User Guide Part 5: Creating Your Own Application

Author: John Hayes

The Problem Generator Subroutine

A new application is created via the construction of a problem generator subroutine which is identified in subroutine **setup** by the CPP macro "PROBLEM". During compilation, the C preprocessor searches for the string PROBLEM and replaces it with the name of the appropriate subroutine specified in the Makefile via "-DPROBLEM=..." in the ZMP_CPP command line. The tar file is shipped with the Makefile written to define **blast** as the subroutine of choice. A second macro, PROBRES, defines the name of a subroutine to be called if this application is to be continued from a restart file. As an example, I will discuss the **rshock** subroutine as a simple example of how to construct an application.

Module Declaration

Because this version of ZEUS-MP follows a basic Fortran 90 programming model, F77-style INCLUDE statements have been dropped in favor of F90 USE statements which refer to modules that serve the analogous function of the INCLUDE files in F77 codes. The modules referenced by these USE statements are collected in the source file mod_files.F, which is the first file processed during compilation. Of the modules listed in rshock.F, **real_prec, config, param, grid, field, root, bndry, and mpipar** should be regarded as mandatory for your generator regardless of the physics set you wish to include. The CPP #ifdef-#else-#endif construction enclosing **mpiyes and mpino** must also be included. The **radiation and opac** modules are needed if radiation is to be included in your problem.

Looking underneath the module list in rshock.F, you will notice what I regard as the most important line in any Fortran code:

implicit NONE

Always, ALWAYS, **ALWAYS** include this line in every fortran subroutine you write! This is the single most effective bug-filter available to you. End of editorial.

Reading the PGEN Namelist

The first task of the Problem Generator subroutine is naturally to read the user-customized PGEN namelist containing all parameters unique to the application. The READ statement is enclosed by a Fortran IF-ENDIF construct which tells the code that, in the case of a parallel calculation, only the "root" process (myid_w = 0) should execute the READ statement. Inside the Fortran IF-ENDIF is a CPP #ifdef-#endif construct that hides MPI-specific code from the compiler if the macro MPI_USED is not defined in the Makefile. If this macro has been defined, then the enclosed code ensures that data read by the root process is broadcast to all the other processes in a parallel calculation.

Initializing Field Variables and Boundary Variables

Once the PGEN namelist has been read, the main job of the Problem Generator routine is to initialize ZEUS-MP's field arrays for density ("d"), gas energy density ("e"), and the three components of velocity ("v1," "v2," and "v3"). If magnetic fields were included, one would also need to specify initial values of b1, b2, and b3. Because this problem includes radiation, we need to initialize "er," the radiation energy density.

This problem also makes use of prescribed constant boundary values along the outer 1-boundary. In either of the two zmp_inp files that can be used with the **rshock** problem generator, zmp_inp.subrshk and zmp_inp.suprshk (*sub*critical and *super*critical radiating shocks, respectively), the OIB boundary value namelist specifies that "inflow" boundary conditions are to be applied on the outer I boundary (nois(1)=nois(2)=3). Setting nois(1)=3 means that the hydro module will use prescribed values of density, gas energy density, and velocity at the outer boundary. Setting nois(2)=3 further specifies that the radiation module will also rely on a prescribed value for er in the outer boundary cells.

The outer boundary values for density, gas energy, and 3 velocity components are fed into the code directly from the zmp_inp file via fois(1), fois(2), and fois(3-5), respectively. The value of fois(2) is interpreted by **rshock** as a material temperature from which the boundary value of gas energy is derived. Rather than specifying the boundary radiation energy in zmp_inp, subroutine **rshock** computes fois(12) internally from the value for gas temperature. Because the initial model is constructed in radiative equilibrium, the gas and radiation temperatures will be identical, so it makes sense to compute fois(12) inside **rshock** rather than specifying it independently in zmp_inp.

With the field variables and required boundary conditions determined, the problem is initialized. The file rshock.F contains a second subroutine, **rshockres**, which would be called in place of **rshock** if the calculation were picked up from a restart file. When a calculation is begun from scratch, subroutine **mstart** calls subroutine **setup**, with then calls subroutine PROBLEM (your problem generator) to configure the application. When a calculation is begun from a restart dump (irestart=1 in namelist rescon), subroutine **mstart** calls **restart** in place of **setup**; subroutine **restart** calls subroutine PROBRES -- your problem-restart routine -- in place of PROBLEM. As with the macro PROBLEM, PROBRES is specified in the ZMP_CPP line in the Makefile.