Kunming School 2018: Lab assignments day 2

Rony Keppens & Jannis Teunissen

1 Installing and running MPI-AMRVAC

We will today start using the general purpose code `MPI-AMRVAC`, which is written in Fortran, is parallelized using MPI, solves generic sets of (nonlinear, especially hyperbolic) PDEs, and has the capability to adjust the grid resolution dynamically, according to the evolving solution of the PDEs itself: a technique called Adaptive Mesh Refinement or AMR. The code is a modern successor to the Versatile Advection Code (hence VAC), and its versatility lies in the fact that it can handle any-dimensional simulations (1D, 2D or 3D, or things in between like 1.5, 1.75, 2.5D computations), as well as Cartesian, cylindrical, polar or spherical coordinate systems. The code is freely available from `github`, and is extensively documented on the dedicated website http://amrvac.org.

Follow the instructions from this website to install and test `MPI-AMRVAC`. The requirements are that you are on a Unix-based system (any desktop or laptop with a Linux operating system will do, so your Mac as well), which has a Fortran 90 compiler installed (`gfortran` works fine), and this Fortran compiler must be used in the MPI version you have available (`openMPI` works). You also rely on the availability of `perl`, which is standard on Unix platforms, and for maintaining and updating the code we need `git`.

If you have all these prerequisites, just follow the **installation instructions** http://amrvac.org/md_doc_installation.html.
After this, you can do your first testrun, by following the **Getting Started** instructions at
http://amrvac.org/md_doc_getting_started.html.
This first testrun will produce 2D data of an advection problem, the data is best visualized using `Paraview` or `Visit`. This software can be freely downloaded from
https://www.paraview.org
https://wci.llnl.gov/simulation/computer-codes/visit

2 Advect yourself with MPI-AMRVAC

The advection equation in 2D just solves

$$\frac{\partial \rho}{\partial t} = -\mathbf{v}\cdot\nabla\rho = -v_x\frac{\partial \rho}{\partial x} - v_y\frac{\partial \rho}{\partial y}$$

where we solve for $\rho(x,y,t)$ under a given constant velocity field $\mathbf{v} = (v_x, v_y)$. It simply advects whatever profile $\rho_0(x,y) = \rho(x,y,t=0)$ you prescribe at

$t = 0$ in the constant direction defined by $\mathbf{v}$, and with the given speed. The test you just ran took a speed $\mathbf{v} = (1, 1)$, and was advecting a discontinuous initial profile $\rho_0(x, y)$ which resembles the letters `VAC`, on a 2D domain, using doubly periodic boundary conditions. This means that this logo will cycle around the diagonal of the chosen domain $[0, 1] \times [0, 1]$ for as long as you care to simulate. Deviations from the initial profile tell you how much error the scheme produces: it may diffuse the discontinuities, may cause errors near strong gradients, etc.

We now ask you to advect a picture of yourself (or your favorite teacher or moviestar) through a 2D or a 3D domain. This will be done by solving the same advection equation, prescribing a purely horizontal flow speed $\mathbf{v} = (1, 0)$ (2D)or $\mathbf{v} = (1, 0, 0)$ (3D), and feed the picture as a 'density' profile (noting that it may contain negative values as well, so density has no physical meaning here) from the $x = 0$ boundary. In 2D, you then use it as if you are faxing a picture through the fax machine. In this manner, you are in fact prescribing spatio-temporal boundary conditions at the line where $x = 0$. In 3D, you prescribe the picture as a time-independent image that will be advected through the 3D domain. We will always use a domain with unit side lengths, so perform the simulation on $[0, 1] \times [0, 1]$ in 2D, or on the cube $[0, 1]^3$ in 3D.

In the amrvac folder `tests/rho/bc_from_vtk`, you find a python script called `picture_to_vtk.py` which you can use to convert your favorite image (any `*.jpg` will do) to a `*.vtk` file. This file will need to be changed in 2 ways, once to serve as the 2D spatio-temporal, and once to serve for the 3D time-independent boundary. You will use the `mod_usr.t` and `amrvac.par` example provided in the folder `tests/rho/bc_from_vtk`, where the example is set up in 3D, and adjust it to first do your own 3D advection, and next modify it to the 2D time-dependent BC variant.

If you succeeded in doing this on a uniform grid (in both 2D and 3D), and you still have free time, then modify the `mod_usr.t` and your `amrvac.par` to ensure you use a finer grid, but only for the center part of the image, by coding up a `usr_refine_grid` subroutine.