Enzo Homework

Preliminaries

Through this tutorial and homework, we're going to explore different aspects of running Enzo simulations as well as analyzing them and digging into data using yt. Both Enzo and yt have documentation online:

- http://lca.ucsd.edu/projects/enzo/wiki/UserGuide
- http://yt.enzotools.org/doc/

as well as attendant websites:

- http://lca.ucsd.edu/projects/enzo/wiki/
- http://yt.enzotools.org/

where many different aspects of their use and extension can be found. For the most part, links to relevant sections have been provided within this document, but you should feel free to explore both sets of documentation.

Before we can get started, we need to ensure that a working Enzo and yt installation are set up. For the purposes of the workshop, we're going to try to rely on an installation of Enzo and yt on Apollo so that we can maximize our time exploring and experimenting.

Running on Apollo

On Apollo, you can source the file */home/pitp097/enzo.sh* to get all of the appropriate paths and environment variables. You can then submit jobs using the Apollo queue. The following script will submit a very simple Enzo job, using the Enzo binary precompiled for Apollo. (Note that you *must source /home/pitp097/enzo.sh* first!):

```
#!/bin/bash
#$ -N enzo-sedov
#$ -pe orte 4
#$ -cwd
#$ -V
#$ -R y

mpirun /home/pitp097/packages/bin/enzo.exe -d SedovBlastAMR.enzo
```

For the nested cosmology run (below) you can up the number of processes as appropriate. However, for most of these simulations, four should be more than enough.

Installing Enzo

If, however, you would like to install Enzo, there are a couple ways, and we'll provide some hints to doing so. Specifically, if you're going to run on a cluster or a supercomputing center, it's almost certainly much better to use an existing Make.mach file than it is to write your own; the Enzo-1.5 distribution

comes with Makefiles for most major supercomputing installations, and Peter Teuben has provided a makefile for the Apollo cluster at PiTP.

Information about the Enzo installation procedure is described in detail on the wiki. For the purposes of the workshop, you will *only* need enzo.exe, inits.exe and yt. You do not need ring.exe or hop or any of the other components.

OS X

Attached are statically linked binaries of both enzo.exe and inits.exe for OSX Leopard. It's recommended that if you have trouble compiling, you use these.

If you choose to install on OS X, you're most likely installing on your laptop. The Makefile that comes with Enzo for OS X (Make.mach.darwin) is designed to be run using binary-packaged Fortran compilers from http://hpc.sf.net/ and HDF5 compiled without any command line arguments, using the GNU toolchain and the system installation of OpenMPI.

For OS X, the GNU toolchain is not only sufficient, but probably the best and easiest solution. You do *not* need the Intel compilers, any version of MPI other than what OS X provides (and places in `/usr/local/`) and any special arguments to the HDF5 compilation.

Ubuntu

The Make.mach.ubuntu-hardy file describes how to apt-get the necessary packages for Hardy Heron and should work out of the box.

Installing yt

If you are running and analyzing data on a machine without an existing yt installation (Apollo has one), unless you are conversant in Python installation and file layouts, using the installer script will probably be the easiest way to install yt.

There are two, one for Linux and one for OS X. Matt Turk wrote these, and he'll be more than happy to help you get yt working. To run the linux installation script::

```
$ svn export http://svn.enzotools.org/yt/branches/yt-
1.5/doc/install_script.sh
$ nano install_script.sh # turn options on and off
$ bash install_script.sh
```

The second step lets you turn things on and off. For the most part, leaving the options as in the default is probably okay.

For OS X, the installation is a bit different as a result of the fundamentally single-user nature of the OS X 'Framework' system. This script should work, btu there will be points where it might appear to hang as .dmg files are mounted and executed.

```
$ svn export http://svn.enzotools.org/yt/branches/yt-
1.5/doc/install_script_osx.sh
$ nano install_script.sh # turn options on and off
$ bash install_script_osx.sh
```

Again, the default installation options are probably okay.

When done, yt will say hello and tell you how to start it up. Give it a go and see if you can get the help screen for the command-line plotter::

```
$ yt help
```

Getting Started with Enzo

AMR Sedov Blast

Our first steps with Enzo will be with a relatively simple hydro problem that will demonstrate some of the things at which Enzo excels. Specifically, we're going to run a Sedov blast wave in two dimensions and allow the code to insert higher-resolution grid patches as necessary -- we're going to watch the adaptive mesh refinement process.

The Enzo test problem page has a comprehensive list of the test problems that come with Enzo, including a set of data against which you can compare your own results.

We'll be running the test problem entitled SedovBlastAMR, which you can find either online on the Enzo website or in the Enzo distribution at `doc/examples/SedovBlastAMR.enzo`. This test problem sets up a Sedov blast starting in the center of a 2D domain (100x100 cells) and allows it to evolve until roughly the time that it touches the boundary. You should skim the parameter file and look up any parameters that aren't clear to you in the Enzo parameter list . This test problem requires no initial conditions that are not specified in the parameter file. It can be executed on a laptop with::

```
$ mpirun -np 2 ./enzo.exe SedovBlastAMR.enzo
```

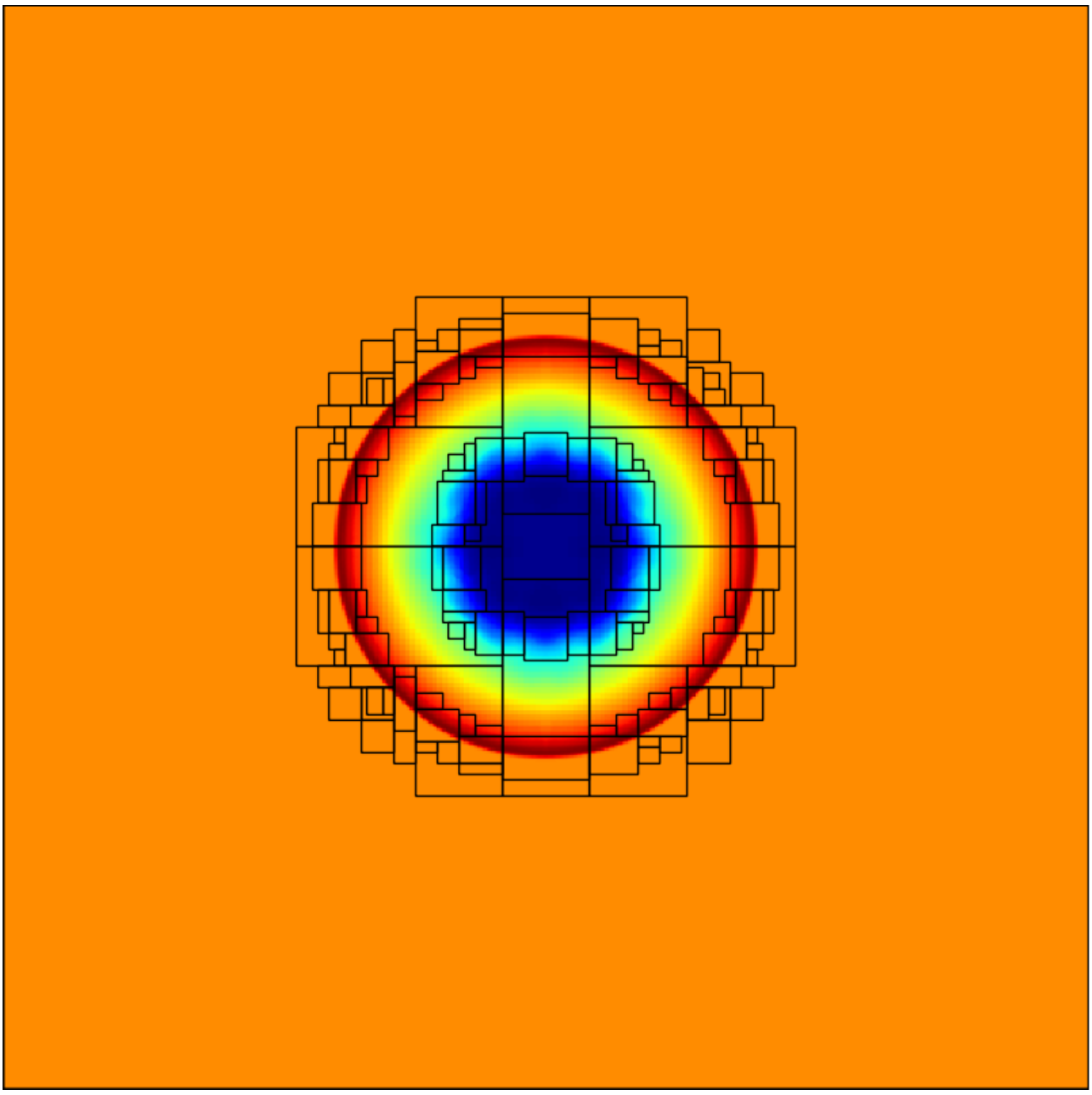
But for submission to a queue, the "mpirun -np 2" can be left off and the queueing system will handle the MPI execution. This test problem takes a few minutes to run to completion on my laptop, but you can start to see results relatively quickly, as it will output every few cycles.

To plot this data, you can either write and run a script, or use the command-line yt tool with the 'plot' subcommand. The options available to you are available through the '--help' command::

```
$ yt plot --help
```

For instance, the attached image was created using the command::

```
$ yt plot -f Density -a 2 --show-grids DD0004/sb_L2x2_0004
```



The first argument, '-f Density' means to plot the "Density" field. The second means to slice along the z-axis (axes are 0 for x, 1 for y, 2 for z -- since we're in a 2D simulation, we'll always want to slice along z.) Finally, we ask yt to show the grid boundaries so we can see where the refinement is happening, and we tell it the name of the output to plot. Once it's plotted it, it will save the image to a png and give you the filename.

Alternatively, scripts that can be run through yt can be written. A simple Python script to do the above would be as follows::

```
from yt.mods import *
pf = load("DD0004/sb_L2x2_0004")
pc = PlotCollection(pf, center=[.5, .5, .5])
p = pc.add_slice("Density", 2)
p.modify["grids"]()
```

```
pc.save("DD0004")
```

This is a bit more verbose, but hidden in there is that more complicated things can be done -- for instance, velocity vectors can be overlaid. (This is a sedov blast, so they are kind of fun to see!) This would be done with `p.modify["velocity"]()`)

Collapsing Spheres and Collisions

To extend upon our initial steps experimenting with the Sedov Blast problem, there are a few other test problems that might be fun. Specifically, the CollapseTest problem will let us explore some fun things Enzo can do. Even *more* specifically, it'll help us take a look at gravity.

We'll explore CollapseTest in two stages. The first will just set up a moving sphere and allow it to collapse -- this is the vanilla, standard parameter file CollapseTest that comes with Enzo. This problem type will set up between 0 and 10 spheres, optionally refined to some higher level, of varying types. You can give these spheres different characteristics -- velocity, density, temperature -- and then watch as they collapse under their own weight.

Collapsing in Isolation

You should run with the test problem parameter file first, to get a feel for how long a run will take. You can watch the sphere collapse to as high a level as you like, which is governed by the `MaximumRefinementLevel` parameter in the parameter file. Note also that the parameter governing frequency of data output (`dtDataDump`) is set quite high in the example parameter file; you might consider dividing it by ten or even a hundred for more frequent output. Additionally, note that the parameter `TopGridDimensions` is set to only 16^3. Feel free to change this to something higher -- 64^3 should still run quite comfortably on a laptop.

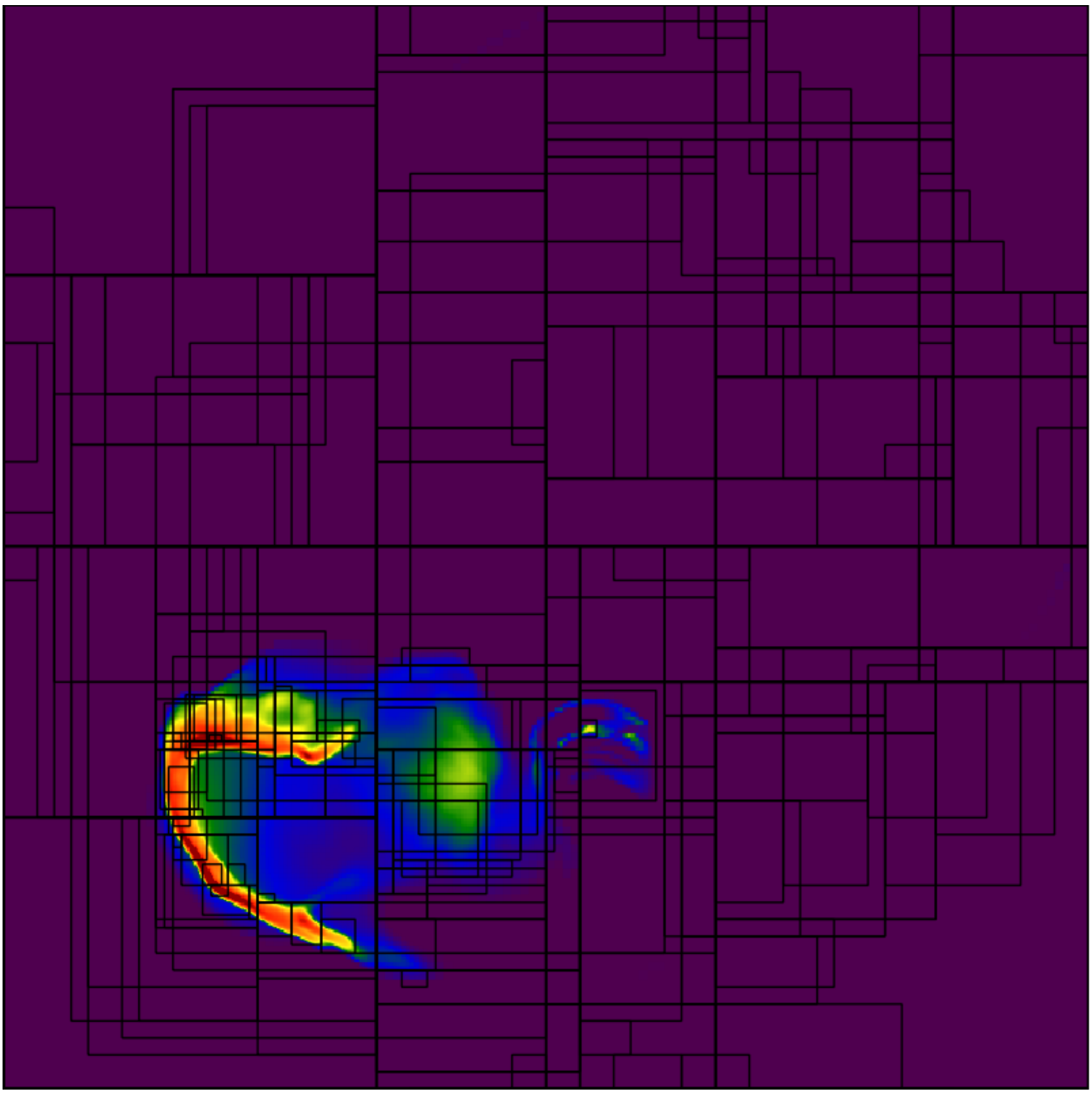
The attached parameter file `CollapseTest.enzo`, which is taken from the Enzo distribution, will run this test. Plot this and see how it looks!

Collapsing and Collisions

The next stage will be to add a second sphere, and see if you can get them to run into each other. An example parameter file for doing this has been attached (`SphereCollision.enzo`), along with some plots of the process, but you should feel free to play with the parameter file yourself and explore different mechanisms and methods for colliding the two spheres. There are a couple tricks, however -- one is that the refine region in the sample parameter file is confined to the center of the box. Another is that the maximum refinement level might be a bit too high; perhaps try 4 or 5.

An image has been attached that shows what my modification to this parameter file looks like after 16 outputs, made using the command::

```
$ yt plot --colormap=bds_highcontrast -z 1e-26 1e-24 -g Density -
f Density \
      -p --show-grids DD0016/moving7_0016
```



The `-p` switch means to project, and `-g Density` means to take the average with a weight of `Density`. This helps us to see the average value of our projected quantity along the line of sight. We also change the colormap and fix the colorbar limits.

Cosmology

Enzo is a fully cosmological hydrodynamic code and has been used to simulate many different cosmological phenomena -- from the formation of the first stars and galaxies to large scale structure and the attendant observational implications.

To get started with cosmology in Enzo, we're going to be doing two sets of cosmological simulations. The first will be relatively straightforward, with no adaptive mesh refinement. The second will being with nested initial conditions and will follow structure formation to several levels of refinement.

Unigrid

The first calculation we will perform will be a so-called 'unigrid' calculation. The initial conditions are generated at a uniform, fixed resolution, and Enzo will not add higher-resolution elements.

To start, we have provided some sample parameter files for both inits and enzo. (`UnigridCosmologySimulation.inits` and `UnigridCosmologySimulation.enzo`.) Inits is the Enzo cosmological initial conditions generator, which accepts specifications for the power spectrum, a random seed and some physical characterics of the generated initial conditions. We'll be generating a 64^3 simulation. You should feel free to choose a random seed -- 10 digits, negative (which happens to be the length of a phone number!) -- that will determine the initial seed from which the simulation will be generated.

The file `UnigridCosmologySimulation.inits` is a good starting place. Be sure to modify the random seed and to choose a length scale for the calculation; the default here is 4 Mpc, but you might find you're interested in larger scales.

To run inits, you just have to execute it as follows::

```
$ ./inits.exe -d UnigridCosmologySimulation.inits
```

This simulation will output quite a bit of data -- on the order of 2.0 gigabytes. These will be separated into "DataDumps" (DD) and "RedshiftDumps" (RD). Redshift-based output is triggered by the reaching of a given redshift by Enzo; datadumps are triggered by a fixed amount of time. These are governed by the parameters `CosmologyOutputRedshift[N]` where N is some integer and the parameter `dtDataDump=` some float value. The value for dtDataDump is in code units of time; code units of time are generated by a combination of various parameters, described on the Enzo wiki. For a 4.0 Mpc box with roughly LCDM parameters, 1.0 corresponds to about 20 million years.

Attached is a movie generated with yt, using some slightly more advanced techniques, such as adding an annotation to the plot to show the current redshift. (This was generated with outputs taken every 40 million years or so.) The script that created the frames for this movie is called `annotate_projections.py` and can simply be run inside your simulation directory.

yt has a built in halo finder; you can use this from the command line by running::

```
$ yt hop RD0009/RD0009
```

(or on any parameter file, not just RD0009) and it will run the halo finder "hop" and output the results in a text file. The halo finder in yt also presents and object-interface, so more complicated analysis can be performed. (To see an example of this, check the yt documentation.)

Nested-Grid

We'll now try running a nested grid calculation, where we initialize our domain with a second, higher-resolution grid embedded in the coarse grid. Additionally, we will allow our calculation to refine. Because of the increased time that this simulation will take to run, along with the increased memory requirements, it will have to be run on multiple processors, with increased memory requirements. Either four or eight processors should work, but if you have access to sixteen it should be able to domain decompose very well across that number. Note that to help with computational time, I've increased the box size by a factor of ten. This reduces the overhead from the Courant condition, and allows the code to take bigger timesteps.

Some parameter files have been provided (`AMRCosmologySimulation_2L.enzo` for the enzo parameter file, and `AMRCosmologySimulation_L0.inits` and `AMRCosmologySimulation_L1.inits` for inits.) Note that the generation of nested initial conditions can be a bit tricky; we don't do any recentering here, but if you wanted to, you'd have to familiarize yourself with the Enzo inits parameter file format. Also,if you use my initial condition, there's no real halo of consequence inside the nested grid. (Bonus problem: run once at unigrid, center on the most massive halo using inits parameters.) So recentering or choosing a different random seed might be extremely useful.

To generate our initial conditions, we have to tell inits that we are running a nested-grid calculation -- this ensures that our particles are not duplicated on multiple levels. We can run inits with::

```
$ ./inits.exe -d -
s AMRCosmologySimulation_L1.inits AMRCosmologySimulation_L0.inits
$ ./inits.exe -d AMRCosmologySimulation_L1.inits
```

The first should run relatively quickly, and the second might take a minute or two. For more information about inits and how to run it, along with how to set up inits parameter files, see RunningInits. The parameter files supplied generate a topgrid of 128^3 cells and particles, and then a centered, nested grid of 96^3 cells and particles.

Using the attached parameter file `AMRCosmologySimulation_2L.enzo`, run Enzo on this set of initial conditions. If you run on 4 processors, it will probably take a couple hours -- but the problem should scale nicely, if you'd like to run on more. If you'd like to add another level of refinement, or change the size of the nested grid, the Enzo wiki has more information about how to write nested-grid parameter files.

Once done, we're going to start flexing our analysis muscles. Here are a list of things you might feel like examining, all of which can be done with yt.

- How does the halo mass function evolve with redshift?
- At the final output, what percentage of the baryonic material in the box is enclosed in halos?
- How does the ionization fraction (HII_Fraction) evolve with redshift?
- Does this change if only material inside halos contributes to the average?

(For some of these, you might find some hints in yt recipes that have yet to be integrated into the main documentation at http://hg.enzotools.org/cookbook/)

There are many more quantities and studies that can be conducted, even on this simple problem set. Explore!

Appendix: yt tricks

yt is written in Python, which means it's very easy to extend and builds on a large base of packages. As a result, there are some fun features in yt that you might enjoy.

The Pastebin

Often, when running into errors, the error message must be copied and pasted between the user and the developer or mailing list. yt will automatically paste error messages, if you ask it to, to a pastebin (http://paste.enzotools.org/). Just supply the `--paste` option on the command line.

Physical Regions

yt is structured around the idea of identify physical regions and then performing analysis on them. This means you can grab a 'sphere' of data centered on a point and examine it; you can grab a cylinder, a rectangular prism, a ray, and several other types of data as well. But, in addition to this, you can extract subsets of these regions -- and then join subsets together, too.

Clump Finding

With yt, connected sets can be identified. This enables the location and analysis of a hierarchy of clumps; in star formation, for instance, this allows the construction of diagrams describing the density at which fragmentation occurs.