Radhyd Outline

Initialization or Re-Initialization **radhyd**

Program **radhyd**

..........Initialization

- Call **initialize**

  1. Call **model_initialize**
     (a) Assign unit 'nread' to Data3/reset.d
     (b) Assign unit 'nprint' to Data3/superdump.d
     (c) Assign unit 'nrstd1' to Data3/rstdmp1.d
     (d) Assign unit 'nrstd2' to Data3/rstdmp2.d
     (e) Call **genrd**
        i. Assign values to array dimensions *nz*, *nez*, *nnu*, and *nnc*
        ii. Call **mgfld_array_initialize** Initialize the dimensions of the mgfld arrays
           A. Call **set_abem_arrays**
           B. Call **set_brem_arrays**
           C. Call **set_e_advct_arrays**
           D. Call **set_eos_snc_arrays**
           E. Call **set_incrmnt_arrays**
           F. Call **set_mdl_cnfg_arrays**
           G. Call **set_nu_dist_arrays**
           H. Call **set_pair_arrays**
           I. Call **set_scat_a_arrays**
           J. Call **set_scat_e_arrays**
           K. Call **set_scat_i_arrays**
           L. Call **set_scat_n_arrays**
           M. Call **set_scat_nn_arrays**
           N. Call **set_t_cntrl_arrays**
        iii. Initialize the dimensions of the non-mgfld arrays
        iv. Call **set_boundary_arrays**
        v. Call **set_convect_arrays**
        vi. Call **set_eos_bck_arrays**
        vii. Call **set_eos_ls_arrays**
        viii. Call **set_hydro_arrays**
        ix. Call **set_mgfld_remap_arrays**
        x. Call **set_psi0p_arrays**
        xi. Call **set_nucbrn_arrays**
        xii. Call **set_shock_arrays**

```
        xiii. Initialize q0
        xiv. Call read_init—Reads 'head' (problem description) and 'nrst' (cycle number)
        xv. If nrst = 0
            A. Call mgfld_var_initialize Initializes some of the mglfd variables
            B. Call init_var Initializes some of the non-mglfd variables
            C. Call mgfld_read—Reads in mglfd keys
                .....Open 'Data3/transport_keys.d'
                .....Call read_transport_keys
                .....Close 'Data3/transport_keys.d'
                .....Open 'Data3/edit_keys.d'
                .....Call read_edit_keys
                .....Close 'Data3/edit_keys.d'
            D. Call model_read—Reads in non-mglfd keys
                .....Open 'Data3/hydro_keys.d'
                .....Call read_hydro_keys
                .....Close 'Data3/hydro_keys.d'
                .....Open 'Data3/initial_model.d'
                .....Call read_initial_model
                .....Close 'Data3/initial_model.d'
        xvi. If nrst /= 0
            A. Call reset_var—Initializes nse
            B. Call readst—Reads in restart file from rstdmp1 or rstdmp2
            C. Call readst—Reads in changes from reset.d
    (f) Close unit 'nrstd1'
    (g) Close unit 'nrstd2'
    (h) Close unit 'nread'
    (i) Call genst_hy
        i. Sets quantities at inner edge of configuration
        ii. Compute rest masses of zones assuming Newtonian description
        iii. Loads equation of state tables
2. If lagr /= 0
   .....lagrangian= .true. if jj = 1 and kk = 1
   ELSE
   .....lagrangian= .false. END IF lagr /= 0
3. Check that 1D arrays are large enough to accommodate all 3 sweeps
4. Set number of zones per PE
5. Call evh1_load
    (a) Set EVH1 globals
    (b) Set the geometry and boundary conditions
    (c) Get radial limits from MGFLD
    (d) IF rezn = ye
        .....IF lagrangian
        ..........Call lagregrid
        ..........Load radial grid into sweep arrays, offseting for ghosts
```

..........Call **coord_bc**—Set coordinates in ghost zones of sweep arrays

..........Call **volume**—Calculate zone volumes

..........Call **paraset**—Set up coefficients for parabola subroutine

..........Set jm and jnumax to imax+1

.....IF not lagrangian

..........Call **eulregrid**

..........Set jnumax & jm to maximum zone number

.....END IF lagrangian

.....Put state variables into 1D arrays, padding with 6 ghost zones

.....Load EVH1 boundary conditions from MGFLD

.....Call **e_compose**—Compute the total energy e(n)

.....Reload values changed by rezoning into MGFLD variables

.....Build a j grid

.....Build a k grid

.....Set transverse velocities initially to zero

ELSE iF rezn = no

.....Set imax to maximum zone number

.....Call **mgfld_to_evh1_restart**—Load evh1 arrays

i. Load quantities from MGFLD to EVH1 arrays
ii. Put radial grid and sweep arrays, offsetting for ghosts
iii. Call **coord_bc**—Set coordinates in ghost zones of sweep arrays
iv. Call **volume**—Calculate zone volumes
v. Call **paraset**—Set up coefficients for parabola subroutine
vi. Load EVH1 boundary conditions from MGFLD
vii. Put state variables into 1D arrays, offsetting for ghosts
viii. Compute the total energy e(n)

END IF rezn

6. Call **mgfld_setup**
   (a) Call **genst** if nrst = 0, otherwise Call **genrst**
   (b) Call **time_step_select**
   (c) Call **mgfld_reset**
   (d) Call **mgfld_edit**
7. Call **mgfld_to_evh1**
   (a) Load arrays zte, zei, and zye
8. Call **svel_init**

..........Problem Cycling—Program **radhyd**

- Set $nmin = 7, \quad nmax = imax + 6$
- Call **cycle**
  1. Update cycle number
  2. Print cycle number to **'Data3/cycle.d'**

- Zero increment variables
- Call **evh1_evolve_xy**—Perform hydro step with x-sweeps preceding y-sweeps
  1. Compute and print the total energy
  2. Save pre-Lagrange step variables in __i arrays for mgfld (xxx make these multi-D arrays)
  3. Call **sweepx**
     (a) Set $nmin = 7, \quad nmax = imax + 6, \quad ntot = imax + 12$
     (b) Loop over j, k (i.e., different (radial) rays)
     (c) Put state variables for a given radial ray into 1D arrays, padding with 6 ghost zones
     (d) Load grid coordinates in xa0, dxa0, xa, and dx
     (e) Call **eos_result** (Computes T from ei, and then p, s, and $gamma$)
     (f) Call **sweepbc**
         i. Call **coord_bc** (Loads ghost coordinates with boundary coordinates)
         ii. Load ghost zones with state variables
     (g) Call **volume** (Computes volume elements)
     (h) Call **paraset** (Updates parabolic coefficients with initial grid coordinates for later use in obtaining parabolic interpolants of flow variables inside each grid zone)
     (i) Call **e_compose** (Computes $e(n)$ from $ei(n)$, $ekin(n)$, and $egrav(n)$, which is needed for subroutine **evolve** to advance the total energy)
     (j) Call **ppm**
         i. Call **flatten** (Calculate flattening coefficients for smoothing near shocks)
         ii. Call **parabola** (Computes parabolic interpolants for flow variables in each grid zone)
         iii. Call **states** (Integrate parabolae over causal domain to get input states for Riemann problem)
         iv. Call **riemann** (Obtain the zone face averages, $umid$ and $pmid$)
         v. Call **evolve** (Lagrangian update is performed—mass conservation (ρ update), momentum conservation (u update; energy conservation (e update))
            A. Grid positions are updated from $umid$ and dt
            B. Call **zone_center** (Calculates volume averaged zone centers, which are used to calculate external forces)
            C. Call **forces** (Calculate forces using zone-centered coordinates at t (0) and at t+dt (1))
            D. Calculate dvolume and average area based on geometry of sweep
            E. Update the density from the new zone positions
            F. Update the velocity due to pressure gradients and forces
            G. Update the energy due to net work performed on the zone surfaces by pressure and by the zone-centered forces times the zone-centered displacement
            H. Call **sweepbc** (Grid change requires updated boundary conditions
               .....Call **coord_bc** (Loads ghost coordinates with boundary coordinates)
               .....Load ghost zones with state variables
            I. Call **paraset** (Grid change requires updated parabolic coefficients
            J. Call **e_decompose** (Extract the internal energy from the updated total energy)

K. Call **etotal** (Total fluid energy check)

L. Compute and store *dei*, the internal energy change due to the hydro step

(k) **sweepx** $\leftarrow$

(l) Save Lagrangian updated variables (except T) in _l arrays for mgfld

(m) For a Lagrangian run, updated the coordinates to *zxa*, *xdz*, *zxc*

4. **evh1_evolve_xy** $\leftarrow$
5. Call **sweepy** (This has not yet been interfaced with MGFLD)

- **radhyd** $\leftarrow$
- Call **evh1_to_mgfld_hydro** (Loads results of EVH1 Lagrangian hydro (*roi*, *ye0i*, *t0i*, *rol*, *dei*, *u0l*, *x0l*) to advance temperature and compute pseudoviscosities)
    1. Call **hydro_t_change** (Compute temperature change from *dei* and store in $dtmpmn(j, 1)$)
    2. Call **mgfld_hydro**
        (a) Call **pseudo** Computes pseudoviscosities for editting purposes
        (b) Call **snuc** Updates composition from nuclear reactions
        (c) Call **nsetest** Flashes or deflashed zones, as appropriate
- Ramp up an explosion, if criteria are satisfied
- Call **evh1_to_mgfld_transport** (Loads arrays _i into arrays for transfer to MGFLD transport)
    1. $roi \rightarrow rhop$
       $t0i \rightarrow tp$
       $ye0i \rightarrow yep$
       $u0i \rightarrow up$
       $xai \rightarrow rp$
       $psi0p$ from **psi0p$_\mathbf{m}$odule**
       $dtnphn\_aetr \rightarrow dtime$
    2. Call **mgfld_transport_in**
        (a) $rhop \rightarrow rho$
            $tp \rightarrow t$
            $yep \rightarrow ye$
            $up \rightarrow u$
            $psi0p \rightarrow psi0$
            $up \rightarrow u$
            compute dr
            compute $dmrst$, $rstmss$ $rho \rightarrow rhoa$
            $t \rightarrow ta$
            $dr \rightarrow dra$
            $r \rightarrow ra$
            $u \rightarrow ua$
            $dtime \rightarrow dtnphn\_aetr$
        (b) Call **mgfld_transport** (Do the neutrino transport)
            i. Call **agr_nu_cal** (Transfer the lapse functions to transport variables (GR runs))

ii. Call **enu_cal** (Updates the neutrino bin energies on the basis of the new lapse functions (GR runs))

iii. Call **gamgr_nu_call** (Transfers the GR gammas for transport modules (GR runs))

iv. Call **extrap** (At the moment, a dummy subroutine)

v. $r \rightarrow r\_nu$
$rho \rightarrow rho\_nu$
$t \rightarrow t\_nu$
$ye \rightarrow ye\_nu$
$ra \rightarrow ra\_nu$
$rhoa \rightarrow rhoa\_nu$
$ta \rightarrow ta\_nu$
$yea \rightarrow yea\_nu$

vi. Call **eddington** (Compute Eddington factors)

vii. Call **comvcf_cal** (Compute the stress-energy coupling (for editing purposes)

viii. Call **w_cal** (Update the relativistic enthalpy using *rhoa_nu*, *ta_nu*, and *ye_nu* (GR runs))

ix. Call **agr_cal** (Update the lapse functions *rhoa_nu*, *ta_nu*, *ye_nu*, and *ra_nu* (GR runs))

x. Call **agr_cal_cal** (Transfer the lapse functions to transport variables (GR runs))

xi. Call **gamgra_nu_call** (Transfers the GR gammas for transport modules (GR runs))

xii. Call **enua_cal** (Updates the neutrino bin energies on the basis of the new lapse functions (GR runs))

xiii. Transfer updated neutrino energies, lapse functions, and GR gammas into original arrays
$unuea \rightarrow unue$
$dunuea \rightarrow dunue$
$unubea \rightarrow unube$
$unua \rightarrow unu$
$dunua \rightarrow dunu$
$unuba \rightarrow unub$
$ncoefaa \rightarrow ncoefa$
$ecoefaa \rightarrow ecoefa$
$ecoefaea \rightarrow ecoefae$
$agra_nu \rightarrow agr_nu$
$agrajmh_nu \rightarrow agrjmh_nu$
$gamgra_nu \rightarrow gamgr_nu$

xiv. Call **eqstz** (Update thermodynamic quantities for transport)

xv. Call **nu_adv** Perform the source and transport step

   A. Transfer variables from calling statement to transport module
   $r\_in \rightarrow ra\_nu$
   $rho\_in \rightarrow rhoa\_nu$
   $t\_in \rightarrow ta\_nu$
   $ye\_in \rightarrow yea\_nu$

```
B. Call **pre_trans** (Compute $area$, $vol$, $dr$, $drjmh$, $c\_r$, $c\_e$)
C. Call **nu_abemtr** (Advance neutrino occupation probabilities due to emission,
   absorption and transport)
   .....Initialize increment variables $dye\_emabtr$, $dt\_emabtr$, $dt\_emabtrk$, $dye\_emabtrk$,
   $dpsi_emabtr$
   .....$ta\_nu \rightarrow t_nu_0 \quad yea\_nu \rightarrow ye\_nu\_0$
   .....Call **dtau_aetr** (Get time step for transport)
   .....$psi0 \rightarrow psi0\_0$
   .....$psi0\_0 \rightarrow psi0\_i$
   .....$psi1 \rightarrow psi1\_i$
   .....$t\_nu\_0 \rightarrow ta\_nu$
   .....$t\_nu\_0 \rightarrow t\_nu\_i$
   .....$ye\_nu\_0 \rightarrow yea\_nu$
   .....$ye\_nu\_0 \rightarrow ye\_nu\_i$
   .....Iterate
   ..........$t\_nu\_i \rightarrow ta\_nu$
   ..........$ye\_nu\_i \rightarrow yea\_nu$
   ..........Call **c_psi_set** (Update thermodynamic quantities for transport)
   ...............Update absorption and emission scattering rates
   ...............Update isoenergetic scattering rates
   ...............Compute inverse mean free paths
   ...............Compute diffusion coefficients
   ...............Compute $psi1\_i$
   ...............Compute $d_ye$, $d\_ye\_t$, $d\_ye\_ye$, $d\_ye\_psi$, $d\_t$, $d\_t\_t$, $d\_t\_ye$, $d\_t\_psi$
   ..........Call **pre_c_psi_set** (Implement switches $iyenu$, $itnu$)
   ..........Call **a_psi_set** (Compute recursion coefficients)
   ..........Call **psi_bd** (Implement boundary conditions)
   ..........Call **d_sub** (Compute $dpsi0\_iph(j)$, this iterations increment of $pis0$)
   ..........$psi0\_i + dpsi0\_iph \rightarrow psi0\_ip1$
   ..........$dpsi\_emabtr + dpsi0\_iph \rightarrow dpsi_emabtr$
   ..........$t\_nu\_i + cf\_t + cf\_t\_psi(j,k) \times dpsi0\_iph \rightarrow t\_nu\_i$
   ..........$ye\_nu\_i + cf\_ye + cf\_ye\_psi \times dpsi0\_iph \rightarrow ye\_nu\_i$
   ..........$dt\_emabtrk + cf\_t + cf\_t\_psi \times dpsi0\_iph \rightarrow dt\_emabtrk$
   ..........$dye\_emabtrk + cf\_ye + cf\_ye\_psi \times dpsi0\_iph \rightarrow dye\_emabtrk$
   ..........Test for convergence
   .....End iteration
   .....$t\_nu\_0 \rightarrow ta\_nu$ (Restore initial value to $ta\_nu$
   .....$ye\_nu\_0 \rightarrow yea\_nu$ (Restore initial value to $yea\_nu$
   .....$dt\_emabtr + dt\_emabtrk \rightarrow dt\_emabtr$ (Add changes to $t$ due to all energy
   zones)
   .....$dye\_emabtr + dye\_emabtrk \rightarrow dye\_emabtr$ (Add changes to $ye$ due to all
   energy zones)
   .....$psi0 + dpsi\_emabtr \rightarrow psi0$ (Update $psi0$)
   .....Compute $psi1$
   .....$dc \rightarrow dcr$
D. $dye\_emabtr \rightarrow dye$
E. $dt\_emabtr \rightarrow dtmpnn$
```

        F. Call **nu_scat** (Advance neutrino occupation probabilities due to inelastic scattering and pair production)

   xvi. **mgfld_transport** $\leftarrow$
   xvii. Call **t_adv** (Update the temperatures)
   xviii. Call **ye_adv** (Update the electron fractions)
   xix. Switch arrays
   $r \rightarrow rr$
   $ra \rightarrow r$
   $ra\_nu \rightarrow r\_nu$
   etc.
   xx. Call **eqstz** (Recompute thermodynamic functions)
   xxi. Call **gammaz** (Recompute thermodynamic gammas)
   xxii. Call **nu_stress** (Compute neutrino stresses)

(c) **mgfld_transport_in** $\leftarrow$

(d) Call **mgfld_reset** (Reset thermodynamic and rate tables)
   i. Call **eqstt** (Compute the internal energy prior to resetting tables)
   ii. Call **esrgnz** (Reset EOS tables)
   iii. Call **eqstz** (Recompute thermodynamic quantities)
   iv. Call **gammaz** (Recompute thermodynamic gammas)
   v. Call **abemset, bremset, scataset, scateset, scatiset, scatnnset, scatnset, pairset** (Recompute rate tables)
   vi. Call **nucset** (Recompute nuclear reaction rate tables)

(e) **mgfld_transport_in** $\leftarrow$

3. **evh1_to_mgfld_transport** $\leftarrow$

- **radhyd** $\leftarrow$
- Call **mgfld_transport_to_evh1**

1. Call **mgfld_transport_out** (Load results of MGFLD transport into arrays for export)
   (a) $ta \rightarrow tp$
   $yea \rightarrow yep$
   $aesv \rightarrow ep$
   $psi0 \rightarrow psi0p$
   $stress \rightarrow nu\_stress$ (Combine stresses of different neutrino flavors)
2. **mgfld_transport_to_evh1** $\leftarrow$
3. $tp \rightarrow zte$
   $ep \rightarrow zei$
   $yep \rightarrow zye$
   $nu\_stress \rightarrow znu\_str$

- **radhyd** $\leftarrow$
- Call **evh1_to_mgfld_e_advct** (Load variables for export to energy advection)

1. $r0i \rightarrow rhop$
   $r0l \rightarrow rhoap$
   $t0i \rightarrow tp$
   $zte \rightarrow tap$
   $ye0i \rightarrow yep$
   $zye \rightarrow yeap$
   $u0i \rightarrow up$
   $xai \rightarrow rp$
   $xal \rightarrow rap$
   $dtnphn\_aetr \rightarrow dtime$
2. Call **mgfld_nu_energy_advct_in** (Export variables to neutrino energy advection modules)
   (a) $rhop \rightarrow rho$
       $rhoap \rightarrow rhoa$
       $tp \rightarrow t$
       $tap \rightarrow ta$
       $yep \rightarrow ye$
       $yeap \rightarrow yea$
       $up \rightarrow u$
       $psi0p \rightarrow psi0$
       $rp \rightarrow r$
       $rap \rightarrow ra$
       Compute $dmrst$ and $rstmss$
       $ua \rightarrow u$
       $dtime \rightarrow dtj$
   (b) Call **nu_energy_advct** (Perform the energy advection step)
      i. $r\_in \rightarrow r\_nu$
         $ra\_in \rightarrow ra\_nu$
         $rho\_in \rightarrow rho\_nu$
         $rhoa\_in \rightarrow rhoa\_nu$
         $t\_in \rightarrow t\_nu$
         $ta\_in \rightarrow ta\_nu$
         $ye\_in \rightarrow ye\_nu$
         $yea\_in \rightarrow yea\_nu$
      ii. Call **eddington** (Compute Eddington factors)
      iii. Call **comvcf_cal** (Compute the stress-energy coupling (for editing purposes)
      iv. Call **w_cal** (Update the relativistic enthalpy using rho_nu, t_nu, and ye_nu (GR runs))
      v. Call **agr_cal** (Update the lapse functions rho_nu, t_nu, ye_nu, and r_nu (GR runs))
      vi. Call **agr_cal_cal** (Transfer the lapse functions to transport variables (GR runs))
      vii. Call **enu_cal** (Compute the neutrino bin energies))
      viii. Call **nu_U** (Compute the initial neutrino energy per unit mass)
      ix. Call **w_cal** (Update the relativistic enthalpy using $rhoa\_nu$, ta_nu, and ye_nu (GR runs))

x. Call **agr_cal** (Update the lapse functions *rhoa_nu*, *ta_nu*, *ye_nu*, and *ra_nu* (GR runs))

xi. Call **agra_cal_cal** (Transfer the lapse functions to transport variables (GR runs))

xii. Call **enua_cal** (Compute the neutrino bin energies)

xiii. Call **e_advct** (Perform the neutrino energy advection)

xiv. $psi0\_a \rightarrow \psi 0$ (Restore updated neutrino distribution to psi0 array)

xv. Call **rebal** (Prevent overfilling of neutrinos states)

xvi. Call **nu_Ua** (Compute the final neutrino energy per unit mass)

xvii. Call **nu_stress** (Update the neutrino stresses)

3. **mgfld_nu_energy_advct_in** $\leftarrow$

- **evh1_to_mgfld_e_advct** $\leftarrow$
- Call **mgfld_e_advct_to_evh1**
  1. Call **mgfld_nu_energy_advct_out**
     (a) $nu\_stress \rightarrow znu\_str$
     (b) $psi0p \rightarrow$ **psi0p_module**
- **evh1_to_mgfld_e_advct** $\leftarrow$
- Call **mgfld_to_evh1** (Load undated T, ye, and ei for EVH1)
- Loop over y and z and put x variables in 1D arrays, padding with 6 ghost zones
- Call **pre_remap_psi** (Prepare for $psi0$ remap)
  1. Divide $psi0$'s by Larangian updated rho and store in array $psi0_re$, padding with 6 ghost zones)
  2. Put initial (Lagrangian updated) and final (Eulerian) grids, rho's and $psi0$'s into arrays, padding with 6 ghost zones)
  3. Call **coordbc_psi** (Find grid coordinates of ghost zones)
  4. Call **CALL sweepbc_r** (Fill rho ghost zones with boundary values)
- Call **remap_psi**
  1. Call **paraset** (Updates parabolic coefficients for later use with Lagrangian grid coordinates in obtaining parabolic interpolants of flow variables inside each grid zone)*****
  2. Call **parabola** (Computes parabolic interpolants for rho and $psi0$ in each grid zone)
  3. Calculate the volume of the overlapping subshells
  4. Integrate over the parabolic profiles to calculate the total mass and neutrino number in the overlapping subshells
  5. Calculate the volumes before and after remap
  6. Advect $psi0$ by moving the subshell quantities into the appropriate Eulerian zone
  7. Restore $psi0$ by multiplying remapped $psi0_re$ by remapped rho

- Call **pre_remap_comp** (Prepare for composition remap)

  1. Put initial (Lagrangian updated) and final (Eulerian) grids, *rho*'s into arrays, padding with 6 ghost zones)
  2. Find nse - nonnse boundary
  3. Put initial (Lagrangian updated) *xn*'s into array *comp*, padding with 6 ghost zones)
  4. Call **coordbc_psi** Find grid coordinates of ghost zones
  5. Call **sweepbc_r** (Fill *rho* ghost zones with boundary values) *****
  6. Fill left and right ghosts with boundary *rho*'s and *comp*'s

- Call **remap_comp** (Remap composition)

  1. Call **paraset** (Updates parabolic coefficients for later use with Lagrangian grid coordinates in obtaining parabolic interpolants of flow variables inside each grid zone)*****
  2. Call **parabola** (Computes parabolic interpolants for *rho* and *comp* in each grid zone)
  3. Calculate the volume of the overlapping subshells
  4. Integrate over the parabolic profiles to calculate the total mass and mass of each specie in the overlapping subshells
  5. Calculate the volumes before and after remap
  6. Advect *comp* by moving the subshell quantities into the appropriate Eulerian zone
  7. Restore *comp* to *xn*

  1. Add six ghost zones to *psi*0 array, and divide by the updated density
  2. Find ghost coordinates

- Call **remap** (Remap flow variables)

  1. Call **paraset** (Updates parabolic coefficients for later use in obtaining parabolic interpolants of flow variables inside each grid zone)*****
  2. Call **parabola** (Computes parabolic interpolants for flow variables in each grid zone)
  3. Call **e_compose** (Computes $e(n)$ from $ei(n)$, $ekin(n)$, and $egrav(n)$, which is needed remap the total energy)
  4. Call **parabola** (Computes parabolic interpolants for e variables in each grid zone)
  5. Calculate the volume of the overlapping subshells
  6. Calculate the total mass (fluxr), etc. in overlap
  7. Advect mass, etc. by moving the subshell quantities into the appropriate Eulerian zone
  8. Reload Eulerian grid coordinates
  9. Call **paraset** (Updates parabolic coefficients for later use with Eulerian grid coordinates in obtaining parabolic interpolants of flow variables inside each grid zone)
  10. Call **e_decompose** (Extract the internal energy from the total energy)

- Call **eos_result** (Computes T from ei, and then p, s, and $gamma$)

- Fill in multi-D arrays, *zro*, etc.

- Put updated values into zone arrays (__e) for **mgfld_edit**
- Call **evh1_to_mgfldseted** (Transfer remapped variables to mgfld variables (*rho*, *t*, *ye*, etc.) for edit)
- Call **mgfld_edit** (Edit)
- Call **mgfld_terminate** (Examine termination criteria)
- Call **time_step_select** (Compute timestep for next cycle)
- **Repeat**, but call **sweepy** before **sweepx**