Introduction

Here you will find a step by step guide to downloading, configuring, and running the Einstein Toolkit. You may use this tutorial on a workstation or laptop, or on a supported cluster. Configuring the Einstein Toolkit on an unsupported cluster is beyond the scope of this tutorial. If you find something that does not work, please feel free to mail users@einsteintoolkit.org (mailto:users@einsteintoolkit.org).

Prerequisites

When using the Einstein Toolkit on a laptop or workstation you will want a number of packages installed in order to download, compile and use the Einstein Toolkit components. If this is a machine which you control (i.e. you have root), you can install using one of the recipes that follow:

On Mac, please first install MacPorts, if you do not have it installed already (https://www.macports.org/install.php) (https://www.macports.org/install.php)). Next, please install the following packages: using the command:

```
sudo port -N install pkgconfig gcc7 openmpi fftw-3 gsl jpeg zli
b hdf5 +fortran +gfortran openssl
```

On Debian/Ubuntu/Mint use this command:

```
sudo apt-get install -y subversion gcc git numactl libgsl-dev l
ibpapi-dev python libhwloc-dev make libopenmpi-dev libhdf5-open
mpi-dev libfftw3-dev libssl-dev liblapack-dev g++ curl gfortran
patch pkg-config libhdf5-dev libjpeg-turbo?-dev
```

On Fedora use this command:

```
sudo dnf install -y libjpeg-turbo-devel gcc git lapack-devel ma
ke subversion gcc-c++ which papi-devel python hwloc-devel openm
pi-devel hdf5-openmpi-devel openssl-devel libtool-ltdl-devel nu
mactl-devel gcc-gfortran findutils hdf5-devel fftw-devel patch
 gsl-devel pkgconfig
module load mpi/openmpi-x86_64
```

You will have to repeat the `module load` command each time you would like to compile or run the code.

On Centos use this command:

```
sudo yum install -y epel-release
sudo yum install -y libjpeg-turbo-devel gcc git lapack-devel ma
ke subversion gcc-c++ which papi-devel hwloc-devel openmpi-deve
l hdf5-openmpi-devel openssl-devel libtool-ltdl-devel numactl-d
evel gcc-gfortran hdf5-devel fftw-devel patch gsl-devel
```

On OpenSuse use this command:

```
sudo zypper install -y curl gcc git lapack-devel make subversio
n gcc-c++ which papi-devel hwloc-devel openmpi-devel libopenssl
-devel libnuma-devel gcc-fortran hdf5-devel libfftw3-3 patch gs
l-devel pkg-config
```

Download

A script called GetComponents is used to fetch the components of the Einstein Toolkit. GetComponents serves as convenient wrapper around lower level tools like git and svn to download the codes that make up the Einstein toolkit from their individual repositories. You may download and make it executable as follows:

Note: By default, the cells in this notebook are Python 3 commands. However, lines that begin with the ! character are run inside a bash shell. Bash commands to set environment variables or change the working directory will not work if executed with ! (their effects are forgotten immediately). Therefore, when setting the directory a special magic %cd command is used. If you wish to run these commands outside the notebook and in a bash shell, cut and paste only the characters following the initial ! or %.

In [1]:
```
%cd ~/
```

```
/home/jovyan
```

In [2]:
```
!curl -kLO https://raw.githubusercontent.com/gridaphobe/CRL/ET_2018_09/G
!chmod a+x GetComponents
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100   99k  100   99k    0     0   408k      0 --:--:-- --:--:-- --:--:
--  406k
```

GetComponents accepts a thorn list as an argument. To check out the needed components:

In [3]:
```
!./GetComponents --parallel https://bitbucket.org/einsteintoolkit/manif
```

```
      from repository: https://bitbucket.org/einsteintoolkit/einstein
base.git (https://bitbucket.org/einsteintoolkit/einsteinbase.git)
                 into: Cactus/arrangements
```

In [4]:
```
%cd ~/Cactus
```

```
/home/jovyan/Cactus
```

Configure and build

The recommended way to compile the Einstein Toolkit is to use the Simulation Factory ("SimFactory").

Configuring SimFactory for your machine

The ET depends on various libraries, and needs to interact with machine-specific queueing systems and MPI implementations. As such, it needs to be configured for a given machine. For this, it uses SimFactory. Generally, configuring SimFactory means providing an optionlist, for specifying library locations and build options, a submit script for using the batch queueing system, and a runscript, for specifying how Cactus should be run, e.g. which mpirun command to use.

In [6]:
```
!./simfactory/bin/sim setup-silent
# If you are on Mac, uncomment and use the next line instead:
# !./simfactory/bin/sim setup-silent --optionlist osx-macports.cfg --ru
```

```
Here we will define some necessary Simulation Factory defaults.


Determining local machine name: jupyter-gabella

--------------------SUMMARY--------------------:


[default]
user            = jovyan
email           = jovyan
allocation      = NO_ALLOCATION


------------------END SUMMARY------------------:

Contents successfully written to /home/jovyan/Cactus/repos/simfactory
2/etc/defs.local.ini
```

After this step is complete you will find your machine's default setup under ./simfactory/mdb/machines/<hostname >.ini You can edit some of these settings freely, such as "description", "basedir" etc. Some entry edits could result in simulation start-up warnings and/or errors such as "ppn" (processor-per-node meaning number of cores on your machine), "num-threads" (number of threads per core) so such edits must be done with some care.

Building the Einstein Toolkit

Assuming that SimFactory has been successfully set up on your machine, you should be able to build the Einstein Toolkit with the command below. The option "--mdbkey make 'make --j2'" sets the make command that will be used by the script. The number used is the number of processors used when building. Even in parallel, this step may take a while, as it compiles all the thorns specified in manifest/einsteintoolkit.th.

Note that the "cat" command on the end of the line is to prevent problems with the display in Jupyter when output comes from multiple threads.

Note: Using too many threads to compile on the test machine may result in compiler failures.

In [7]:
```
!./simfactory/bin/sim build --mdbkey make 'make -j2' --thornlist ../ein
```

```
Compiling /home/jovyan/Cactus/arrangements/PITTNullCode/SphericalHarm
onicRecon/src/util/printtime.c
Compiling /home/jovyan/Cactus/arrangements/PITTNullCode/SphericalHarm
onicRecon/src/util/readmeta.c
Creating Riemann1d in /home/jovyan/Cactus/exe/sim from /home/jovyan/C
actus/configs/sim/build/GRHydro/Riemann1d.o
Creating ascii_output in /home/jovyan/Cactus/exe/sim from /home/jovya
n/Cactus/configs/sim/build/SphericalHarmonicRecon/ascii_output.o
Creating fftwfilter in /home/jovyan/Cactus/exe/sim from /home/jovyan/
Cactus/configs/sim/build/SphericalHarmonicRecon/fftwfilter.o
Creating setmeta in /home/jovyan/Cactus/exe/sim from /home/jovyan/Cac
tus/configs/sim/build/SphericalHarmonicRecon/setmeta.o
Creating findlast in /home/jovyan/Cactus/exe/sim from /home/jovyan/Ca
ctus/configs/sim/build/SphericalHarmonicRecon/findlast.o
Creating printtime in /home/jovyan/Cactus/exe/sim from /home/jovyan/C
actus/configs/sim/build/SphericalHarmonicRecon/printtime.o
Creating readmeta in /home/jovyan/Cactus/exe/sim from /home/jovyan/Ca
ctus/configs/sim/build/SphericalHarmonicRecon/readmeta.o
Done.
```

Running a simple example

You can now run the Einstein Toolkit with a simple test parameter file.

In [8]:
```
!./simfactory/bin/sim create-submit helloworld \
    --parfile arrangements/CactusExamples/HelloWorld/par/HelloWorld.par
```

```
Parameter file: /home/jovyan/Cactus/arrangements/CactusExamples/HelloW
orld/par/HelloWorld.par
Skeleton Created
Job directory: "/home/jovyan/simulations/helloworld"
Executable: "/home/jovyan/Cactus/exe/cactus_sim"
Option list: "/home/jovyan/simulations/helloworld/SIMFACTORY/cfg/Optio
nList"
Submit script: "/home/jovyan/simulations/helloworld/SIMFACTORY/run/Sub
mitScript"
Run script: "/home/jovyan/simulations/helloworld/SIMFACTORY/run/RunScr
ipt"
Parameter file: "/home/jovyan/simulations/helloworld/SIMFACTORY/par/He
lloWorld.par"
Assigned restart id: 0
Warning: Total number of threads and number of cores per node are inco
nsistent: procs=1, ppn-used=4 (procs must be an integer multiple of pp
n-used)
Executing submit command: exec nohup /home/jovyan/simulations/hellowor
ld/output-0000/SIMFACTORY/SubmitScript < /dev/null > /dev/null 2> /de
v/null & echo $!
Submit finished, job id is 27931
```

The above command will submit the simulation to the queue naming it "helloworld" and ask for a 5 minutes long job time, if you are running on a cluster, or run it immediately in the background if

you are on a personal laptop or workstation. You can check the status of the simulation with the command below. You can run this command repeatedly until the job shows

```
[ACTIVE (FINISHED)...
```

as it's state. Prior to that, it may show up as QUEUED or RUNNING.

In [10]:
```
!simfactory/bin/sim list-simulations helloworld
```

```
Warning: job status is U
   helloworld              [ACTIVE (FINISHED), restart 0000, job id 27
931]
```

Once it finished you can look at the output with the command below.

In [11]:
```
!simfactory/bin/sim show-output helloworld
```

```
Simulation name: helloworld
Warning: job status is U
Warning: job status is U
Warning: Job is not running, cannot retreive exechost
================================================================================
==========
The job's Formaline output is:
================================================================================
==========
(file does not exist)
================================================================================
==========
The job's stdout is:
================================================================================
==========
(file does not exist)
================================================================================
==========
The job's stderr is:
================================================================================
==========
(file does not exist)
================================================================================
==========
```

If you see

```
INFO (HelloWorld): Hello World!
```

anywhere in the above output, then congratulations, you have successfully downloaded, compiled and run the Einstein Toolkit! You may now want to try some of the other tutorials to explore some interesting physics examples.

Running single star simulation

What follows is the much more computationally intensive example of simulating a static TOV star. We will first use two sed commands to strip this down from a production quality run to something more manageable. Note that we use 2 procs for this simulation, which may be more than your machine supports. If that is the case, please adjust procs and ppn-used accordingly.

In [12]:
```
# modify parameter file for smaller memory footprint using sed by changi
!sed '/CoordBase::d[xyz]/s/8/12/' < par/static_tov.par > par/static_tov_
# Shorten the run time to 250
!sed '/cctk_final_time/s/1000/250/' <  par/static_tov_small.par >  par/s
```

In [13]:
```
# start simulation, watch log output
!./simfactory/bin/sim create-run static_tov \
  --parfile=par/static_tov_small_short.par --procs=2 --num-threads=1 --
```

```
Parameter file: /home/jovyan/Cactus/par/static_tov_small_short.par
Skeleton Created
Job directory: "/home/jovyan/simulations/static_tov"
Executable: "/home/jovyan/Cactus/exe/cactus_sim"
Option list: "/home/jovyan/simulations/static_tov/SIMFACTORY/cfg/Optio
nList"
Submit script: "/home/jovyan/simulations/static_tov/SIMFACTORY/run/Sub
mitScript"
Run script: "/home/jovyan/simulations/static_tov/SIMFACTORY/run/RunScr
ipt"
Parameter file: "/home/jovyan/simulations/static_tov/SIMFACTORY/par/st
atic_tov_small_short.par"
Simulation name: static_tov
Assigned restart id: 0
Running simulation static_tov
+ Preparing:
set -e
+ cd /home/jovyan/simulations/static_tov/output-0000-active
+ echo Checking:
+ Checking:
pwd
+ /home/jovyan/simulations/static_tov/output-0000-active
hostname
jupyter-gabella
+ date
Thu Mar 14 16:27:58 UTC 2019
+ echo Environment:
+ Environment:
export CACTUS_NUM_PROCS=2
+ export CACTUS_NUM_THREADS=1
+ export GMON_OUT_PREFIX=gmon.out
+ export OMP_NUM_THREADS=1
+ env
+ sort
+ echo Starting:
Starting:
+ date +%s
+ export CACTUS_STARTTIME=1552580878
+ [ 2 = 1 ]
+ mpirun -np 2 /home/jovyan/simulations/static_tov/SIMFACTORY/exe/cact
us_sim -L 3 /home/jovyan/simulations/static_tov/output-0000/static_tov
_small_short.par
--------------------------------------------------------------------------
----
The value of the MCA parameter "plm_rsh_agent" was set to a path
that could not be found:

  plm_rsh_agent: ssh : rsh

Please either unset the parameter, or check that the path is correct
--------------------------------------------------------------------------
----
```

```
Thu Mar 14 16:28:00 UTC 2019
Simfactory Done at date: 0
```

Plotting the Output

The following commands will generate a simple line plot of the data. They will work in a python script as easily as they do in the notebook (just remove the "%matplotlib inline" directive).

In [14]:
```
# This cell enables inline plotting in the notebook
%matplotlib inline

import matplotlib
import numpy as np
import matplotlib.pyplot as plt
```

Numpy has a routine called genfromtxt() which is an extremely efficient reader of textual arrays of floating point numbers. This is well-suited to Cactus .asc files.

In [15]:
```
import os
home = os.environ["HOME"]
lin_data = np.genfromtxt(home+"/simulations/static_tov/output-0000/stati
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-15-180f99e7dbb0> in <module>
      1 import os
      2 home = os.environ["HOME"]
----> 3 lin_data = np.genfromtxt(home+"/simulations/static_tov/output-0000/static_tov_small_short/hydrobase-rho.maximum.asc")

/opt/conda/lib/python3.6/site-packages/numpy/lib/npyio.py in genfromtxt(fname, dtype, comments, delimiter, skip_header, skip_footer, converters, missing_values, filling_values, usecols, names, excludelist, deletechars, replace_space, autostrip, case_sensitive, defaultfmt, unpack, usemask, loose, invalid_raise, max_rows)
   1549                 fhd = iter(np.lib._datasource.open(fname, 'rbU'))
   1550             else:
-> 1551                 fhd = iter(np.lib._datasource.open(fname, 'rb'))
   1552             own_fhd = True
   1553         else:

/opt/conda/lib/python3.6/site-packages/numpy/lib/_datasource.py in open(path, mode, destpath)
    149
    150     ds = DataSource(destpath)
--> 151     return ds.open(path, mode)
    152
    153

/opt/conda/lib/python3.6/site-packages/numpy/lib/_datasource.py in open(self, path, mode)
    499             return _file_openers[ext](found, mode=mode)
    500         else:
--> 501             raise IOError("%s not found." % path)
    502
    503

OSError: /home/jovyan/simulations/static_tov/output-0000/static_tov_small_short/hydrobase-rho.maximum.asc not found.
```

This is all you need to do to plot the data once you've loaded it. Note, this uses Python array notation to grab columns 1 and 2 of the data file.

In [ ]:
```
plt.plot(lin_data[:,1],lin_data[:,2])
```

In [ ]: