'upper region'
c                  core - radius of initial energy core (cm)
c                  delt - overall problem time step (control hydro and output)
c                  dtrad- lagrangian radiation energy transfer time step only
c                         note: dt2 << dt1
c                  weight-weight factor for 9 pt diffusion stencil
c                         for 'normal' axes (rotated axes is 1.-weight)
c                  con(23)--relative error in tp (subroutine e2t)
c                  con(24)--relative temp difference in RMFP
c                  con(25)--floor tp in input
c                  con(26)--floor value for Er (in ev)
c
c              assume He to be the 'upper' gas.  Let He be treated as
c                     transparent. (see variable wtmol2)
c
c         units--- er     J/cm3  (radiation density)
c                  x,y    cm
c                  P,Pr   Mpa or J/cm3  (output only)
c                             dyne/cm2 (erg/cm3) -- internal use
c                  rho    gm/cm3
c                  tp,tr  ev
c                  e      J/gm   (plasma specific energy density)
c
c**********************************************************************
c
c       this is a one phase 2-d compressible code for the full navier-stokes
c       equations  (e.g. includes an energy equation)
c         the present convective transport formulation is first order accurate
c
c       the solution procedure is sola-ice  ref la-6236 (1976)
```

```
c            modified for variable mesh
c                        pressure iteration modification
c
c         vectorized version---single vector instead of 2-d matrix
c         note:  k     = i,j
c                k+1   = i+1,j
c                k-1   = i-1,j
c                k+imx = i,j+1
c                k-imx = i,j-1
c
c                imx = length of i subscript (x or r dimension)
c                jmx = length of j subscript (y or z dimension)
c
c                kpta = (2,2)
c                kptb = (imx-1, jmx-1)
c                kptc = (imx, jreg1p1)
c                kptd = (imx, jmx)
c                kpte = (imx, jmx-1)
c                kptf = (imx, jmx-2)
c         recall---boundary conditions meshes around physical mesh--imx=imesh+2
c
c          note---cdir$ ivdep forces do loop vectorization
c
c         input values:
c
c           tinit---initial problem start time
c           tend----problem end time
c           iterimt-maximum # pressure iterations
c           rstrt---0 = no   1 = yes
c           xmu-----absolute viscosity (constant)
c           cyl----- 0 for cartesian  and 1 for cylindrical
c           espi----pressure iteration convergence criterion
c           xkap----thermal conductivity
c           gx------body acceleration in the positive x-direction
c           gy------body acceleration in the positive y-direction
c           ui------initial x velocity
c           vi------initial y velocity
c           omega---sor relaxation factor for pressure iteration
c           alpha---controls the amount of donor cell fluxing
c                   1.0----full donor cell
c                   0.0----centered differencing
c           wl------bounadry condtion-left side
c           wr------boundary condition-right side
c           wt------boundary condition-top side
c           wb------boundary condition-bottom
c                   1--rigid, free-slip wall
c                   2--rigid, no-slip wall
c                   3--continuative outflow wall
c                   4--periodic (needs symmetric bc)
c                   5--user defined
c           gamma---ratio of specific heats
c           asq-----square of sound speed used in the stiffened gas eos
c                        not currently implemented
c           rhoi----reference density
c           eamb----ambient temperature for i.c. and/or b.c.
```

```
111     c          edrive--temperature of right boundary for thermally driven
112     c                  cavity problem
113     c          wtmol---molecular weight in amu
114     c          rgas-----gas constant (8.3143e7 in cgs)
115     c          ireg1,2,3--number of real x (r) meshes in region
116     c          dx1,2,3----width of mesh in corresponding region
117     c          jreg1,2,3--number of real y (z) meshes in region
118     c          dy1,2,3----width of mesh in corresponding region
119     c                grid system such that ireg1 and jreg1 are low numbered
120     c                mesh points
121     c          cnrst---number of time steps per restart dump
122     c          cndmpa--number of time steps per dump type a
123     c                  pressure iter, mesh ke, total energy, mass
124     c          cndmpb-----number of time steps per dump type b
125     c          icntrl-----control for type b dump fileds
126     c                     format field = 87654321  (0-no dmp, 1-dmp data)
127     c                     1--u velocity
128     c                     2--v velocity
129     c                     3--pressure
130     c                     4--density
131     c                     5--specific internal energy
132     c                     6--temperature
133     c                     7--radiation energy density
134     c                     8--gamma-1 in ideal gas eos
135     c        dt1--------time step 1
136     c        ndt1-------number of time steps of dt1
137     c        dt2,ndt2---time step 2 control
138     c        bvar1------boundary condition 1
139     c        bvar2
140     c        bvar3
141     c        bvar4
142     c        bvar5
143     c        bvar6
144     c        bvar7
145     c        bvar8
146     c
147     c
148     c          stiffened gas eos   p = asq *(rho-rhoi) + (gamma-1)*rho*i
149     c                  ref  fluid dynamics  la-4700
150     c
151     c          note--restart option used to define velocity, pressure, energy
152     c                and density fields.  cycle 0 calculation is required to
153     c                determine beta for first 'real' time step
154     c
155     c
156     c          output file descriptions:
157     c
158     c            gswrst---echo input data and contains array dumps for
159     c                     printing or editing for restart
160     c                     also contains job timing info in last few lines
161     c            gsdmpa---global mesh data for conservation checks
162     c            gsdmpb---particular time and spatial data for post process plottin
163     c
164           integer cycle, wl, wr, wt, wb
165           real ke
```

```
      integer*4 itemp, icntrl, ndt1, ndt2
      integer rstrt
      dimension xput(61), mask(9), value(8)
c
      character*80 header
      character*80 scratch
      character*3  depen(8)
      character*8 gasinp
      character*8 gsdmpa
      character*8 gsdmpb
      character*8 gsdmpc
      character*8 gsdmpd
      character*8 gswrst
      character*8 gsinrst
      character*8 eostab
      common /mesh/ xc(9999)  , yc(9999)  , dx(9999)   , dy(9999)  ,
     1              rdx(9999) , rdy(9999) ,
     2              rdxc(9999), rdyc(9999), rxc(9999),   r2dx(9999),
     3              r2dy(9999)
      common /wrk/  un(9999)  , vn(9999)  , rhon(9999), en(9999)  ,
     1              pr(9999)  , work(9999,7)
      common /np1/  u(9999)   , v(9999)   , rho(9999) , e(9999)   ,
     1              er(9999)  , p(9999)   , tp(9999)
      common /prop/  gam1(9999), rosmfp(9999), rmfp1t(9999),
     1             rmfp2t(9999), rcsubv(9999),   xkap(9999)
      common /a/    beta(9999)  , xmask(9999), con(30)
      common /e/  cycle, wl, wr, wt, wb, iter, imx, jmx, delt, xmu,
     1            im1, jm1, im2, jm2, iterlmt, cyl, epsi, gaminit,
     2            gx, gy, ui, vi, cnrst, cndmp, omega, alpha, t, am1,
     3            gamma, asq, rhoi, eamb, dtrad, otd, ftd, xkap2,
     4            kpta, kptb, kptc, kptd, kpte, kptf, eamb2, rcsubin,
     5            bvar1, ivar2, bvar3, bvar4, bvar5, bvar6, bvar7,bvar8
      data length /9999/
      data pi , ixero /3.1415927, 0/
      data nimp /61/
6data wtmol2 / 14. /
      data depen /'U  ','V  ','Pr ','Rho','Ep ','Tp ','Tr ','gm1'/
c
c        set file definitions
c
      data ir,lwrst,ireos,idmpa,idmpb,irst/ 5,6,4,1,3,2/
      data idmpc, idmpd /7,8/
      data gsdmpa, gsdmpb, gasinp, gswrst, gsinrst
     1     /'gsdmpa', 'gsdmpb', 'gasinp', 'gswrst', 'gsinrst'/
      data eostab / 'eostab'/
      data gsdmpc, gsdmpd /'gsdmpc', 'gsdmpd'/
c
c        ir--input problem file
c        iwrst--output restart file
c        irw-console
c        idmpa-plot save file a
c        idmpb-plot save file b
c        irst-initial condition or restart file
c
      call dropfile (0)
```

```
c
c        define i/o files
c
      open (1, file=gsdmpa ,status='new')
      open (3, file=gsdmpb ,status='new')
      open (5, file=gasinp ,status='old')
      open (6, file=gswrst ,status='new')
      open (4, file=eostab ,status='old')
      open (7, file=gsdmpc ,status='new')
      open (8, file=gsdmpd ,status='new')
c
c        input problem parameters
c
      read(ir,10) header
      read(ir,11) (xput(i),i=1,nimp)
 10   format(a)
 11   format(e10.0)
         tinit= xput(1)
         tend = xput(2)
         iterlmt=xput(3) + .001
         rstrt= xput(4) + .001
         xmu  = xput(5)
         cyl  = xput(6)
         epsi = xput(7)
         xkap2= xput(8)
         gx   = xput(9)
         gy   = xput(10)
         ui   = xput(11)
         vi   = xput(12)
         omega= xput(13)
         alpha= xput(14)
         wl   = xput(15) + .001
         wr   = xput(16) + .001
         wt   = xput(17) + .001
         wb   = xput(18) + .001
         gamma= xput(19)
         asq  = xput(20)
         rhoi = xput(21)
         eamb = xput(22)
         edrive=xput(23)
         wtmol= xput(24)
         rgas = xput(25)
         ireg1= xput(26) + .001
         dx1  = xput(27)
         ireg2= xput(28) + .001
         dx2  = xput(29)
         ireg3= xput(30) + .001
         dx3  = xput(31)
         jreg1= xput(32) + .001
         dy1  = xput(33)
         jreg2= xput(34) + .001
         dy2  = xput(35)
         jreg3= xput(36) + .001
         dy3  = xput(37)
         er1  = xput(38)
```

```
276             cnrst= xput(39)
277             cndmpa=xput(40)
278             cndmpb=xput(41)
279             icntrl=xput(42) + .001
280             delt = xput(43)
281             ndt1 = xput(44) + .001
282             dtrad= xput(45)
283             weight=xput(46)
284             bvar1= xput(47)
285             ivar2= xput(48) + .001
286             bvar3= xput(49)
287             bvar4= xput(50)
288             bvar5= xput(51)
289             bvar6= xput(52)
290             bvar7= xput(53)
291             bvar8= xput(54)
292             dmpcd= xput(55)
293             iloche=xput(56) + .001
294             core = xput(57)
295             con(24)=xput(58)
296             con(23)=xput(59)
297             con(25)=xput(60)
298             con(21) = xput(61)
299     c
300     c         echo print input file
301     c
302           rewind ir
303           read(ir,10) scratch
304           write(iwrst,101) scratch
305           do 91 i=1,nimp
306              read(ir,10) scratch
307              write(iwrst,102) i,xput(i),scratch
308      102     format(1x,i2,1x,e12.5,5x,a80)
309      101     format(1x,a80)
310      91   continue
311     c*********************************************************
312     c
313     c         compute constant terms
314     c
315     c*********************************************************
316           ibar = ireg1 + ireg2 + ireg3
317           jbar = jreg1 + jreg2 + jreg3
318           imx  = ibar + 2
319           jmx  = jbar + 2
320           im1  = imx - 1
321           jm1  = jmx - 1
322           im2  = imx - 2
323           jm2  = jmx - 2
324           ileft= ivar2*imx + 1
325           iright=ileft + im1
326           kpta = imx + 2
327           kptb = imx * jm1 - 1
328           kptc = imx * (iloche+1) - 1
329           kptd = imx * jmx
330           kpte = imx * jm1
```

```
      kptf = imx * (jmx-2)
      if (kptd.gt.length) go to 990
c
      j1 = 1
      j2 = kptc + 1
c
      otd   = 1./3.
      ftd   = 4./3.
      am1   = 1. - alpha
c
c
c        con array initialization
c
      con(1)  = 7.6387e-22
      con(2)  = 0.7500e+10
      con(3)  = 2.2916e-11
      con(4)  = 3.0000e+10
      con(5)  = 1./ con(1)
      con(6)  = 6.0230e+23 / wtmol
      con(8)  = 1.2175e+02
      con(9)  = 1.6020e-19
      con(11) = wtmol / rgas
      con(12) = 1./11605.
      con(13) = 11605.
      con(14) = er1
      con(15) = 0.001
      con(16) = 1.e+07
      con(17) = 1.e-07
      con(18) = core
      con(19) = (1.41421 * dx1)
      con(10) = 1./con(19)
      con(7)  = 0.5*dx1
      con(20) = weight
c
c        convert from K to ev
c
      con(1)  = con(1) * (con(13))**4
      con(3)  = con(3) * (con(13))**4
      con(5)  = 1./con(1)
      con(11) = con(11) * con(12)
      xkap2   = xkap2 * con(13)
      rgas    = rgas * con(13)
      con(26) = con(1) * 0.25**4
      con(22) = con(26)
6con(29) = .00001 * rhoi
6con(30) = 1.0
c
      gaminit = gamma - 1.
      rcsubin = wtmol * gaminit / rgas
      rcsub2  = wtmol2 * gaminit / rgas
      tpinit  = eamb
      con(27) = 0.1 / rcsubin
c
c*************************************************************
c
```

```
c        initialize mesh stuff for variable width option
c        do x mesh first, then y mesh
c
c*************************************************************
      dx(1)  = dx1
      rdx(1) = 1./dx1
      xc(1)  = -0.5 * dx1
      rxc(1) = 1. / xc(1)
c
      do 15 i=2,im1
         dxa = dx3
         if (i. le. (ireg1+ireg2+1) ) dxa = dx2
         if (i. le. (ireg1 + 1) )     dxa = dx1
         dx(i) = dxa
         rdx(i) = 1./dxa
         xc(i)  = xc(i-1) + 0.5*( dx(i-1) + dx(i) )
         rxc(i) = 1./xc(i)
 15   continue
      dx(imx) = dx(im1)
      rdx(imx) = 1./dx(imx)
      xc(imx)  = xc(im1) + 0.5*( dx(im1)+dx(imx))
      rxc(imx)  = 1. / xc(imx)
c
      do 16 i=1,im1
         rdxc(i) = 1./(dx(i) + dx(i+1))
 16   continue
      rdxc(imx)= 0.5/dx(imx)
c
c        now fill entire matrix
c
ctim
      do 17 i=1,imx
         do 17 j=1,jm1
            k = i + imx*j
            dx(k)  = dx(i)
            rdx(k) = rdx(i)
            xc(k)  = xc(i)
            rxc(k) = rxc(i)
            rdxc(k)= rdxc(i)
 17   continue
c
c
c        now the y mesh variables
c
c
      dy(1)  = dy1
      rdy(1) = 1./dy1
      yc(1)  = -0.5 * dy1
      do 20 j=2,jm1
         dya = dy3
         if ( j. le. (jreg1+jreg2+1) ) dya = dy2
         if ( j. le. (jreg1+1) )       dya = dy1
         kpt      = (j-1) * imx + 1
         dy(kpt)  = dya
         rdy(kpt) = 1./dya
```

```
441            yc(kpt)  = yc(kpt-imx) + 0.5*( dy(kpt-imx) + dy(kpt) )
442     20   continue
443          kpt      = kpte + 1
444          dy(kpt)  = dy(kpt-imx)
445          rdy(kpt) = 1./dy(kpt-imx)
446          yc(kpt)  = yc(kpt-imx) + 0.5*( dy(kpt-imx)+dy(kpt))
447          do 21 k=1,kpte,imx
448             rdyc(k) = 1./(dy(k) + dy(k+imx))
449     21   continue
450          rdyc(kpt)  = 0.5/dy(kpt)
451    c
452    c        fill up entire mesh
453    c
454    ctim
455          do 22 kj=1,kptd,imx
456             do 22 i=1,im1
457                k       = kj + i
458                dy(k)   = dy(kj)
459                rdy(k)  = rdy(kj)
460                yc(k)   = yc(kj)
461                rdyc(k) = rdyc(kj)
462     22   continue
463    c
464    c        compute 2nd order diffusion delx, dely
465    c
466          do 18 k=kpta,kptb
467             r2dx(k) = 1. / ( dx(k-1)   + 2.*dx(k) + dx(k+1)   )
468             r2dy(k) = 1. / ( dy(k-imx) + 2.*dy(k) + dy(k+imx) )
469     18   continue
470    c*********************************************************************
471    c
472    c        initialize mask vector---1.0 interior mesh,  0.0 bc mesh
473    c
474    c*********************************************************************
475    c
476          do 23 k=kpta,kptb
477             xmask(k)      = 1.
478     23   continue
479    c
480    cdir$ ivdep
481          do 24 k=1,imx
482             xmask(k)      = 0.0
483             xmask(k+kpte) = 0.0
484     24   continue
485    c
486    cdir$ ivdep
487          do 25 k=1,kptd,imx
488             xmask(k)      = 0.0
489             xmask(k+im1)  = 0.0
490     25   continue
491    c*********************************************************************
492    c
493    c        initialize counters
494    c
495    c*********************************************************************
```

```
      t     = tinit
      iter  = 0
      cycle = 0
      trst  = tinit
      tdmpa = tinit
      tdmpb = tinit
      tdmpcd= tinit
c*********************************************************************
c
c        initialize dependent variables
c          call subroutine init2 to input eos tables
c
c          check for standard/restart option
c
c*********************************************************************
c
      call init2 (ireos)
c
      if (rstrt.eq.1) then
         open (2, file=gsinrst, status='old')
         call restart (irst,iwrst)
         call eos (j1,j2)
         go to 58
      endif
c
      do 56 kk=1,2
c
c         note:  eamb is temperature on input
c
         if (kk.eq.1) then
            rhoinit = rhoi
            einit   = eamb / rcsubin
            pinit   = gaminit * rhoinit * einit * con(16)
            erinit  = con(1) * con(14)**4
            ka      = 1
            kb      = kptc + 1
         else
            rhoinit = rhoi * wtmol2 / wtmol
            eamb2   = eamb / rcsub2
            einit   = eamb2
            pinit   = gaminit * rhoinit * einit * con(16)
            erinit  = con(1) * con(14)**4
            ka      = kptc + 2
            kb      = kptd
         endif
         do 56 k=ka,kb
            u(k)      = ui
            v(k)      = vi
            e(k)      = einit
            rho(k)    = rhoinit
            p(k)      = pinit
            rhon(k)   = rhoinit
            en(k)     = einit
            er(k)     = erinit
            tp(k)     = tpinit
```

```
551                pr(k)     = pinit + otd * erinit
552     56   continue
553     c
554     c**********************************************************************
555     c                                                                     *
556     c          initial energy ball profile                                *
557     c                                                                     *
558     c**********************************************************************
559     c
560           call epinit
561     c
562      58      continue
563     c
564     c         set properties for He region -- constant
565     c
566               do 57 k=kptc+2,kptd
567                  xkap(k)   = xkap2
568                  rosmfp(k) = 1.e-6
569                  rmfp1t(k) = 1.e-6
570                  rmfp2t(k) = 1.e-6
571                  rcsubv(k) = rcsub2
572                  gam1(k)   = gaminit
573      57      continue
574     c**********************************************************************
575     c
576     c         unscrunch icntrl
577     c
578     c**********************************************************************
579           ndepb = 0
580           do 30 i=1,8
581              itemp = 0.1 * icntrl
582              itemp = icntrl - 10 * itemp
583              icntrl = 0.1 * icntrl
584              if (itemp.eq.0) go to 30
585                 ndepb = ndepb + 1
586                 mask(ndepb) = i
587      30   continue
588     c**********************************************************************
589     c
590     c         initialize dump files
591     c
592     c**********************************************************************
593     c
594               ndmpb  = 0
595               ndmpcd = 0
596               nrst   = 0
597           write(idmpa,810) header
598      810  format(a80)
599     c
600           write(idmpb,810) header
601           write(idmpb,811) ndepb
602      811  format(1x,i2)
603           write(idmpb,812) ibar,jbar,(depen(mask(i)),i=1,ndepb)
604      812  format(1x,'time (sec)',i3,i3,8a3)
605           do 813 k=2,im1
```

```
606            write(idmpb,814) xc(k)
607     814    format(1x,e12.5)
608     813  continue
609          do 816 k=kpta,kpte,imx
610            write(idmpb,814) yc(k)
611     816  continue
612     c
613          write(idmpc,810) header
614          write(idmpc,811) ndepb
615          write(idmpc,815) (depen(mask(i)),i=1,ndepb)
616     815  format(' time (sec)         ',' cell centre (cm)    ',8a3)
617     c
618          write(idmpd,810) header
619          write(idmpd,811) ndepb
620          write(idmpd,815) (depen(mask(i)),i=1,ndepb)
621     c
622     c**************************************************************************
623     c**************************************************************************
624     c                                                                         *
625     c        call subroutine solaice for each time step                       *
626     c             note: time step 0 skips hydro and just sets up              *
627     c                   beta array and properties                             *
628     c                                                                         *
629     c        add time splitting lagrangian energy calculation e2t here        *
630     c        note:  jump around this subroutine for cycle 0                   *
631     c                                                                         *
632     c        computational loop begins now                                    *
633     c                                                                         *
634     c**************************************************************************
635     c**************************************************************************
636     c
637      100  continue
638          if (cycle.eq.0) go to 111
639     c                                                                    *
640     c        2T energy equation solver section                           *
641     c         solve for e-tilde, p-tilde, and er-tilde                   *
642     c        note---split time step solution technique used for speed    *
643     c               perform lagrangian radiation energy transport and    *
644     c               exchange only using time step dtrad                  *
645     c               grad (velocity) assumed constant for this loop       *
646     c                                                                    *
647     c         sub eosset iterates on Tp to satisfy Ep from EOS tables
648     c
649     c
650          ja = 1
651          jb = kptc + 1
652          call eosset (ja,jb)
653          call e2t
654     c
655     c
656     c        set time n vectors
657     c
658      111  continue
659          do 650 k=1,kptd
660            vn(k)   = v(k)
```

```
661            un(k)   = u(k)
662            rhon(k) = rho(k)
663            pr(k)   = p(k) + otd * er(k) * con(16)
664     650    continue
665     c
666     c         update He thermal conductivity (Spitzer Conductivity)
667     c         k = J/cm-s-ev     t--ev
668     c
669           do 651 k=kptc+2,kptd
670              xkap(k) = 274. * tp(k)**2.5
671     651    continue
672     c
673     c
674           call solaice
675     c
676     c
677     c*****************************************************************
678     c
679     c         check for output options
680     c
681     c         check for dmp file a output
682     c
683     110    if (t.lt.tdmpa.and.(t+delt).lt.tend) go to 799
684              tdmpa = tdmpa + cndmpa*delt
685     c
686     c         calculate grid ke, mass, and internal energy
687     c
688              ke     = 0.
689              energy = 0.
690              xmass  = 0.
691              do 701 k=kpta,kptb
692                 vol    = (1. - cyl + 2.*pi*xc(k)*cyl) * dx(k) * dy(k)
693                 vol    = vol * xmask(k)
694                 uave   = u(k-1) + u(k)
695                 vave   = v(k)   + v(k-imx)
696                 ke     = ke + rho(k) * vol * (uave*uave + vave*vave)
697                 xmass  = xmass + rho(k) * vol
698                 energy = energy + e(k) * vol * rho(k)
699     701    continue
700              ke = 0.125 * ke * con(17)
701              write(idmpa,800) t,iter,ke,xmass,energy
702     800      format(2x,e12.5,2x,i3,'.',2x,3(e12.5,2x))
703     c
704     c         check restart file control
705     c
706     799    if (t.lt.trst.and.(t+delt).lt.tend) go to 580
707              trst = trst + cnrst*delt
708              write(iwrst,880) t
709     880      format('1      time = ',e12.5)
710              write(iwrst,801)
711     801      format('    k',10x,'u',12x,'v',12x,'p',11x,'rho',11x,
712         1             'e',11x,'Tp  ',11x,'Er')
713              do 575 k=1,kptd
714                 write(iwrst,802)k,u(k),v(k),p(k),rho(k),e(k),tp(k),er(k)
715     575      continue
```

```
716     802     format(1x,i5,3x,7(1x,e12.5))
717             nrst = nrst + 1
718             write(idmpa,777) nrst,t
719     777     format(23x,'wrst  file --  ',i3,'  time = ',e12.5)
720     c
721     c       check dump file b control
722     c
723     580   if (t.lt.tdmpb.and.(t+delt).lt.tend) go to 590
724             tdmpb = tdmpb + cndmpb*delt
725             write(idmpb,804) t,ibar,jbar
726     804     format(1x,e12.5,2x,i3,2x,i3)
727             do 585 k=1,kptd
728                value(1) = u(k)
729                value(2) = v(k)
730                value(3) = pr(k) * con(17)
731                value(4) = rho(k)
732                value(5) = e(k)
733                value(6) = tp(k)
734                value(7) = (con(5)*er(k))**0.25
735                value(8) = gam1(k)
736                write(idmpb,805)(value(mask(kk)),kk=1,ndepb)
737     585     continue
738     805     format(1x,9(e12.5,1x))
739             ndmpb = ndmpb + 1
740             write(idmpa,778) ndmpb,t
741     778     format(23x,'dmpb  file --  ',i3,'  time = ',e12.5)
742     590   continue
743     c
744     c       check for dump file c/d control
745     c
746           if (t.lt.tdmpcd.and.(t+delt).lt.tend) go to 599
747             tdmpcd = tdmpcd + dmpcd*delt
748             write(idmpc,804) t,imx
749             do 592 k=ileft,iright
750                value(1) = u(k)
751                value(2) = v(k)
752                value(3) = pr(k) * con(17)
753                value(4) = rho(k)
754                value(5) = e(k)
755                value(6) = tp(k)
756                value(7) = (con(5)*er(k))**0.25
757                value(8) = gam1(k)
758                write(idmpc,805) xc(k),(value(mask(kk)),kk=1,ndepb)
759     592     continue
760             write(idmpd,804) t,jmx
761             do 593 k=2,kptd,imx
762                value(1) = u(k)
763                value(2) = v(k)
764                value(3) = pr(k) * con(17)
765                value(4) = rho(k)
766                value(5) = e(k)
767                value(6) = tp(k)
768                value(7) = (con(5)*er(k))**0.25
769                value(8) = gam1(k)
770                write(idmpd,805) yc(k),(value(mask(kk)),kk=1,ndepb)
```

```
 593     continue
         ndmpcd = ndmpcd + 1
         write(idmpa,779) ndmpcd,t
 779     format(23x,'dmpcd file --  ',i3,'  time = ',e12.5)
 599  continue
c
c        advance time  t = t + delt
c
      t = t + delt
      if (t.gt.tend) go to 999
         call timeleft (isec)
         if (isec.lt.40) then
            tend = 0.
            go to 110
         endif
         cycle = cycle + 1
         go to 100
 990  continue
c
c*************************************************************
c
      write(iwrst,991)
 991  format('  input error')
 999  continue
c
c        housekeeping and file closing
c
      t = t - delt
      write(idmpa,776) t,cycle
 776  format('   last dump time = ',e12.5,'  cycle = ',i5)
      call timeused (icpu,iio,isys,imem)
      tcpu = icpu* 1.e-6
      tio  = iio * 1.e-6
      tsys = isys* 1.e-6
      tmem = imem*1.e-12
      write(idmpa,775)tcpu,tio,tsys,tmem
 775  format(2x,'cpu(s) = ',e12.5,'  i/o(s) = ',e12.5,' sys(s) = ',
     a          e12.5,'  mem(s) = ',e12.5)
c
      close (1)
      close (3)
      close (6)
      close (7)
      close (8)
c
c        save dropfile for possible restart
c
      call exit (1)
c
      end
```

```
      SUBROUTINE EOS (j1,j2)
C
c     eos calculates equation of state quantities by table look-up
c     output---
c              xkap --- thermal conductivity
c              gam1 --- (gamma-1) for ideal gas eos
c              rcsubv - recipical of Cv
c              rmfp --- radiation mfp
c              ross2t - rosseland mfp for 2 temperatures
c              rosmfp - rosseland mfp for 1 temperature
c     input--
c        ja, jb  limits on input array
c
c      this eos package is constructed around those in MF-FIRE  FDM-458
c
c      note:  an effort was made to conserve arrays here
c             several arrays perform double duty here
c             en, rhon, un, and vn are fair game here
c
c      con array:
c               5 --- c/(4.*sigma)
c               6 --- number density / mass density
c               9 --- pressure conversion constant (from MF-FIRE)
c              11 --- mol wt / rgas  (constant)
c              22 --- eos floor value energy density (j/cm3)
c
       common/eos1/   ad,  bd,  at,  bt, nmat, nfg, ja, jb,
     1             rad, rdb, rat, rbt
       common/eostab/ ztab(20,10,2),      entab(20,10,2),
     1              rostab(20,20,10,2), rmftab(20,20,10,2)
       common/wrk   / work(9999,12)
       common/eoswrk/  trl(9999),  tpl(9999), rhol(9999),
     1              meos(9999), leos(9999), keos(9999)
       common /np1/   u(9999)   , v(9999)   , rho(9999) , e(9999)   ,
     1              er(9999)  , p(9999)   , tp(9999)
       common /prop/  gam1(9999), rosmfp(9999), rmfp1t(9999),
     1            rmfp2t(9999), rcsubv(9999),   xkap(9999)
       common /a/    beta(9999)  , xmask(9999), con(30)
       common /e/  cycle, wl, wr, wt, wb, iter, imx, jmx, delt, xmu,
     1            im1, jm1, im2, jm2, iterlmt, cyl, epsi, gaminit,
     2            gx, gy, ui, vi, cnrst, cndmp, omega, alpha, t, am1,
     3            gamma, asq, rhoi, eamb, dtrad, otd, ftd, xkap2,
     4            kpta, kptb, kptc, kptd, kpte, kptf, eamb2, rcsubin,
     5            bvar1, ivar2, bvar3, bvar4, bvar5, bvar6, bvar7,bvar8
C
       dimension z(9999), ee(9999), dummy(9999), dedt(9999)
       dimension xflg(9999)
C
       equivalence (ee(1), work(1,11))  ,   (z(1), work(1,12)),
     1            (dummy(1),work(1,10)), (dedt(1), work(1,9))
C
       ja = j1
       jb = j2
C
C  COMPUTE THE LOGARITHMS OF THE TEMPERATURES AND DENSITY FOR USE IN
```

```
C  THE TABLE LOOK-UP ROUTINES
c
      DO 10 J = ja,jb
         erad    = er(j)
         erad    = cvmgp ( erad  , con(22) , (erad-con(22)) )
         tfluid  = tp(j)
         trl(j)  = 0.25 * alog10( con(5)*erad )
         tpl(j)  = alog10( tfluid )
         rhol(j) = alog10( rho(j)*con(6) )
 10   continue
C
C  FIRST FIND THE (K,L) POINTS THAT CORRESPOND TO THE TEMPERATURE AND
C  DENSITY IN EACH ZONE
c
      CALL POINT
C
C  NOW FIND THE VALUES IN THE CHARGE STATE TABLE
c
      CALL TABLE2 (ZTAB,Z,dummy,tp)
C
C  NEXT FIND THE VALUES IN THE SPECIFIC ENERGY TABLE
c
      CALL TABLE2 (ENTAB,EE,dedt,tp)
c
c        calulate xkap (from subroutine kappa)
c                 gam1 and rcubv from Z and EE
c
      call kappa (Z, ja, jb)
c
c       note: p = gam1 * rho * e = c9 * c6 * rho * tp * (1 + z)
c       check for values below table (leos=0) use initial values then
c
      do 40 j = ja,jb
         xgam1     = con(9)*con(6) * tp(j) * (1.+z(j)) / e(j)
ceos     gam1(j)   = cvmgn ( xgam1, gaminit, float(leos(j)) )
ceos     rcsubv(j) = cvmgn ( 1./dedt(j), rcsubin, float(leos(j)) )
         gam1(j) = xgam1
         rcsubv(j) = 1./dedt(j)
 40   continue
c
C  FIND THE VALUES IN THE RADIATION MFP TABLE
c
      CALL TABLE3 (RMFTAB,RMFP2T)
C
C  FIND THE VALUES IN THE ROSSELAND MFP TABLE
c
      CALL TABLE3 (ROSTAB,ROSMFP)
C
C  FIND THE VALUES WHEN TR = TP
c
      do 60 j= ja,jb
         KEOS(J) = LEOS(J)
         trl(j)  = tpl(j)
   60 CONTINUE
c
```

```
      CALL TABLE3 (rmftab,rmfp1t)
c
      return
      END
```

```
      SUBROUTINE EOSSET (j1,j2)
C
c     eosset iterates on tp to satisfy e with the EOS tables
c
c     input--
c        ja, jb  - limits on input array
c        con(21) - relative error on Tp iteration
c
c
      common/eos1/   ad,  bd,  at,  bt, nmat, nfg, ja, jb,
     1              rad, rdb, rat, rbt
      common/eostab/ ztab(20,10,2),      entab(20,10,2),
     1               rostab(20,20,10,2), rmftab(20,20,10,2)
      common/wrk   / work(9999,12)
      common/eoswrk/  trl(9999),  tpl(9999), rhol(9999),
     1               meos(9999), leos(9999), keos(9999)
      common /np1/   u(9999)   , v(9999)   , rho(9999) , e(9999)   ,
     1               er(9999)  , p(9999)   , tp(9999)
      common /prop/  gam1(9999), rosmfp(9999), rmfp1t(9999),
     1             rmfp2t(9999), rcsubv(9999),   xkap(9999)
      common /a/    beta(9999)  , xmask(9999), con(30)
      common /e/  cycle, wl, wr, wt, wb, iter, imx, jmx, delt, xmu,
     1            im1, jm1, im2, jm2, iterlmt, cyl, epsi, gaminit,
     2            gx, gy, ui, vi, cnrst, cndmp, omega, alpha, t, am1,
     3            gamma, asq, rhoi, eamb, dtrad, otd, ftd, xkap2,
     4            kpta, kptb, kptc, kptd, kpte, kptf, eamb2, rcsubin,
     5            bvar1, ivar2, bvar3, bvar4, bvar5, bvar6, bvar7,bvar8
c
      dimension z(9999), ee(9999), dummy(9999), dedt(9999)
      dimension xflg(9999)
      data limit / 25/
c
      equivalence (ee(1), work(1,11))  ,   (z(1), work(1,12)),
     1            (dummy(1),work(1,10)), (dedt(1), work(1,9))
c
      ja = j1
      jb = j2
      nloop = 0
C
C  COMPUTE THE LOGARITHMS OF THE TEMPERATURES AND DENSITY FOR USE IN
C  THE TABLE LOOK-UP ROUTINES
c
      DO 10 J = ja,jb
         erad    = er(j)
         erad    = cvmgp ( erad  , con(22) , (erad-con(22)) )
9tfluid  = cvmgp (tp(j), 0.25, (tp(j) - 0.25))
         trl(j)  = 0.25 * alog10( con(5)*erad )
         tpl(j)  = alog10( tfluid )
         rhol(j) = alog10( rho(j)*con(6) )
 10   continue
C
C  FIRST FIND THE (K,L) POINTS THAT CORRESPOND TO THE TEMPERATURE AND
C  DENSITY IN EACH ZONE
c
 50   continue
```

```
      CALL POINT
C
C  NEXT FIND THE VALUES IN THE SPECIFIC ENERGY TABLE
c
      CALL TABLE2 (ENTAB,EE,dedt,tp)
c
c        now iterate on Tp until the convergence criterian is achieved
c
      xflag = 1.0
      nloop = nloop + 1
      do 20 j = ja,jb
         tnew  = tp(j) + (e(j) - ee(j)) / dedt(j)
         relt  = abs (tnew-tp(j)) / tp(j)
         xflag = xflag * cvmgp ( 0.0, 1.0, (relt-con(21)) )
9tp(j) = cvmgp (tnew, 0.25, (tnew - 0.25))
         tpl(j)= alog10 (tp(j))
 20   continue
c
      if (xflag.gt.0.01) go to 999
      if (nloop.lt.limit) go to 50
c
 999  continue
      return
      END
```

```
      SUBROUTINE EOSINIT (j1,j2)
C
c     eosinit initializes energy density and pressure for epinit
c
c     input--
c        ja, jb  - limits on input array
c
c
      common/eos1/   ad,  bd,  at,  bt, nmat, nfg, ja, jb,
     1              rad, rdb, rat, rbt
      common/eostab/ ztab(20,10,2),      entab(20,10,2),
     1               rostab(20,20,10,2), rmftab(20,20,10,2)
      common/wrk   / work(9999,12)
      common/eoswrk/  trl(9999),  tpl(9999), rhol(9999),
     1               meos(9999), leos(9999), keos(9999)
      common /np1/   u(9999)   , v(9999)   , rho(9999) , e(9999)   ,
     1               er(9999)  , p(9999)   , tp(9999)
      common /prop/  gam1(9999), rosmfp(9999), rmfp1t(9999),
     1             rmfp2t(9999), rcsubv(9999),   xkap(9999)
      common /a/    beta(9999)  , xmask(9999), con(30)
      common /e/  cycle, wl, wr, wt, wb, iter, imx, jmx, delt, xmu,
     1            im1, jm1, im2, jm2, iterlmt, cyl, epsi, gaminit,
     2            gx, gy, ui, vi, cnrst, cndmp, omega, alpha, t, am1,
     3            gamma, asq, rhoi, eamb, dtrad, otd, ftd, xkap2,
     4            kpta, kptb, kptc, kptd, kpte, kptf, eamb2, rcsubin,
     5            bvar1, ivar2, bvar3, bvar4, bvar5, bvar6, bvar7,bvar8
c
      dimension z(9999), ee(9999), dummy(9999), dedt(9999)
      dimension xflg(9999)
c
      equivalence (ee(1), work(1,11))  ,   (z(1), work(1,12)),
     1            (dummy(1),work(1,10)), (dedt(1), work(1,9))
c
      ja = j1
      jb = j2
C
C  COMPUTE THE LOGARITHMS OF THE TEMPERATURES AND DENSITY FOR USE IN
C  THE TABLE LOOK-UP ROUTINES
c
      DO 10 J = ja,jb
         erad    = er(j)
         erad    = cvmgp ( erad  , con(22) , (erad-con(22)) )
         tfluid  = tp(j)
         trl(j)  = 0.25 * alog10( con(5)*erad )
         tpl(j)  = alog10( tfluid )
         rhol(j) = alog10( rho(j)*con(6) )
 10   continue
C
C  FIRST FIND THE (K,L) POINTS THAT CORRESPOND TO THE TEMPERATURE AND
C  DENSITY IN EACH ZONE
c
      CALL POINT
C
C  NEXT FIND THE VALUES IN THE SPECIFIC ENERGY TABLE
c
```

```
1069          CALL TABLE2 (ENTAB,EE,dedt,tp)
1070     c
1071          call table2 (ztab,z,dummy,tp)
1072     c
1073          do 40 j = ja,jb
1074             xgam1     = con(9)*con(6) * tp(j) * (1.+z(j)) / ee(j)
1075     ceos    gam1(j)   = cvmgn (xgam1, gaminit, float(leos(j)) )
1076     ceos    rcsubv(j) = cvmgn (1./dedt(j), rcsubin, float(leos(j)) )
1077             gam1(j) = xgam1
1078             rcsubv(j) = 1./dedt(j)
1079             e(j)      = ee(j)
1080      40   continue
1081          return
1082          END
```

```
      subroutine epinit
c
c        this subroutine determines the initial plasma specific energy
c        density (En) and initial radiation density (Er) for a fireball.
c        It uses a linear fit for the initial plasma temperature.
c
c
      integer cycle, wl, wr, wt, wb
      common /mesh/ xc(9999)  , yc(9999)  , dx(9999)   , dy(9999)  ,
     1              rdx(9999) , rdy(9999) ,
     2              rdxc(9999), rdyc(9999), rxc(9999),   r2dx(9999),
     3              r2dy(9999)
      common /wrk/  un(9999)  , vn(9999)  , rhon(9999), en(9999)  ,
     1              pr(9999)  , work(9999,7)
      common /np1/  u(9999)   , v(9999)   , rho(9999) , e(9999)   ,
     1              er(9999)  , p(9999)   , tp(9999)
      common /prop/  gam1(9999), rosmfp(9999), rmfp1t(9999),
     1             rmfp2t(9999), rcsubv(9999),   xkap(9999)
      common /a/    beta(9999)  , xmask(9999), con(30)
      common /e/  cycle, wl, wr, wt, wb, iter, imx, jmx, delt, xmu,
     1            im1, jm1, im2, jm2, iterlmt, cyl, epsi, gaminit,
     2            gx, gy, ui, vi, cnrst, cndmp, omega, alpha, t, am1,
     3            gamma, asq, rhoi, eamb, dtrad, otd, ftd, xkap2,
     4            kpta, kptb, kptc, kptd, kpte, kptf, eamb2, rcsubin,
     5            bvar1, ivar2, bvar3, bvar4, bvar5, bvar6, bvar7, bvar8
      dimension xr(16),tr(16)
      data xr /1.0, 2.5, 6.12, 8.0, 11.5, 12.3, 18.3, 21., 25.4, 30.0,
     1        32.6,41.5,60.0 ,114., 189., 250. /
6data tr / 1226., 853., 450., 310., 235., 220., 200., 195., 190.,
     1          160., 110., 30., 4.7, 1.0, 0.5, 0.37 /
c
c        find limits first
c
      kcenter= ivar2*imx + 2
      ycent  = yc(kcenter)
      xcent  = xc(kcenter) - 0.5 * dx(kcenter)
      k1     = 1
      k2     = kptc + 1
      ntab   = 16
c
c         profile initilization
c         set Tr profile using linear fits
c         determine Ep from EOS tables
c
c
      do 100 k=k1,k2
         d    = sqrt( (xcent-xc(k))**2 + (ycent-yc(k))**2 )
         do 90 i=2,ntab
            if (xr(i).lt.d) go to 90
               tp(k) = tr(i) + (tr(i-1)-tr(i))*(d-xr(i))/(xr(i-1)-xr(i))
?go to 91
 90      continue
 917continue
         if (tp(k).lt.con(25).or.d.ge.xr(ntab)) tp(k) = con(25)
 100  continue
```

```
c
         call eosinit (k1,k2)
c
c        set pressure profile with eos properties
c
      do 110 k=k1,k2
         p(k)  = gam1(k) * rho(k) * e(k) * con(16)
         en(k) = e(k)
         pr(k) = p(k) + otd*er(k)*con(16)
 110  continue
c
      return
      end
6subroutine e2t
c
c*********************************************************************
c                                                                    *
c        2T energy equation solver section                           *
c        use implicit technique to solve for e-tilde and er-tilde    *
c        also calculate p-tilde                                      *
c         note---use centered differencing on Er diffusion term      *
c                                                                    *
c        special update---use split time step to solve lagrangian    *
c        portion only in this subroutine:                            *
c                diffusion + radiation exhange + pdv work term       *
c                                                                    *
c        new update---incorporate quickie nine pt diffusion stencil
c                    assume uniform mesh  weight controled by        *
c                    variable con(20)                                *
c                                                                    *
c        set properties to n+1 with 2 loop iteration                 *
c                                                                    *
c*********************************************************************
c
      integer cycle, wl, wr, wt, wb
      common /mesh/ xc(9999)  , yc(9999)  , dx(9999)   , dy(9999)  ,
     1              rdx(9999) , rdy(9999) ,
     2              rdxc(9999), rdyc(9999), rxc(9999),   r2dx(9999),
     3              r2dy(9999)
      common /wrk/  un(9999)  , vn(9999)  , rhon(9999), en(9999)  ,
     1              pr(9999)  , work(9999,7)
      common /np1/  u(9999)   , v(9999)   , rho(9999) , e(9999)   ,
     1              er(9999)  , p(9999)   , tp(9999)
      common /prop/  gam1(9999), rosmfp(9999), rmfp1t(9999),
     1             rmfp2t(9999), rcsubv(9999),   xkap(9999)
      common /a/    beta(9999)  , xmask(9999), con(30)
      common /e/  cycle, wl, wr, wt, wb, iter, imx, jmx, delt, xmu,
     1            im1, jm1, im2, jm2, iterlmt, cyl, epsi, gaminit,
     2            gx, gy, ui, vi, cnrst, cndmp, omega, alpha, t, am1,
     3            gamma, asq, rhoi, eamb, dtrad, otd, ftd, xkap2,
     4            kpta, kptb, kptc, kptd, kpte, kptf, eamb2, rcsubin,
     5            bvar1, ivar2, bvar3, bvar4, bvar5, bvar6, bvar7,bvar8
c
      dimension   fluxx(9999), fluxy(9999), fluxxy(9999), fluxyx(9999),
     1            etrans(9999), gradv(9999), ern(9999)
```

```
1193    c
1194          equivalence (fluxx(1) ,work(1,1)),   (fluxy(1) ,work(1,2)),
1195         1            (fluxxy(1),work(1,3)),   (fluxyx(1),work(1,4)),
1196         2            (etrans(1),work(1,5)),
1197         3            (ern(1)   ,work(1,7))
1198          equivalence (tt, con(20)), (delxy, con(19)),  (rdelxy, con(10)),
1199         1            (rerror, con(23)),(halfdx, con(7)),(dtemp,con(24))
1200          data loopimt / 10/
1201    c
1202    c          first calculate number of lagrangian loops to perform
1203    c
1204          nloop = delt / dtrad
1205          j1    = 1
1206          j2    = kptc + 1
1207          xtt   = 1. - tt
1208    c
1209    c          now calculate velocity gradient -- constant for all loops
1210    c
1211          do 5 k=kpta,kptd
1212             gradv(k)  =( rdx(k) * ( un(k) - un(k-1)   ) +
1213         1                rdy(k) * ( vn(k) - vn(k-imx) ) +
1214         2                cyl    * ( un(k) + un(k-1)   ) * 0.5*rxc(k) )
1215     5    continue
1216    c*************************************************************************
1217    c
1218    c          the lagranian loop begins now
1219    c
1220    c*************************************************************************
1221    c
1222          do 100 loop=1,nloop
1223    c
1224    c
1225    c        call eos routine for opacities, conductivity, gam1
1226    c        determine tp from constant volume process
1227          call eos (j1, j2)
1228    c
1229    c        set n level plasma and radiation energy densities
1230    c       also put fix in for RMFP2T
1231    c
1232          do 51 k=kpta,kptd
1233             en(k) = e(k)
1234             ern(k)= er(k)
1235             trad  = (con(5)*er(k))**0.25
1236             temp  = abs(trad-tp(k))/tp(k)
1237             rmfp2t(k) = cvmgp( rmfp2t(k), rmfp1t(k), (temp-dtemp) )
1238    9rmfp2t(k) =cvmgp(rmfp2t(k),rmfp1t(k),(tp(k)-con(30)))
1239     51   continue
1240    c
1241    c        point-jacobi iteration loop
1242    c
1243     349  iter = 0
1244     350  flag = 1.
1245    c
1246    c        set lefthand b.c.  (zero gradient)
1247    c
```

```
1248    cdir$ ivdep
1249          do 52 k=1,kptd,imx
1250             er(k) = er(k+1)
1251             tp(k) = tp(k+1)
1252     52   continue
1253    c
1254    c        set top and bottom bc
1255    c
1256    cdir$ ivdep
1257          do 53 k=1,imx
1258             er(k)      = er(k+imx)
1259             er(k+kpte) = er(k+kpte-imx)
1260             tp(k)      = tp(k+imx)
1261             tp(k+kpte) = tp(k+kpte-imx)
1262     53   continue
1263    c
1264    c          first determine misc intermediate quantities for Er
1265    c
1266          do 10 k=kpta,kptd
1267             delxeb   = er(k-1)    - er(k)
1268    cupwind  ebxstar  = cvmgp( er(k-1),   er(k), delxeb)
1269             ebxstar  = 0.5 * ( er(k-1)     + er(k) )
1270             fluxx(k) = con(2) *ebxstar * delxeb /
1271         1               (.375*ebxstar*( rho(k-1)*rosmfp(k-1)*dx(k-1) +
1272         2                               rho(k)  *rosmfp(k)  *dx(k) ) +
1273         3                abs(delxeb) + bvar5 )
1274     10   continue
1275          do 11 k=kpta,kptd
1276             delyeb   = er(k-imx)  - er(k)
1277             ebystar  = 0.5 * ( er(k-imx)   + er(k) )
1278             fluxy(k) =  con(2)*ebystar * delyeb /
1279         1              (.375*ebystar*(rho(k-imx)*rosmfp(k-imx)*dy(k-imx)+
1280         2                             rho(k)    *rosmfp(k)    *dy(k)   )+
1281         3                abs(delyeb) + bvar5 )
1282     11   continue
1283          do 12 k=kpta,kptd
1284             delxye   = er(k-imx-1)- er(k)
1285             ebxystr  = 0.5 * ( er(k-imx-1) + er(k) )
1286             fluxxy(k)= con(2) * ebxystr * delxye /
1287         1              (.375*ebxystr*delxy*(rho(k-imx-1)*rosmfp(k-imx-1)+
1288         2                                   rho(k)      *rosmfp(k)      )+
1289         3               abs(delxye) + bvar5 )
1290     12   continue
1291          do 13 k=kpta,kptd
1292             delyxe   = er(k-imx+1)- er(k)
1293             ebyxstr  = 0.5 * ( er(k-imx+1) + er(k) )
1294             fluxyx(k)= con(2) * ebyxstr * delyxe /
1295         1              (.375*ebyxstr*delxy*(rho(k-imx+1)*rosmfp(k-imx+1)+
1296         2                                   rho(k)      *rosmfp(k)      )+
1297         3               abs(delyxe) + bvar5 )
1298     13   continue
1299    c
1300    c          calculate energy transfer by emission and absorption
1301    c          ( assume positive for Er )
1302    c
```

```
1303          do 15 k=kpta,kptd
1304             etrans(k)= rho(k) * ( con(3)*rmfp1t(k)*tp(k)**4 -
1305         1                          con(4)*rmfp2t(k)*er(k) )
1306      15   continue
1307     c
1308     c          now for the solution of er-tilde
1309     c
1310          do 25 k=kpta,kptb
1311             r1       = xc(k) - halfdx
1312             r2       = xc(k) + halfdx
1313             r3       = xc(k)
1314     c
1315             ebx      = rdx(k)*( fluxx(k) - fluxx(k+1)   )
1316             eby      = rdy(k)*( fluxy(k) - fluxy(k+imx) )
1317             ebc      = 0.5*cyl*rxc(k)* (fluxx(k) + fluxx(k+1))
1318             ebxy     = rdelxy*(fluxxy(k)*r1-fluxxy(k+imx+1)*r2)/r3
1319             ebyx     = rdelxy*(fluxyx(k)*r2-fluxyx(k+imx-1)*r1)/r3
1320             erwork   = -otd * er(k) * gradv(k)
1321             ediff    = tt * (ebx+eby-ebc) + xtt * (ebxy+ebyx)
1322     c
1323             deler    = dtrad * xmask(k) * ( ediff + etrans(k) + erwork )
1324             ernp1    = ern(k) + deler
1325             er(k)    = ernp1
1326             er(k)    = cvmgp ( er(k), con(26), (er(k)-con(26)) )
1327      25   continue
1328     c
1329     c          determine intermediate quantities for E
1330     c
1331          do 20 k=kpta,kptd
1332             fluxx(k)= 2.* (tp(k-1)-tp(k)) * (xkap(k-1)*xkap(k)) /
1333         1                ( dx(k-1)*xkap(k) + dx(k)*xkap(k-1) )
1334      20   continue
1335          do 21 k=kpta,kptd
1336             fluxy(k)= 2.* (tp(k-imx) -tp(k)) * (xkap(k-imx)*xkap(k)) /
1337         1                ( dy(k-imx)*xkap(k) + dy(k)*xkap(k-imx) )
1338      21   continue
1339          do 22 k=kpta,kptd
1340             fluxxy(k)=2. * rdelxy *
1341         1             (tp(k-imx-1) - tp(k)) * (xkap(k-imx-1)*xkap(k))/
1342         2                ( xkap(k-imx-1) + xkap(k) )
1343      22   continue
1344          do 23 k=kpta,kptb
1345             fluxyx(k)=2. * rdelxy *
1346         1             (tp(k-imx+1) - tp(k)) * (xkap(k-imx+1)*xkap(k))/
1347         2                ( xkap(k-imx+1) + xkap(k) )
1348      23      continue
1349     c
1350     c          the solution for e-tilde
1351     c
1352          do 30 k=kpta,kptb
1353             r1       = xc(k) - halfdx
1354             r2       = xc(k) + halfdx
1355             r3       = xc(k)
1356     c
1357             ex       = rdx(k)*( fluxx(k) - fluxx(k+1)   )
```

```
1358            ey      = rdy(k)*( fluxy(k) - fluxy(k+imx) )
1359            ec      = 0.5*cyl*rxc(k)* (fluxx(k) + fluxx(k+1))
1360            exy     = rdelxy*(fluxxy(k)*r1-fluxxy(k+imx+1)*r2)/r3
1361            eyx     = rdelxy*(fluxyx(k)*r2-fluxyx(k+imx-1)*r1 )/r3
1362            epwork  = -p(k) * con(17) * gradv(k)
1363            ediff   = tt * (ex+ey-ec) + xtt * (exy+eyx)
1364    c
1365            dele    = dtrad * xmask(k) * (ediff -etrans(k) +epwork) /rho(k)
1366            enp1    = en(k) + dele
1367            enp1    = cvmgp( enp1, con(27), (enp1-con(27)))
1368            temp    = abs(enp1-e(k)) - e(k) * rerror
1369            flag    = flag * cvmgp ( 0.0, 1.0, temp )
1370    c
1371            p(k)    = gam1(k) * rho(k) * enp1 * con(16)
1372            tp(k)   = tp(k) + rcsubv(k) * (enp1 -e(k))
1373            e(k)    = enp1
1374     30   continue
1375    c
1376          iter = iter + 1
1377          if (flag.gt.0.01) go to 360
1378          if (iter.lt.loopimt) go to 350
1379             write(1,700) cycle,loop
1380     700     format('  iteration e2t exceeded ',i6,2x,i4)
1381    ctim     close (1)
1382    ctim     close (3)
1383    ctim     close (6)
1384    ctim     close (7)
1385    ctim     close (8)
1386    ctimn    stop 1
1387     360  continue
1388    c
1389    c        update time value of e for later plasma temp calculations
1390    c
1391          do 40 k=kpta,kptb
1392             en(k) = e(k)
1393     40   continue
1394    c
1395     100  continue
1396          return
1397          end
```

```
1398          SUBROUTINE INIT2 (ireos)
1399     C
1400     C  INIT2 reads in opacity tables
1401     c         taken from MF-FIRE   ref. UWFDM-458
1402     C
1403          common/eos1/   ad,  bd,  at,  bt, nmat, nfg, ja, jb,
1404         1              rad, rdb, rat, rbt
1405          common/eostab/ ztab(20,10,2),      entab(20,10,2),
1406         1               rostab(20,20,10,2), rmftab(20,20,10,2)
1407          character*40 headr1, headr2
1408          dimension hnu1(1),hnu11(1),hnu12(1),rrtab(20,20,10,2),
1409         1                                    rptab(20,20,10,2)
1410          equivalence (rostab(1,1,1,1), rrtab(1,1,1,1))
1411          equivalence (rmftab(1,1,1,1), rptab(1,1,1,1))
1412     C
1413          nmat = 1
1414          nfg  = 0
1415     C
1416     C  READ IN EOS DATA FROM ireos FOR MATERIAL 1 &
1417     c       FROM ireos2 FOR MATERIAL 2
1418          READ(ireos,1002) HEADR1
1419          READ(ireos,1001) AD1, BD1, AT1, BT1, NFG1
1420          IF (NMAT .EQ. 2) then
1421             read(ireos2,1002) headr2
1422             READ(ireos2,1001) AD2, BD2, AT2, BT2, NFG2
1423          endif
1424      5   AD = AD1
1425          BD = BD1
1426          AT = AT1
1427          BT = BT1
1428          rad= 1./ad
1429          rbd= 1./bd
1430          rat= 1./at
1431          rbt= 1./bt
1432     c
1433          READ(ireos,1000) ((ZTAB(L,M,1), L=1,20), M=1,10)
1434          READ(ireos,1000) ((ENTAB(L,M,1), L=1,20), M=1,10)
1435          READ(ireos,1000) (((ROSTAB(K,L,M,1), K=1,20), L=1,20), M=1,10)
1436          READ(ireos,1000) (((RMFTAB(K,L,M,1), K=1,20), L=1,20), M=1,10)
1437          IF (NMAT .EQ. 1) GOTO 15
1438             rEAD(ireos2,1000) ((ZTAB(L,M,2), L=1,20), M=1,10)
1439             READ(ireos2,1000) ((ENTAB(L,M,2), L=1,20), M=1,10)
1440             READ(ireos2,1000) (((ROSTAB(K,L,M,2), K=1,20), L=1,20), M=1,10)
1441             READ(ireos2,1000) (((RMFTAB(K,L,M,2), K=1,20), L=1,20), M=1,10)
1442      15   CONTINUE
1443      1000 FORMAT(4(1x,E12.6))
1444      1001 FORMAT(4(1x,E12.6),I12)
1445      1002 FORMAT(40A1)
1446        20 DO 50 M = 1,10
1447             DO 50 L = 1,20
1448                DO 50 KMAT = 1, nmat
1449                   IF(ZTAB(L,M,KMAT) .NE. 0.0) ZTAB(L,M,KMAT) =
1450         1                                     ALOG10(ZTAB(L,M,KMAT))
1451                   IF(ENTAB(L,M,KMAT) .NE. 0.0) ENTAB(L,M,KMAT) =
1452         1                                     ALOG10(ENTAB(L,M,KMAT))
```

```
                  DO 50 K = 1,20
                     IF(RMFTAB(K,L,M,KMAT).NE.0.0) RMFTAB(K,L,M,KMAT) =
     1                                          ALOG10(RMFTAB(K,L,M,KMAT))
                     IF(ROSTAB(K,L,M,KMAT).NE.0.0) ROSTAB(K,L,M,KMAT) =
     1                                          ALOG10(ROSTAB(K,L,M,KMAT))
   50 CONTINUE
c
      RETURN
      END
```

```
      subroutine kappa (z,ja,jb)
c
c        this subroutine computes the plasma thermal conductivity
c        it uses a curve fit to loglamda
c        see subs. llam and pcond in MF-FIRE   FDM-458
c        vectorized
c
      common/wrk   / work(9999,12)
      common /np1/  u(9999)   , v(9999)   , rho(9999) , e(9999)   ,
     1              er(9999)  , p(9999)   , tp(9999)
      common /prop/  gam1(9999), rosmfp(9999), rmfp1t(9999),
     1             rmfp2t(9999), rcsubv(9999),   xkap(9999)
      common /a/    beta(9999)  , xmask(9999), con(30)
c
      real lnxb(10), lnyb(14)
      dimension xt(10), yt(14), rxt(10), ryt(14), z(*)
      dimension      x(9999), dens(9999), xisx(9999), xisy(9999),
     1          xlogim(9999), sqtp(9999)
c
      equivalence (x(1)     , work(1,1)), (dens(1), work(1,2)),
     1            (xisx(1)  , work(1,3)), (xisy(1), work(1,4)),
     2            (xlogim(1), work(1,5)), (sqtp(1), work(1,6))
c
      data xt / 1.e3, 100., 10., 1., .1, .01, .001, .0001, 1.e-5, 1.e-6/
      data rxt/ .001, .01, .1, 1., 10., 100., 1000., 1.e4, 1.e+5, 1.e+6/
      data yt / 1.e28, 1.e27, 1.e26, 1.e25, 1.e24, 1.e23, 1.e22, 1.e21,
     1          1.e20, 1.e19, 1.e18, 1.e17, 1.e16, 1.e15 /
      data ryt/ 1.e-28, 1.e-27, 1.e-26, 1.e-25, 1.e-24, 1.e-23, 1.e-22,
     1          1.e-21, 1.e-20, 1.e-19, 1.e-18, 1.e-17, 1.e-16, 1.e-15 /
      data lnxb /  6.91,  4.60,  2.30, 0.0, -2.30, -4.60, -6.91, -9.21,
     1           -11.51,-13.81 /
      data lnyb / 64.47, 62.17, 59.87, 57.56, 55.26, 52.96, 50.66,
     1            48.35, 46.05, 43.75, 41.45, 39.14, 36.84, 34.54 /
c
c        define misc goodies here
c
      do 150 j=ja,jb
         sqtp(j) = sqrt( tp(j) )
         x(j)    = 3.1623e-5 * tp(j) * sqtp(j) / z(j)
         dens(j) = con(6) * rho(j)
         xisx(j) = 1.0
         xisy(j) = 1.0
  150 continue
c
c        search for limits on x
c        switch order of loops for efficiency
c
      do 155 i=1,9
         do 155 j=ja,jb
            xtemp   = float(i) + 1.
            delx    = x(j) - xt(i)
            xisx(j) = cvmgp ( xisx(j), xtemp, delx )
  155 continue
c
c        now search for limits on density
```

```
1517    c
1518          do 160 i=1,13
1519             do 160 j=ja,jb
1520                xtemp   = float(i) + 1.
1521                delrho  = dens(j) - yt(i)
1522                xisy(j) = cvmgp ( xisy(j), xtemp, delrho )
1523     160    continue
1524    c
1525    c           determine loglamdas in a scalar loop
1526    c
1527          do 170 j=ja,jb
1528             isx       = int( xisx(j) )
1529             isy       = int( xisy(j) )
1530             xlog      = 33.825 + lnxb(isx) + .26 * x(j) * rxt(isx) -
1531           1               0.5 * ( lnyb(isy) + .26*dens(j) * ryt(isy) )
1532             xlogim(j) = cvmgp ( xlog, 1.0, (xlog-1.0) )
1533     170    continue
1534    c
1535    c           now put all this stuff together to get kappa
1536    c
1537          do 180 j=ja,jb
1538             xkap(j) = con(8) * tp(j) * tp(j) * sqtp(j) /
1539           1                          ( xlogim(j) * (4. + z(j) ) )
1540     180    continue
1541    c
1542          return
1543          end
```

```
      SUBROUTINE POINT
C
C  POINT COMPUTES THE (K,L,M) INDEX IN THE EOS TABLES THAT CORRESPOND
C  TO (R-TEMP,P-TEMP,DENSITY) IN EACH ZONE OF THE PLASMA
c       modified for cray vectorization
C
      common/eos1/   ad,  bd,  at,  bt, nmat, nfg, ja, jb,
     1              rad, rdb, rat, rbt
       common/eoswrk/  trl(9999),  tpl(9999), rhol(9999),
     1               meos(9999), leos(9999), keos(9999)
C
C  FIND THE INDEX FOR THE DENSITY
c
      DO 100 J = ja,jb
         temp    = (rhol(j) - bd) * rad + 1.
         x       = cvmgp( temp,  0., temp-1.)
9x7= cvmgp(4x,  9.,39.-x )
         MEOS(J) = ifix(x)
  100 CONTINUE
C
C  FIND THE INDEX FOR radiation TEMPERATURE
c
      DO 200 J = ja,jb
         temp    = (trl(j) - bt ) * rat + 1
         x       = cvmgp( temp, 0.0, temp-1.)
9x7= cvmgp(4x, 19.,  19.-x )
         keos(J) = ifix(x)
  200 CONTINUE
C
C  FIND THE INDEX FOR plasma TEMPERATURE
c
      DO 300 J = ja,jb
         temp    = (tpl(j) - bt ) * rat + 1
         x       = cvmgp( temp, 0.0, temp-1.)
9x7= cvmgp(4x, 19.,  19.-x )
         leos(J) = ifix(x)
  300 CONTINUE
c
      return
      END
```

```
      subroutine restart (irst,iwrst)
      integer cycle, wl, wr, wt, wb
      common /wrk/  un(9999)  , vn(9999)  , rhon(9999), en(9999)  ,
     1              pr(9999)  , work(9999,7)
      common /np1/  u(9999)   , v(9999)   , rho(9999) , e(9999)   ,
     1              er(9999)  , p(9999)   , tp(9999)
      common /a/    beta(9999)  , xmask(9999), con(30)
      common /e/  cycle, wl, wr, wt, wb, iter, imx, jmx, delt, xmu,
     1            im1, jm1, im2, jm2, iterlmt, cyl, epsi, gaminit,
     2            gx, gy, ui, vi, cnrst, cndmp, omega, alpha, t, am1,
     3            gamma, asq, rhoi, eamb, dtrad, otd, ftd, xkap2,
     4            kpta, kptb, kptc, kptd, kpte, kptf, eamb2, rcsubin,
     5            bvar1, ivar2, bvar3, bvar4, bvar5, bvar6, bvar7,bvar8
      character*80 header
c
      read(irst,1) ir,jr,header
 1    format(i3,i3,a80)
      write(iwrst,2) header
 2    format(a80)
      do 20 k=1,kptd
            read(irst,*) ipt,u(k),v(k),p(k),rho(k),e(k),tp(k),er(k)
            rhon(k) = rho(k)
            en(k)   = e(k)
            pr(k)   = p(k) + otd*er(k)*con(16)
 20         continue
      return
      end
```

```
      subroutine solabc (iflag)
c
c        boundary condition subroutine
c          bc flags (wl, wr, wt, wb) +
c             1--rigid, free slip wall
c             2--rigid, no-slip wall
c             3--continuative outflow wall
c             4--periodic (symmetric bc)
c             5--user defined
c
c         note: since explicit routines used for e, er, and rho
c               only n+1 bc values needed--called at end of time step
c
       integer cycle, wl, wr, wt, wb
       common /mesh/ xc(9999)  , yc(9999)  , dx(9999)   , dy(9999)  ,
     1               rdx(9999) , rdy(9999) ,
     2               rdxc(9999), rdyc(9999), rxc(9999),   r2dx(9999),
     3               r2dy(9999)
       common /wrk/  un(9999)  , vn(9999)  , rhon(9999), en(9999)  ,
     1               pr(9999)  , work(9999,7)
       common /np1/  u(9999)   , v(9999)   , rho(9999) , e(9999)   ,
     1               er(9999)  , p(9999)   , tp(9999)
       common /prop/  gam1(9999), rosmfp(9999), rmfp1t(9999),
     1              rmfp2t(9999), rcsubv(9999),   xkap(9999)
       common /a/    beta(9999)  , xmask(9999), con(30)
       common /e/  cycle, wl, wr, wt, wb, iter, imx, jmx, delt, xmu,
     1             im1, jm1, im2, jm2, iterlmt, cyl, epsi, gaminit,
     2             gx, gy, ui, vi, cnrst, cndmp, omega, alpha, t, am1,
     3             gamma, asq, rhoi, eamb, dtrad, otd, ftd, xkap2,
     4             kpta, kptb, kptc, kptd, kpte, kptf, eamb2, rcsubin,
     5             bvar1, ivar2, bvar3, bvar4, bvar5, bvar6, bvar7,bvar8
c
       do 140 k=1,kptb+2,imx
          rho(k)   = rho(k+1)
          e(k)     = e(k+1)
          en(k)    = en(k+1)
          er(k)    = er(k+1)
          gam1(k)  = gam1(k+1)
          rcsubv(k)= rcsubv(k+1)
          xkap(k)  = xkap(k+1)
          tp(k)    = tp(k+1)
ctim      rho(k+im1)  = rho(k+im2)
ctim      e(k+im1)    = e(k+im2)
ctim      en(k+im1)   = en(k+im2)
ctim      er(k+im1)   = er(k+im2)
c
c         left bc
c
          go to (102, 104, 106, 108, 110) wl
 102        u(k)   = 0.0
            v(k)   = v(k+1)
            go to 111
 104        u(k)   = 0.0
            v(k)   = -v(k+1)
            go to 111
```

```
 106        e(k)    = e(k+1)
            er(k)   = er(k+1)
            if(iflag.gt.0) go to 111
              u(k)   = u(k+1)
              v(k)   = v(k+1)
              go to 111
 108        u(k)     = u(k+im2)
            v(k)     = v(k+im2)
            rho(k)   = rho(k+im2)
            e(k)     = e(k+im2)
            en(k)    = en(k+im2)
            er(k)    = er(k+im2)
            go to 111
 110        continue
            u(k)   = 0.0
            v(k)   = v(k+1)
c
c        right bc
c
 111     go to (122, 124, 126, 128, 130) wr
 122        u(k+im2) = 0.0
            v(k+im1) = v(k+im2)
            go to 140
 124        u(k+im2)  = 0.0
            v(k+im1) = -v(k+im2)
            go to 140
 126        if(iflag.gt.0) go to 140
              u(k+im2)  = u(k+im2-1)
              v(k+im1) = v(k+im2)
              go to 140
 128        u(k+im1)   = u(k+1)
            v(k+im1)   = v(k+1)
            rho(k+im1) = rho(k+1)
            e(k+im1)   = e(k+1)
            er(k+im1)  = er(k+1)
            go to 140
 130        continue
c
c
 140  continue
         do 190 k=2,im1
            rho(k)      = rho(k+imx)
            e(k)        = e(k+imx)
            en(k)       = en(k+imx)
            en(k+kpte) = en(k+kptf)
            rho(k+kpte)= rho(k+kptf)
            e(k+kpte)  = e(k+kptf)
            er(k)      = er(k+imx)
            er(k+kpte) = er(k+kptf)
            tp(k)      = tp(k+imx)
            tp(k+kpte) = tp(k+kptf)
c
c        top bc
c
            go to (152, 154, 156, 158, 160) wt
```

```
1721     152        v(k+kptf) = 0.0
1722                u(k+kpte) = u(k+kptf)
1723                go to 161
1724     154        v(k+kptf) = 0.0
1725                u(k+kpte) = -u(k+kptf)
1726                go to 161
1727     156        if(iflag.gt.0) go to 161
1728                  v(k+kptf) = v(k+kptf-imx)
1729                  u(k+kpte) = u(k+kptf)
1730                  go to 161
1731     158        v(k+kpte)   = v(k+imx)
1732                u(k+kpte)   = u(k+imx)
1733                rho(k+kpte) = rho(k+imx)
1734                e(k+kpte)   = e(k+imx)
1735                er(k+kpte)  = er(k+imx)
1736                go to 161
1737     160        continue
1738    c
1739    c         bottom bc
1740    c
1741     161     go to (172, 174, 176, 178, 180) wb
1742     172        v(k)   = 0.0
1743                u(k)   = u(k+imx)
1744                go to 190
1745     174        v(k)   = 0.0
1746                u(k)   = -u(k+imx)
1747                go to 190
1748     176        if(iflag.gt.0) go to 190
1749                  v(k)   = v(k+imx)
1750                  u(k)   = u(k+imx)
1751                  go to 190
1752     178        v(k)     = v(k+kptf)
1753                u(k)     = u(k+kptf)
1754                rho(k)   = rho(k+kptf)
1755                e(k)     = e(k+kptf)
1756                er(k)    = er(k+kptf)
1757                go to 190
1758     180        continue
1759     190  continue
1760          return
1761          end
```

```
      subroutine solaice
c
c        this subroutine solves a single time step of a 2-d
c             compressible flow problem (with energy equation)
c             using the sola-ice technique
c             modified for
c                          vectorized loops
c                          pressure iteration
c        note:    k   ----- i,j
c                 k-1  ---- i-1,j
c                 k+1  ---- i+1,j
c                 k-imx --- i,j-1
c                 k+imx --- i,j+1
c
       integer cycle, wl, wr, wt, wb
       common /mesh/ xc(9999)  , yc(9999)  , dx(9999)   , dy(9999)  ,
     1               rdx(9999) , rdy(9999) ,
     2               rdxc(9999), rdyc(9999), rxc(9999),   r2dx(9999),
     3               r2dy(9999)
       common /wrk/  un(9999)  , vn(9999)  , rhon(9999), en(9999)  ,
     1               pr(9999)  , work(9999,7)
       common /np1/  u(9999)   , v(9999)   , rho(9999) , e(9999)   ,
     1               er(9999)  , p(9999)   , tp(9999)
       common /prop/  gam1(9999), rosmfp(9999), rmfp1t(9999),
     1              rmfp2t(9999), rcsubv(9999),   xkap(9999)
       common /a/    beta(9999)  , xmask(9999), con(30)
       common /e/  cycle, wl, wr, wt, wb, iter, imx, jmx, delt, xmu,
     1             im1, jm1, im2, jm2, iterlmt, cyl, epsi, gaminit,
     2             gx, gy, ui, vi, cnrst, cndmp, omega, alpha, t, am1,
     3             gamma, asq, rhoi, eamb, dtrad, otd, ftd, xkap2,
     4             kpta, kptb, kptc, kptd, kpte, kptf, eamb2, rcsubin,
     5             bvar1, ivar2, bvar3, bvar4, bvar5, bvar6, bvar7,bvar8
c
       dimension enp1(9999),  ernp1(9999)
       equivalence ( enp1(1), work(1,1) ), ( ernp1(1), work(1,2) )
c
c        start cycle
c
       if (cycle.eq.0) go to 480
c
c**********************************************************************
c                                                                     *
c        first compute u-tilde and v-tilde using explicit solver      *
c                                                                     *
c**********************************************************************
c        compute u tilde
c
       do 60 k=kpta, kptb
          fux =  am1*un(k)*rdxc(k)*(un(k+1)-un(k-1)) +
     1           0.5*alpha*(rdx(k+1)*(un(k+1)-un(k))*(un(k)-abs(un(k)))+
     2                      rdx(k)*(un(k)-un(k-1))*(un(k)+abs(un(k))) )
          vstar= 0.5*rdxc(k)*( dx(k)*(vn(k)+vn(k-imx)) +
     1                         dx(k+1)*(vn(k+imx+1)+vn(k-imx+1)) )
          fuy =  2.*am1*vstar*( un(k+imx)-un(k-imx) ) * r2dy(k) +
     1           alpha*(rdyc(k)*(un(k+imx)-un(k))*(vstar-abs(vstar)) +
```

```
1817        2                 rdyc(k-imx)*(un(k)-un(k-imx))*(vstar-abs(vstar)))
1818             vk   = 2.*xmu/(rhon(k)+rhon(k+1))
1819             visx = vk*( ftd * rdx(k) * ( rdxc(k-1) * (un(k-1) - un(k)) -
1820        a                                 rdxc(k)   * (un(k) - un(k+1)) ) +
1821        1               rdy(k) * ( rdyc(k-imx) * (un(k-imx) - un(k)) -
1822        a                          rdyc(k)     * (un(k) - un(k+imx)) ) +
1823        2          (vn(k+1)-vn(k-imx+1)-vn(k)+vn(k-imx))*otd*rdy(k)*
1824        3                                                 2.*rdxc(k) +
1825        4          ftd*cyl*( (un(k+1)-un(k-1) )*2.*r2dx(k) -
1826        5                    un(k)/(abs(xc(k))+.5*dx(k)) ) /
1827        6                  ( abs(xc(k))+.5*dx(k) )   )
1828             dp   = 2.*(pr(k)-pr(k+1)) / (dx(k)*rhon(k) + dx(k+1)*rhon(k+1))
1829    c
1830    c
1831             u(k) = un(k) + xmask(k) * delt*( dp + gx - fux - fuy + visx )
1832     60   continue
1833    c
1834    c        compute v tilde
1835    c
1836          do 61 k=kpta, kptb
1837            ustar= 0.5*rdyc(k)*( dy(k)*(un(k)+un(k-1)) +
1838        1                        dy(k+imx)*(un(k+imx)+un(k+imx-1)) )
1839             fvx = 2.*am1*ustar*( vn(k+1) - vn(k-1) ) * r2dx(k) +
1840        1         alpha*(rdxc(k)*(vn(k+1)-vn(k))*(ustar - abs(ustar)) +
1841        2                rdxc(k-1)*(vn(k)-vn(k))*(ustar - abs(ustar)) )
1842             fvy = am1*vn(k)*rdyc(k)*( vn(k+imx) - vn(k-imx) ) +
1843        1        .5*alpha*(rdy(k+imx)*(vn(k+imx)-vn(k))*(vn(k)-abs(vn(k)))+
1844        2                  rdy(k)*(vn(k)-vn(k-1))*(vn(k)-abs(vn(k))) )
1845             vk   = 2.*xmu/(rhon(k)+rhon(k+imx))
1846             visy= vk*( rdx(k-1) * ( rdxc(k-1) * (vn(k-1) - vn(k)) -
1847        a                            rdxc(k)    * (vn(k) - vn(k+1)) ) +
1848        1               ftd*rdy(k) * ( rdyc(k-imx) * (vn(k-imx) - vn(k)) -
1849        b                            rdyc(k)     * (vn(k) - vn(k+imx)) ) +
1850        2          (un(k+imx)-un(k)-un(k+imx-1)+un(k-1))*otd*rdx(k)*
1851        3                                                 2.*rdyc(k) +
1852        4          cyl*rxc(k)*( (vn(k+1)-vn(k-1))*2.*r2dx(k) +
1853        5                      2.*otd*rdyc(k)*( un(k+imx)+un(k+imx-1)-
1854        6                                       un(k)-un(k-1)) ))
1855             dp  =2.*(pr(k)-pr(k+imx))/(dy(k)*rhon(k)+dy(k+imx)*rhon(k+imx))
1856    c
1857    c
1858             v(k)= vn(k) + xmask(k)*delt*( dp + gy - fvx - fvy + visy )
1859     61   continue
1860    c*****************************************************************
1861    c                                                                *
1862    c        pressure solver sequence -- determine u,v,p (n+1)       *
1863    c        satisfy continuity and reduced momentum equations       *
1864    c        implicit technique                                      *
1865    c                                                                *
1866    c*****************************************************************
1867          iter = 0
1868          flg  = 0.
1869    c
1870    c         now set boundary conditions
1871    c
```

```
 75   continue
      iflag = iter
      call solabc (iflag)
c
c        check for pressure convergence
c
      iter = iter + 1
      if (abs(flg-1.0).lt.0.001) go to 400
         if (iter.lt.iterlmt) go to 255
            if (cycle.lt.2) go to 400
               t = 1.e+32
               go to 502
c
c        vector function cvmgp sets iflg:
c           = 0  if temp.ge.0  (not converged)
c           = 1  if temp.lt.0  (converged)
c
 255  continue
c
      flg = 1.
c
      do 301 k=kpta,kptb
         d    = rdx(k)*(u(k)-u(k-1))+rdy(k)*(v(k)-v(k-imx))
     1          +cyl*(u(k)+u(k-1))*(0.5*rxc(k))
         rot  = rhon(k)/(1.0+delt*d)
         et   = e(k) - p(k)*delt*d/rot*con(17)
         pt   = gam1(k) * rot * et * con(16)
         delp = -beta(k)*(p(k) - pt) * xmask(k)
         p(k) = p(k) + delp
         temp = abs(delp) - p(k)*epsi
         flg  = flg * cvmgp(0.0, 1.0, temp)
         u(k) = u(k)+2.0*delt*rdx(k)*delp/(rhon(k)+rhon(k+1))
         u(k-1)=u(k-1)-2*delt*rdx(k)*delp/(rhon(k-1)+rhon(k))
         v(k) = v(k)+ 2.*delt*rdy(k)*delp/(rhon(k)+rhon(k+imx))
         v(k-imx)=v(k-imx)-2*delt*rdy(k)*delp/(rhon(k-imx)+rhon(k))
 301  continue
      go to 75
 400  continue
c*********************************************************************
c                                                                    *
c        converged step--now update scalar transport equation        *
c                                                                    *
c*********************************************************************
c
c        density equation
c
      do 450 k=kpta, kptb
         f1       = rhon(k)
         f2       = -rdx(k)*( (u(k)*( rhon(k) + rhon(k+1) ) +
     1                         alpha*abs(u(k))*( rhon(k) - rhon(k+1)))
     2                      - (u(k-1)*( rhon(k-1)+ rhon(k) ) +
     3                         alpha*abs(u(k-1))*( rhon(k-1)- rhon(k))))
         f3       = -rdy(k)*( (v(k)*( rhon(k) + rhon(k+imx) ) +
     1                         alpha*abs(v(k))*( rhon(k) - rhon(k+imx)))
     2                      - (v(k-imx)*(rhon(k)+rhon(k-imx)) +
```

```
1927        3                       alpha*abs(v(k-imx))*( rhon(k-imx)-rhon(k))))
1928            f4       = -rxc(k)*( u(k)*( rhon(k)+ rhon(k+1) ) +
1929        1                         alpha*abs(u(k))*( rhon(k) - rhon(k+1) )
1930        2                       + u(k-1)*( rhon(k-1)+ rhon(k) ) +
1931        3                         alpha*abs(u(k-1))*( rhon(k-1)- rhon(k)) )
1932            rho(k)   = f1 + xmask(k) * delt * 0.5 * ( f2 + f3 + cyl*f4 )
1933    9rho(k)3= cvmgp (rho(k), con(29), (rho(k) - con(29)) )
1934     450  continue
1935    c
1936    c        energy equation--convective portion only
1937    c        for both e and er energy densities
1938    c
1939          do 475 k=kpta, kptb
1940             f1  = e(k)
1941             f2  = am1*(u(k)+u(k-1))*( e(k+1) - e(k-1) ) * r2dx(k)  +
1942        1          am1*(v(k)+v(k-imx))*( e(k+imx) - e(k-imx) ) * r2dy(k)+
1943        2          0.5*alpha*( rdxc(k)*(u(k)+u(k-1) -abs(u(k)+u(k-1)) ) *
1944        3                              ( e(k+1) - e(k) ) +
1945        4                     rdxc(k-1)*(u(k)+u(k-1)+abs(u(k)+u(k-1)) ) *
1946        5                              ( e(k) - e(k-1) ) ) +
1947        6          0.5*alpha*(rdyc(k)*(v(k)+v(k-imx)-abs(v(k)+v(k-imx)))*
1948        7                              ( e(k+imx) - e(k) ) +
1949        8                   rdyc(k-imx)*(v(k)+v(k-imx)+abs(v(k)+v(k-imx)))*
1950        9                              ( e(k) - e(k-imx) ) )
1951    c
1952          enp1(k)= f1 - xmask(k) * delt* f2
1953     475  continue
1954    c
1955          do 476 k=kpta, kptb
1956             f1  = er(k)
1957             f2  = am1*(u(k)+u(k-1))*( er(k+1) - er(k-1) ) * r2dx(k)  +
1958        1          am1*(v(k)+v(k-imx))*( er(k+imx) - er(k-imx) )*r2dy(k)+
1959        2          0.5*alpha*( rdxc(k)*(u(k)+u(k-1) -abs(u(k)+u(k-1)) ) *
1960        3                              ( er(k+1) - er(k) ) +
1961        4                     rdxc(k-1)*(u(k)+u(k-1)+abs(u(k)+u(k-1)) ) *
1962        5                              ( er(k) - er(k-1) ) ) +
1963        6          0.5*alpha*(rdyc(k)*(v(k)+v(k-imx)-abs(v(k)+v(k-imx)))*
1964        7                              ( er(k+imx) - er(k) ) +
1965        8                   rdyc(k-imx)*(v(k)+v(k-imx)+abs(v(k)+v(k-imx)))*
1966        9                              ( er(k) - er(k-imx) ) )
1967    c
1968          ernp1(k)= f1 - xmask(k) * delt* f2
1969     476  continue
1970    c
1971          do 477 k=kpta,kptb
1972             tp(k) = tp(k) + rcsubv(k) * (enp1(k) - e(k))
1973             e(k)  = enp1(k)
1974             erad  = ernp1(k)
1975             er(k) = cvmgp ( erad, con(26), (erad-con(26)) )
1976     477  continue
1977    c
1978    c           now update the hydro pressure and total pressure
1979    c
1980          do 478 k=1,kptd
1981             p(k)  = gam1(k) * rho(k) * e(k) * con(16)
```

```
1982            pr(k) = p(k) + otd * er(k) * con(16)
1983            en(k) = e(k)
1984     478  continue
1985    c
1986    c
1987     480  continue
1988    c
1989    c         call boundary conditons since loops go to im1, and jm1
1990    c
1991          iflag = 0
1992          call solabc (iflag)
1993    c*****************************************************************
1994    c                                                               *
1995    c         compute the relaxation factors--constant for time step  *
1996    c                                                               *
1997    c*****************************************************************
1998          do 490 k=kpta, kptb
1999             pto   = gam1(k) * rho(k) * e(k) * con(16)
2000             delp  = 1.e-4 * pto
2001             ur    = 2.0*delt * rdx(k) * delp /(rho(k)+rho(k+1))
2002             ul    =-2.0*delt * rdx(k) * delp /(rho(k-1)+rho(k))
2003             vt    = 2.0*delt * rdy(k) * delp /(rho(k)+rho(k+imx))
2004             vb    =-2.0*delt * rdy(k) * delp /(rho(k-imx)+rho(k))
2005             dt    = delt *(rdx(k)*(ur-ul) + rdy(k)*(vt-vb)
2006          1           + cyl*(ur+ul)*0.50*rxc(k))
2007             rot   = rho(k)/(1.0 + dt)
2008             et    = e(k) - p(k)/rho(k)*dt*con(17)
2009             pt    = gam1(k)*rot*et*con(16)
2010             beta(k) = omega*delp/(delp-(pt-pto))
2011     490  continue
2012     502  continue
2013          return
2014          end
2015          END
```

```
      SUBROUTINE TABLE2(A,AA,AADER,tp)
C
C  TABLE2 INTERPOLATES IN THE TWO-DIMENSIONAL EOS TABLES USING THE
C  LEOS AND MEOS INDICES
c       modified for cray vectorization
c                cvmgz (=0, <>0, test)
C
C
       common/eos1   / ad,  bd,  at,  bt, nmat, nfg, ja, jb,
     1                 rad, rdb, rat, rbt
       common/wrk   / work(9999,12)
       common/eoswrk/  trl(9999),  tpl(9999), rhol(9999),
     1                meos(9999), leos(9999), keos(9999)
c
       dimension A(*), AA(*), AADER(*), tp(*),
     1           dtp(9999), dden(9999), xpt(9999)
c
       equivalence (work(1,5), dtp(1)), (work(1,6), dden(1)),
     1             (work(1,7), xpt(1))
c
       data kmat, xtab, itab, xytab /1, 20., 20, 200./
c
       const = float(kmat-1) * xytab
C
C  FIND THE EOS QUANTITIES
c
       DO 100 J = ja,jb
          l       = LEOS(J)
          M       = MEOS(J)
          dtp(j)  = tpl(J) - AT * (L-1) - BT
          dden(j) = rhol(J) - AD * (M-1) - BD
C
C  TEST FOR LOWER TABLE LIMIT
c
          xl       = float(l)
          dtp(j)  = cvmgz (0.0, dtp(j), xl)
          xl      = cvmgz (1.0,     xl, xl)
          xm      = float(m)
          dden(j) = cvmgz (0.0, dden(j), xm)
          xm      = cvmgz (1.0,      xm, xm)
          xpt(j)  = (xm - 1.)*xtab + xl + const
 100  continue
C
C  IF THE DENSITY BECOMES LOWER THAN THE MINIMUM IN THE TABLE, WE MUST
C  EXTRAPOLATE TO GET Z2B AND EN2B USING Y=SLOPE*X + B FORMULA.
C  EXTRAPOLATE THE DENSITY AT L.
C     SLOPE1 = -(A(L,1,KMAT) - A(L,2,KMAT))/AD
C     B1 = A(L,1) - SLOPE1 * BD
C     ALMS = SLOPE1 * rhol(J) + B1
C  EXTRAPOLATE THE DENSITY AT L + 1
C     SLOPE2 = -(A(L+1,1,KMAT) - A(L+1,2,KMAT))/AD
C     B2 = A(L+1,1,KMAT) - SLOPE2 * BD
C     ALMSS = SLOPE2 * rhol(J) + B2
C     GO TO 30
C
```

```
2071    c
2072    c  set table values in a scalar loop
2073    c
2074          do 110 j=ja,jb
2075             ipt       = ifix(xpt(j))
2076             work(j,1) = a(ipt)
2077             work(j,2) = a(ipt+itab)
2078             work(j,3) = a(ipt+1)
2079             work(j,4) = a(ipt+itab+1)
2080     110  continue
2081    c
2082    c  now finish with a vectorized loop
2083    c
2084          do 150 j=ja,jb
2085    c
2086    C  INTERPOLATE THE DENSITY AT L and l+1
2087    c
2088             ALMS  = work(j,1) + (work(j,2) - work(j,1)) * rad * dden(j)
2089             almss = work(j,3) + (work(j,4) - work(j,3)) * rad * dden(j)
2090    C
2091    C  COMPUTE THE PLASMA TEMPERATURE DERIVATIVE
2092    c
2093             ADER = (ALMSS - ALMS) * rat
2094             QUAN =    ALMS + ADER * DTP(j)
2095    C
2096    C  CONVERT FROM LOGARITHMIC BACK TO REAL UNITS
2097    c
2098             AA(J)    = 10.**QUAN
2099             AADER(J) = ADER * AA(J) / tp(j)
2100    C
2101      150 CONTINUE
2102          RETURN
2103          END
```

```
      SUBROUTINE TABLE3(B,AA)
C
C  TABLE3 INTERPOLATES IN THE THREE-DIMENSIONAL EOS TABLES USING THE
C  KEOS, LEOS, AND MEOS INDICES
c     modified for cray vectorization
C
      common/eos1  /   ad,  bd,  at,  bt, nmat, nfg, ja, jb,
     1               rad, rdb, rat, rbt
      common/wrk   / work(9999,12)
      common/eoswrk/  trl(9999),  tpl(9999), rhol(9999),
     1              meos(9999), leos(9999), keos(9999)
c
      dimension B(*), aa(*), xpt(9999), dden(9999),
     1                       dtp(9999),  dtr(9999)
c
      equivalence (xpt(1), work(1,11)), (dden(1), work(1,9)),
     1            (dtp(1), work(1,12)), ( dtr(1), work(1,10))
c
      data kmat,xtab,xytab,itab,ijtab
     1    /   1, 20., 400.,  20,  400/
c
      DO 200 J = ja,jb
         xk      = float(KEOS(J))
         xl      = float(LEOS(J))
         xm      = float(MEOS(J))
         dden(j) = rhol(J)- AD * (xm-1.) - BD
         dtp(j)  = tpl(J) - AT * (xl-1.) - BT
         dtr(j)  = trl(J) - AT * (xk-1.) - BT
C
C  TEST FOR LOWER TABLE LIMIT
c
         dtr(j)  = cvmgn ( dtr(j), 0.0, xk)
         xk      = cvmgn (     xk, 1.0, xk)
         dtp(j)  = cvmgn ( dtp(j), 0.0, xl)
         xl      = cvmgn (     xl, 1.0, xl)
         dden(j) = cvmgn (dden(j), 0.0, xm)
         xm      = cvmgn (     xm, 1.0, xm)
         xpt(j)  = (xm -1.)*xytab + (xl -1.)*xtab + xk
 200  continue
c
c  setup property arrays---scalar mode
c
      do 250 j=ja,jb
         ipt       = ifix(xpt(j))
         work(j,1) = b(ipt)
         work(j,2) = b(ipt+itab)
         work(j,3) = b(ipt+1)
         work(j,4) = b(ipt+itab+1)
         work(j,5) = b(ipt+ijtab)
         work(j,6) = b(ipt+ijtab+itab)
         work(j,7) = b(ipt+ijtab+1)
         work(j,8) = b(ipt+ijtab+itab+1)
 250   continue
c
c  now finish the job with vectorized loop
```

```
2159    c
2160          do 300 j=ja,jb
2161    C
2162    C  INTERPOLATE THE PLASMA TEMPERATURE AT M and M+1
2163    c
2164               AMS  = work(j,1) + (work(j,2) - work(j,1)) * rat * dtp(j)
2165               AMSS = work(j,3) + (work(j,4) - work(j,3)) * rat * dtp(j)
2166    c
2167               AM1S = work(j,5) + (work(j,6) - work(j,5)) * rat * dtp(j)
2168               AM1SS= work(j,7) + (work(j,8) - work(j,7)) * rat * dtp(j)
2169    C
2170    C  COMPUTE THE RADIATION TEMPERATURE DERIVATIVE AT M AND M+1
2171    c
2172               ADER  = (AMSS  - AMS)  * rat
2173               ADER1 = (AM1SS - AM1S) * rat
2174    C
2175    C  INTERPOLATE THE RADIATION TEMPERATURE AT M AND M+1
2176    c
2177               QUAN  = AMS  + ADER  * DTR(j)
2178               QUAN1 = AM1S + ADER1 * DTR(j)
2179    C
2180    C  INTERPOLATE THE DENSITY
2181    c
2182               QUAN  = QUAN + (QUAN1-QUAN) * rad * dden(j)
2183    C
2184    C  CONVERT FROM LOGARITHMIC BACK TO REAL UNITS
2185    c
2186               AA(J) = 10.0**QUAN
2187      300 CONTINUE
2188          RETURN
2189          END
```