Numerical Simulations for Forest Wildfires

New Mexico

Supercomputing Challenge

Final Report

March 30, 2009

Team 58

Los Alamos High School

Team Members:

Edward J. Dai

Contents

1 Executive Summary **4**

2 Introduction **4**

2.1 Forest Wildfires 4

2.2 Problem Statement 5

2.3 Scope of the Project 6

3 Physics Models and Basic Equations **6**

3.1 Physics Models 6

3.2 Basic Equations 7

4 Numerical Methods **8**

4.1 Difference Equations 9

4.2 Explicit Treatment for Heat Conduction 12

4.3 Numerical Schemes for Multi-dimensional Flows 13

4.4 Time Step Control 13

4.5 Simulation Procedure 14

5 Numerical Results **15**

5.1 Convergence and Accuracy 15

5.2 Parameters in Simulations of Forest Fires 16

5.3 Development of Forest Fires 16

6 Conclusions and Discussions **21**

7 Acknowledgments **22**

References **23**

A Source Codes **24**

1 Executive Summary

In this project, we have proposed a simplified model for the development of forest wildfires, modified an existing numerical algorithm for gas dynamics, and developed a three-dimensional computer code. We have obtained preliminary numerical results through three-dimensional computer simulations for forest fires.

In our model, we treat forest trees as an energy source. The abundance of trees and oxygen that are needed for energy release are together described by a density function for energy source. IN this project, radiation effects of fires are ignored in this project. Therefore, we approximately use the set of equations for gas dynamics together with the energy source to describe the development of forest fires.

To our knowledge, the idea to use gas dynamics equations together with the energy source to simulate the development of forest wildfires is original. The numerical algorithm we use to solve the equations are dimension splitting and Brian Van Leer's MUSCL scheme[5]. The algorithm to solve multi-dimensional gas dynamics equations is second order accurate in both space and time. To get the second order accuracy in time, we have not explicitly evaluate any variable at the half time step. Instead, through characteristic theory [3], we obtain the time-averaged flux, i.e., the flux at a half time step, through spatial domain-averaged values.

The capability of the code has been demonstrated through numerical examples. The left image in Fig.1 shows the convergence and accuracy of the numerical algorithm for gas dynamics.

The other two images in the figure show a forest fire at its initial stage and the stage after 20 minutes.

2 Introduction

2.1 Forest Wildfires

Forest wildfires are a natural occurrence with billions of dollars being lost each year [10]. For example, Los Alamos Cerro Grande Fire caused one billion dollar damage in 2000. Fires

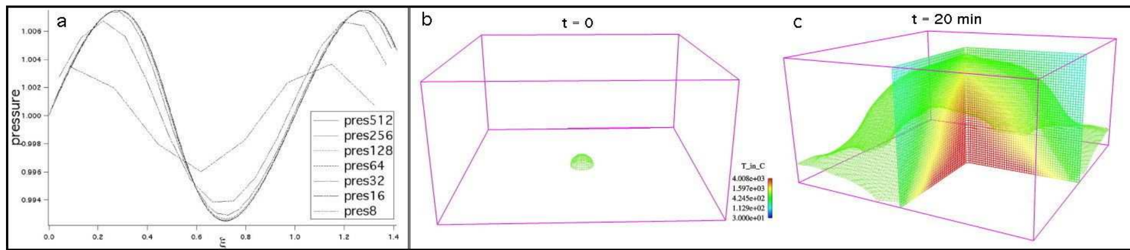

Figure 1: (a) A convergence study of seven two-dimensional simulations for the propagation of a sound wave with resolutions, 8×8, 16×16, 32×32, 64×64, 128×128, 256×256, and 512×512, and iso-surfaces of temperature $492^{o}C$ of an forest fire (b) at the inital time and (c) after 20 minutes.

become a problem when they burn in wrong places, or in right places but at the wrong frequency or the wrong temperatures. Fires in forests that burn under natural circumstances become a problem when those forests are used for a particular purpose, such as settlement or timber production. A controlled burn of forest becomes a problem when it become uncontrolled.

Accurate prediction of forest fire behavior, in particular the most active portion of fire fronts, can give valuable information to forest fire controllers. Modeling can predict the behavior of a fire in progress and its response to different extinguishing strategies, aiding in the optimal placement of manpower and better informed decisions concerning the evacuation of communities. Accurate modeling can also enable the evaluation of the effects of different shapes and placements of fire lines (i.e. regions either bulldozed or burnt ahead of the fire, intended to stop its progress). It is also possible to formulate fire plans for a particular region, for given weather and fuel conditions, so that optimal strategies can be calculated ahead of time.

Modeling is an important research topic in studying forest fires [7-9]. Fire spread is a complex natural propagation phenomenon that requires a great deal of computer storage space and computing power to resolve multi spatial and temporal scales [8].

2.2 Problem Statement

This project is to propose a simplified model for the development of forest wildfires, modify an existing numerical algorithm to the model, and develop three-dimensional computer codes to simulate the development of forest fires.

2.3 Scope of the Project

In this project, we have proposed a simplified model for the development of forest wildfires. As the result of the model, the set of differential equations for gas dynamics together with an energy source is used to described the fires. Modern numerical algorithms for gas dynamics have been modified and used for the development of forest fires. To our knowledge, the idea to use gas dynamics equations together with energy source to simulate the development of forest wildfires is original.

The numerical algorithms we use to solve the equations is dimension splitting [4], and Brian Van Leer's MUSCL scheme [5]. The dimension splitting is to decompose three-dimensional problems into three one-dimensional passes.

The algorithm to solve gas dynamics equations is second order accurate in both space and time. To get the second order accuracy in space, linear interpolations are used together with a monotone condition being applied. To get the second order accuracy in time, we follow the characteristic theory [3] and obtain the time-averaged flux through spatial domain-averaged values. We have not explicitly evaluate any variable at the half time step.

Three-dimensional computer code has been generated and preliminary numerical results have been obtained for the development of forest fires.

3 Physics Models and Basic Equations

In this section, we will describe our physics model for the development of forest wildfires and give resulting basic equations we will use to simulate the fires.

3.1 Physics Models

In our model, the forest will be simplified as a three-dimensional rectangular volume. The forest trees are treated as an energy source. The source is not zero only when local temperature increases to the igniting point of woods. We will not resolve each tree in the forest. Instead, a density function will be used to describe the trees. Also, we will ignore any details

of chemical reaction and combustion. Instead, we take into consideration the results of the combustion, i.e., the energy release from the burning of trees with oxygen. The abundance of trees and oxygen is further described by the strength of the density function and the time period during which local trees could burn and release energy. The period of the burning will be called the local burn-interval, during which local trees could burn to release energy. Furthermore, in this project, we will ignore radiation effects of fires.

Within this model, the development of forest fires could be described by the differential equations for gas dynamics [2].

3.2 Basic Equations

Three-dimensional gas dynamics equations are

$$\frac{\partial \rho}{\partial t} + \bigtriangledown \cdot (\rho \mathbf{u}) = 0, \tag{1}$$

$$\frac{\partial}{\partial t}(\rho u_x) + \frac{\partial}{\partial x}(\rho u_x u_x + p) + \frac{\partial}{\partial y}(\rho u_x u_y) + \frac{\partial}{\partial z}(\rho u_x u_z) = 0, \tag{2}$$

$$\frac{\partial}{\partial t}(\rho u_y) + \frac{\partial}{\partial x}(\rho u_y u_x) + \frac{\partial}{\partial y}(\rho u_y u_y + p) + \frac{\partial}{\partial z}(\rho u_y u_z) = 0, \tag{3}$$

$$\frac{\partial}{\partial t}(\rho u_z) + \frac{\partial}{\partial x}(\rho u_z u_x) + \frac{\partial}{\partial y}(\rho u_z u_y) + \frac{\partial}{\partial z}(\rho u_z u_z + p) = 0, \tag{4}$$

$$\frac{\partial}{\partial t}(\rho E) + \bigtriangledown \cdot [(\rho E + p)\mathbf{u}] = s(t, x, y, z) + \kappa \bigtriangledown^2 T. \tag{5}$$

Here, ρ, p, $\mathbf{u}$, ϵ, and T are air mass density, air pressure, air velocity, specific internal energy of air, and air temperature. $s(t, x, y, z)$ is the source term due to the burning of forest trees with oxygen, and κ is the coefficient of heat conductivity. E is the specific total energy density of air, $E \equiv (\epsilon + \frac{1}{2}\mathbf{u}^2)$. We assume the γ-law, $p = (\gamma - 1)\rho\epsilon$, for the equation of state. Here γ is the ratio of specific heats. The temperature T is related to the specific internal energy through $kT = \frac{2}{3}\frac{m}{N_A}$. Here k is the Boltzmann constant, N_A is the Avogadro constant, and m is the mass of air per mole, which is 0.0289 kg/mole in our calculation.

The energy source s is non-zero constant only at the bottom layer of the simulation domain, and is zero anywhere else. The time-dependent of s is simplified so that s is not zero only when local temperature increases to the igniting point of woods, and only when it is within the local burn-interval described before.

The equations (1-5) can be written in a compact form,

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_x}{\partial x} + \frac{\partial \mathbf{F}_y}{\partial y} + \frac{\partial \mathbf{F}_z}{\partial z} = \mathbf{S}. \tag{6}$$

Here

$$\mathbf{U} \equiv \begin{pmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \rho u_z \\ \rho E \end{pmatrix}, \quad \mathbf{S} \equiv \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \kappa \bigtriangledown^2 T + s \end{pmatrix},$$

$$\mathbf{F}_x(\mathbf{U}) \equiv \begin{pmatrix} \rho u_x \\ \rho u_x^2 + p \\ \rho u_x u_y \\ \rho u_x u_z \\ u_x(\rho E + p) \end{pmatrix}, \quad \mathbf{F}_y(\mathbf{U}) \equiv \begin{pmatrix} \rho u_y \\ \rho u_y u_x \\ \rho u_y^2 + p \\ \rho u_y u_z \\ u_y(\rho E + p) \end{pmatrix}, \quad \mathbf{F}_z(\mathbf{U}) \equiv \begin{pmatrix} \rho u_z \\ \rho u_z u_x \\ \rho u_z u_y \\ \rho u_z^2 + p \\ u_z(\rho E + p) \end{pmatrix}.$$

To solve the set of equations on a numerical grid, we will use the technique of dimensional-splitting [10]. Therefore we will first look into the one-dimensional form of Eqs.(1-5),

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho u_x) = 0, \tag{7}$$

$$\frac{\partial}{\partial t}(\rho u_x) + \frac{\partial}{\partial x}(\rho {u_x}^2 + p) = 0, \tag{8}$$

$$\frac{\partial}{\partial t}(\rho u_y) + \frac{\partial}{\partial x}(\rho u_x u_y) = 0, \tag{9}$$

$$\frac{\partial}{\partial t}(\rho u_z) + \frac{\partial}{\partial x}(\rho u_x u_z) = 0, \tag{10}$$

$$\frac{\partial}{\partial t}(\rho E) + \frac{\partial}{\partial x}[u_x(\rho E + p)] = s_1(t, x) + \kappa \frac{\partial^2 T}{\partial x^2}. \tag{11}$$

Here the different notation $s_1(t, x)$ has been used for the source term and it is one third of s.

We may write the set of equations above in the form,

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_x}{\partial x} = \mathbf{S}_1. \tag{12}$$

4 Numerical Methods

In this section, we will describe key steps in the numerical algorithm we use.

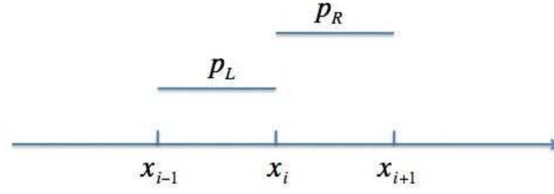

Figure 2: The left and right states at a grid interface x_i.

4.1 Difference Equations

Considering Eq.(12) and one cell $x_i < x < x_{i+1}$ in a one-dimensional grid $\{x_i\}$, we integrate Eq.(12) over the cell and one time step, $0 < t < \triangle t$, to get the following

$$\mathbf{U}_i(\triangle t) = \mathbf{U}_i(0) + \frac{\triangle t}{\triangle x}[\overline{\mathbf{F}}_x(x_i) - \overline{\mathbf{F}}_x(x_{i+1})] + \mathbf{S}_1 \triangle t. \tag{13}$$

Here $\triangle x$ is the width of the grid cell, $\mathbf{U}_i(0)$ and $\mathbf{U}_i(\triangle t)$ are space-averaged values,

$$\mathbf{U}_i(t) \equiv \frac{1}{\triangle x_i}\int_{x_i}^{x_{i+1}} \mathbf{U}(t,x)dx, \tag{14}$$

and $\overline{\mathbf{F}}_x(x_i)$ and $\overline{\mathbf{F}}_x(x_{i+1})$ are time-averaged values of flux at two grid interfaces,

$$\overline{\mathbf{F}}_x(x) \equiv \frac{1}{\triangle t}\int_0^{\triangle t} \mathbf{F}_x(x)dt. \tag{15}$$

The time-averaged flux, Eq.(15), may be approximately written as

$$\overline{\mathbf{F}}_x(x) = \mathbf{F}_x[\overline{\mathbf{U}}(x)]. \tag{16}$$

To get Eq.(16), we have approximately used the product of time-averaged values as the time-averaged value of the product, which is second order accurate. Therefore, one of the important steps in the numerical algorithm is to calculate the time-averaged flux in Eq.(13). We will notate u_{xl} and p_l for the initial values of u_x and p in the cell left to x_i, u_{xr} and p_r for the initial values in the cell right to x_i, as shown in Fig.2, and $\overline{u}_x$ and $\overline{p}$, for the time-averaged values of u_x and p at the grid interface x_i. According to the characteristic theory[3], along the characteristic curves defined by

$$\frac{dx}{dt} = u_x \pm c_s,$$

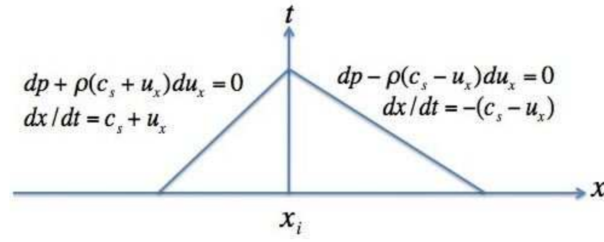

Figure 3: To calculate the time-averaged values of p and u_x during one time step, two characteristic curves are approximately treated as straight lines. Along the characteristic curves, two differentials are zero respectively.

we have

$$dp \pm \rho c_s du_x = 0.$$

Here c_s is the speed of sound wave. As shown in Fig.3, the time-averaged values, $\overline{u}_x$ and $\overline{p}$, could be calculated through solving the following set of algebraic equations.

$$\overline{p} - p_l + \rho_l(c_{sl} + u_{xl})(\overline{u}_x - u_{xl}) = 0, \tag{17}$$

$$\overline{p} - p_r - \rho_r(c_{sr} - u_{xr})(\overline{u}_x - u_{xr}) = 0. \tag{18}$$

Here the subscript l and r stand for the values evaluated at the cells at the left and right sides of x_i respectively. In Eqs.(17,18) and in Fig.3 we have used the assumption that the characteristics are straight lines. If $\overline{p}$ and $\overline{u}_x$ obtained through Eqs.(17,18) are used to calculate the flux needed in Eqs.(13), the resulting numerical scheme is first-order accurate in space. Equations(17,18) require the smoothness of p and u_x, but allow discontinuity in flow density ρ, and more importantly, they are simple. Therefore, they are perfect for the forest fires in which pressure and flow velocity are rather smooth, but flow density experience dramatic changes in space.

To get the second order accuracy of the numerical algorithm, we have to find the internal structures of variables within a grid cell. For this purpose, we use the central difference to determine the slope of a variable, for example, p, as shown in Fig.4, where one dashed line connects the mid-points of the two neighboring cells, and the solid line passes through the mid-point of the middle cell. The solid line is the internal structure of the cell for $p(x)$. If D is the domain of dependence of the wave propagating to the left, then the space-averaged value of $p(x)$ on D, p_{rd}, is the actual value that is used in Eqs.(17,18) for p_r. Therefore, Eqs.(17,18)

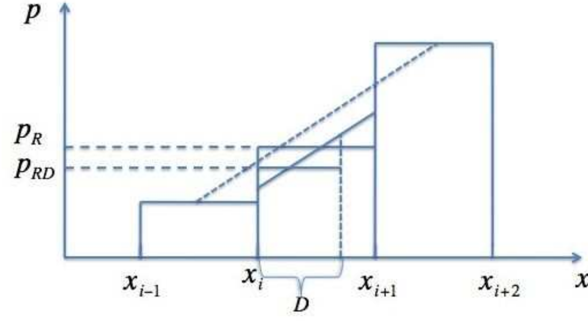

Figure 4: The slope of $p(x)$ in the cell $x_i < x < x_{i+1}$ is determined by the values at the two neighboring cells, as shown by a dashed line. Since only the state in the region indicated by D can influence the state at x_i through the sound wave traveling to the left during $\triangle t$, the actual right state is the averaged-state in the region D, p_{rd} that is called domain-average, but not p_r.

become

$$\overline{p} - p_{ld} + \rho_l(c_{sl} + u_{xl})(\overline{u}_x - u_{xld}) = 0, \tag{19}$$

$$\overline{p} - p_{rd} - \rho_r(c_{sr} - u_{xr})(\overline{u}_x - u_{xrd}) = 0. \tag{20}$$

To avoid negative values of $p(x)$ within the internal structure within a grid cell, we apply a monotone condition after we find the slope through the central difference, as shown in Fig.5. The simple central difference gives us the slope shown by a dashed line. Since the value of $p(x)$ at x_i is lower than the averaged value of the left cell, the slope is adjusted to the form as shown by the solid line in the figure. The same montone condition is applied to a few other similar situations to adjust the slopes obtained by the central difference.

After obtaining the time-averaged values for u_x and p at grid interfaces, we may explicitly find the time-averaged value of ρ at the grid point, $\overline{\rho}_i$, through the following equation

$$\overline{p} - p_d - c_s^2(\overline{\rho} - \rho_d) = 0. \tag{21}$$

Here, the subscript d denotes the domain-average over the domain between x_i and $(x_i + \overline{u}_x \triangle t)$. A similar technique applies to u_y and u_z.

Through this explicit treatment for flow signals, we may obtain the time-averaged values $\overline{\rho}_i$, $\overline{p}_i$, $\overline{u}_x$, $\overline{u}_y$, and $\overline{u}_z$. Using the time-averaged values $\overline{\mathbf{U}}$, we may obtain the cell-averaged

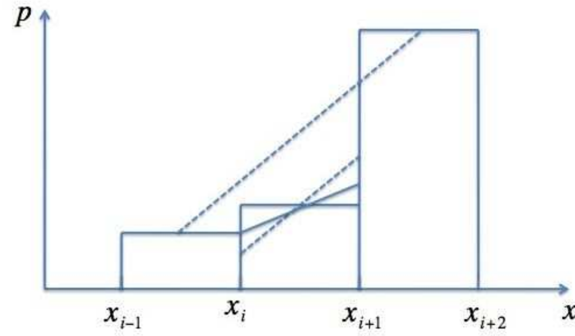

Figure 5: The illustration of monotone condition. The dashed line in the cell $x_i < x < x_{i+1}$ is the slope obtained from the central difference of values in two neighboring cells. Since the interpolated value of p at x_i exceeds the range defined on the cells (x_{i-1}, x_i) and (x_i, x_{i+1}), the slope is flattened to fit into the range as the solid line shows.

value $\mathbf{U}_i(\triangle t)$ through Eq.(13).

Although there are no shock waves present during the forest wildfires, the distribution of air density could experience dramatic spatial changes. Equations(19,20) do not allow dramatic changes in u_x and p, they allow dramatic jumps in ρ between grid cells. Therefore, Equations (19,20) are appropriate for our calaculations. To verify this, we have implemented more sophisticated calculations [1,6] in our computer code for the time-averaged flux in our computer code. Through comparison in simulation results, we conclude that more sophisticated treatment for the time-averaged flux than Eqs.(19,20) is not necessary for our case, and Eqs.(19,20) are accurate enough for our calculations.

4.2 Explicit Treatment for Heat Conduction

In this project, as a preliminary study, the heat conduction in Eqs.(5,11), or Eqs.(6,12), is explicitly treated. The temperature at the previous time step is used to calculate the heat flux. Therefore, the treatment for the heat conduction is first-order accurate in time.

4.3 Numerical Schemes for Multi-dimensional Flows

The multi-dimensional Eq.(6) is solved through a dimensionally split technique [4]. Each time step of a multi-dimensional problem is broken down into one-dimensional passes in which derivatives in other dimensions are set to zero.

The solution of two-dimensional flows may be written in the following form,

$$\mathbf{U}_{i,j}(2\triangle t) = L^x_{\triangle t} L^y_{\triangle t} L^y_{\triangle t} L^x_{\triangle t} \mathbf{U}_{i,j}(0). \tag{22}$$

Here $L^x_{\triangle t}$ is the one-dimensional operator in the x-direction with the time step $\triangle t$, which is described before. If the one-dimensional operator is second order accurate, then the scheme constructed in Eq.(22) is second order accurate at every other time step.

Similarly, the solution of three-dimensional flows may be written as

$$\mathbf{U}_{i,j,k}(2\triangle t) = L^x_{\triangle t} L^y_{\triangle t} L^z_{\triangle t} L^z_{\triangle t} L^y_{\triangle t} L^x_{\triangle t} \mathbf{U}_{i,j,k}(0). \tag{23}$$

In Eq.(23) the y-dimension is treated differently from the other two dimensions. For our simulations of forest fires, $x-$ and $y-$dimensions are used for two horizontal directions, and $z-$dimension will be for the vertical direction. The two horizontal directions should be treated in the exactly same way. Therefore, in our simulations for forest fires, we will use the following formula,

$$\mathbf{U}_{i,j,k}(2\triangle t) = L^x_{\triangle t} L^z_{\triangle t} L^y_{\triangle t} L^y_{\triangle t} L^z_{\triangle t} L^x_{\triangle t} \mathbf{U}_{i,j,k}(0). \tag{24}$$

4.4 Time Step Control

Since explicit numerical schemes are used in this project, the size of time step in our simulations is controlled by the local speed of sound wave in the moving gas and the width of the cell,

$$\frac{\triangle t(c_s + |u_i|)}{\triangle l} < 1. \tag{25}$$

Here $\triangle l$ is the minimum width of any cell in three dimensions, u_i is the flow velocity in that direction, and c_s is the local sound speed. The physical meaning of Eq.(25) is that the sound signals can not move more than the width of a cell in one time step. The left side

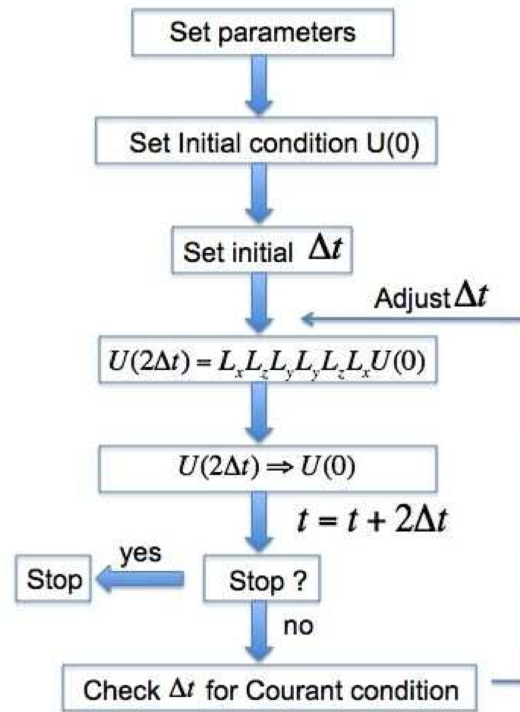

Figure 6: The illustration for the procedure used in simulations.

of Eq.(25) is called Courant number of a simulation. If the condition, Eq.(25), is violated, the numerical scheme discussed above will be mathematically unstable. The inequality in Eq.(25) is often called ”Courant condition”, and the maximum value of the left side is called ”Courant number”.

4.5 Simulation Procedure

Fig.6 shows the flow chart of a simulation. In our simulations, we set the safety factor 0.8. After each pair of time steps, we check the value of the left side of Eq.(25). If it is smaller than 0.8, we increase the time step appropriately for the next two time step. If it is larger than 0,8, we reduce the time step by an appropriate amount.

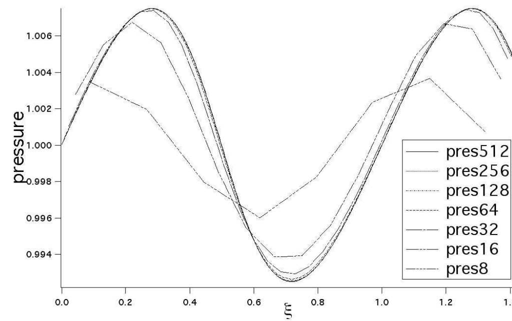

Figure 7: The profiles of pressure along the diagonal direction ξ of a two-dimensional square domain after a sound wave propagates five wavelengths in the diagonal direction. The profiles are obtained in seven simulations with resolutions 8×8, 16×16, 32×32, 64×64, 128×128, 256×256, and 512×512.

5 Numerical Results

In this section we will present some preliminary results from the numerical scheme described in the last section.

5.1 Convergence and Accuracy

Theoretically, the numerical scheme described in the last section for gas dynamics is second order accurate in both space and time for one and multi-dimensional problems. To test the accuracy, we set a single sound wave propagating in the diagonal direction of a two-dimensional square domain. We simulate the propagation of the wave through different spatial and temporal resolutions. For this set of simulations, the periodic boundary condition is used, and the courant number is set to 0.8.

Fig.7 shows the pressure along the diagonal direction after the wave propagates five wavelengths with seven different grid resolutions, 8×8, 16×16, 32×32, 64×64, 128×128, 256×256, and 512×512. Although we don't know the exact solution of this problem due to the nonlinearity, the figure clearly shows how the solutions with different resolutions approach to the one with the grid resolution 512×512. If we consider the solution with the grid of 512x512 the exact solution, the error between a numerical solution and the exact solution decreases nonlinearly with the size of grid cells.

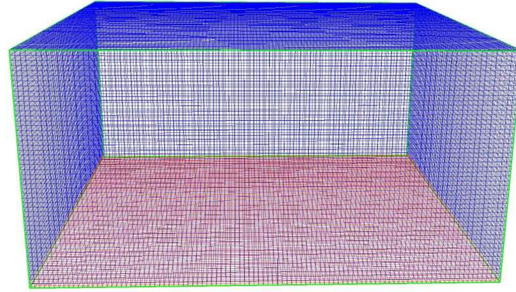

Figure 8: The mesh used in simulations for forest fires. The resolution is $100x100x50$. Only on the bottom layer of the mesh, the energy source is not zero.

5.2 Parameters in Simulations of Forest Fires

For the simulations presented in this report, the simulation domain is a box with $20km \times 20km \times 10km$. As stated before, the forest terrains, which could have significant impact on forest fires, are ignored in our calculations. The density function $s(x, y, z)$ for forest trees and oxygen is assumed to be uniformly distributed at the bottom of simulation box, as shown by the red color in Fig.8. The value of s is $2MJ/(m^3min)$. We assume that the igniting temperature of forest trees is 492^oC.

Initially, the air density is $1kg/m^3$, air pressure is 1 atm, and air velocity in zero inside the simulation box. There will be wind (air motion) generated even if there is no wind initially and there is no wind from simulation boundaries.

5.3 Development of Forest Fires

In this subsection, we will present preliminary simulation results for a forest fire. Open boundary conditions are used at all the sides of the simulation domain except for the ground, where reflection boundary condition is used.

Initially, there is a fire localized at the center on the ground, the highest temperature of the initial fire is 887^oC at its center. The initial fire is shown in the first image in Fig.9, where the "semi-sphere" is an iso-surface of temperature at the value 492^oC at which woods will

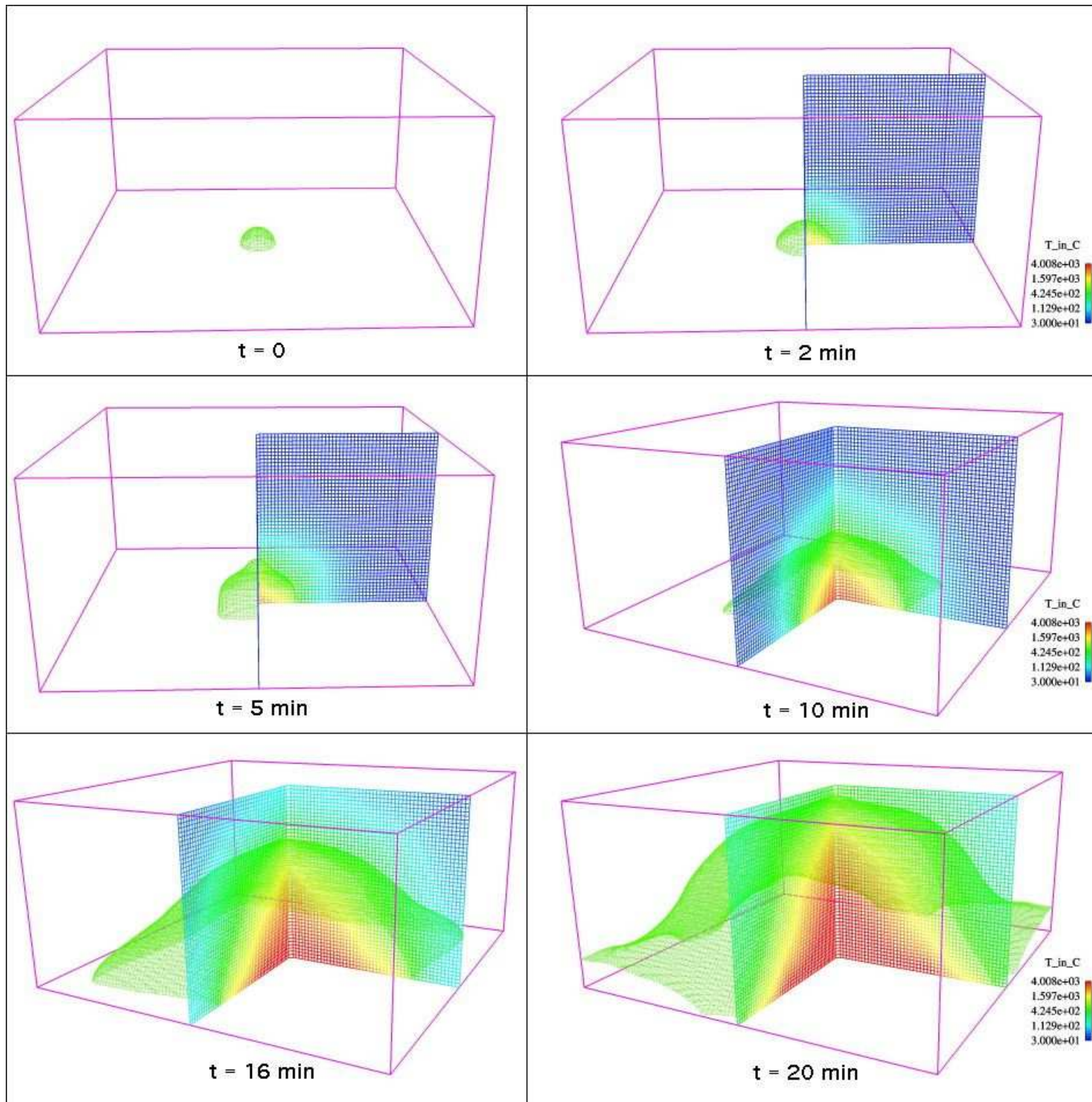

Figure 9: The distribution of temperature on two cutting faces and iso-surfaces of temperature at $T = 492^oC$ at six instants.

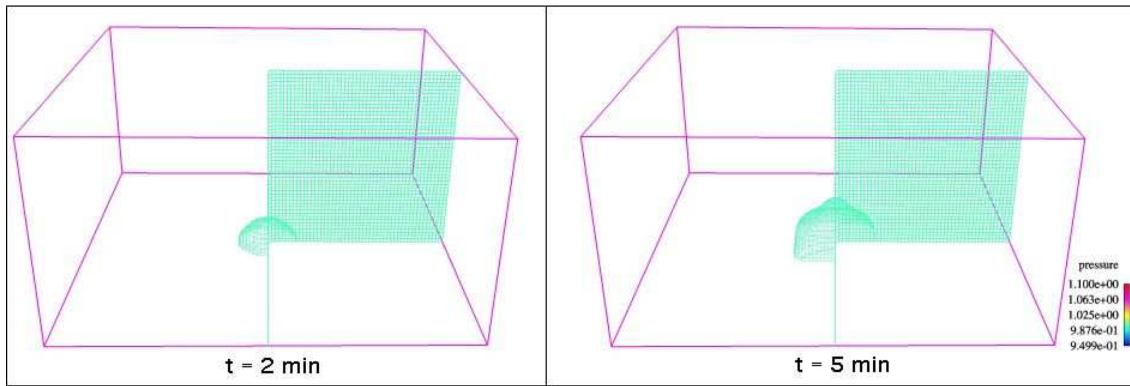

Figure 10: The pressure distributions on two cutting faces and temperature iso-surfaces at two instants.

start burning.

The remaining images in Figs.9 show the distributions of temperature after 2, 5, 10, 16, and 20 minutes. The surfaces in the images are the temperature iso-surfaces at 492^oC, the igniting temperature of woods. The "square-like" shape of the fire front is the footprint of the rectangular simulation domain. Sound waves have been bounced back and fourth from the simulation boundaries a few times every minute. This footprint suggests that it will be better to use the cylindrical coordinate in numerical simulations for forest fires.

Fig.10 shows the air pressure on the temperature iso-surface and distribution on two cutting faces after t = 2 min and 5 min. The pressure is in the unit of atm. It is clear that the distrbution of pressure is quite smooth. Fig.11 shows the air density at the temperature iso-surface and on the two cutting surfaces at t = 2 min, 5 min, 16 min, and 20 min. The unit of air density is $1kg/m^3$. This figure indicates that air density experiences dramatic changes across the region. Fig.12 displays the vertical component of air velocity at t = 2 min, 5 min, 16 min, and 20 min. As shown in the figure, winds are generated through the fire even if there is no winds coming in through the simulation boundaries. The unit of the velocity is $6\sqrt{10}km/min \approx 20km/min$.

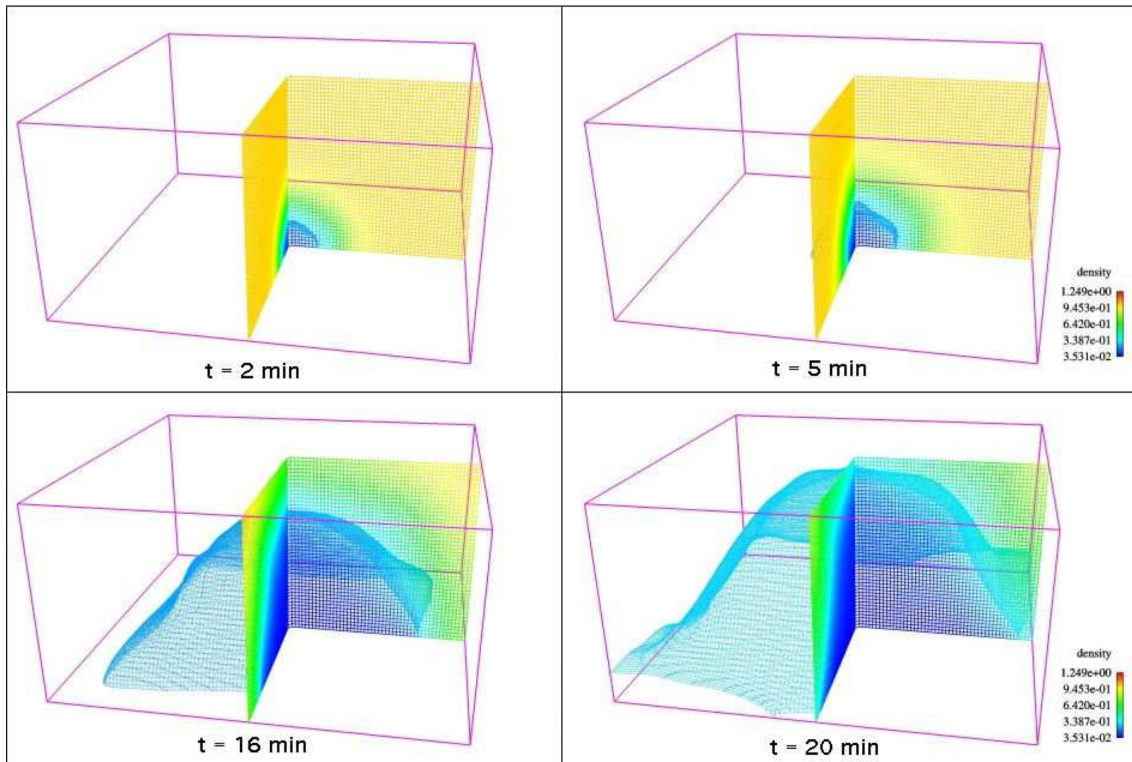

Figure 11: The air density distribution on two cutting surfaces and temperature iso-surfaces at four instants.

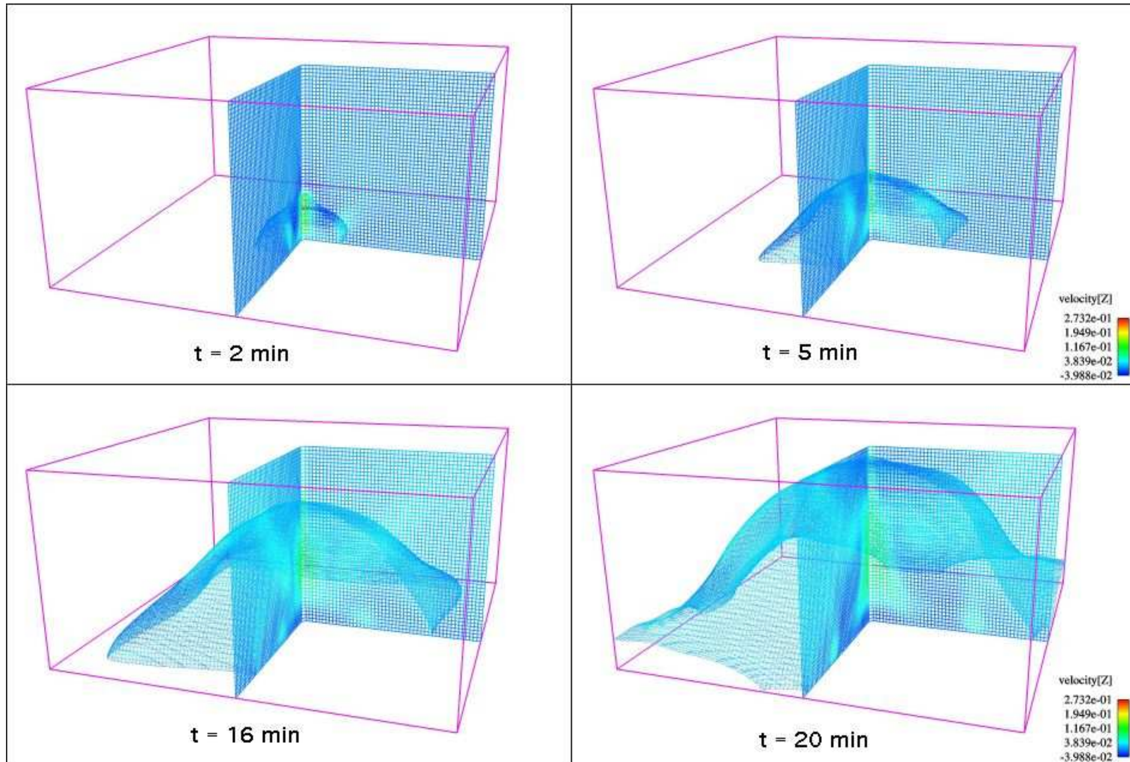

Figure 12: The vertical component of air velocity on two cutting faces and temperature iso-surfaces at four instants. The velocity is normalized by $6\sqrt{10}km/min$.

6 Conclusions and Discussions

In this project, we have proposed a simplified model for the development of forest wildfires. As the result of the model, the development of forest fires could be described by the set of equations for gas dynamics with energy source.

We have modified modern numerical techniques for gas dynamics and developed a three-dimensional computer code for the development of the fires. The algorithm for gas dynamics is second order accurate in both space and time. The scheme for heat conduction is currently first-order accurate in time.

We have obtained preliminary numerical results for the development of forest fires, through which We have demonstrated that the computer code could be used to simulate three-dimensional forest fires.

There are several aspects the model could be further improved in future. First, more realistic non-uniform density function could be included for non-uniform tree distributions in forests. Second the effect of wetness could be included, so that the tree will not immediately start burning after getting to igniting temperature. Third, the radiation effect of fires could be included.

The numerical algorithm used in this project could be further improved. For heat conduction, we could use modern techniques, for example, multi-grid method, which is second order accurate in both space and time. We should also include the cylindrical coordinate system in our computer code.

The computer code could be extended to be parallel. The parallel feature of a computer code will significantly increase the size and speed of simulations.

Finally, we plan to use the parameters in an actual fire, for example, Los Alamos 2000 Cerro Grande Fire to see how this simple model works.

7 Acknowledgments

We are very grateful to our parents and teachers for their full support, specifically as the role of mentors. We also thank Eric Dai, who is a high school student intern at Los Alamos National Laboratory, for his help in producing the images presented in this report through the graphics software called EnSight.

References

[1] S.K. Godunov., *Difference methods for the numerical calculation of the equations of fluid dynamics*, *Math Sb.*, **47**, 271, 1959.

[2] L.D. Landau and E.M. Lifshitz, *Fluid Mechanics*, Pergamon Press, New York, 1959.

[3] R. Courant and K. O. Friedrichs, *Supersonic Flow and Shock Wave*, 5th ed., Interscience, New York, 1967.

[4] G. Strang, *On construction and comparison of differential schemes*, *SIAM J. Numer. Anal.* **5**, 506, 1968.

[5] B. Van Leer, *Toward the ultimate conservative difference scheme. V. A second-order sequel to Godunov method*, *J. Comput. Physics*, **32**, 101, 1979.

[6] P.R. Woodward and P. Colella, *High resolution difference schemes for compressible gas dynamics*, in *Lecture Notes in Physics*, **141**, 434, 1981, Springer-Verlag, New York.

[7] M. E. Alexander, *Estimating the length-to-breadth ratio of elliptical forest fire patterns*, *Proc. of the 8th Conference on Fire and Forest Meteorology*, 287-304, 1985.

[8] G.D. Richards, *Numerical simulation of forest fires*, *Int. J. Numerical Meth. Eng*, **25**, 625, 1988.

[9] G.D. Richards, *An elliptical growth model of forest fire fronts and its numerical solution*, *Int. J. Numerical Meth. Eng*, **30**, 1163, 1990.

[10] E.A. Johnson and K. Miayanishi, *Forest Fires: Behavior and Ecological Effects*, Academic Press, 2001.

A Source Codes

```
/** File defs.h    **/

#ifndef FF_DEFINES_H
#define FF_DEFINES_H

#ifdef __cplusplus
extern "C" {
#endif

#ifdef DOUBLE
#define ifdouble  1
#else
#define ifdouble  0
#endif

#ifdef LLINT
#define ifllong   1
#else
#define ifllong   0
#endif

#if ifdouble == 1
#define real double
#else
#define real float
#endif

#if ifllong == 1
#define llint long long
#define myabs llabs
#else
#define llint int
#define myabs abs
#endif

#if ifdouble == 1
#define myllrint llround
#define myexp  exp
#define myfmax fmax
#define myfmin fmin
#define myfabs fabs
#define mycos  cos
#define mysin  sin
#define myacos acos
#define myasin asin
#define mysqrt sqrt
#define mytanh tanh
#define mypow  pow
#else
```

```
#define myllrint llroundf
#define myexp expf
#define myfmax fmaxf
#define myfmin fminf
#define myfabs fabsf
#define mycos  cosf
#define mysin  sinf
#define myacos acosf
#define myasin asinf
#define mysqrt sqrtf
#define mytanh tanhf
#define mypow  powf
#endif

#define mymax(x,y) (((x) >= (y)) ? (x):(y))
#define mymin(x,y) (((x) <= (y)) ? (x):(y))

#define GET_2D_FORM(type, rho, rho2d, k, sizes)          \
{                                                        \
    rho2d = (type **) malloc(sizes[1] * sizeof(type *)); \
    rho2d[0] = rho;                                      \
    for (k = 1; k < sizes[1]; k++) {                     \
        rho2d[k] = rho2d[k-1] + sizes[0];                \
    }                                                    \
}

/* to free, free(rho3d[0]; free(rho3d)   */

#define GET_3D_FORM(type, rho, rho3d, j, k, sizes, offset, rho2d)    \
{                                                                    \
    rho3d = (type ***)malloc(sizes[2] * sizeof(type **));            \
    rho2d = (type **) malloc(sizes[1] * sizes[2] * sizeof(type *));  \
    offset = 0;                                                      \
    for (k = 0; k < sizes[2]; k++) {                                 \
        rho2d[0] = rho + offset;                                     \
        for (j = 1; j < sizes[1]; j++) {                             \
            rho2d[j] = rho2d[j-1] + sizes[0];                        \
        }                                                            \
        rho3d[k] = rho2d;                                            \
        offset += (sizes[0] * sizes[1]);                             \
        rho2d  += sizes[1];                                          \
    }                                                                \
}

#ifdef __cplusplus
}
#endif

#endif

/** File consts.h    **/

#ifndef FF_CONSTS_H
```

```
#define FF_CONSTS_H

#ifdef __cplusplus
extern "C" {
#endif

int dim = 3;
int ff_nbdy = 3;

int npes = 1;
int mype = 0;

int ff_next_id = 0;
ff_Domain *ff_domains = NULL;

int  ff_nstop = 1000000;
real ff_tstop = 30.0;
real ff_dtstart = 0.0002;
int  ff_dt_fixed = 0;
real ff_safety = 0.8;
int  ff_ndump = 10000000;
real ff_dtdump = 1.0/10.0;

int ff_idump = 0;

real ff_rho0  = 1.0;    /* 1 kg/ m^3 */
real ff_rho1  = 0.0;    /* to be calculated */
real ff_e1i0  = 2.61;   /* 30 C,  T = 116 * ei - 273 */
real ff_p0  = 0.0;      /* to be calculated */
real ff_ei1 = 10.0;     /* 887 C */
real ff_vx0 = 0.000;
real ff_vy0 = 0.000;
real ff_vz0 = 0.0;

real ff_flowin_vx0 = 0.00;
real ff_flowin_vy0 = 0.00;

// sizes of simulation domain
real ff_sz_z  = 0.5;            /*                */
real ff_sz_xy = 1.0;            /*  about 20 km   */
real ff_forest_height = 0.01;
real ff_z0 = 0.00;              /*  location vx and vy turn to vx0 from 0 at bottom */
real ff_width = 0.01;

llint ff_gsize_z  = 50;
llint ff_gsize_xy = 100;

real ff_gap_width = 0.0000001;
real ff_gap_disp = 5.0;   // 0.5 * ff_sz_xy;

real ff_forest_energy_src = 20.0;
real ff_igniting_e  = 6.6;               /* 492 C degree */
real ff_forest_max_dt_burning = 400.0;   /* 120/2 = 60 min */
```

```
real ff_heat_conductivity = 0.001;

real gm = 5.0/3.0;

real small = 1.0e-30;
real ff_dt_smallest = 1.0e-06;

#ifdef __cplusplus
}
#endif

#endif

/*** File ff.h   ***/

#ifndef FF_H
#define FF_H

#ifdef __cplusplus
extern "C" {
#endif

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 *****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <assert.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/file.h>
#include <sys/types.h>

#include "defs.h"

#define MAX_VAR_RANK  3

enum ff_Data_Type {
     ff_char             = 1,
     ff_double           = 2,
     ff_float            = 3,
     ff_int              = 4,
     ff_long             = 5,
     ff_long_long        = 6,
     ff_datatype_invalid = 10
};
typedef enum ff_Data_Type ff_Data_Type;

enum ff_Var_Type {
     ff_zone = 1,
```

```
     ff_face = 2,
     ff_vartype_invalid  = 10
};
typedef enum ff_Var_Type ff_Var_Type;

struct ff_Var_Comps {
       int which_comp[MAX_VAR_RANK];
       void *buffer;
};
typedef struct ff_Var_Comps ff_Var_Comps;

struct ff_Mesh_Var {
       char *name;
       ff_Var_Type type;
       int  rank;
       ff_Data_Type datatype;
       int comp_sizes[MAX_VAR_RANK];
       ff_Var_Comps comps[MAX_VAR_RANK*MAX_VAR_RANK*MAX_VAR_RANK];
  };
typedef struct ff_Mesh_Var ff_Mesh_Var;

enum ff_Bdy_Type {
         ff_periodic   = 1,
         ff_continued  = 2,
         ff_reflected  = 3,
         ff_flow_in  = 4,
         ff_flow_out = 5,
         ff_open       = 6,
         ff_fixed      = 7,
         ff_not_bdy    = 20
      };
typedef enum ff_Bdy_Type ff_Bdy_Type;

struct ff_Bdy_Condition {
       ff_Bdy_Type typel[3];
       ff_Bdy_Type typer[3];
};
typedef struct ff_Bdy_Condition ff_Bdy_Condition;

struct ff_Domain;

struct ff_Domain {
       int  id;
       int  owner;
       int  nbdy;
       llint sizes[3];
       real coordl[3];
       real coordr[3];
       ff_Bdy_Condition *bdy;
       int nvar;
       ff_Mesh_Var *vars;

       real ***rho3d;
```

```
        real ***p3d;
        real ***ux3d;
        real ***uy3d;
        real ***uz3d;

        real ***src3d;
        real ***dt_heated3d;
        real ***dt_burned3d;

        real **rho2d;    /**< alias for variale density */
        real **p2d;
        real **ux2d;
        real **uy2d;
        real **uz2d;

        real **src2d;
        real **dt_heated2d;
        real **dt_burned2d;
};
typedef struct ff_Domain ff_Domain;

#ifdef __cplusplus
}
#endif
#endif

/*** file main.c ***/

#include "ff.h"
#include "consts.h"
#include "files.h"

int main(int argc, char **argv)
{
   int k, ndomain, too_fine;

   int ndomain_d[3] = {1, 1, 1};
   llint gsizes[] = {ff_gsize_xy, ff_gsize_xy, ff_gsize_z};
   real gcoordl[] = {0, 0, 0};
   real gcoordr[] = {ff_sz_xy, ff_sz_xy, ff_sz_z};

   int nchild;
   llint nelem;
   real **coords;

   ff_Bdy_Type bdy_typel[] = {ff_flow_in, ff_open, ff_reflected};
   ff_Bdy_Type bdy_typer[] = {ff_open, ff_open, ff_open};
   ff_Domain *domains, *domain, *children;

    nchild = ndomain_d[0];
    for (k = 1; k < dim; k++) {
        nchild *= ndomain_d[k];
    }
```

```
    domains = NULL;
    init_mesh_var_hydro(gcoordl, gcoordr, gsizes, ndomain_d,
                        bdy_typel, bdy_typer, &domains, &ndomain);

    side_bdy(ndomain, domains, NULL);
    ztop_bdy(ndomain, domains, NULL);
    zbot_bdy(ndomain, domains, NULL);

    io_for_debug("data_dbg", ff_idump, ndomain, domains);
    ff_idump++;

    control(gcoordl, gcoordr, gsizes, ndomain, domains);

#ifdef MPI
    MPI_Finalize();
#endif

    return 0;
}

int control(real *gcoordl, real *gcoordr,
            llint *gsizes, int ndomain, ff_Domain *domains)
{
    int  k, ncycle, idump, ntodump, ntorefine, ntdump, ntrefine, order, nchild;
    real tdump, trefine, t, dtime,courno, cournmx, ddtime;
    real *courno_pe;
    ff_Domain *domain;

    extern int  ff_next_id;
    extern real ff_dt_smallest;

    if (npes > 1) {
        courno_pe = (real *) malloc(npes * sizeof(real));
    }
    else {
        courno_pe = &courno;
    }
    order = 0;
    ncycle = 0;
    idump = 0;;
    tdump = 0.0;
    ntodump = 0;

    t = 0.0;
    dtime = ff_dtstart;
    while ((t < ff_tstop) && (ncycle < ff_nstop)) {

        courno = 0.0;
        run_hydro(ndomain, domains, order, t, dtime, &courno);

        order = 1 - order;

        cournmx = courno;
```

```
        t += dtime;
        ncycle++;

        side_bdy(ndomain, domains, NULL);
        ztop_bdy(ndomain, domains, NULL);
        zbot_bdy(ndomain, domains, NULL);

        courno = 0.0;
        run_hydro(ndomain, domains, order, t, dtime, &courno);
        cournmx = mymax(cournmx, courno);
        t += dtime;
        ncycle++;
        tdump += (dtime + dtime);
        ntodump += 2;

        printf("ncycle = %5d, time = %12.5e, dt = %12.5e, cournmx = %12.5e\n",
                    ncycle, t, dtime, cournmx);
        if ((t < ff_tstop) && (ncycle < ff_nstop)) {
            side_bdy(ndomain, domains, NULL);
            ztop_bdy(ndomain, domains, NULL);
            zbot_bdy(ndomain, domains, NULL);

            if ((tdump >= ff_dtdump) || (ntodump >= ff_ndump)) {

               io_for_debug("data_dbg", ff_idump, ndomain, domains);
               ff_idump++;

                idump++;
                ntdump = (int)(t / ff_dtdump);
                tdump  = t - (float) ntdump * ff_dtdump;
                ntodump = 0;
            }
            ddtime = (ff_safety / cournmx - 1.0) * dtime;
            ddtime = myfmin (0.1 * ddtime, (ddtime + ddtime));

            dtime += ddtime;
            if (dtime < ff_dt_smallest) {
                printf("ERROR: dtime = %e <  ff_ds_smallest\n", dtime);
                return -1;
            }
        }           /* if ((t < ff_tstop) && (ncycle < ff_nstop))  */
    }               /* while  */
    return 0;
 }

int side_bdy(int ndomain, ff_Domain *domains, ff_Domain **pdomains)
{
    int b, d, nbdy;
    llint k, k0, km, kp, j, j0, jm, jp, i, i0, im, ip;
    llint *sizes, sizes_ext[3];
    real ***rho3d, ***p3d, ***ux3d, ***uy3d, ***uz3d;
    ff_Domain *domain;
    ff_Bdy_Condition *bdy;
```

```
ff_Bdy_Type *typel, *typer;
ff_Bdy_Type type_null[] = {ff_not_bdy, ff_not_bdy, ff_not_bdy};

extern int dim;

if (dim != 3) return 0;

for (b = 0; b < ndomain; b++) {
    domain = domains + b;
    bdy = domain->bdy;
    if (bdy) {
        typel = bdy->typel;
        typer = bdy->typer;
    }
    else {
        typel = type_null;
        typer = type_null;
    }
    rho3d = domain->rho3d;
    p3d   = domain->p3d;
    ux3d  = domain->ux3d;
    uy3d  = domain->uy3d;
    uz3d  = domain->uz3d;
    nbdy  = domain->nbdy;
    sizes = domain->sizes;
    for (d = 0; d < dim; d++) {
        sizes_ext[d] = sizes[d] + nbdy + nbdy;
    }
    if (typel[0] == ff_open) {
        for (k = nbdy; k < sizes[2] + nbdy; k++) {
            for (j = nbdy; j < sizes[1] + nbdy; j++) {
                for (i = 0; i < nbdy; i++) {
                    i0 = nbdy - i;
                    im = i0 - 1;
                    ip = i0 + 1;
                    rho3d[k][j][im] = rho3d[k][j][i0];
                      p3d[k][j][im] =   p3d[k][j][i0];
                     ux3d[k][j][im] =  ux3d[k][j][i0];
                     uy3d[k][j][im] =  uy3d[k][j][i0];
                     uz3d[k][j][im] =  uz3d[k][j][i0];
                }
            }
        }
    }
    else if (typel[0] == ff_flow_in) {
        for (k = nbdy; k < sizes[2] + nbdy; k++) {
            for (j = nbdy; j < sizes[1] + nbdy; j++) {
                for (i = 0; i < nbdy; i++) {
                    i0 = nbdy - i;
                    im = i0 - 1;
                    ip = i0 + 1;
                    rho3d[k][j][im] =  ff_rho0;
                      p3d[k][j][im] =  ff_p0;
```

```
                    ux3d[k][j][im] =  ff_flowin_vx0;
                    uy3d[k][j][im] =  ff_flowin_vy0;
                    uz3d[k][j][im] =  0.0;
                }
            }
        }
    }
    if (typer[0] == ff_open) {
        for (k = nbdy; k < sizes[2] + nbdy; k++) {
            for (j = nbdy; j < sizes[1] + nbdy; j++) {
                for (i = 0; i < nbdy; i++) {
                    i0 = nbdy + sizes[0] - 1 + i;
                    im = i0 - 1;
                    ip = i0 + 1;
                    rho3d[k][j][ip] = rho3d[k][j][i0];
                      p3d[k][j][ip] =   p3d[k][j][i0];
                     ux3d[k][j][ip] =  ux3d[k][j][i0];
                     uy3d[k][j][ip] =  uy3d[k][j][i0];
                     uz3d[k][j][ip] =  uz3d[k][j][i0];
                }
            }
        }
    }
    else if (typer[0] == ff_flow_in) {
        for (k = nbdy; k < sizes[2] + nbdy; k++) {
            for (j = nbdy; j < sizes[1] + nbdy; j++) {
                for (i = 0; i < nbdy; i++) {
                    i0 = nbdy + sizes[0] - 1 + i;
                    im = i0 - 1;
                    ip = i0 + 1;
                    rho3d[k][j][ip] = ff_rho0;
                      p3d[k][j][ip] = ff_p0;
                     ux3d[k][j][ip] = -ff_flowin_vx0;
                     uy3d[k][j][ip] =  ff_flowin_vy0;
                     uz3d[k][j][ip] =  0.0;
                }
            }
        }
    }
    if (typel[1] == ff_open) {
        for (k = nbdy; k < sizes[2] + nbdy; k++) {
            for (j = 0; j < nbdy; j++) {
                j0 = nbdy - j;
                jm = j0 - 1;
                jp = j0 + 1;
                for (i = 0; i < sizes_ext[0]; i++) {
                    rho3d[k][jm][i] = rho3d[k][j0][i];
                      p3d[k][jm][i] =   p3d[k][j0][i];
                     ux3d[k][jm][i] =  ux3d[k][j0][i];
                     uy3d[k][jm][i] =  uy3d[k][j0][i];
                     uz3d[k][jm][i] =  uz3d[k][j0][i];
                }
            }
```

```
            }
        }
        else if (typel[1] == ff_flow_in) {
            for (k = nbdy; k < sizes[2] + nbdy; k++) {
                for (j = 0; j < nbdy; j++) {
                    j0 = nbdy - j;
                    jm = j0 - 1;
                    jp = j0 + 1;
                    for (i = 0; i < sizes_ext[0]; i++) {
                        rho3d[k][jm][i] = ff_rho0;
                          p3d[k][jm][i] = ff_p0;
                         ux3d[k][jm][i] = ff_flowin_vx0;
                         uy3d[k][jm][i] = ff_flowin_vy0;
                         uz3d[k][jm][i] = 0.0;
                    }
                }
            }
        }
        if (typer[1] == ff_open) {
            for (k = nbdy; k < sizes[2] + nbdy; k++) {
                for (j = 0; j < nbdy; j++) {
                    j0 = nbdy + sizes[1] - 1 + j;
                    jm = j0 - 1;
                   jp = j0 + 1;
                    for (i = 0; i < sizes_ext[0]; i++) {
                        rho3d[k][jp][i] = rho3d[k][j0][i];
                          p3d[k][jp][i] =   p3d[k][j0][i];
                         ux3d[k][jp][i] =  ux3d[k][j0][i];
                         uy3d[k][jp][i] =  uy3d[k][j0][i];
                         uz3d[k][jp][i] =  uz3d[k][j0][i];
                    }
                }
            }
        }
        else if (typer[1] == ff_flow_in) {
            for (k = nbdy; k < sizes[2] + nbdy; k++) {
                for (j = 0; j < nbdy; j++) {
                    j0 = nbdy + sizes[1] - 1 + j;
                    jm = j0 - 1;
                    jp = j0 + 1;
                    for (i = 0; i < sizes_ext[0]; i++) {
                        rho3d[k][jp][i] = ff_rho0;
                          p3d[k][jp][i] = ff_p0;
                         ux3d[k][jp][i] = ff_flowin_vx0;
                         uy3d[k][jp][i] = -ff_flowin_vy0;
                         uz3d[k][jp][i] =  0.0;
                    }
                }
            }
        }
    }
    return 0;
}
```

```
int ztop_bdy(int ndomain, ff_Domain *domains, ff_Domain **pdomains)
{
    int b, d, nbdy;
    llint k, k0, km, kp, j, i;
    llint *sizes, sizes_ext[3];
    real ***rho3d, ***p3d, ***ux3d, ***uy3d, ***uz3d;
    ff_Domain *domain;
    ff_Bdy_Condition *bdy;
    ff_Bdy_Type *typel, *typer;
    ff_Bdy_Type type_null[] = {ff_not_bdy, ff_not_bdy, ff_not_bdy};

    extern int dim;

    if (dim != 3) return 0;

    for (b = 0; b < ndomain; b++) {
        domain = domains + b;
        bdy = domain->bdy;
        if (bdy) {
            typel = bdy->typel;
            typer = bdy->typer;
        }
        else {
            typel = type_null;
            typer = type_null;
        }
        if (typer[2] != ff_open) continue;

        rho3d = domain->rho3d;
        p3d   = domain->p3d;
        ux3d  = domain->ux3d;
        uy3d  = domain->uy3d;
        uz3d  = domain->uz3d;
        nbdy  = domain->nbdy;
        sizes = domain->sizes;
        for (d = 0; d < dim; d++) {
            sizes_ext[d] = sizes[d] + nbdy + nbdy;
        }
        for (k = 0; k < nbdy; k++) {
            k0 = nbdy + sizes[2] - 1 + k;
            km = k0 - 1;
            kp = k0 + 1;
            for (j = 0; j < sizes_ext[1]; j++) {
                for (i = 0; i < sizes_ext[0]; i++) {
                    rho3d[kp][j][i] = rho3d[k0][j][i];
                      p3d[kp][j][i] =   p3d[k0][j][i];
                     ux3d[kp][j][i] =  ux3d[k0][j][i];
                     uy3d[kp][j][i] =  uy3d[k0][j][i];
                     uz3d[kp][j][i] =  uz3d[k0][j][i];
                }
            }
        }
    }
```

```
    return 0;
 }

int zbot_bdy(int ndomain, ff_Domain *domains, ff_Domain **pdomains)
{
    int b, d, nbdy;
    llint k, k1, j, i;
    llint *sizes, sizes_ext[3];
    real ***rho3d, ***p3d, ***ux3d, ***uy3d, ***uz3d;
    ff_Domain *domain;
    ff_Bdy_Condition *bdy;
    ff_Bdy_Type *typel, *typer;
    ff_Bdy_Type type_null[] = {ff_not_bdy, ff_not_bdy, ff_not_bdy};

    extern int dim;

    if (dim != 3) return 0;

    for (b = 0; b < ndomain; b++) {
        domain = domains + b;
        bdy = domain->bdy;
        if (bdy) {
            typel = bdy->typel;
            typer = bdy->typer;
        }
        else {
            typel = type_null;
            typer = type_null;
        }
        if (typel[2] != ff_reflected) continue;

        rho3d = domain->rho3d;
        p3d   = domain->p3d;
        ux3d  = domain->ux3d;
        uy3d  = domain->uy3d;
        uz3d  = domain->uz3d;
        nbdy  = domain->nbdy;
        sizes = domain->sizes;
        for (d = 0; d < dim; d++) {
            sizes_ext[d] = sizes[d] + nbdy + nbdy;
        }
        for (k = 0; k < nbdy; k++) {
            k1 = nbdy + nbdy - 1 - k;
            for (j = 0; j < sizes_ext[1]; j++) {
                for (i = 0; i < sizes_ext[0]; i++) {
                    rho3d[k][j][i] = rho3d[nbdy][j][i];
                      p3d[k][j][i] =   p3d[nbdy][j][i];
                     ux3d[k][j][i] =  ux3d[k1][j][i];
                     uy3d[k][j][i] =  uy3d[k1][j][i];
                     uz3d[k][j][i] = -uz3d[k1][j][i];
                }
            }
        }
```

```
    }
    return 0;
 }

/*** file init.c ***/

#include "ff.h"
#include "files.h"

int init_mesh_var_hydro(real *gcoordl, real *gcoordr, llint *gsizes,
                        int *ndomain_d,
                        ff_Bdy_Type *bdy_typel, ff_Bdy_Type *bdy_typer,
                        ff_Domain **domains, int *ndomain)
{
    int put_layer, k, nc;
    llint i1, i2, ndomain_tot;
    ff_Domain *domain, **new_children, **children;

    extern int dim, mype, ifamr;
    extern ff_Domain *ff_domains;
    extern real small;
    extern int ff_next_id;

    init_mesh(gcoordl, gcoordr, gsizes, ndomain_d,
              bdy_typel, bdy_typer, domains, ndomain);

    init_owner(*ndomain, *domains);

    for (k = 0; k < *ndomain; k++) {
        domain = *domains + k;
        init_domain_hydro_var(domain, gcoordl, gcoordr, gsizes);
    }
    return 0;
 }

int init_domain_hydro_var(ff_Domain *domain, real *gcoordl,
                          real *gcoordr, llint *gsizes)
{
    int extra, d, c, nvar, nc, nbdy;
    int comp_sizes[1], which_comp[1];
    llint i, j, k, size, ic, jc, kc;
    llint *sizes, sizes_ext[3], lsize, offset;
    real p0, p1, tmp, xc, yc, r0, dr, z, y, x;
    real dx, dy, dz, hdx, hdy, hdz, ds, dz0, vx0, vy0;
    real *buffer_tot, *buffer[10], *flux_x, *flux_y, *flux_z;
    real ***rho3d, ***p3d, ***ux3d, ***uy3d, ***uz3d, ***src3d, ***dt_burned3d;
    real  **rho2d,  **p2d,  **ux2d,  **uy2d, **src2d, **dt_burned2d;
    char *names[] = {"density", "pressure", "velocity", "src", "dt_burned"};
    ff_Mesh_Var *vars, *var;
    ff_Data_Type datatype;
    ff_Var_Type type;

    extern int dim, mype, ff_ifconservd;
```

```
    extern real gm, ff_rho0, ff_ei0, ff_ei1, ff_p0, ff_vx0, ff_vy0, ff_vz0;
    extern real ff_sz_z, ff_sz_xy, ff_width, ff_z0;
    extern llint ff_gsize_z, ff_gsize_xy;

#if ifdouble == 1
    datatype = ff_double;
#else
    datatype = ff_float;
#endif

    p0 = (gm - 1.0) * ff_rho0 * ff_ei0;
    ff_p0 = p0;
    p1 = (gm - 1.0) * ff_rho0 * ff_ei1;

    nvar = 5;
    if (dim == 3) {
        nc = 7;
        type = ff_zone;
    }
    else if (dim == 2) {
        nc = 6;
        type = ff_face;
    }
     sizes = domain->sizes;
     nbdy  = domain->nbdy;
     lsize = 1;
     for (d = 0; d < dim; d++) {
         sizes_ext[d] = sizes[d] + nbdy + nbdy;
         lsize *= (sizes[d] + nbdy + nbdy);
     }
     buffer_tot = (real *)malloc(nc * lsize * sizeof(real));
     if (!buffer_tot) {
          printf("ERROR: allocate buffer_tot failed in  init_domain_hydro_var\n");
          return -1;
     }
     buffer[0] = buffer_tot;
     for (c = 1; c < nc; c++) {
         buffer[c] = buffer[c-1] + lsize;
     }
     domain->nvar = nvar;
     vars = (ff_Mesh_Var *) malloc((nvar + 1) * sizeof(ff_Mesh_Var));
     for (c = 0; c <  nvar; c++) {
         var = vars + c;
         init_var_null(var);
     }
     init_one_var(buffer[0], domain->owner, names[0], type, datatype, 0, NULL, NULL,
              gcoordl, gcoordr, domain->coordl, domain->coordr,
              domain->sizes, domain->nbdy, NULL, vars, ff_rho0);
     var = vars + 1;
     init_one_var(buffer[1], domain->owner, names[1], type, datatype, 0, NULL, NULL,
              gcoordl, gcoordr, domain->coordl, domain->coordr,
              domain->sizes, domain->nbdy, NULL, var, p0);
     comp_sizes[0] = dim;
```

```
    which_comp[0] = 0;
    var = vars + 2;
    init_one_var(buffer[2], domain->owner, names[2],
             type, datatype, 1, comp_sizes, which_comp,
             gcoordl, gcoordr, domain->coordl, domain->coordr,
             domain->sizes, domain->nbdy, NULL, var, ff_vx0);
    which_comp[0] = 1;
    init_one_var(buffer[3], domain->owner, names[2],
             type, datatype, 1, comp_sizes, which_comp,
             gcoordl, gcoordr, domain->coordl, domain->coordr,
             domain->sizes, domain->nbdy, NULL, var, ff_vy0);
    extra = 0;
    if (dim == 3) {
        extra = 1;
        which_comp[0] = 2;
        init_one_var(buffer[4], domain->owner, names[2],
             type, datatype, 1, comp_sizes, which_comp,
             gcoordl, gcoordr, domain->coordl, domain->coordr,
             domain->sizes, domain->nbdy, NULL, var, ff_vz0);
    }
    var = vars + 3;
    init_one_var(buffer[4+extra], domain->owner,
             names[3], type, datatype, 0, NULL, NULL,
             gcoordl, gcoordr, domain->coordl, domain->coordr,
             domain->sizes, domain->nbdy, func_src, var, 1.0);
    var = vars + 4;
    init_one_var(buffer[5 + extra], domain->owner,
             names[4], type, datatype, 0, NULL, NULL,
             gcoordl, gcoordr, domain->coordl, domain->coordr,
             domain->sizes, domain->nbdy, NULL, var, 0.0);

    domain->nvar = nvar;
    domain->vars = vars;

    if (dim == 3) {
        GET_3D_FORM(real, buffer[0], rho3d, j, k, sizes_ext, offset, rho2d);
        GET_3D_FORM(real, buffer[1], p3d,   j, k, sizes_ext, offset, rho2d);
        GET_3D_FORM(real, buffer[2], ux3d,  j, k, sizes_ext, offset, rho2d);
        GET_3D_FORM(real, buffer[3], uy3d,  j, k, sizes_ext, offset, rho2d);
        GET_3D_FORM(real, buffer[4], uz3d,  j, k, sizes_ext, offset, rho2d);
        GET_3D_FORM(real, buffer[5], src3d,  j, k, sizes_ext, offset, rho2d);
        GET_3D_FORM(real, buffer[6], dt_burned3d,  j, k, sizes_ext, offset, rho2d);

        domain->rho3d = rho3d;
        domain->p3d   = p3d;
        domain->ux3d  = ux3d;
        domain->uy3d  = uy3d;
        domain->uz3d  = uz3d;
        domain->src3d = src3d;
        domain->dt_burned3d = dt_burned3d;

        xc = 0.5 * ff_sz_xy;
        yc = xc;
```

```
        dx = ff_sz_xy / (real) ff_gsize_xy;
        dy = dx;
        dz = ff_sz_z/(real)ff_gsize_z;
        hdz = 0.5 * dz;
        hdx = 0.5 * dx;
        hdy = 0.5 * dy;
        r0 = 0.04 * ff_sz_xy;

        for (k = 0; k < sizes[2]; k++) {
            z = (real) k * dz + hdz;
            for (j = 0; j < sizes[1]; j++) {
                y = hdy + (real)j * dy - yc;
                for (i = 0; i < sizes[0]; i++) {
                    x = hdx + (real)i * dx - xc;
                    dr = (mysqrt(x * x + y * y + z * z) - r0)/ff_width;
                    tmp = mytanh(dr);
                    p3d[k+nbdy][j+nbdy][i+nbdy] =
                         0.5 *((1.0 + tmp)*p0 + (1.0 - tmp)*p1);
                }
            }
        }
        if ((ff_vx0 != 0.0) || (ff_vy0 != 0.0) ) {

            dz = ff_sz_z/(real)ff_gsize_z;
            z  = -0.5 * dz;
            for (k = nbdy; k < sizes[2] + nbdy; k++) {
                z += dz;
                dz0 = (z - ff_z0)/ff_width;
                tmp = mytanh(dz0);
                vx0 = 0.5 * (1.0 + tmp) * ff_vx0;
                vy0 = 0.5 * (1.0 + tmp) * ff_vy0;
                for (j = 0; j < sizes[1] + nbdy + nbdy; j++) {
                    for (i = 0; i < sizes[0] + nbdy + nbdy; i++) {
                        ux3d[k][j][i] = vx0;
                        uy3d[k][j][i] = vy0;
                    }
                }
            }
        }
    }
    else if (dim == 2) {
        GET_2D_FORM(real, buffer[0], rho2d, j, sizes_ext);
        GET_2D_FORM(real, buffer[1], p2d,   j, sizes_ext);
        GET_2D_FORM(real, buffer[2], ux2d,  j, sizes_ext);
        GET_2D_FORM(real, buffer[3], uy2d,  j, sizes_ext);
        GET_2D_FORM(real, buffer[4], src2d, j, sizes_ext);
        GET_2D_FORM(real, buffer[5], dt_burned2d, j, sizes_ext);

        domain->rho2d = rho2d;
        domain->p2d = p2d;
        domain->ux2d = ux2d;
        domain->uy2d = uy2d;
        domain->src2d = src2d;
```

```
            domain->dt_burned2d = dt_burned2d;
        }
        return 0;
    }


int init_owner(int ndomain, ff_Domain *domains)
{
        int k, ndomain_per_pe, pe, nremain, nb_this_pe;
        ff_Domain *domain, *list;

        extern int npes, mype, dim;

        for (k = 0; k < ndomain; k++) {
            domain = domains + k;
            domain->owner = k;
        }
        return 0;
    }


int init_mesh(real *gcoordl, real *gcoordr, llint *gsizes,
              int *ndomain_d,
              ff_Bdy_Type *bdy_typel, ff_Bdy_Type *bdy_typer,
              ff_Domain **domain_list, int *ndomain)
{
    /**
        ndomain_d:   the number of domains in each dimension
        bdy_typel:  boundary conditions at the low ends.
        bdy_typer:  boundary conditions at the higher ends.
    **/
        extern int ff_next_id;
        extern int dim;

        int k, j, i, id, b, n, isbdy0, isbdy1, isbdy2, isbdy;
        int ndomain_layer[3];
        llint sizes[3];
        llint **size_db, **offset_db;
        real dc[3], coordmin[3], coordmax[3];
        real *c0, *c1;

        ff_Bdy_Type typel[3], typer[3];
        ff_Domain *domains, *domain;

        extern int ff_nbdy;

        *ndomain = 1;
        for (k = 0; k  < dim; k++) {
            *ndomain *= ndomain_d[k];
        }
        domains = (ff_Domain *) malloc((*ndomain + 1) * sizeof(ff_Domain));
        *domain_list = domains;
```

```
for (k = 0; k < dim; k++) {
    sizes[k] = gsizes[k]/ndomain_d[k];
}
size_db = (llint **) malloc((dim + dim) * sizeof(llint *));
offset_db = size_db + dim;
size_db[0] = (llint *) malloc((dim + dim) * (*ndomain) * sizeof(llint));
offset_db[0] = size_db[0] + (dim * (*ndomain));

for (k = 1; k < dim; k++) {
    size_db[k] = size_db[k-1] + (*ndomain);
    offset_db[k] = offset_db[k-1] + (*ndomain);
}
for (k = 0; k < dim; k++) {
    n = ndomain_d[k];
    for (b = 0; b < n - 1; b++) {
        size_db[k][b] = sizes[k];
        offset_db[k][b] = b * sizes[k];
    }
      size_db[k][n-1]  = gsizes[k] - (n - 1) * sizes[k];
    offset_db[k][n-1] = (n - 1) * sizes[k];
}
for (k = 0; k < dim; k++) {
    dc[k] = (gcoordr[k] - gcoordl[k])/(real)(gsizes[k]);
}
ndomain_layer[0] = 1;
for (k = 1; k < dim; k++) {
    ndomain_layer[k] = ndomain_layer[k-1] * ndomain_d[k-1];
}
init_bdy_null(typel, typer);

if (dim == 3) {
    for (k = 0; k < ndomain_d[2]; k++) {
        if (k == 0) {
           typel[2] = bdy_typel[2];
           isbdy2 = 1;
        }
        else {
           typel[2] = ff_not_bdy;
        }
        if (k == ndomain_d[2] - 1) {
           typer[2] = bdy_typer[2];
           isbdy2 = 1;
        }
        else {
           typer[2] = ff_not_bdy;
        }
        coordmin[2] = gcoordl[2]  + (real)(offset_db[2][k]) * dc[2];
        coordmax[2] = coordmin[2] + (real)(  size_db[2][k]) * dc[2];
        for (j = 0; j < ndomain_d[1]; j++) {
            if (j == 0) {
               typel[1] = bdy_typel[1];
               isbdy1 = 1;
            }
```

```
                else {
                   typel[1] = ff_not_bdy;
                }
                if (j == ndomain_d[1] - 1) {
                   typer[1] = bdy_typer[1];
                   isbdy1 = 1;
                }
                else {
                   typer[1] = ff_not_bdy;
                }
                coordmin[1] = gcoordl[1]  + (real)(offset_db[1][j]) * dc[1];
                coordmax[1] = coordmin[1] + (real)(size_db[1][j]) * dc[1];
                for (i = 0; i < ndomain_d[0]; i++) {
                    if (i == 0) {
                       typel[0] = bdy_typel[0];
                       isbdy0 = 1;
                    }
                    else {
                       typel[0] = ff_not_bdy;
                    }
                    if (i == ndomain_d[0] - 1) {
                       typer[0] = bdy_typer[0];
                       isbdy0 = 1;
                    }
                    else {
                       typer[0] = ff_not_bdy;
                    }
                    coordmin[0] = gcoordl[0]  + (real)(offset_db[0][i]) * dc[0];
                    coordmax[0] = coordmin[0] + (real)(  size_db[0][i]) * dc[0];

                    id = k * ndomain_layer[2] + j * ndomain_layer[1] + i;

                    domain = domains + id;

                    sizes[0] = size_db[0][i];
                    sizes[1] = size_db[1][j];
                    sizes[2] = size_db[2][k];

                    isbdy = isbdy0 + isbdy1 + isbdy2;
                    init_domain(domain, id, -1, coordmin, coordmax, sizes,
                                ff_nbdy, isbdy, typel, typer);
                    isbdy0 = 0;
                }
                isbdy1 = 0;
            }
            isbdy2 = 0;
        }
    }
    else if (dim == 2) {
        for (j = 0; j < ndomain_d[1]; j++) {
            if (j == 0) {
                typel[1] = bdy_typel[1];
                isbdy1 = 1;
```

```
            }
            else {
                typel[1] = ff_not_bdy;
            }
            if (j == ndomain_d[1] - 1) {
               typer[1] = bdy_typer[1];
               isbdy1 = 1;
            }
            else {
               typer[1] = ff_not_bdy;
            }
            coordmin[1] = gcoordl[1]  + (real)(offset_db[1][j]) * dc[1];  /* y0 */
            coordmax[1] = coordmin[1] + (real)(  size_db[1][j]) * dc[1];  /* y1 */
            for (i = 0; i < ndomain_d[0]; i++) {
                if (i == 0) {
                    typel[0] = bdy_typel[0];
                    isbdy0 = 1;
                }
                else {
                    typel[0] = ff_not_bdy;
                }
                if (i == ndomain_d[0] - 1) {
                    typer[0] = bdy_typer[0];
                    isbdy0 = 1;
                }
                else {
                    typer[0] = ff_not_bdy;
                }
                coordmin[0] = gcoordl[0]  + (real)(offset_db[0][i]) * dc[0];  /* x0 */
                coordmax[0] = coordmin[0] + (real)(  size_db[0][i]) * dc[0];  /* x1 */

                id = j * ndomain_layer[1] + i;
                domain = domains + id;

                sizes[0] = size_db[0][i];
                sizes[1] = size_db[1][j];

                isbdy = isbdy0 + isbdy1;
                init_domain(domain, id, -1, coordmin, coordmax, sizes,
                           ff_nbdy, isbdy, typel, typer);
                isbdy0 = 0;
            }
            isbdy1 = 0;
        }
    }
    else {
        printf("ERROR: dim != 2 or 3\n");
        return -1;
    }
    ff_next_id = id + 1;
    return 0;
}
```

```
int init_domain(ff_Domain *domain, int id, int owner,
               real *coordmin, real *coordmax, llint *sizes,
               int nbdy, int isbdy, ff_Bdy_Type *typel, ff_Bdy_Type *typer)
{
    int k;
    ff_Bdy_Condition *bdy;

    extern int dim;

    domain->id = id;
    domain->owner = owner;
    domain->nbdy  = nbdy;
    memcpy(domain->coordl, coordmin, (size_t)(dim * sizeof(real)));
    memcpy(domain->coordr, coordmax, (size_t)(dim * sizeof(real)));
    memcpy(domain->sizes,  sizes,    (size_t)(dim * sizeof(llint)));

    if (isbdy) {
        bdy = (ff_Bdy_Condition *) malloc(sizeof(ff_Bdy_Condition));
        memcpy(bdy->typel, typel, (size_t)(dim * sizeof(int)));
        memcpy(bdy->typer, typer, (size_t)(dim * sizeof(int)));
        domain->bdy = bdy;
    }
    else {
        domain->bdy = NULL;
    }
    return 0;
}

int init_bdy_null(ff_Bdy_Type *typel, ff_Bdy_Type *typer)
{
   int k;
   extern int dim;

   for (k = 0; k < dim; k++) {
       typel[k] = ff_not_bdy;
       typer[k] = ff_not_bdy;
   }
   return 0;
}

int init_one_var(void *buffer, int owner, char *name,
             ff_Var_Type type, ff_Data_Type datatype,
             int rank, int *comp_sizes, int *which_comp,
             real *gcoordl, real *gcoordr,
             real *coordl, real *coordr, llint *sizes, int nbdy,
             int (*func)(int, real*, real*,
                         real*, real*, llint*, void*, ff_Data_Type, real),
             ff_Mesh_Var *var, real v0)
{
/*  set one component of a variable.
    If *var  = NULL, create the varibale
    If *var != NULL, attach the new component or reset an component
```

```
*/
    int k, c, slen, ncomp, found;
    int *my_which_comp, *my_comp_sizes;
    llint lsize, i;

    real *dbuffer;

    ff_Var_Comps *my_comps, *my_comp, *this_comp;

    extern int npes, mype, dim;

    if (!var) {
        printf("ERROR: null var in init_var\n");
        return -1;
    }
    my_comp_sizes = var->comp_sizes;
    if (var->name) {
        if (strcmp(var->name, name)) {
            printf("ERROR: name != var->name in init_var\n");
            return -1;
        }
        if (var->type != type) {
            printf("ERROR: var->type != type in init_var\n");
            return -1;
        }
        if (var->rank != rank) {
            printf("ERROR: var->rank != rank in init_var\n");
            return -1;
        }
        if (var->datatype != datatype) {
           printf("ERROR: var->datatype != datatype in init_var\n");
            return -1;
        }
        for (k = 0; k < rank; k++) {
            if (my_comp_sizes[k] != comp_sizes[k]) {
               printf("ERROR: comp_sizes inconsistent in init_var\n");
               return -1;
            }
        }
    }
    else {
       slen = strlen(name) + 1;
       var->name = (char *) malloc(slen);
       strcpy(var->name, name);
       var->type = type;
       var->rank = rank;
       var->datatype = datatype;
       for (k = 0; k < rank; k++) {
           my_comp_sizes[k] = comp_sizes[k];
       }
    }
    for (k = 0; k < rank; k++) {
        if (which_comp[k] < comp_sizes[k]) continue;
```

```
        printf("ERROR: which_comp[k] >= comp_sizes[k] in init_var\n");
        return -1;
    }
    my_comps = var->comps;
    ncomp = 1;
    for (k = 0; k < rank; k++) {
        ncomp *= comp_sizes[k];
    }
    /* find this component if it already exits */

    this_comp = NULL;
    if (rank == 0) {
        my_comp = my_comps;
        if (my_comp->buffer) {
            printf("Warning: scalar variable is reset in init_var\n");
        }
        this_comp = my_comp;
    }
    else {
        for (c = 0; c < ncomp; c++) {
            my_comp = my_comps + c;
            my_which_comp = my_comp->which_comp;
            for (k = 0; k < rank; k++) {
                if ((my_which_comp[k] >= 0) &&
                    (my_which_comp[k] < my_comp_sizes[k])) continue;
                this_comp = my_comp;
                break;
            }
            if (this_comp) break;
        }
    }
    if (!this_comp) {
        if (rank == 0) {
            this_comp = my_comps;
        }
        else {
            for (c = 0; c < ncomp; c++) {
                my_comp = my_comps + c;
                my_which_comp = my_comp->which_comp;
                for (k = 0; k < rank; k++) {
                    if ((my_which_comp[k] < 0) ||
                        (my_which_comp[k] >= my_comp_sizes[k])) {
                        this_comp = my_comp;
                        break;
                    }
                }
                if (this_comp) break;
            }
        }
    }
    if (!this_comp) {
        printf("ERROR: component not determined in init_var\n");
        return -1;
```

```
    }
    if (this_comp->buffer) free(this_comp->buffer);
    this_comp->buffer = NULL;
    my_which_comp = this_comp->which_comp;
    for (k = 0; k < rank; k++) {
        my_which_comp[k] = which_comp[k];
    }
    this_comp->buffer = buffer;

    if (func) {
        func(nbdy, gcoordl, gcoordr,
             coordl, coordr, sizes, buffer, datatype, v0);
    }
    else {
        lsize = 1;
        for (k = 0; k < dim; k++) {
            lsize *= (sizes[k] + nbdy + nbdy);
        }
        dbuffer = (real *)buffer;
        for (i = 0; i < lsize; i++) {
            dbuffer[i] = v0;
        }
    }
    return 0;
}

int init_var_null(ff_Mesh_Var *var)
{
     int n, k, i;
     int *comp_sizes, *which_comp;
     ff_Var_Comps *comps;

     var->name = NULL;
     var->type = ff_vartype_invalid;
     var->rank = 0;
     var->datatype = ff_datatype_invalid;
     comp_sizes = var->comp_sizes;
     for (k = 0; k < MAX_VAR_RANK; k++) {
         var->comp_sizes[k] = 0;
     }
     comps = var->comps;
     n = MAX_VAR_RANK*MAX_VAR_RANK*MAX_VAR_RANK;
     for (k = 0; k < n; k++) {
         which_comp = comps[k].which_comp;
         for (i = 0; i < MAX_VAR_RANK; i++) {
             which_comp[i] = -1;
         }
         comps[k].buffer = NULL;
     }
     return 0;
 }

int func_src(int nbdy,
```

```
              real *gcoordl, real *gcoordr,
              real *coordl, real *coordr, llint *sizes,
              void *buffer, ff_Data_Type datatype, real v0)
{

    real *fp, *fp2, *fp1, z, dz, y, dy, x, dx, ds;
    llint sizes_ext[3], i, j, k, d;
    real dc[3], c0[3], ctr[3], c[3];

    extern real ff_forest_height, ff_gap_width, ff_gap_disp;
    extern real ff_forest_energy_src;
    extern int npes, mype, dim;

    for (d = 0; d < dim; d++) {
        sizes_ext[d] = sizes[d] + nbdy + nbdy;
        dc[d] = (coordr[d] - coordl[d])/(real)sizes[d];
        c0[d] = coordl[d] + 0.5 * dc[d];
        ctr[d] = 0.5 *(gcoordl[d] + gcoordr[d]);
    }
    ctr[2] = 0.0;

    fp = (real *)buffer;
    for (k = 0; k < sizes_ext[2]; k++) {
        z = c0[2] + (real)(k - nbdy) * dc[2];
        dz = z - ctr[2];
        fp2 = fp + (k * sizes_ext[0] * sizes_ext[1]);
        for (j = 0; j < sizes_ext[1]; j++) {
            y = c0[1] + (real)(j - nbdy) * dc[1];
            dy = y - ctr[1];
            fp1 = fp2 + (j * sizes_ext[0]);
            for (i = 0; i < sizes_ext[0]; i++) {
                x = c0[0] + (real)(i - nbdy) * dc[0];
                dx = x - ctr[0];
                if (dz > ff_forest_height) {
                    fp1[i] = 0.0;
                }
                else {
                    ds = mysqrt(dx * dx + dy * dy);
                    if ((ds > ff_gap_disp) &&
                        (ds < ff_gap_disp + ff_gap_width)) {
                         fp1[i] = 0.0;
                    }
                    else {
                         fp1[i] = ff_forest_energy_src;
                    }
                }
            }
        }
    }
    return 0;
  }

#include "ff.h"
```

```
#include "files.h"

int run_hydro(int ndomain, ff_Domain *domains,
              int order, real t, real dt, real *cournmx)
{
    int k;
    ff_Domain *domain;

    extern real gm;
    extern int mype;

    for (k = 0; k < ndomain; k++) {
        domain  = domains + k;
        run_domain_hydro(domain, order, t, dt, gm, cournmx);
    }
    return 0;
}

int run_domain_hydro(ff_Domain *domain, int pass,
                     real t0, real dt0, real gm, real *cournmx)
{
    int k, ncon2, order, indent;
    llint size, i;
    real dt, t;
    extern int dim, ff_nbdy;

    dt  = dt0;
    t   = t0;
    order = pass;
    indent = 0;

    if (dim == 3) {
            euler_3D(domain, order, indent, t, dt, gm, dim, cournmx);
            t += dt;
            order = 1 - order;
            indent += ff_nbdy;
    }
    else if (dim == 2) {
            euler_2D(domain, order, indent, t, dt, gm, dim, cournmx);
            t += dt;
            order = 1 - order;
            indent += ff_nbdy;
    }
    return 0;
 }

int euler_3D(ff_Domain *domain,
             int order, int indent, real t, real dt, real gm, int dim,
             real *cournmx)
{
    int err, nbdy, nconservd, d;
    llint *sizes;
    real  *coordl, *coordr;
```

```
real ***rho, ***p, ***ux, ***uy, ***uz, ***src, ***dt_burned;
llint i, j, k, size_mx;
llint sizes_ind[3];
real *var1d, *scratch, *rho1d, *p1d, *ux1d, *uy1d, *uz1d;
real *src1d, *dt_burned1d;

extern int ff_nbdy;

nbdy = domain->nbdy;
sizes = domain->sizes;
coordl = domain->coordl;
coordr = domain->coordr;
rho  = domain->rho3d;
p    = domain->p3d;
ux   = domain->ux3d;
uy   = domain->uy3d;
uz   = domain->uz3d;
src  = domain->src3d;
dt_burned = domain->dt_burned3d;

for (d = 0; d < dim; d++) {
    sizes_ind[d] = sizes[d] + nbdy + nbdy - indent;
}
size_mx = 0;
for (d = 0; d < dim; d++) {
    if (size_mx < sizes[d]) size_mx = sizes[d];
}
size_mx += (nbdy + nbdy + 1);

var1d = (real *)malloc(28 * size_mx * sizeof(real));
scratch = var1d + (7 * size_mx);

if (order == 0) {
 /* x-pass */
    for (k = indent;  k < sizes_ind[2]; k++) {
        for (j = indent; j < sizes_ind[1]; j++) {
             rho1d = rho[k][j];
               p1d =   p[k][j];
              ux1d =  ux[k][j];
              uy1d =  uy[k][j];
              uz1d =  uz[k][j];
              src1d = src[k][j];
              dt_burned1d = dt_burned[k][j];

             if (ff_nbdy >= 3) {
                 err = muscl_3_3D(indent, t, dt, gm, nbdy,
                         sizes[0], coordl[0], coordr[0],
                         rho1d, p1d, ux1d, uy1d, uz1d,
                         src1d, dt_burned1d,
                         cournmx, scratch);
             }
             else if (ff_nbdy == 2) {
                 err = muscle_2_3D(indent, t, dt, gm, nbdy,
```

```
                        sizes[0], coordl[0], coordr[0],
                        rho1d, p1d, ux1d, uy1d, uz1d,
                        src1d, dt_burned1d,
                        cournmx, scratch);
            }
        }
    }
    rho1d = var1d;
      p1d = rho1d + size_mx;
     ux1d = p1d   + size_mx;
     uy1d = ux1d  + size_mx;
     uz1d = uy1d  + size_mx;
    src1d = uz1d  + size_mx;
    dt_burned1d = src1d + size_mx;

/*  z-pass */

    for (j = indent; j < sizes_ind[1]; j++) {
        for (i = indent + ff_nbdy; i < sizes_ind[0] - ff_nbdy; i++) {
            for (k = indent;  k < sizes_ind[2]; k++) {
                rho1d[k] = rho[k][j][i];
                  p1d[k] =   p[k][j][i];
                 ux1d[k] =  uz[k][j][i];
                 uy1d[k] =  ux[k][j][i];
                 uz1d[k] =  uy[k][j][i];
                src1d[k] = src[k][j][i];
                dt_burned1d[k] = dt_burned[k][j][i];
            }
            if (ff_nbdy >= 3) {
                err = muscl_3_3D(indent, t, dt, gm, nbdy,
                         sizes[2], coordl[2], coordr[2],
                         rho1d, p1d, ux1d, uy1d, uz1d,
                         src1d, dt_burned1d,
                         cournmx, scratch);
            }
            else if (ff_nbdy == 2) {
                err = muscle_2_3D(indent, t, dt, gm, nbdy,
                         sizes[2], coordl[2], coordr[2],
                         rho1d, p1d, ux1d, uy1d, uz1d,
                         src1d, dt_burned1d,
                         cournmx, scratch);
            }
            for (k = indent + ff_nbdy;
                 k < sizes_ind[2] - ff_nbdy; k++) {
                rho[k][j][i] = rho1d[k];
                  p[k][j][i] =   p1d[k];
                 uz[k][j][i] =  ux1d[k];
                 ux[k][j][i] =  uy1d[k];
                 uy[k][j][i] =  uz1d[k];
                dt_burned[k][j][i] = dt_burned1d[k];
            }
        }
    }
```

```
/*    y-pass */

      for (i = indent + ff_nbdy; i < sizes_ind[0] - ff_nbdy; i++) {
          for (k = indent + ff_nbdy;  k < sizes_ind[2] - ff_nbdy; k++) {
              for (j = indent; j < sizes_ind[1]; j++) {
                  rho1d[j] = rho[k][j][i];
                    p1d[j] =   p[k][j][i];
                   ux1d[j] =  uy[k][j][i];
                   uy1d[j] =  uz[k][j][i];
                   uz1d[j] =  ux[k][j][i];
                  src1d[j] = src[k][j][i];
                  dt_burned1d[j] = dt_burned[k][j][i];
               }
               if (ff_nbdy >= 3) {
                   err = muscl_3_3D(indent, t, dt, gm, nbdy,
                           sizes[1], coordl[1], coordr[1],
                           rho1d, p1d, ux1d, uy1d, uz1d,
                           src1d, dt_burned1d,
                           cournmx, scratch);
               }
               else if (ff_nbdy == 2) {
                   err = muscle_2_3D(indent, t, dt, gm, nbdy,
                           sizes[1], coordl[1], coordr[1],
                           rho1d, p1d, ux1d, uy1d, uz1d,
                           src1d, dt_burned1d,
                           cournmx, scratch);
               }
               for (j = indent + ff_nbdy; j < sizes_ind[1] - ff_nbdy; j++) {
                  rho[k][j][i] = rho1d[j];
                    p[k][j][i] =   p1d[j];
                   uy[k][j][i] =  ux1d[j];
                   uz[k][j][i] =  uy1d[j];
                   ux[k][j][i] =  uz1d[j];
                  dt_burned[k][j][i] = dt_burned1d[j];
               }
           }
      }
  }
  else {
     rho1d = var1d;
       p1d = rho1d + size_mx;
      ux1d = p1d   + size_mx;
      uy1d = ux1d  + size_mx;
      uz1d = uy1d  + size_mx;
     src1d = uz1d  + size_mx;
     dt_burned1d   = src1d + size_mx;

 /*  y-pass  */

     for (i = indent; i < sizes_ind[0]; i++) {
          for (k = indent;  k < sizes_ind[2]; k++) {
              for (j = indent; j < sizes_ind[1]; j++) {
                  rho1d[j] = rho[k][j][i];
```

```
                p1d[j] =   p[k][j][i];
               ux1d[j] =  uy[k][j][i];
               uy1d[j] =  uz[k][j][i];
               uz1d[j] =  ux[k][j][i];
              src1d[j] = src[k][j][i];
              dt_burned1d[j] = dt_burned[k][j][i];
          }
           if (ff_nbdy >= 3) {
               err = muscl_3_3D(indent, t, dt, gm, nbdy,
                       sizes[1], coordl[1], coordr[1],
                       rho1d, p1d, ux1d, uy1d, uz1d,
                       src1d, dt_burned1d,
                       cournmx, scratch);
           }
           else if (ff_nbdy == 2) {
               err = muscle_2_3D(indent, t, dt, gm, nbdy,
                       sizes[1], coordl[1], coordr[1],
                       rho1d, p1d, ux1d, uy1d, uz1d,
                       src1d, dt_burned1d,
                       cournmx, scratch);
          }
          for (j = indent + ff_nbdy; j < sizes_ind[1] - ff_nbdy; j++) {
              rho[k][j][i] = rho1d[j];
                p[k][j][i] =   p1d[j];
               uy[k][j][i] =  ux1d[j];
               uz[k][j][i] =  uy1d[j];
               ux[k][j][i] =  uz1d[j];
              dt_burned[k][j][i] = dt_burned1d[j];
          }
       }
    }
 /* z-pass */

    for (j = indent + ff_nbdy; j < sizes_ind[1] - ff_nbdy; j++) {
        for (i = indent; i < sizes_ind[0]; i++) {
            for (k = indent;  k < sizes_ind[2]; k++) {
                rho1d[k] = rho[k][j][i];
                  p1d[k] =   p[k][j][i];
                 ux1d[k] =  uz[k][j][i];
                 uy1d[k] =  ux[k][j][i];
                 uz1d[k] =  uy[k][j][i];
                src1d[k] = src[k][j][i];
                dt_burned1d[k] = dt_burned[k][j][i];
            }
            if (ff_nbdy >= 3) {
                err = muscl_3_3D(indent, t, dt, gm, nbdy,
                         sizes[2], coordl[2], coordr[2],
                         rho1d, p1d, ux1d, uy1d, uz1d, src1d, dt_burned1d,
                         cournmx, scratch);
            }
            else if (ff_nbdy == 2) {
                err = muscle_2_3D(indent, t, dt, gm, nbdy,
                         sizes[2], coordl[2], coordr[2],
```

```
                             rho1d, p1d, ux1d, uy1d, uz1d, src1d, dt_burned1d,
                             cournmx, scratch);
                }
                for (k = indent + ff_nbdy;  k < sizes_ind[2] - ff_nbdy; k++) {
                     rho[k][j][i] = rho1d[k];
                       p[k][j][i] =   p1d[k];
                      uz[k][j][i] =  ux1d[k];
                      ux[k][j][i] =  uy1d[k];
                      uy[k][j][i] =  uz1d[k];
                    dt_burned[k][j][i] = dt_burned1d[k];
                 }
             }
        }
   /*   x-pass */
        for (k = indent + ff_nbdy;  k < sizes_ind[2] - ff_nbdy; k++) {
            for (j = indent + ff_nbdy; j < sizes_ind[1] - ff_nbdy; j++) {
                 rho1d = rho[k][j];
                   p1d =   p[k][j];
                  ux1d =  ux[k][j];
                  uy1d =  uy[k][j];
                  uz1d =  uz[k][j];
                 src1d = src[k][j];
                 dt_burned1d = dt_burned[k][j];

                 if (ff_nbdy >= 3) {
                     err = muscl_3_3D(indent, t, dt, gm, nbdy,
                             sizes[0], coordl[0], coordr[0],
                             rho1d, p1d, ux1d, uy1d, uz1d, src1d, dt_burned1d,
                             cournmx, scratch);
                 }
                 else if (ff_nbdy == 2) {
                     err = muscle_2_3D(indent, t, dt, gm, nbdy,
                             sizes[0], coordl[0], coordr[0],
                             rho1d, p1d, ux1d, uy1d, uz1d, src1d, dt_burned1d,
                             cournmx, scratch);
                 }
            }
        }
    }
    free(var1d);

    return 0;
 }

int euler_2D(ff_Domain *domain,
             int order, int indent, real t, real dt, real gm, int dim,
             real *cournmx)
{
    int nbdy, nconservd, ncon2, d, num;
    llint *sizes;
    real  *coordl, *coordr;
    real **rho, **p, **ux, **uy;
    llint i, i0, j, j0, k, k0, size_mx, idx;
```

```
llint sizes_ind[3];
real *var1d, *scratch, *rho1d, *p1d, *ux1d, *uy1d, *uz1d;

ff_Domain **children, *child;

extern int ff_nbdy;

nbdy   = domain->nbdy;
sizes  = domain->sizes;
coordl = domain->coordl;
coordr = domain->coordr;
rho    = domain->rho2d;
p      = domain->p2d;
ux     = domain->ux2d;
uy     = domain->uy2d;

for (d = 0; d < dim; d++) {
    sizes_ind[d] = sizes[d] + nbdy + nbdy - indent;
}
size_mx = 0;
for (d = 0; d < dim; d++) {
    if (size_mx < sizes[d]) size_mx = sizes[d];
}
size_mx += (nbdy + nbdy + 1);

var1d = (real *)malloc(22 * size_mx * sizeof(real));
scratch = var1d + (5 * size_mx);

if (order == 0) {
    for (j = indent; j < sizes_ind[1]; j++) {
        j0 = j - nbdy - nbdy;
        rho1d = rho[j];
          p1d =   p[j];
         ux1d =  ux[j];
         uy1d =  uy[j];

         if (ff_nbdy >= 3) {
             muscl_3_2D(indent, t, dt, gm, nbdy,
                   sizes[0], coordl[0], coordr[0],
                   rho1d, p1d, ux1d, uy1d,
                   cournmx, scratch);
         }
         else if (ff_nbdy == 2) {
            muscle_2_2D(indent, t, dt, gm, nbdy,
                   sizes[0], coordl[0], coordr[0],
                   rho1d, p1d, ux1d, uy1d,
                   cournmx, scratch);
        }
    }
    rho1d = var1d;
      p1d = rho1d + size_mx;
     ux1d = p1d   + size_mx;
     uy1d = ux1d  + size_mx;
```

```
    for (i = indent + ff_nbdy; i < sizes_ind[0] - ff_nbdy; i++) {
        i0 = i - nbdy - nbdy;
        for (j = indent; j < sizes_ind[1]; j++) {
            rho1d[j] = rho[j][i];
              p1d[j] =   p[j][i];
             ux1d[j] =  uy[j][i];
             uy1d[j] =  ux[j][i];
        }
        if (ff_nbdy >= 3) {
            muscl_3_2D(indent, t, dt, gm, nbdy,
                    sizes[1], coordl[1], coordr[1],
                    rho1d, p1d, ux1d, uy1d,
                    cournmx, scratch);
        }
        else if (ff_nbdy == 2) {
            muscle_2_2D(indent, t, dt, gm, nbdy,
                    sizes[1], coordl[1], coordr[1],
                    rho1d, p1d, ux1d, uy1d,
                    cournmx, scratch);
        }
        for (j = indent + ff_nbdy; j < sizes_ind[1] - ff_nbdy; j++) {
             rho[j][i] = rho1d[j];
               p[j][i] =   p1d[j];
              uy[j][i] =  ux1d[j];
              ux[j][i] =  uy1d[j];
        }
    }
}
else {
   rho1d = var1d;
     p1d = rho1d + size_mx;
    ux1d = p1d   + size_mx;
    uy1d = ux1d  + size_mx;

    for (i = indent; i < sizes_ind[0]; i++) {
        i0 = i - nbdy - nbdy;
        for (j = indent; j < sizes_ind[1]; j++) {
            rho1d[j] = rho[j][i];
              p1d[j] =   p[j][i];
             ux1d[j] =  uy[j][i];
             uy1d[j] =  ux[j][i];
        }
         if (ff_nbdy >= 3) {
             muscl_3_2D(indent, t, dt, gm, nbdy,
                     sizes[1], coordl[1], coordr[1],
                     rho1d, p1d, ux1d, uy1d,
                     cournmx, scratch);
         }
         else if (ff_nbdy == 2) {
             muscle_2_2D(indent, t, dt, gm, nbdy,
                     sizes[1], coordl[1], coordr[1],
                     rho1d, p1d, ux1d, uy1d,
```

```
                        cournmx, scratch);
            }
            for (j = indent + ff_nbdy; j < sizes_ind[1] - ff_nbdy; j++) {
                rho[j][i] = rho1d[j];
                  p[j][i] =   p1d[j];
                 uy[j][i] =  ux1d[j];
                 ux[j][i] =  uy1d[j];
            }
        }
        for (j = indent + ff_nbdy; j < sizes_ind[1] - ff_nbdy; j++) {
            j0 = j - nbdy - nbdy;
            rho1d = rho[j];
              p1d =   p[j];
             ux1d =  ux[j];
             uy1d =  uy[j];

             if (ff_nbdy >= 3) {
                  muscl_3_2D(indent, t, dt, gm, nbdy,
                      sizes[0], coordl[0], coordr[0],
                      rho1d, p1d, ux1d, uy1d,
                      cournmx, scratch);
             }
             else if (ff_nbdy == 2) {
                 muscle_2_2D(indent, t, dt, gm, nbdy,
                      sizes[0], coordl[0], coordr[0],
                      rho1d, p1d, ux1d, uy1d,
                      cournmx, scratch);
             }
        }
    }
    free(var1d);

    return 0;
 }

int muscle_2_2D(int indent, real t, real dt, real gm, int nbdy,
                llint size, real coordl, real coordr,
                real *rho, real *p, real *ux, real *uy,
                real *cournmx, real *scratch)
{
  /**
    scratch[0, 17*(nbdy+nbdy+size):    working array
  **/
    int k, iter;
    llint i, i0, i1, size_ext, size_ind;
    real dx, dtbydx, cs, tmp, factor, dda, pavl, uxavl, rho1, rho2;
    real gaminv, gamma1, gm1inv, gammp1, hgamp1;
    real dpavl, wlfac, hrhol, wl;
    real dpdul, wrfac, hrhor, wr, dpdur, ustrl, ustrr;
    real sr, sl, p_euler, ux_euler, rho_euler, uy_euler, ei_euler;
    real rhoux, ek;

    real *cc, *courno, *drho, *dp, *dux, *duy;
```

```
real *rhol, *rhor, *pl, *pr, *uxl, *uxr, *uyl, *uyr;
real *rhonu, *momx, *momy, *tote;
real *fmassl, *fmomxl, *fmomyl, *ftotel;

extern real small;
extern int mype;

size_ext = size + nbdy + nbdy;
size_ind = size_ext - (llint) indent;

i1 = size_ext + 1;

cc = scratch;
courno = cc + i1;
drho   = courno + i1;
dp     = drho   + i1;
dux    = dp     + i1;
duy    = dux    + i1;

rhol   = duy    + i1;
rhor   = rhol   + i1;
pl     = rhor   + i1;
pr     = pl     + i1;
uxl    = pr     + i1;
uxr    = uxl    + i1;
uyl    = uxr    + i1;
uyr    = uyl    + i1;

dx = (coordr - coordl)/(real)size;
dtbydx = dt / dx;
gaminv = 1.0/gm;
gamma1 = gm - 1.0;
gm1inv = 1.0/gamma1;
gammp1 = gm + 1.0;
hgamp1 = 0.5 * gammp1;

for (i = indent; i < size_ind; i++) {
    cs = mysqrt(gm * p[i]/rho[i]);
    cc[i] = cs * rho[i];
    courno[i] = dtbydx * cs;

    tmp = courno[i] + dtbydx * myfabs(ux[i]);
    if (*cournmx < tmp) *cournmx = tmp;
}
getslope(indent, nbdy, size, rho, drho);
getslope(indent, nbdy, size, p,   dp);
getslope(indent, nbdy, size, ux,  dux);
getslope(indent, nbdy, size, uy,  duy);

for (i = indent + 1; i < size_ind - 1; i++) {
    factor = 0.5 * (1.0 - courno[i]);

    dda = factor * drho[i];
```

```
        rhor[i]   = rho[i] - dda;
        rhol[i+1] = rho[i] + dda;

        dda = factor * dp[i];
        pr[i]   = p[i] - dda;
        pl[i+1] = p[i] + dda;

        dda = factor * dux[i];
        uxr[i]   = ux[i] - dda;
        uxl[i+1] = ux[i] + dda;

        dda = factor * duy[i];
        uyr[i]   = uy[i] - dda;
        uyl[i+1] = uy[i] + dda;
    }
    fmassl = drho;
    fmomxl = dux;
    fmomyl = duy;
    ftotel = dp;

    for (i = indent + 2; i < size_ind - 1; i++) {

        dpavl = (pl[i] - pr[i]  +  cc[i-1] * (uxl[i] - uxr[i]))/
                (cc[i-1] + cc[i]);
        pavl  = mymax (small, (pr[i]  +  dpavl * cc[i]));
        uxavl = uxr[i] + dpavl;
        iter = 0;
        while (iter < 0) {
            wlfac = gamma1 * pl[i]  +  gammp1 * pavl;
            wl = mysqrt (0.5 * rho[i-1] * wlfac);
            tmp = wlfac  -  hgamp1 * (pavl - pl[i]);
            dpdul = wl * wlfac / tmp;
            wrfac = gamma1 * pr[i]  +  gammp1 * pavl;
            wr = mysqrt (0.5 * rho[i] * wrfac);
            tmp = wrfac - hgamp1 * (pavl - pr[i]);
            dpdur = wr * wrfac / tmp;
            ustrl = uxl[i] -  (pavl - pl[i]) / wl;
            ustrr = uxr[i] +  (pavl - pr[i]) / wr;
            tmp = (ustrr - ustrl) * dpdur / (dpdur + dpdul);
            uxavl = ustrl + tmp;
            pavl = pavl - tmp * dpdul;
            pavl = mymax(small, pavl);
            iter++;
        }
        sr = uxr[i] + wr / rho[i];
        sl = uxl[i] - wl / rho[i-1];

        tmp = dtbydx * myfmax(myfabs(sr), myfabs(sl));
        *cournmx = myfmax(*cournmx, tmp);

        if (sl > 0.0) {
            p_euler   = pl[i];
            ux_euler  = uxl[i];
```

```
            uy_euler  = uy[i-1];
            rho_euler = rho[i-1];
            ei_euler  = gm1inv * pl[i] / rho[i-1];
        }
        else if (sr < 0.0) {
            p_euler   = pr[i];
            ux_euler  = uxr[i];
            uy_euler  = uy[i];
            rho_euler = rho[i];
            ei_euler  = gm1inv * pr[i] / rho[i];
        }
        else if ((myfabs(sl) < small) || (myfabs(sr) < small) ) {
            p_euler  = pavl;
            ux_euler = 0.0;
            rho_euler = 0.5 *(rho[i-1] + rho[i]);
            uy_euler  = 0.5 *(uy[i-1] + uy[i]);
        }
        else {
            p_euler  = pavl;
            ux_euler = uxavl;
            if (uxavl > 0.0) {
               /** the post shock state of w- for rho  **/

                tmp = wl + rho[i-1] * mymax(0.0, (uxl[i] - uxavl));
                rho_euler = rho[i-1] * wl/tmp;
                uy_euler  = uy[i-1];
                ei_euler  = gm1inv * pavl / rho_euler;
            }
            else if (uxavl < 0.0) {
               /** the post shock state of w+ for rho **/

                tmp = wr - rho[i] * mymin(0.0, (uxr[i] - uxavl));
                rho_euler = rho[i] * wr/tmp;
                uy_euler  = uy[i];
                ei_euler  = gm1inv * pavl / rho_euler;
            }
            else {
                tmp = wl + rho[i-1] * mymax(0.0, (uxl[i] - uxavl));
                rho1 = rho[i-1] * wl/tmp;
                tmp = wr - rho[i] * mymin(0.0, (uxr[i] - uxavl));
                rho2 = rho[i] * wr/tmp;

                rho_euler = 0.5 *(rho1 + rho2);
                uy_euler  = 0.5 *(uy[i-1] + uy[i]);
                ei_euler  = gm1inv * pavl * rho_euler;
            }
        }
        rhoux     = rho_euler * ux_euler;
        fmassl[i] = rhoux;
        fmomxl[i] = rhoux * ux_euler + p_euler;
        fmomyl[i] = rhoux * uy_euler;
        ftotel[i] = ux_euler * (p_euler + rho_euler *
                   (ei_euler + 0.5 *(ux_euler * ux_euler +
```

```
                    uy_euler * uy_euler)));
    }
    rhonu = rhol;
    momx  = uxl;
    momy  = uyl;
    tote  = pl;
    for (i = indent; i < size_ind; i++) {
        ek = 0.5 * rho[i] *(ux[i] * ux[i] + uy[i] * uy[i]);

        momx[i] = rho[i] * ux[i];
        momy[i] = rho[i] * uy[i];
        tote[i] = gm1inv * p[i] + ek;
    }
    for (i = indent + 2; i < size_ind - 2; i++) {
        i1 = i + 1;

        rhonu[i] =  (dtbydx *(fmassl[i] - fmassl[i1])) + rho[i];
         momx[i] += (dtbydx *(fmomxl[i] - fmomxl[i1]));
         momy[i] += (dtbydx *(fmomyl[i] - fmomyl[i1]));
         tote[i] += (dtbydx *(ftotel[i] - ftotel[i1]));
    }
    for (i = indent + 2; i < size_ind - 2; i++) {
        tmp = 1.0/rhonu[i];
        momx[i] = tmp * momx[i];
        momy[i] = tmp * momy[i];
        ek    = 0.5 * rhonu[i] *(momx[i] * momx[i] + momy[i] * momy[i]);
        tote[i] = gamma1 * (tote[i] - ek);
    }
    for (i = indent + 2; i < size_ind - 2; i++) {
        rho[i] = rhonu[i];
          p[i] =  tote[i];
         ux[i] =  momx[i];
         uy[i] =  momy[i];
    }
    return 0;
}

int muscle_2_3D(int indent, real t, real dt, real gm, int nbdy,
                llint size, real coordl, real coordr,
                real *rho, real *p, real *ux, real *uy, real *uz,
                real *src, real *dt_burned,
                real *cournmx, real *scratch)
{
  /**
    scratch[0, 17*(nbdy+nbdy+size):    working array
  **/
    int k, iter;
    llint i, i1, size_ext, size_ind;
    real s, dx, dtbydx, cs, tmp, factor, dda, pavl, uxavl, rho1, rho2;
    real gaminv, gamma1, gm1inv, gammp1, hgamp1;
    real dpavl, wlfac, hrhol, wl, dpdul;
    real wrfac, hrhor, wr, dpdur, ustrl, ustrr;
    real sr, sl, p_euler, ux_euler;
```

```
real rho_euler, uy_euler, uz_euler, ei_euler;
real rhoux, ek;

real *cc, *courno, *drho, *dp, *dux, *duy, *duz;
real *rhol, *rhor, *pl, *pr, *uxl, *uxr, *uyl, *uyr, *uzl, *uzr, *ei;
real *rhonu, *momx, *momy, *momz, *tote;
real *fmassl, *fmomxl, *fmomyl, *fmomzl, *ftotel;

extern real small;
extern int mype, ff_nbdy;
extern real ff_forest_max_dt_burning, ff_igniting_e;

size_ext = size + nbdy + nbdy;
size_ind = size_ext - (llint) indent;

i1 = size_ext + 1;

cc = scratch;
courno = cc + i1;
drho   = courno + i1;
dp     = drho   + i1;
dux    = dp     + i1;
duy    = dux    + i1;
duz    = duy    + i1;

rhol   = duz    + i1;
rhor   = rhol   + i1;
pl     = rhor   + i1;
pr     = pl     + i1;
uxl    = pr     + i1;
uxr    = uxl    + i1;
uyl    = uxr    + i1;
uyr    = uyl    + i1;
uzl    = uyr    + i1;
uzr    = uzl    + i1;

ei     = uzr    + i1;

dx = (coordr - coordl)/(real)size;
dtbydx = dt / dx;
gaminv = 1.0/gm;
gamma1 = gm - 1.0;
gm1inv = 1.0/gamma1;
gammp1 = gm + 1.0;
hgamp1 = 0.5 * gammp1;

for (i = indent; i < size_ind; i++) {
    ei[i] = gm1inv * p[i]/rho[i];
    cs = mysqrt(gm * p[i]/rho[i]);
    cc[i] = cs * rho[i];
    courno[i] = dtbydx * cs;

    tmp = courno[i] + dtbydx * myfabs(ux[i]);
```

```
        if (*cournmx < tmp) *cournmx = tmp;
    }
    getslope(indent, nbdy, size, rho, drho);
    getslope(indent, nbdy, size, p,   dp);
    getslope(indent, nbdy, size, ux,  dux);
    getslope(indent, nbdy, size, uy,  duy);
    getslope(indent, nbdy, size, uz,  duz);

    for (i = indent + 1; i < size_ind - 1; i++) {
        factor = 0.5 * (1.0 - courno[i]);

        dda = factor * drho[i];
        rhor[i]   = rho[i] - dda;
        rhol[i+1] = rho[i] + dda;

        dda = factor * dp[i];
        pr[i]   = p[i] - dda;
        pl[i+1] = p[i] + dda;

        dda = factor * dux[i];
        uxr[i]   = ux[i] - dda;
        uxl[i+1] = ux[i] + dda;

        dda = factor * duy[i];
        uyr[i]   = uy[i] - dda;
        uyl[i+1] = uy[i] + dda;

        dda = factor * duz[i];
        uzr[i]   = uz[i] - dda;
        uzl[i+1] = uz[i] + dda;
    }
    fmassl = drho;
    fmomxl = dux;
    fmomyl = duy;
    fmomzl = duz;
    ftotel = dp;

    for (i = indent + 2; i < size_ind - 1; i++) {

        dpavl = (pl[i] - pr[i]  +  cc[i-1] * (uxl[i] - uxr[i])) /
                (cc[i-1] + cc[i]);
        pavl  = mymax (small, (pr[i]  +  dpavl * cc[i]));
        uxavl = uxr[i] + dpavl;
        iter = 0;
        while (iter < 0) {
            wlfac = gamma1 * pl[i]  +  gammp1 * pavl;
            wl = mysqrt (0.5 * rho[i-1] * wlfac);
            tmp = wlfac  -  hgamp1 * (pavl - pl[i]);
            dpdul = wl * wlfac / tmp;
            wrfac = gamma1 * pr[i]  +  gammp1 * pavl;
            wr = mysqrt (0.5 * rho[i] * wrfac);
            tmp = wrfac - hgamp1 * (pavl - pr[i]);
            dpdur = wr * wrfac / tmp;
```

```
        ustrl = uxl[i] -  (pavl - pl[i]) / wl;
        ustrr = uxr[i] +  (pavl - pr[i]) / wr;
        tmp = (ustrr - ustrl) * dpdur / (dpdur + dpdul);
        uxavl = ustrl + tmp;
        pavl  = pavl - tmp * dpdul;
        pavl = mymax(small, pavl);
        iter++;
    }
    sr = uxr[i] + wr / rho[i];
    sl = uxl[i] - wl / rho[i-1];

    tmp = dtbydx * myfmax(myfabs(sr), myfabs(sl));
    *cournmx = myfmax(*cournmx, tmp);

    if (sl > 0.0) {
        p_euler   = pl[i];
        ux_euler  = uxl[i];
        uy_euler  = uy[i-1];
        uz_euler  = uz[i-1];
        rho_euler = rho[i-1];
        ei_euler  = gm1inv * pl[i] / rho[i-1];
    }
    else if (sr < 0.0) {
        p_euler   = pr[i];
        ux_euler  = uxr[i];
        uy_euler  = uy[i];
        uz_euler  = uz[i];
        rho_euler = rho[i];
        ei_euler  = gm1inv * pr[i] / rho[i];
    }
    else if ((myfabs(sl) < small) || (myfabs(sr) < small) ) {
        p_euler  = pavl;
        ux_euler = 0.0;
        rho_euler = 0.5 *(rho[i-1] + rho[i]);
        uy_euler  = 0.5 *(uy[i-1] + uy[i]);
        uz_euler  = 0.5 *(uz[i-1] + uz[i]);
    }
    else {
        p_euler  = pavl;
        ux_euler = uxavl;
        if (uxavl > 0.0) {
            /** the post shock state of w- for rho  **/

            tmp = wl + rho[i-1] * mymax(0.0, (uxl[i] - uxavl));
            rho_euler = rho[i-1] * wl/tmp;
            uy_euler  = uy[i-1];
            uz_euler  = uz[i-1];
            ei_euler  = gm1inv * pavl / rho_euler;
        }
        else if (uxavl < 0.0) {
            /** the post shock state of w+ for rho **/

            tmp = wr - rho[i] * mymin(0.0, (uxr[i] - uxavl));
```

```
                rho_euler = rho[i] * wr/tmp;
                uy_euler  = uy[i];
                uz_euler  = uz[i];
                ei_euler  = gm1inv * pavl / rho_euler;
            }
            else {
                tmp = wl + rho[i-1] * mymax(0.0, (uxl[i] - uxavl));
                rho1 = rho[i-1] * wl/tmp;
                tmp = wr - rho[i] * mymin(0.0, (uxr[i] - uxavl));
                rho2 = rho[i] * wr/tmp;

                rho_euler = 0.5 *(rho1 + rho2);
                uy_euler  = 0.5 *(uy[i-1] + uy[i]);
                uz_euler  = 0.5 *(uz[i-1] + uz[i]);
                ei_euler  = gm1inv * pavl * rho_euler;
            }
        }
        rhoux = rho_euler * ux_euler;
        fmassl[i] = rhoux;
        fmomxl[i] = rhoux * ux_euler + p_euler;
        fmomyl[i] = rhoux * uy_euler;
        fmomzl[i] = rhoux * uz_euler;
        ftotel[i] = ux_euler * (p_euler + rho_euler *
                   (ei_euler + 0.5 *(ux_euler * ux_euler +
                    uy_euler * uy_euler + uz_euler * uz_euler)));
    }
    rhonu = rhol;
    momx  = uxl;
    momy  = uyl;
    momz  = uzl;
    tote  = pl;
    for (i = indent; i < size_ind; i++) {
        ek = 0.5 * rho[i] *(ux[i] * ux[i] + uy[i] * uy[i] + uz[i] * uz[i]);
        tote[i] = gm1inv * p[i] + ek;
        momx[i] = rho[i] * ux[i];
        momy[i] = rho[i] * uy[i];
        momz[i] = rho[i] * uz[i];
    }
    for (i = indent + 2; i < size_ind - 2; i++) {
        i1 = i + 1;

       if ((ei[i] >= ff_igniting_e) &&
           (dt_burned[i] < ff_forest_max_dt_burning)) {
            s = src[i] * dt;
            if (s > small) {
                 dt_burned[i] += dt;
            }
        }
        else {
            s = 0.0;
        }
        rhonu[i] = (dtbydx *(fmassl[i] - fmassl[i1])) + rho[i];
        momx[i] += (dtbydx *(fmomxl[i] - fmomxl[i1]));
```

```
        momy[i] += (dtbydx *(fmomyl[i] - fmomyl[i1]));
        momz[i] += (dtbydx *(fmomzl[i] - fmomzl[i1]));
        tote[i] += (dtbydx *(ftotel[i] - ftotel[i1]) + s);
    }
    for (i = indent + 2; i < size_ind - 2; i++) {
        tmp = 1.0 / rhonu[i];
        momx[i] = tmp * momx[i];
        momy[i] = tmp * momy[i];
        momx[i] = tmp * momz[i];
        ek = 0.5 * rhonu[i] *(momx[i] * momx[i] +
             momy[i] * momy[i] + momz[i] * momz[i]);
        tote[i] = gamma1 * (tote[i] - ek);
    }
    for (i = indent + 2; i < size_ind - 2; i++) {
        rho[i] = rhonu[i];
          p[i] =  tote[i];
         ux[i] =  momx[i];
         uy[i] =  momy[i];
         uz[i] =  momz[i];
    }
    return 0;
}


int muscl_3_2D(int indent, real t, real dt, real gm, int nbdy,
                llint size, real coordl, real coordr,
                real *rho, real *p, real *ux, real *uy,
                real *cournmx, real *scratch)
{
/** scratch is at least 17 * (size + nbdy  nbdy + 1) long. */

    int k, iter;
    llint size_ind, i, i1;

    real *drho, *rhol, *rhor, *dp,  *pl,  *pr;
    real *dux, *uxl, *uxr, *duy, *uyl, *uyr;
    real *cc, *courno, *rhonu, *uxnu, *uynu, *eenu;
    real *dmassl, *dmomxl, *dmomyl, *detotl;
    real *uxavl, *pavl, *dxnu;

    real mycourno, dx, dxinv, dtbydx, dtbydm;
    real ddxl, factor, tmp, sgm, dda, cs, ei, ee;
    real gaminv, gamma1, gm1inv, gammp1, hgamp1;
    real dpavl, wlfac, hrhol, wl, dpdul;
    real wrfac, hrhor, wr, dpdur, ustrl, ustrr;
    real rho_lft, p_lft, ux_lft, gm_lft;
    real rho_rgt, p_rgt, ux_rgt, gm_rgt;

    extern real small;
    extern int  mype;

    size_ind = size + nbdy + nbdy - (llint) indent;
```

```
i1 = size + nbdy + nbdy + 1;

cc     = scratch;
courno = cc + i1;
pavl   = courno + i1;
uxavl  = pavl  + i1;

drho = uxavl + i1;
rhol = drho  + i1;
rhor = rhol  + i1;

dp = rhor + i1;
pl = dp + i1;
pr = pl + i1;

dux = pr  + i1;
uxl = dux + i1;
uxr = uxl + i1;

duy = uxr + i1;
uyl = duy + i1;
uyr = uyl + i1;

dxnu  = uyr  + i1;

dx = (coordr - coordl)/(real)size;
dxinv  = 1.0/dx;
dtbydx = dt * dxinv;
gaminv = 1.0/gm;
gamma1 = gm - 1.0;
gm1inv = 1.0/gamma1;
gammp1 = gm + 1.0;
hgamp1 = 0.5 * gammp1;

for (i = indent; i < size_ind; i++) {
    cs = mysqrt(gm * p[i]/rho[i]);
    cc[i] = cs * rho[i];
    courno[i] = dtbydx * cs;
    mycourno = courno[i] + (dtbydx * myfabs(ux[i]));
    if (*cournmx < mycourno) {
        *cournmx = mycourno;
    }
}
getslope(indent, nbdy, size, rho, drho);
getslope(indent, nbdy, size, p,   dp);
getslope(indent, nbdy, size, ux,  dux);
getslope(indent, nbdy, size, uy,  duy);

for (i = indent + 1; i < size_ind - 1; i++) {
    i1 = i + 1;
    factor = 0.5 * (1.0 - courno[i]);

    dda = factor * drho[i];
```

```
        rhor[i]  = rho[i] - dda;
        rhol[i1] = rho[i] + dda;

        dda = factor * dp[i];
        pr[i]  = p[i] - dda;
        pl[i1] = p[i] + dda;

        dda = factor * dux[i];
        uxr[i]  = ux[i] - dda;
        uxl[i1] = ux[i] + dda;
    }
    for (i = indent + 2; i < size_ind - 1; i++) {
      dpavl    = (pl[i] - pr[i]  +  cc[i-1] * (uxl[i] - uxr[i])) /
                 (cc[i-1] + cc[i]);
      pavl[i]  = mymax (small, (pr[i]  +  dpavl * cc[i]));
      uxavl[i] = uxr[i] + dpavl;

      iter = 0;
      while (iter < 0) {
          wlfac = gamma1 * pl[i]  +  gammp1 * pavl[i];
          wl = mysqrt (0.5 * rho[i-1] * wlfac);
          dpdul = wl * wlfac / (wlfac  -  hgamp1 * (pavl[i] - pl[i]));
          wrfac = gamma1 * pr[i]  +  gammp1 * pavl[i];
          wr = mysqrt (0.5 * rho[i] * wrfac);
          dpdur = wr * wrfac / (wrfac - hgamp1 * (pavl[i] - pr[i]));
          ustrl = uxl[i] -  (pavl[i] - pl[i]) / wl;
          ustrr = uxr[i] +  (pavl[i] - pr[i]) / wr;

          tmp = (ustrr - ustrl) * dpdur / (dpdur + dpdul);
          uxavl[i] = ustrl + tmp;
          pavl[i]  = mymax(small, (pavl[i] - tmp * dpdul));

          iter++;
       }
    }
    for (i = indent + 2; i < size_ind - 1; i++) {

        if (uxavl[i] >= 0.0) {
            tmp = courno[i] + uxavl[i] * dtbydx;
        }
        else {
            tmp = courno[i-1] - uxavl[i] * dtbydx;
        }
        if (*cournmx < tmp) *cournmx = tmp;
    }
    rhonu = rhor;
    eenu  = pr;
    uxnu  = uxr;
    uynu  = uyr;

    for (i = indent + 2; i < size_ind - 2; i++) {
        i1 = i + 1;
        dxnu[i] = dx + dt *(uxavl[i1] - uxavl[i]);
```

```
    courno[i] = 1.0 - dxnu[i] * dxinv;
    dtbydm  = dtbydx/rho[i];

    rhonu[i] = (dx/dxnu[i]) * rho[i];
    uxnu[i] = ux[i] + dtbydm *(pavl[i] - pavl[i1]);
    ee      = gm1inv * p[i]/
              rho[i] + 0.5 *(ux[i] * ux[i] + uy[i] * uy[i]);
    eenu[i] = ee  + dtbydm *
             (pavl[i] * uxavl[i] - pavl[i1] * uxavl[i1]);

    uynu[i] = uy[i];
}
for (i = indent + 2; i < size_ind - 2; i++) {
    if (*cournmx < courno[i]) {
        *cournmx = courno[i];
    }
}
for (i = indent + 2; i < size_ind - 2; i++) {
    uxnu[i] *= rhonu[i];
    uynu[i] *= rhonu[i];
    eenu[i] *= rhonu[i];
}
getslope1(indent + 2, nbdy, size, dxnu, rhonu, drho);
getslope1(indent + 2, nbdy, size, dxnu, eenu,  dp);
getslope1(indent + 2, nbdy, size, dxnu, uxnu,  dux);
getslope1(indent + 2, nbdy, size, dxnu, uynu,  duy);

dmassl = rhol;
dmomxl = uxl;
dmomyl = uyl;
detotl =  pl;

for (i = indent + 3; i < size_ind - 2; i++) {
     i1 = i - 1;
     ddxl = dt * uxavl[i];


     if (ddxl <= 0.0) {
         factor = 0.5 * (1.0 + ddxl/dxnu[i]);
         dmassl[i] =  ddxl *(rhonu[i] - drho[i] * factor);
         dmomxl[i] =  ddxl *( uxnu[i] -  dux[i] * factor);
         dmomyl[i] =  ddxl *( uynu[i] -  duy[i] * factor);
         detotl[i] =  ddxl *( eenu[i] -   dp[i] * factor);
     }
     else {
        factor = 0.5 *(1.0 - ddxl/dxnu[i1]);
        dmassl[i] = ddxl *(rhonu[i1] + drho[i1] * factor);
        dmomxl[i] = ddxl *( uxnu[i1] +  dux[i1] * factor);
        dmomyl[i] = ddxl *( uynu[i1] +  duy[i1] * factor);
        detotl[i] = ddxl *( eenu[i1] +   dp[i1] * factor);
     }
}
for (i = indent + 3; i < size_ind - 3; i++) {
```

```
        i1 = i + 1;
        rhonu[i] = (rhonu[i] * dxnu[i] + (dmassl[i] - dmassl[i1])) * dxinv;
         eenu[i] = ( eenu[i] * dxnu[i] + (detotl[i] - detotl[i1])) * dxinv;
         uxnu[i] = ( uxnu[i] * dxnu[i] + (dmomxl[i] - dmomxl[i1])) * dxinv;
         uynu[i] = ( uynu[i] * dxnu[i] + (dmomyl[i] - dmomyl[i1])) * dxinv;
    }
    for (i = indent + 3; i < size_ind - 3; i++) {
        tmp   = 1.0/rhonu[i];
        uxnu[i] =  uxnu[i] * tmp;
        uynu[i] =  uynu[i] * tmp;
        eenu[i] =  eenu[i] * tmp;
    }
    for (i = indent + 3; i < size_ind - 3; i++) {
        ei = eenu[i] - 0.5 * (uxnu[i] * uxnu[i] + uynu[i] * uynu[i]);
        eenu[i] = gamma1 * rhonu[i] * ei;
    }
    for (i = indent + 3; i < size_ind - 3; i++) {
        rho[i] = rhonu[i];
          p[i] =  eenu[i];
         ux[i] =  uxnu[i];
         uy[i] =  uynu[i];
    }
    return 0;
  }

int muscl_3_3D(int indent, real t, real dt, real gm, int nbdy,
                llint size, real coordl, real coordr,
                real *rho, real *p, real *ux, real *uy, real *uz,
                real *src, real *dt_burned,
                real *cournmx, real *scratch)
{
/** scratch is at least 20 * (size + nbdy  nbdy + 1) long. */

    int k, iter;
    llint size_ind, i, i1;

    real *drho, *rhol, *rhor, *dp,  *pl,  *pr;
    real *dux, *uxl, *uxr, *duy, *uyl, *uyr, *duz, *uzl, *uzr;
    real *cc, *courno, *rhonu, *uxnu, *uynu, *uznu, *eenu;
    real *dmassl, *dmomxl, *dmomyl, *dmomzl, *detotl;
    real *uxavl, *pavl, *dxnu, *ei;

    real mycourno, dx, dxinv, dtbydx, dtbydm;
    real ddxl, factor, tmp, sgm, dda, cs, s, eii, ee;
    real gaminv, gamma1, gm1inv, gammp1, hgamp1;
    real dpavl, wlfac, hrhol, wl, dpdul, wrfac, hrhor, wr, dpdur, ustrl, ustrr;
    real rho_lft, p_lft, ux_lft, gm_lft;
    real rho_rgt, p_rgt, ux_rgt, gm_rgt;
    real cdtbydx2, dheat;

    extern real small;
    extern int mype;
    extern real ff_forest_max_dt_burning, ff_igniting_e;
```

```
extern real ff_heat_conductivity;

i1 = size + nbdy + nbdy + 1;
size_ind = size + nbdy + nbdy - (llint) indent;

cc     = scratch;
courno = cc + i1;
pavl   = courno + i1;
uxavl  = pavl  + i1;

drho = uxavl + i1;
rhol = drho + i1;
rhor = rhol + i1;

dp = rhor + i1;
pl = dp + i1;
pr = pl + i1;

dux = pr  + i1;
uxl = dux + i1;
uxr = uxl + i1;

duy = uxr + i1;
uyl = duy + i1;
uyr = uyl + i1;

duz = uyr + i1;
uzl = duz + i1;
uzr = uzl + i1;

dxnu  = uzr  + i1;
ei = dxnu + i1;

dx = (coordr - coordl)/(real)size;
dxinv = 1.0/dx;
dtbydx = dt * dxinv;
cdtbydx2 = ff_heat_conductivity * dtbydx * dxinv;
gaminv = 1.0/gm;
gamma1 = gm - 1.0;
gm1inv = 1.0/gamma1;
gammp1 = gm + 1.0;
hgamp1 = 0.5 * gammp1;

for (i = indent; i < size_ind; i++) {
    ei[i] = gm1inv * p[i]/rho[i];
    cs = mysqrt(gm * p[i]/rho[i]);
    cc[i] = cs * rho[i];
    courno[i] = dtbydx * cs;
    mycourno = courno[i] + (dtbydx * myfabs(ux[i]));
    if (*cournmx < mycourno) {
        *cournmx = mycourno;
    }
}
```

```
getslope(indent, nbdy, size, rho, drho);
getslope(indent, nbdy, size, p,   dp);
getslope(indent, nbdy, size, ux,  dux);
getslope(indent, nbdy, size, uy,  duy);
getslope(indent, nbdy, size, uz,  duz);

for (i = indent + 1; i < size_ind - 1; i++) {
    i1 = i + 1;
    factor = 0.5 * (1.0 - courno[i]);

    dda = factor * drho[i];
    rhor[i]  = rho[i] - dda;
    rhol[i1] = rho[i] + dda;

    dda = factor * dp[i];
    pr[i]  = p[i] - dda;
    pl[i1] = p[i] + dda;

    dda = factor * dux[i];
    uxr[i]  = ux[i] - dda;
    uxl[i1] = ux[i] + dda;
}
for (i = indent + 2; i < size_ind - 1; i++) {
    tmp = 1.0/(cc[i-1] + cc[i]);
    pavl[i]  = tmp *(cc[i-1]*pr[i] +
                cc[i]*pl[i] - cc[i-1]*cc[i]*(uxr[i] - uxl[i]));
    uxavl[i] = tmp *(cc[i]*uxr[i] +
                      cc[i-1]*uxl[i] - (pr[i] - pl[i]));
    pavl[i]  = mymax (small, pavl[i]);
    iter = 0;
    while (iter < 0) {
         wlfac = gamma1 * pl[i]  +  gammp1 * pavl[i];
         wl = mysqrt (0.5 * rho[i-1] * wlfac);
         tmp = wlfac  -  hgamp1 * (pavl[i] - pl[i]);
         dpdul = wl * wlfac / tmp;
         wrfac = gamma1 * pr[i]  +  gammp1 * pavl[i];
         wr = mysqrt (0.5 * rho[i] * wrfac);
         tmp = wrfac - hgamp1 * (pavl[i] - pr[i]);
         dpdur = wr * wrfac / tmp;
         ustrl = uxl[i] -  (pavl[i] - pl[i]) / wl;
         ustrr = uxr[i] +  (pavl[i] - pr[i]) / wr;
         uxavl[i] = (dpdul/(dpdur + dpdul)) *
                      ustrl +(dpdur/(dpdur + dpdul)) * ustrr;
         pavl[i]  = pavl[i] - (ustrr - ustrl) *
                      (dpdur * dpdul/(dpdur + dpdul));
         pavl[i]  = mymax(small, pavl[i]);
         iter++;
    }
}
for (i = indent + 2; i < size_ind - 1; i++) {

    if (uxavl[i] >= 0.0) {
        tmp = courno[i] + uxavl[i] * dtbydx;
```

```
        }
        else {
            tmp = courno[i-1] - uxavl[i] * dtbydx;
        }
        if (*cournmx < tmp) *cournmx = tmp;
    }
    rhonu = rhor;
    eenu  = pr;
    uxnu  = uxr;
    uynu  = uyr;
    uznu  = uzr;

    for (i = indent + 2; i < size_ind - 2; i++) {
        i1 = i + 1;
        dxnu[i] = dx + dt *(uxavl[i1] - uxavl[i]);
        courno[i] = 1.0 - dxnu[i] * dxinv;
        dtbydm  = dtbydx/rho[i];

        rhonu[i] = (dx/dxnu[i]) * rho[i];
        uxnu[i] = ux[i] + dtbydm *(pavl[i] - pavl[i1]);
        if ((ei[i] >= ff_igniting_e) &&
            (dt_burned[i] < ff_forest_max_dt_burning)) {
            s = src[i] * dt;
            if (s > small) {
                dt_burned[i] += dt;
            }
        }
        else {
            s = 0.0;
        }
        ee = ei[i] + 0.5 *(ux[i] * ux[i] + uy[i] * uy[i] + uz[i] * uz[i]);
        eenu[i] = ee  + dtbydm *(pavl[i] *
                  uxavl[i] - pavl[i1] * uxavl[i1]) + s;
        dheat = cdtbydx2 *(ei[i1] - ei[i] - ei[i] + ei[i-1]);
        eenu[i] += dheat;

        uynu[i] = uy[i];
        uznu[i] = uz[i];
    }
    for (i = indent + 2; i < size_ind - 2; i++) {
        if (*cournmx < courno[i]) {
            *cournmx = courno[i];
        }
    }
    for (i = indent + 2; i < size_ind - 2; i++) {
        uxnu[i] *= rhonu[i];
        uynu[i] *= rhonu[i];
        uznu[i] *= rhonu[i];
        eenu[i] *= rhonu[i];
    }
    getslope1(indent + 2, nbdy, size, dxnu, rhonu, drho);
    getslope1(indent + 2, nbdy, size, dxnu, eenu,  dp);
    getslope1(indent + 2, nbdy, size, dxnu, uxnu,  dux);
```

```
getslope1(indent + 2, nbdy, size, dxnu, uynu,  duy);
getslope1(indent + 2, nbdy, size, dxnu, uznu,  duz);

dmassl = rhol;
dmomxl = uxl;
dmomyl = uyl;
dmomzl = uzl;
detotl =  pl;

for (i = indent + 3; i < size_ind - 2; i++) {
     i1 = i - 1;
     ddxl = dt * uxavl[i];

     if (ddxl <= 0.0) {
         factor = 0.5 * (1.0 + ddxl/dxnu[i]);
         dmassl[i] =  ddxl *(rhonu[i] - drho[i] * factor);
         dmomxl[i] =  ddxl *( uxnu[i] -  dux[i] * factor);
         dmomyl[i] =  ddxl *( uynu[i] -  duy[i] * factor);
         dmomzl[i] =  ddxl *( uznu[i] -  duz[i] * factor);
         detotl[i] =  ddxl *( eenu[i] -   dp[i] * factor);
     }
     else {
        factor = 0.5 *(1.0 - ddxl/dxnu[i1]);
        dmassl[i] = ddxl *(rhonu[i1] + drho[i1] * factor);
        dmomxl[i] = ddxl *( uxnu[i1] +  dux[i1] * factor);
        dmomyl[i] = ddxl *( uynu[i1] +  duy[i1] * factor);
        dmomzl[i] = ddxl *( uznu[i1] +  duz[i1] * factor);
        detotl[i] = ddxl *( eenu[i1] +   dp[i1] * factor);
     }
}
for (i = indent + 3; i < size_ind - 3; i++) {
    i1 = i + 1;
    rhonu[i] = (rhonu[i] * dxnu[i] + (dmassl[i] - dmassl[i1])) * dxinv;
     eenu[i] = ( eenu[i] * dxnu[i] + (detotl[i] - detotl[i1])) * dxinv;
     uxnu[i] = ( uxnu[i] * dxnu[i] + (dmomxl[i] - dmomxl[i1])) * dxinv;
     uynu[i] = ( uynu[i] * dxnu[i] + (dmomyl[i] - dmomyl[i1])) * dxinv;
     uznu[i] = ( uznu[i] * dxnu[i] + (dmomzl[i] - dmomzl[i1])) * dxinv;
}
for (i = indent + 3; i < size_ind - 3; i++) {
    tmp = 1.0/rhonu[i];
    uxnu[i] =  uxnu[i] * tmp;
    uynu[i] =  uynu[i] * tmp;
    uznu[i] =  uznu[i] * tmp;
    eenu[i] =  eenu[i] * tmp;
}
for (i = indent + 3; i < size_ind - 3; i++) {
    eii = eenu[i] - 0.5 * (uxnu[i] * uxnu[i] +
          uynu[i] * uynu[i] + uznu[i] * uznu[i]);
    eenu[i] = gamma1 * rhonu[i] * eii;
}
for (i = indent + 3; i < size_ind - 3; i++) {
    rho[i] = rhonu[i];
      p[i] =  eenu[i];
```

```
         ux[i] =  uxnu[i];
         uy[i] =  uynu[i];
         uz[i] =  uznu[i];
    }

    return 0;
  }



int getslope(int indent, int nbdy, llint size, real *rho, real *drho)
{
   /***
        rho[0:nbdy+size+nbdy)    : input
        drho[0:nbdy+size+nbdy)   :
            output, only drho[nbdy:nbdy+size) are meaningful
   ****/

    llint size_ind, i, ip, im;
    real ddrho, rhor, rhol, rhomax, rhomin;

    size_ind = nbdy + nbdy + size - indent - 1;
    for (i = indent + 1; i < size_ind; i++) {
        ip = i + 1;
        im = i - 1;
        ddrho = 0.25 *(rho[ip] - rho[im]);
        rhor  = rho[i] + ddrho;
        rhol  = rho[i] - ddrho;

        if (rho[i] >= rho[ip]) {
            rhomax = rho[i];
            rhomin = rho[ip];
        }
        else {
            rhomax = rho[ip];
            rhomin = rho[i];
        }
        if (rhor > rhomax) {
            rhor = rhomax;
        }
        else if (rhor < rhomin) {
            rhor = rhomin;
        }
        if (rho[i] >= rho[im]) {
            rhomax = rho[i];
            rhomin = rho[im];
        }
         else {
             rhomax = rho[im];
             rhomin = rho[i];
         }
         if (rhol > rhomax) {
             rhol = rhomax;
```

```
        }
        else if (rhol < rhomin) {
            rhol = rhomin;
        }
        if (rhor >= rhol) {
           ddrho = myfmin(rhor - rho[i], rho[i] - rhol);
        }
        else {
           ddrho = -myfmin(rhol - rho[i], rho[i] - rhor);
        }
        drho[i] = ddrho + ddrho;
    }
    return 0;
}

int getslope1(int indent, int nbdy,
              llint size, real *dx, real *a, real *da)
{
/*  non-uniform 1D grids*/

    llint size_ind, i, ip, im;
    real factor, dda, ap, am, amax, amin;

    size_ind = size + nbdy + nbdy - indent;

    for (i = indent + 1; i < size_ind - 1; i++) {
        ip = i + 1;
        im = i - 1;

        factor = dx[i] /(dx[ip] + dx[i] + dx[i] + dx[im]);
        dda = factor *(a[ip] - a[im]);
        ap = a[i] + dda;
        am = a[i] - dda;

        if (a[i] >= a[ip]) {
            amax = a[i];
            amin = a[ip];
        }
        else {
            amax = a[ip];
            amin = a[i];
        }
        if (ap > amax) {
            ap = amax;
        }
        else if (ap < amin) {
            ap = amin;
        }
        if (a[i] >= a[im]) {
            amax = a[i];
            amin = a[im];
        }
        else {
```

```
            amax = a[im];
            amin = a[i];
        }
        if (am > amax) {
            am = amax;
        }
        else if (am < amin) {
            am = amin;
        }
        if (ap >= am) {
           dda = myfmin(ap - a[i], a[i] - am);
         }
        else {
           dda = -myfmin(am - a[i], a[i] - ap);
        }
        da[i] = dda + dda;
    }
    return 0;
 }

#include "ff.h"
#include "files.h"
#include "mio.h"

int io_for_debug(char *name, int idump, int ndomain, ff_Domain *domains)
{
    int fileid, write_smesh;
    char myname[64];
    extern int mype;

    write_smesh = 1;
    sprintf(myname, "data/%s_domain_%04d", name, idump);
    if (mype == 0) {
        printf("creating %s ... \n", myname);
    }
    mio_create_file(myname, mio_file_create, mio_independent, &fileid);
    write_domains(write_smesh, fileid, ndomain, domains);
    mio_close_file(fileid);
    printf("%s closed.\n", myname);

    return 0;
 }

int write_domains(int write_smesh, int fileid, int ndomain, ff_Domain *domains)
{
    int b;
    ff_Domain *domain, **children;

    if (ndomain == 0) return 0;
    domain = domains;

    for (b = 0; b < ndomain; b++) {
        domain = domains + b;
```

```
        write_domain(write_smesh,fileid, domain);
    }
    return 0;
 }

int write_domain(int write_smesh, int fileid, ff_Domain *domain)
{
    /* ndomain_pe should be allocated and evaluated before the call */

    char name[32];
    char name_ei[] = "ei";
    int  nbdy, b, d, v, n, owner, rank, nvar, ncomp, writing;
    int ndomain_tot, ndomain, nc;
    llint i, j, k, e, offset;
    llint lsize, ncoord, nelem, *sizes;
    real gm1inv;
    real *ei, *rho, *p, *x[3], *c, *coordl, *coordr;
    real **coords;

    mio_Structured_Mesh   smesh;
    mio_Coord *coord;

    ff_Data_Type datatype;
    ff_Mesh_Var *vars, *var;
    int *comp_sizes, *which_comp, *my_comp_sizes, *my_which_comp;
    ff_Var_Comps *comps, *comp;

    mio_Mesh_Var my_var;
    mio_Var_Comps *my_comps, *my_comp;
    mio_Mesh_Var_Type  var_type;

    extern int npes, mype, dim;
    extern real small, gm;

    gm1inv = 1.0/(gm - 1.0);

    ei = NULL;
    if (write_smesh) {

        if (dim == 2) {
            var_type = mio_face;
        }
        else if (dim == 3) {
            var_type = mio_zone;
        }
        owner  = domain->owner;
        sizes  = domain->sizes;
        nbdy   = domain->nbdy;
        coordl = domain->coordl;
        coordr = domain->coordr;

        lsize = 1;
        for (d = 0; d < dim; d++) {
```

```
            lsize *= sizes[d];
        }
        if (!lsize) {
            printf("ERROR:  lsize = 0 in write_domain\n");
            return 0;
        }
        sprintf(name, "s%d_owner%d_id%d", 0, domain->owner, domain->id);
        mio_init(mio_smesh, -1, &smesh);
        smesh.name = name;
        smesh.dims = dim;
        smesh.element_centered = 0;
#if ifdouble == 1
        smesh.datatype = mio_double;
#else
        smesh.datatype = mio_float;
#endif
        for (b = 0; b < dim; b++) {
             d = dim - 1 - b;
             smesh.sizes[b] = sizes[d];
             smesh.nbdyl[b] = domain->nbdy;
             smesh.nbdyr[b] = domain->nbdy;
             smesh.dcoord[b] = (coordr[d] - coordl[d])/(real) sizes[d];
             smesh.coordmin[b] = coordl[d];
        }
        mio_write(mio_smesh, fileid, &smesh);

        nvar = domain->nvar;
        vars = domain->vars;

        for (v = 0; v < nvar; v++) {
            var  = vars + v;
            datatype = var->datatype;
            rank = var->rank;
            comp_sizes = var->comp_sizes;
            comps = var->comps;

            mio_init(mio_mesh_var, -1, &my_var);
            my_var.name = var->name;
            my_var.mesh_ids[0] = smesh.id;
            my_var.num_meshes = 1;
            if (var->type == ff_zone) {
                my_var.type = mio_zone;
            }
            else if (var->type == ff_face) {
                my_var.type = mio_face;
            }
            else {
                printf("ERROR:  var->type in io\n");
                return -1;
            }
            my_var.rank = rank;
            my_var.datatype = (mio_Data_Type) datatype;
```

```
            my_comp_sizes = my_var.comp_sizes;
            my_comps      = my_var.comps;

            ncomp = 1;
            for (d = 0; d < rank; d++) {
                ncomp *= comp_sizes[d];
                my_comp_sizes[d] = comp_sizes[d];
            }
            for (n = 0; n < ncomp; n++) {
                comp = comps + n;
                my_comp = my_comps + n;
                which_comp = comp->which_comp;
                my_which_comp = my_comp->which_comp;
                for (d = 0; d < rank; d++) {
                    my_which_comp[d] = which_comp[d];
                }
                my_comp->buffer = comp->buffer;
            }
            mio_write(mio_mesh_var, fileid, &my_var);
        }
        if (dim == 3) {
            lsize = 1;
            for (d = 0; d < dim; d++) {
                 lsize *= (nbdy + nbdy + sizes[d]);
            }
            ei = (real *) malloc(lsize * sizeof(real));
            rho = domain->rho3d[0][0];
            p   = domain->p3d[0][0];
            for (i = 0; i < lsize; i++) {
                ei[i] = gm1inv * p[i]/mymax(small, rho[i]);
            }
            mio_init(mio_mesh_var, -1, &my_var);
            my_var.name = name_ei;
            my_var.mesh_ids[0] = smesh.id;
            my_var.num_meshes = 1;
            my_var.type = mio_zone;
            my_var.rank = 0;
            my_var.datatype = (mio_Data_Type) datatype;
            my_comp_sizes = my_var.comp_sizes;
            my_comps      = my_var.comps;

            my_comp = my_comps;
            my_comp->buffer = ei;
            mio_write(mio_mesh_var, fileid, &my_var);
            free(ei);
        }
    }
    return 0;
}
```