FLASH UHD Documentation

Christopher C. Lindner, Miloš Milosavljević,

Department of Astronomy, University of Texas, 1 University Station C1400, Austin, TX 78712

ABSTRACT

The purpose of this documentation is to document the FLASH unsplit hydrodynamics solver and the changes needed to add radiation hydrodynamics to this solver.

1. Introduction

The latest version of the FLASH code is available in my directory
`/data1/r900-1/lindner/FLASH4.2.1/source/`

Most of the files relevant to the unsplit solver are in
`/data1/r900-1/lindner/FLASH4.2.1/source/physics/Hydro/HydroMain/unsplit/Hydro_Unspl`

Any file or directory names I mention will be relative to this directory.

Section 2 contains a basic overview flowchart of how the solver works. There are some brief descriptions of select files as well.

2. General Flowchart

Hydro Main FLASH hydro driver that calls the unsplit solver.

hy_uhd_unsplit This basically runs through the entire unsplit solver. It calls the following functions (`hy_uhd_` prefixes have been omitted):

putGravityUnsplit

getRiemannState Calculates and stores Riemann state values at cell interfaces so we can use these to compute fluxes, see MC 4.2.3

- First, some "'hybrid order"' things that we do not use (?).
- Then, slope flattening is carried out: MC 4.2.2 .
- Then, we start calculating Riemann states.
- **dataReconstOneStep** Evolves cell-centered values by $\Delta t/2$ at cell interfaces using PPM characteristic tracing. Most of this is carried out in the function below.

– **DataReconstructNomralDir_PPM** The driver of PPM and characteristic tracing. The steps described are as follows:

1. Calculate $\vec{\lambda}$, which is the vector of characteristic speeds.
2. Estimates the slope of the primitives and some additional variables at the interfaces by calling `hy_uhd_TVDslope` (see below). Projects $\bar{\Delta}$ (AKA `delbar`, *deltaq*), the estimated slope at each interface, to primitives (?).
3. Performs PPM Polynomial interpolation. Carries out $q_{i+\frac{1}{2}} = \frac{1}{2}(q_{i+1} + q_i) - \frac{1}{6}(\delta q_{i+1} - \delta q_i)$, which is equation 66 in MC. Note that they mention CW equation 1.9, which is equivalent to MC equation 67. I don't see this done anywhere, however. I think this might be for some sort of special case or higher order estimate or for use when you have a simpler version of δq, but I'm really not sure.
4. Interpolates species and mass scalars (which we don't use).
5. Carries out steepening (which is turned off in the setup we use).
6. Flattening. The flattening coefficent is actually set up in `hy_uhd_getRiemannState` and carried into this function. Flattening is described in Equations 74-80 in MC. This is applied to the species first, then to `vecL` and `vecR`, which are the left and right interface values, $q_{i-\frac{1}{2}}$ and $q_{i+\frac{1}{2}}$ in MC notation. Note that the form of the flattening matrix does involve pressures, so this may be something we need to look at down the road in our changes. MC equations 70a and 70b are carried out here.
7. Monotonicity Check. This is equivalent to equations 69a, 69b, and 69c of MC.
8. "Take initial guesses for the left and right states." This calculates Δq_i (`delW`) and q_{6i} (`W6`, note the typo in MC; there should not be a Δ here) as defined in MC Equations 72b and 72c. I describe these matrices a bit more in Section 3 below. There are some additional computations relating to the `Wp` and `Wm` arrays in the code. However, it turns out that all of these changes are overwritten later on, so these are apparently vestigial and not used. Everything in the sections labeled 7a, 7b, 7c, and 7d in FLASH can be ignored.
9. "Advance the above interpolated interface values by $\frac{1}{2}$ time step using characteristic Tracing." I have more detailed notes on what happens during this, but it's similar to MC 4.2.3 . See Section 3.
10. Finalizes the calculation of Riemann states?

These functions are called during this:

* **eigenParameters** Calculates several parameters needed for the wave speed eigenvalue calculations.
* **eigenValue** Calculates the eigenvalues (wave speeds) that fill the eigenvectors. Fills λ_0 (see Section 3).
* **eigenVector** Fills the **L** and **R** eigenvectors in primative or conservative form (see Section 3).

    - **upwindTransverseFlux** "This routine advances species by locally an Eulerian algorithm in an unsplit way." (sic) ??? See Section 3 of MC, especially Section 3.2. This computes `sig`, the "transverse flux vector."
    - **TVDslope** "This routine calculates limited slopes depending on the user's choice of the slope limiter." We use a `minmod` slope limiter. `delbar` (the vector $\bar{\Delta}$) is returned, which contains the limited slopes for primitive variables + `gamc`, `game`, and gravity. It is similar to, but not the same as some of the slope limiters described on pages 46 and 47 of MC.
  - Applies geometric terms.
  - Updates Gravity.
  - Stores scratch terms for each direction.
  - Applies transverse correction terms for 3D.
  - **upwindTransverseFlux**

getFaceFlux Computes fluxes at cell faces. Also calls specific solvers (e.g. **HLL**). Calculates min timestep and alters fluxes for artificial viscosity.

- **HLL** This is the most stable solver we've been using. It's in charge of computing the high-order Godunov fluxes based on the L and R Riemann states.
  1. Converts primitives to conserved variables for left and right states.
     prim2con Calculates conversions from primitive to conserved variables. The conserved variables are $\rho, \rho\vec{v}, E_{\text{tot}}$
     prim2flx Converts from variables to fluxes. $F = \rho v$, $F = \rho v v + P$, and $F = v(E+P)$.
  2. Calculates $F^* = [S_R F_L - S_L F_R + S_R S_L (U_R - U_L)] / (S_R - S_L)$

unsplitUpdate Given the fluxes, updates the conserved, cell-centered quantities. This is responsible for getting all the variables and geometric terms in the right form so it can call two subfunctions that are present in this file.

- **updateConservedVariable** Computes $U = U - \frac{\Delta t}{\Delta x}(F_R - F_L)$. See equations 53 and 63a in MC?
- **updateInternalEnergy** When flag `hy_useAuxEintEqn` is set, computes $\epsilon = \epsilon + \frac{\Delta t}{\Delta x}(F_L - F_R + P(F_L - F_R))$.

multiTemp/unsplitUpdateMultiTemp Updates energies for the special 3T conditions. See the FLASH manual for more information on the "Rage-like" approach used here. Some of the weird uhd crashes I've seen occur here. There are comments that explicitly state these crashes occur for "unknown reasons." This calls **hy_uhd_ragelike**.

energyFix Corrects energy in cases where the total energy is advected and `eint` can become negative. In the three temperature case, this step is actually handled by **unsplitUpdateMultiTemp**.

Grid_conserveFluxes

Eos_wrapped

putGravityUnsplit

addGravityUnsplit

energyFix

multiTempAfter This is only needed for when you are using MHD and three temperature, I think.

3. Characteristic Tracing

These are equations I was able to decipher from `hy_uhdDataREconstructNormalDir_PPM`. I'm going to focus on step 8, characteristic tracing, which is similar to MC section 4.2.3 .

$$\lambda_0 = \begin{pmatrix} v_x - c \\ v_x \\ v_x \\ v_x \\ v_x + c \end{pmatrix} \tag{1}$$

Where c is sound speed. This is computed in the `hy_uhd_eigenValue` function. This is the variable `lambda0` in the code and known as Λ in MC.

Our vector of primitive variables is

$$\mathbf{Q} = \begin{pmatrix} \rho \\ v_x \\ v_y \\ v_z \\ P \end{pmatrix}, \tag{2}$$

where P is the total pressure.

The left eigenvector (variable `leig0`) as defined in `hy_uhd_eigenVector` is

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ -\frac{1}{2}c & 0 & 0 & 0 & \frac{1}{2c} \\ 0 & -\rho & 0 & \rho & 0 \\ 0 & 0 & 0 & \rho & 0 \\ \frac{1}{2\rho c^2} & 0 & \frac{1}{c} & 0 & \frac{1}{2\rho c^2} \end{pmatrix} \tag{3}$$

The right eigenvector (variable `reig0`) is

$$\mathbf{R} = \begin{pmatrix} \rho & 0 & 1 & 0 & \rho \\ -c & 0 & 0 & 0 & c \\ 0 & 0 & -\frac{1}{\rho} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\rho} & 0 \\ \rho c^2 & 0 & 0 & 0 & \rho c^2 \end{pmatrix} \tag{4}$$

The array `delW` is

$$\Delta \mathbf{Q} = \mathbf{Q}_{\rm R} - \mathbf{Q}_{\rm L}, \tag{5}$$

where $Q_{\rm R}$ and $Q_{\rm L}$ are the right and left interface values of the primitive variables, which is equivalent to $\delta\mathbf{q}$ defined in equation 64 of MC.

The array `W6` is

$$\mathbf{Q}_6 = 6\mathbf{Q}_{\rm C} - \frac{1}{2}\left(\mathbf{Q}_{\rm R} + \mathbf{Q}_{\rm L}\right), \tag{6}$$

where $Q_{\rm C}$ is the cell-centered value of the primitive vector, $\mathbf{Q}$. This is equivalent to q_{6i}

Section 7a of `DataReconstructNormalDir_PPM` calculates

$$q_i = q_{i-\frac{1}{2}} - \lambda_{\max}\frac{\Delta t}{2\Delta x}\left[\Delta q_i - q_6 i\left(1 - \lambda_{\max}\frac{4}{3}\frac{\Delta t}{\Delta x}\right)\right]. \tag{7}$$

This is similar to, but deviates from equation 71 in MC. It turns out that this calculation is overwritten later in the code, so this is unused.

The final Riemann states are calculated via

$$\mathbf{Q}_{i-\frac{1}{2}}^{t+\frac{\Delta t}{2}} = \mathbf{Q}_i^t + \frac{\mathbf{Q}_{i6}}{12} + \sum_n L_n^*. \tag{8}$$

L_n^*, which is referred to as `vecL` in the code is

$$\begin{aligned} L_n^* &= \frac{1}{2}\left(-1 - \frac{\Delta t}{\Delta x}\lambda_0^n\right)\mathbf{R}^n\left(\mathbf{L}^n \cdot \Delta\mathbf{Q}\right) \\ &+ \frac{1}{4}\left[1 + 2\frac{\Delta t}{\Delta x}\lambda_0^n + \frac{4}{3}\left(\frac{\Delta t}{\Delta x}\lambda_0^n\right)^2\right]\mathbf{R}^n\left(\mathbf{L}^n \cdot -\mathbf{Q}_{i6}\right) \end{aligned} \tag{9}$$

where $\mathbf{R}^n$ is the nth column of $\mathbf{R}$. There is a corresponding form of these equations for the right face states as well.