Torch Quick Start Guide
Last updated: March 4, 2026

Contents

1 Introduction

Torch is a star formation simulation code with magnetohydrodynamics (FLASH), radiative transfer (Fervent), self-gravity & sinks, star particles, feedback (stellar winds, SNe), stellar evolution tracks (SeBa), and N-body dynamics (AMUSE/PeTar). Torch features and add-ons are listed in Figure 1, and references for the major components are in Appendix A.

You can find a high-level overview of the code in:

- Wall+ 2019, arXiv:1901.01132
- Wall 2019, Ph.D thesis

Torch comprises (1) interface code that connects FLASH to AMUSE, (2) a good number of add-on code units for FLASH, and (3) a set of Python scripts that connect the codes and drive thes imulation.

To use Torch, you install additional code into FLASH, and then compile FLASH into an AMUSE "worker" binary that enables FLASH to be called from AMUSE python scripts. You then run a Python script to run the simulation.

Flash (AMUSE interface)		**ph4 / Multiples / SeBa**
MHD (Fryxell+2000)	Atomic cooling (Hill+2012)	N-body (McMillian)
Rad Trans (Bacynzski+2015)	Mol cool (Neufield+1996)	Binary formation (McMillan+)
Winds (Markova+2008,Vink+2000, Kudritzski+2000)	Dust<->Gas (Hollenbach+1989)	Binary dynamics (McMillan+)
SN (Simpson+2016)	Background FUV (Weingartner+2001)	Binary accretion (PZ+1996)
Star formation (Sormani+2017)	Cosmic rays (Galli+2015)	Stellar flux (Lanz+2003)
Ionization fraction solver	Local gas extinction (Banerjee+2006)	SE (PZ+1996)
Ionization heating (Bacynzski+2015)	Radiation momentum	Stellar mass loss rate (other than OB winds) (PZ+1996)
FUV local stellar heating (Weingartner+2001)	EUV on dust (Draine 2011)	

Figure 1. Adapted from Wall+ 2018, MODEST conference proceedings

2 Quick Start

Torch has many moving parts. You'll likely have to adjust the procedure below for your personal setup. In any case, don't despair! Torch has been successfully built and run on a variety of systems.

We assume that you're using (1) the bash shell, and (2) conda or a virtualenv for Python package management. But, the setup procedure should work for any shell and Python package manager with some adjustments.

2.1 Overview

Torch consists of a collection of Fortran files that extend the FLASH MHD code, bindings for AMUSE for that extended FLASH, and a set of Python scripts that use AMUSE to run a simulation combining the extended FLASH with the other codes.

Installing Torch therefore entails 1) creating an environment and installing the dependencies, including AMUSE, 2) installing Torch, with its extended FLASH and Python parts, and 3) installing the FLASH AMUSE bindings to tie it all together.

2.2 Environment

First, start a new shell session. You may want to check your `.bashrc` for any old MPI or HDF5 configuration that could conflict with Torch setup – module load commands, environment variable exports, et cetera – and comment them out.

Next, it is probably a good idea to make a project directory in which you can download and install all the pieces:

```
mkdir torch_project
```

You can call it something different or arrange the files differently if you want. In particular if you're on an HPC cluster with a limited amount of space available in your home directory, then you'll want to use another location instead, for example a scratch directory. The documentation for the machine should tell you what is available. You'll need about 2GB for the base installation, plus space for model output which can also easily grow to multiple gigabytes.

Next, we'll need an environment to install the Python bits into, and possibly the native tools and dependencies as well. There are two options here. If you're on an HPC machine, then you'll probably want to use native tools and dependencies, and a Python virtual environment for the Python bits. On a local machine, you can do that as well, or use conda instead to get everything contained in a single environment. Both options are described here.

2.2.1 Local, with conda

We'll assume here that you already have conda installed and available. If not, we recommend installing the miniforge distribution from https://github.com/conda-forge/miniforge.

First, we'll create and activate a new conda environment, and set it to use conda-forge:

```
conda create -n Torch
conda activate Torch
```

Note that AMUSE is designed to work with the conda-forge package channel. Conda-forge contains a very large variety of scientific and other packages, so you'll find everything you need for Torch as well as other tools you may want to use.

Mixing conda channels is not recommended, as it leads to package incompatibilities, long waits for the solver to try to deal with incongruent sets of packages when installing, and hard to diagnose issues. So we strongly recommend sticking with conda-forge only. Miniforge will do that by default.

If you open a new shell, then the environment needs to be reactivated using the `conda activate Torch` command before you do anything else.

Once we have a conda environment, we can install the prerequisites inside of it, once again making sure our packages come from conda-forge (some versions of conda can be a bit stubborn on this):

```
conda install -c conda-forge --override-channels pip wheel c-compiler cxx-compiler
fortran-compiler \ 'gfortran<14' python coreutils patch curl make openmpi hdf5 zlib
'docutils>=0.6' 'mpi4py>=1.1.0' \ 'numpy>=1.2.2' 'h5py>=1.1.0' scipy astropy jupyter pandas seaborn
    matplotlib yt
```

AMUSE is not yet available directly from conda-forge, so we'll have to install it from source. First, we need to download and unpack it (into our project directory):

```
curl -L -O "https://github.com/amusecode/amuse/archive/refs/tags/v2025.9.0.tar.gz"
tar xzf v2025.9.0.tar.gz
```

AMUSE has had a major upgrade to its installer recently, so be sure to get v2025.9.0 or a later version, older ones won't work.

Next, we can move into the newly created AMUSE source directory, and run the installer:

```
cd amuse-2025.9.0
./setup
```

This will produce quite a bit of output, including an overview of installable codes, disabled codes and the reasons why, and follow-up instructions. If all went well then you should now be able to install the AMUSE packages you need into your conda environment using

```
./setup install framework kepler petar ph4 seba smalln
```

Note that Torch doesn't necessarily use all of these, so if you know which ones you want, then you can install only those. At any rate you can return to the AMUSE directory later and use `./setup` to install more codes.

That concludes setting up the environment and the dependencies, so you can scroll past the next section (one environment is enough, really!) and continue with installing Torch.

2.2.2 HPC, with virtualenv and modules

As an alternative to Conda, you can also use a Python virtual environment to install the Python dependencies into. Dependencies for the Fortran and C code in Torch will then have to be installed in some other way. Linux computers typically have a built-in package manager like `apt` or `dnf`, while on a Mac Homebrew and MacPorts are commonly used. On HPC machines, you typically load the necessary modules.

If you're on a local machine and have one of the above package managers available, then you should continue with the instructions below, but skip the loading of modules and just make the virtual environment and install the Python dependencies into it. If you then continue with installing AMUSE, the AMUSE installer will guide you through installing the dependencies using whichever package manager you have available. (Note that you won't need everything it suggests, because Torch doesn't use all of AMUSE. You can save some disk space by limiting your selection.)

On an HPC cluster or similar environment, the main tools and dependencies are likely already installed and available as modules. To use them, we'll have to activate them using the `module` command.

First, let's see what's installed:

```
module avail
```

This will show a long (potentially very long) list of installed packages, or modules.

On some systems, this will only show a few years, e.g. `2023 2024 2025`. In that case, use `module load 2025` (the latest is usually best) and then try `module avail` again to get the list.

Have a look through the list and see if you can find modules named `GCC`, `HDF5`, `OpenMPI` (or some other MPI implementation, like `MPICH`), `make`, and `Python`. The exact names will vary from machine to machine. If the list is long, it's better to let the computer search for you using e.g. `module keyword HDF5`.

Often, there are multiple versions of a package available. The codes in Torch (and AMUSE) are fairly old, so probably any version will work. If you see different packages with names like `gompi` or `iimpi`, use the `gompi` ones, as those match GCC and OpenMPI.

You may not need `make` if it's already available. If the command

```
make --version
```

prints some text starting with GNU Make, then you're good to go.

Once you have found the needed module names, you can load the modules, for example for Snellius using the 2025 software stack (you'll need to modify the exact module names to suit your machine):

```
module load 2025
module load GCC/14.2.0
module load OpenMPI/5.0.7-GCC-14.2.0
module load HDF5/1.14.6-gompi-2025a
module load make/4.4.1-GCCcore-14.2.0
module load Python/3.13.1-GCCcore-14.2.0
```

Next, we can make a Python virtual environment. This is a directory, which can go into your project directory or wherever you want it.

```
python3 -m venv /path/to/torch_project/Torch-env
```

We can then activate the environment, and install the Python dependencies:

```
. /path/to/torch_project/Torch-env/bin/activate
pip3 install -U pip wheel scipy astropy jupyter pandas seaborn matplotlib yt
```

At this point, it's probably convenient to create a file you can load that will activate the modules and the virtual environment whenever you want to work with Torch. To do that, create a file named `Torch.env` in your project directory containing the module load commands and the activation of the environment, like so:

```
module load 2025
module load GCC/14.2.0
module load OpenMPI/5.0.7-GCC-14.2.0
module load HDF5/1.14.6-gompi-2025a
module load make/4.4.1-GCCcore-14.2.0
module load Python/3.13.1-GCCcore-14.2.0

. /path/to/torch_project/Torch-env/bin/activate
```

with appropriate modifications for your site. Now you can activate everything in one command using `. Torch.env` in your project directory (note the period and the space at the beginning, they're required).

With the environment set up and activated, we can install AMUSE and most of the codes that Torch uses into it. First, we need to download and unpack it (into our project directory):

```
curl -L -O "https://github.com/amusecode/amuse/archive/refs/tags/v2025.9.0.tar.gz"
tar xzf v2025.9.0.tar.gz
```

AMUSE has had a major upgrade to its installer recently, so be sure to get v2025.9.0 or a later version, older ones won't work.

Next, we can move into the newly created AMUSE source directory, and run the installer:

```
cd amuse-2025.9.0
./setup
```

This will produce quite a bit of output, including an overview of installable codes, disabled codes and the reasons why, and follow-up instructions.

If you're on a local machine and don't already have the dependencies installed, then `./setup` will give you the required command to install them using your local package manager. Note that you probably don't need all the dependencies it suggests installing, as that is the complete set and Torch only uses a few of AMUSE's many codes. Compilers for C, C++ and Fortran, MPI, HDF5, make and Python should go a long way.

If all went well then you should now be able to install the AMUSE packages you need into your virtual environment using

```
./setup install framework kepler petar ph4 seba smalln
```

Note that Torch doesn't necessarily use all of these, so if you know which ones you want, then you can install only those. At any rate you can return to the AMUSE directory later and use `./setup` to install more codes.

With that, we have a virtual environment that is ready to install Torch into.

2.3 Installing Torch

The first step to installing Torch is to get a copy of FLASH. To do that, you need to register, which may take a few days, and then download FLASH 4.6.2. The FLASH homepage (http://flash.uchicago.edu/site/flashcode) will show you how.

Once you have a `FLASH4.6.2.tar.gz` file in your project directory, you need to unpack it, and then set the `FLASH_DIR` environment variable so that the Torch installer can find it:

```
cd torch_project
tar xzf FLASH4.6.2.tar.gz
export FLASH_DIR=${PWD}/FLASH4.6.2
```

Next, we can download Torch, and set `TORCH_DIR`:

```
git clone https://bitbucket.org/torch-sf/torch.git
export TORCH_DIR=${PWD}/torch
```

Then we can move into the Torch directory, and run its installer:

```
cd ${TORCH_DIR}
./install.sh
```

This will add the Fortran parts of Torch to FLASH. Next, we need to make the Python parts of Torch available:

```
export PYTHONPATH=$PYTHONPATH:$TORCH_DIR/src
```

You will want to add the exports of `TORCH_DIR` and `FLASH_DIR` to your `Torch.env`, as well as the PYTHONPATH, like so:

```
module load 2025
module load GCC/14.2.0
module load OpenMPI/5.0.7-GCC-14.2.0
module load HDF5/1.14.6-gompi-2025a
module load make/4.4.1-GCCcore-14.2.0
module load Python/3.13.1-GCCcore-14.2.0

FLASH_DIR=/path/to/FLASH4.6.2
TORCH_DIR=/path/to/torch

. /path/to/torch_project/Torch-env/bin/activate

export PYTHONPATH=$PYTHONPATH:$TORCH_DIR/src
```

With Torch installed into FLASH, we can now go to the FLASH directory and configure it:

```
cd $FLASH_DIR
./setup Cube -auto -3d +amuseUSM +amuseMG +rayPE +HandCnew +amuseSinksAndStars \
+amuseWind +enerinj +supportPPMupwind +pm4dev -maxblocks=100 +cube16 --site=torch_auto
```

The final piece of the puzzle are Torch's AMUSE FLASH bindings. These make it possible for the Python parts of Torch to talk to FLASH, including its Torch additions. Check that you still have your virtual environment active, and then you can compile and install FLASH and the bindings like this:

```
cd $TORCH_DIR
make -C src/amuse/torch_amuse_flash develop-torch-amuse-flash
```

With that, your environment should contain a working Torch installation. Time to give it a try!

2.4 Run a Torch simulation (Josh thesis Appendix B)

You are now ready to run a Torch simulation! We will simulate a $3 \times 10^3 M_\odot$, 5 pc radius gas sphere in ± 7 pc cubic domain with outflow boundary conditions. In about a free-fall time (≈ 2 Myr), the cloud will collapse under its own gravity, create a sink particle, and begin forming stars.

Choose a directory for your simulation and `cd` there. Then do:

```
ln -s $TORCH_DIR/cool.dat
ln -s $TORCH_DIR/cube128
cp $TORCH_DIR/torch_user.py                torch_user.py
cp $TORCH_DIR/flash.par.turbsph_standard   flash.par
mkdir data
```

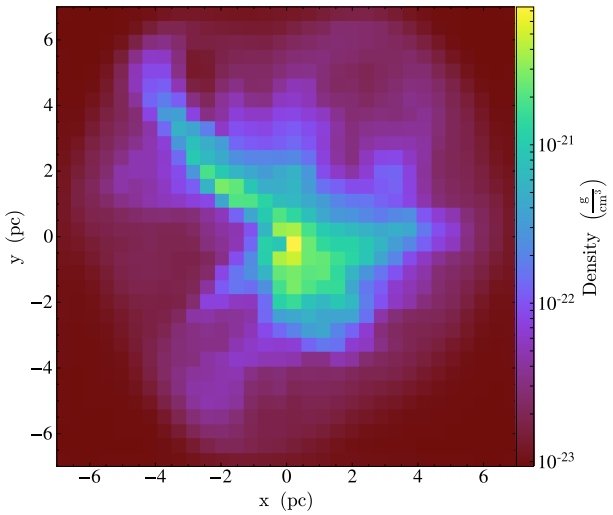

Figure 2. Density in x-y plane.

The file `torch_mainloop.py` is the heart of the simulation. It performs the split time-evolution with both FLASH and the n-body integrator ph4 or PeTar, handles stellar evolution, and lots more. `torch_user.py` is the main source of control for users of Torch, where parameters can be set and initial stellar conditions defined.

The simulation should take 1–10 minutes to reach $t_{\rm max} = 2$ Myr $= 6.31 \times 10^{13}$ s. It will use 7 threads: 2 for FLASH, 1+2 for n-body integration (ph4) and n-body binary interactions (multiples) or 1 for stellar dynamics (PeTar), 1 for stellar evolution, and 1 for AMUSE itself. At around 1 Myr, a sink particle will form and begin creating stars. One or a few $\geq 7 M_\odot$ stars may begin blowing a wind, so the time step may drop; you may wish to end the simulation early.

To run the simulation interactively, execute the command:

```
mpiexec -n 1 python torch_user.py
```

On a compute cluster with Slurm, copy `run.sh` from the `torch` repository to your simulation directory. Edit the `SBATCH` options as needed. Then do:

```
sbatch run.sh
```

If you increase the number of tasks requested, `torch_user.py` will automatically assign more threads (workers) to FLASH.

Log files will be dumped in your current working directory, and simulation outputs will be written to the `data/` directory. If you re-start a simulation, Torch (FLASH) will overwrite existing data, but append to (some) existing logs. You may want to rename or delete your old logs to help keep track of your runs.

2.5 Look at the output

Let's have a quick look at this gestating star cluster. The example plots here were created with the `yt` package, version 3.4.1. To learn more about `yt`, visit http://yt-project.org/. To install `yt`:

```
conda install yt
conda clean --all
```

We can first look at density-slice plots in the three coordinate planes (x-y, x-z, y-z). At the command line, call:

```
yt plot data/turbsph_forced_hdf5_plt_cnt_0000
```

or, if you don't have a "forced" plot output, use the last "plt_cnt" file dumped by the simulation. This will create a new directory `frames/` with three plots similar to Figure 2.

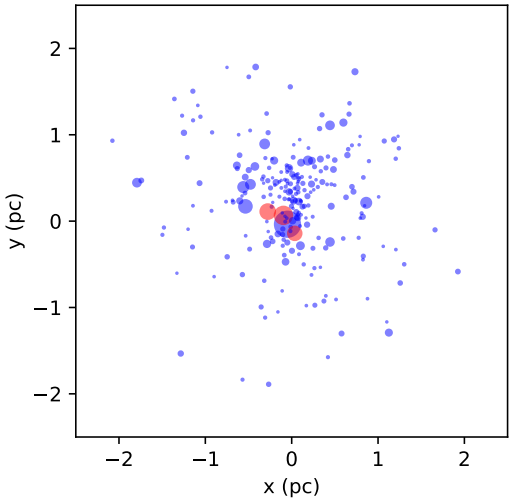

Figure 3. Star particles. Size is proportional to mass; red particles have $\mathrm{d}m/\mathrm{d}t > 0$ (wind-blowing stars), blue particles have $\mathrm{d}m/\mathrm{d}t = 0$ (quiescent).

How about the particles? Let's see how they're distributed, and inspect some of their other properties. In a Python session:

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

ds = yt.load('data/turbsph_hdf5_plt_cnt_0018')

ad = ds.all_data()
ppx = ad['all', 'particle_posx'].to('pc').value
ppy = ad['all', 'particle_posy'].to('pc').value
ppm = ad['all', 'particle_mass'].to('Msun').value
windy = ad['all', 'particle_dmdt'].value
windy[windy > 0] = 1

plt.scatter(ppx, ppy, s=ppm*6, c=windy, cmap='bwr', alpha=0.5)
plt.gca().set_aspect('equal')
plt.xlim(-2.5, +2.5)
plt.ylim(-2.5, +2.5)
plt.xlabel('x (pc)')
plt.ylabel('y (pc)')
plt.show()
```

In this particular simulation, three stars are massive enough to blow a wind (Figure 3).

2.6 Next steps

To tweak the simulation parameters, you will need to edit `flash.par` and `torch_user.py`. These two files specify most of the configuration options for the simulation.

Torch comes packaged with a few simulations in `FLASH4.6.2/source/Simulation/SimulationMain/`

```
Cube
EnergyInjection
StratBox
```

which provide initial conditions for (1) a turbulent gas sphere initialized from a 128^3 array, (2) a uniform medium for testing a stellar wind or supernova, and (3) a planar gas layer in a periodic domain, driven by random thermal supernova explosions.

It's often useful to reduce the number of Torch features for study or debugging. Below are setup calls showing some reduced feature sets:

Full Torch code with AMUSE/FLASH coupling.

```
./setup Cube -a -3d +amuseUSM +amuseMG +rayPE +HandCnew +amuseSinksAndStars \
    +amuseWind +enerinj +supportPPMupwind +pm4dev -maxblocks=100 +cube16
```

FLASH-only simulation, no AMUSE coupling. Sinks will work, but no stars will form.

```
./setup Cube -a -3d +usm +gravMgrid +rayPE +HandCnew  +amuseSinksAndStars \
    +amuseWind +enerinj +supportPPMupwind +pm4dev -maxblocks=100 +cube16
```

FLASH-only simulation, no ray-tracing or sinks/stars. Only self-gravity and heating/cooling.

```
./setup Cube -a -3d +usm +gravMgrid +HandCnew \
    +supportPPMupwind +pm4dev -maxblocks=100 +cube16
```

FLASH-only simulation, only MHD.

```
./setup Cube -a -3d +usm +supportPPMupwind +pm4dev -maxblocks=100 +cube16
```

Note that `+rayPE` generally requires `+amuseSinksAndStars`. These setup flags are aliases for various FLASH units; the flags are defined in `FLASH4.6.2/bin/setup_shortcuts.txt`.

Don't forget that after a fresh setup + compile of FLASH, you will need to recompile the AMUSE worker too.

2.7 PeTar Stellar Dynamics

Torch is now capable of using `PeTar` for stellar dynamics. This is recommended for all runs, but particularly for runs with $> 5,000$ stars or runs containing many binaries such as those with primordial binaries. To use `PeTar`, set

```
p['with_petar'] = True
p['petar_rout'] = 0.001 | units.pc # outer radius for tree
```

in `torch_user.py`. `PeTar` accepts the user parameter `r_out` which is a boundary radius between the tree gravity and direct N-body and SDAR. The default is the standard value used in cluster simulations, but the user should test this value and modify it for their particular use case. For a thorough understanding of the parameters used in `PeTar`, users should familiarize themselves with the README of the `PeTar` github and the `PeTar` methods paper.

3 VorAMR: Expanded Options for Initial Conditions

A normal Torch run initializes from a "cubefile" like `cube128` or one generated from `turb-sphere.py`. These initial conditions are currently restricted to spherical clouds with a turbulent velocity distribution and gaussian density profile or something similar. While such ICs make for easy first steps for users to setup a new Torch simulation, ideal ICs would come from systems that evolved self-consistently in galactic environments.

`VorAMR` was created to allow for output data from other star formation software suites to be used as initial conditions in Torch. Currently, `VorAMR` has only been shown to be able to convert data from the star formation code `AREPO` into Torch ICs, but in theory can be easily adapted to convert ANY moving-mesh, AMR, or SPH code into Torch ICs.

3.1 How VorAMR works

`VorAMR` uses output data from other hydro-codes to build a refined grid in `FLASH` and then populate that grid with field values (density, internal energy, velocity, etc.) via the `AMUSE` interface. By doing this all within the Torch framework, the resulting grid and initial gas/particle properties allows Torch to immediately launch into a simulation but with external data as its initial conditions.

`VorAMR` takes advantage of the "refine_on_particle" routines within `FLASH` to construct a refined grid that as accurately as possible mirrors the local mesh scale of the `AREPO` data. `VorAMR` also uses a nearest-neighbor NDInterpolator to fill the `FLASH` grid cells with the field data of the nearest Voronoi mesh element (and therefore the element which encapsulates the cell center).

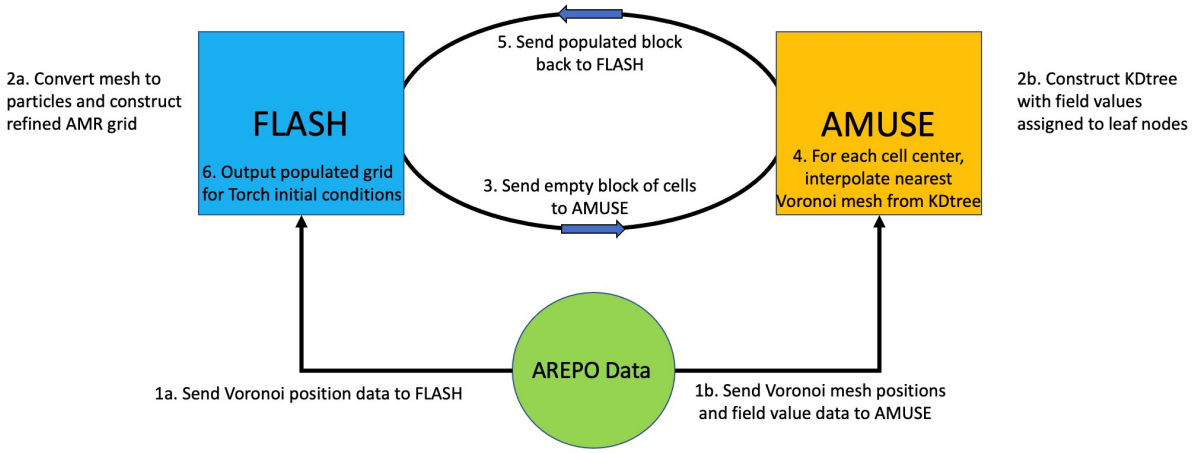

Figure 4. VorAMR logic flow.

3.2 How to use VorAMR

`VorAMR` is installed alongside Torch but is turned off by default; Torch operates normally with `VorAMR` turned off. To turn `VorAMR` on, switches must be flipped in the flash.par file. Note, the file `flash.par.turbsph_standard` does not have `VorAMR` switches, `flash.par.voramr` is meant to be identical but with the `VorAMR` switches included.

```
# ================
# VorAMR extension
# ================
use_voramr = .true. # main control switch
voramr_source = "snapshot_518_9.hdf5" # source data to be converted
voramr_input = "voramr_input.hdf5" # name of file to be read-in by FLASH

refine_on_particle_count = .true. # builds grid based on source data - BE SURE TO TURN OFF FOR NORMAL TORCH
min_particles_per_blk    = 4096 # Assumes 16x16x16 blocks
max_particles_per_blk    = 4096
pt_maxPerProc            = 10000000 # Enough to fit all particles on one proc - CAN BE REDUCED FOR NORMAL
    TORCH
refineonjeanslength = .true.

# Restrict initial refinement by only placing particles within radius
use_localRef = .false.
center_localRef = .false. # Crops FLASH domain centered at localRef_{x,y,z} and centers new domain
# Local refinement center and radius
localRef_x = 3.20621187e+20 # cm
localRef_y = 6.24367575e+20
localRef_z = -1.51873194e+20
localRef_r = 1.543e+20

# Background Potential
sim_withStaticGrav = .true.
             # AREPO 518_9   # Hill et al 2012
sim_aParm1  = 1.31142525e-72  #4.39996789e-9  # [cm/s^2]     # 1.42E-3 [kpc/Myr^2]
sim_aParm2  = 1.35401904e-52  #5.51293615e-31 # [1/s^2]      # 5.49E-4 [1/Myr^2]
sim_aParm3  = -2.36078509e-30 #5.55421965e20  # [cm]         # 0.18E0  [kpc]
sim_aParm4  = 4.06112841e-11  #1.62715932e-53 # [1/(cm s^2)] #5 .0E-5  [1/kpc Myr^2]
```

Some of these switches are reflected in the `torch_user.py` file, all `flashp['..']` data is extracted directly from `flash.par`:

```
    p = {}
    flashp = FlashPar("flash.par")

    # <VorAMR>
```

```
p['with_voramr'] = flashp['use_voramr']
p['source_file'] = flashp['voramr_source']
p['convert_file'] = True # Runs source_file through src/voramr/hdf5_convert.py
p['use_localRef'] = flashp['use_localRef']
p['local_ref'] = [flashp['localRef_x'], flashp['localRef_y'], flashp['localRef_z'],
flashp['localRef_r']]
p['center_local_ref'] = flashp['center_localRef']
p['input_file'] = flashp['voramr_input']
p['pickle_kdtree'] = False # saves kdtree built from source_file used in interpolation. Useful if
memory strained.
p['pickle_file_name'] = "kdtree.pickle"
p['numBlocks'] = 15000 # Quirky parameter, just set to any number larger than total num actual blocks
and FLASH will figure it out.
p['cellsPerBlock'] = 16
```

Once `VorAMR` is turned on, install torch as normal `. ./install.sh`.

Then, setup FLASH with enough MAXBLOCKS to allow for the expected refined grid to fit on a single processor.

```
./setup Cube -a -3d +amuseUSM +amuseMG +rayPE +HandCnew +amuseSinksAndStars \
    +amuseWind +enerinj +supportPPMupwind +pm4dev -maxblocks=10000 +cube16
```

Change the 'lrefine_max' value in flash.par to be whatever level you wish the grid to reach at its highest refinement regions. To acheive maximum consistency with the structure of the input data, 'lrefine_max' should be set such that the highest refinement FLASH blocks will contain as many input data points as cells (4096 for +cube16 blocks).

If your source data contains star particles that you would like to include in Torch, edit `user_initial_conditions()` in `torch_user.py` to read in the particle data, incorporate them into an `AMUSE` particle set, and initialize them via the `hydro` class.

It is then recommended to change the following `flash.par` parameters to force `VorAMR` to output a checkpoint file as fast as possible. That resulting checkpoint can the be restarted from with `VorAMR` turned off.

```
tmax            = 6.30e8 # two init timesteps
dtinit          = 3.15e8
dtmin           = 3.15e7

checkpointFileIntervalTime  = 3.15e7 # dt_min
plotFileIntervalTime        = 3.15e8
particleFileIntervalTime    = 3.15e8
```

Finally, run Torch with only 1 processor assigned to `flash_worker` (if running default worker partitions, this would be 6 total processors: 1 driver, 2 `ph4`, 1 `SeBa`, 1 `multiples`, 1 `FLASH`). Note, when restarting from a `VorAMR` checkpoint, any number of processors can be used as long as `use_voramr = .false.`

In addition to the usual Torch output, `VorAMR` produces two more files:

1. `voramr_input.hdf5` - the particle data extracted from the source data. This is what `FLASH` sees and will refine on.
2. `interp-data.hdf5` - the file from which the interpolation kdtree is built. Only `AMUSE` sees this file, but then passes info from it to `FLASH`. This file will always contain all data from the source file.

These files are both produced as sometimes we only want `FLASH` to refine on a portion of the source data, but we want `AMUSE` to have an accurate kdtree for all data to avoid domain edge effects during interpolation.

3.3 Regional Refinement

Torch is capable of de-refining the grid to a user defined level outside of a user defined rectangular region of interest. This is most useful in large-scale runs such as those initialized with `VorAMR`. To use this capability, the user must set the parameters listed below in the `flash.par` file.

```
# ================================================
# Derefinement outside rectangular region of interest
```

```
# ================================================
use_deref         = .true.
deref_lref        = 2  # level to derefine to
# EXAMPLE: derefine to level deref_lref outside center 5pc square
deref_xl          = -1.543e+19  # 5pc
deref_xr          =  1.543e+19
deref_yl          = -1.543e+19
deref_yr          =  1.543e+19
deref_zl          = -1.543e+19
deref_zr          =  1.543e+19
```

4 Caveats and Known Issues

4.1 Torch Issues

Unphysical hot/empty zones are a common problem in finite volume codes, and are readily generated by under-resolved supernovae and stellar winds in Torch. To combat such zones, a few tactics include:

- Lower CFL
- Restart with more diffusive hydro solver parameters, temporarily
- Manually smooth out extremely sharp gradients
- Enable or increase artificial shock viscosity.
- Use the time-step limiter for positive-definite cell values (already enabled in the example Torch flash.par files). This is controlled by the FLASH runtime parameter `dr_usePosdefComputeDt`, among others.
- `VorAMR` is currently hardcoded to extract data from and refine on data from AREPO file structures. To change to other file structures, edits will need to be made to at least `src/voramr/hdf5_convert.py` and possibly `src/flash/source/Simulation/SimulationMain/Cube/pt_initVoronoiPositions.F90`

See http://flash.uchicago.edu/pipermail/flash-users/2019-May/002908.html for a nice explanation by Sasha Tchekhovskoy.

In the FLASH unsplit solver, the logic for the hybrid Riemann solver is slightly altered; see the `torch` file: `src/flash/source/physics/Hydro/HydroMain/unsplit/hy_uhd_dataReconstOneStep.F90`

Torch does not support the use of BHTree with AMUSE. Code exists to hook FLASH/BHTree and AMUSE together, but it needs a small update and some testing to work.

Several add-on ParticleInitialization, ParticleMapping, etc. units are provided with Torch. These were created to trace supernova and superbubble ejecta in stratified box simulations run by Ibanez-Mejia et al. (2017). However, these have not been tested with the Torch sink/star framework and probably will not work correctly as is.

The module `source/physics/sourceTerms/GridInject` is in development and not recommended for use yet.

4.2 VorAMR Issues

`VorAMR` is capable of building Torch-ready initial conditions from ANY hydrodynamical simulation output. But, `VorAMR` is limited in how quickly it can complete this process. Luckily, a user will only have to use VorAMR once for a major production run, so the initial cost of time is generally worthwhile.

- `VorAMR` can only run with ONE oversubscribed processor for the `flash_worker` due to the use of serial HDF5 open/read calls within `FLASH`. Data conversion and grid build times can be on the order of days.
- Users are recommended to augment a Torch+`VorAMR` run to output a checkpoint file after ONE timestep. Then, restart from the checkpoint in a clean Torch build without `VorAMR` turned on.
- Sometimes `VorAMR` will run fine, but the resulting grid does not appear to be refined. Check in `turbsph.log` and see if the number of particles reported matches the dimensions of the array sent so FLASH (reported in slurm file) and what FLASH actually reads (reported in flash_worker.out). If there is a mismatch, then it's likely that some refinement particles are being placed outside of the computational domain. Make sure your domain dimensions correctly reflect the source data.

5 Help!

5.1 General checks

- Are all packages built using the same compiler set?
- Check your environment variables. Is PATH pointing somewhere it shouldn't? Are two different installs/versions of a module visible in PATH?
- Anaconda or Conda may provide its own HDF5. If you are using this HDF5, great, but if you are using a system HDF5 install, not so great. Check your PATH and make sure that the desired system HDF5 comes before the Anaconda/Conda HDF5.
- Check that FLASH, AMUSE, and mpi4py are using the same HDF5 and MPI libraries.
- If you had to swap MPI modules and reinstall `mpi4py` at any point, `pip install` needs the `--no-cache-dir` flag or else `mpi4py`'s compiled object library will not be reinstalled. You can check that `mpi4py` is linked to the correct MPI library by doing:

```
ldd {your/conda_env/directory}/{env_name}/lib/python2.7/site-packages/mpi4py/MPI.so
```

- Here are some commands that may help diagnose library and path problems:

```
module list
module show some/module/name
which python
which mpiexec
python -c "from mpi4py import MPI; print MPI"
ldd flash4
ldd flash_worker
```

5.2 FLASH setup/compile problems

- **make fails immediately with something like** `/usr/local/mpich2//bin/mpif90: Command not found.`

  Check that you are using the correct `Makefile.h`, with `MPI_PATH` and `HDF5_PATH` appropriate for your system.

- **make is slow.**

  Try `make -j` to use multiple processes.

- **make succeeds, but flash4 or flash_worker fails at runtime with error like** `error while loading shared libraries: libhdf5.so.8: cannot open shared object file.`

  Take a look at http://flash.uchicago.edu/pipermail/flash-users/2013-July/001322.html.

- **FLASH runs out of memory.**

  Reduce MAXBLOCKS or allocate more memory per core. The memory cost can be estimated as:

  ```
  (NUNK_VARS + NFACEVAR + NFLUXVAR) * (NXB + 2*NGUARD) * (NYB + 2*NGUARD) * (NZB + 2*NGUARD)
  * MAXBLOCKS * (8 bytes)
  ```

  where the variables are defined in `FLASH4.6.2/object/Flash.h`. For example, with `NUNK_VARS=46`, `NFACEVAR=2`, `NFLUXVAR=12`, `NXB=NYB=NZB=16`, `NGUARD=6`, and `MAXBLOCKS=100`, the memory usage is around 1 GB.

5.3 AMUSE configure/make problems

- **How do I troubleshoot X configure error?**

  A few places to look include: STDOUT from `configure`, `config.log`, and the source code of `configure` itself.

  Also, `./configure --help` may give an idea of tunable parameters.

- **FLASH alone compiles, but the AMUSE** `flash_worker` **build fails with library errors.**

  Try comparing, piece-by-piece, the command-line invocations of `mpif90` for FLASH alone versus `flash_worker` to make sure that `-L` and `-l` flags are sensible.

You can manually edit some compiler flags in `config.mk`, located in the top-level AMUSE directory.

5.4 Runtime and MPI problems

- **AMUSE fails spawn processes over > 1 cluster node, but works if all processes are on the same node. MPI may hang, or return errors of form (for Intel MPI).**

  ```
  HYDT_dmx_register_fd ({...}demux.c:101): registering duplicate fd 0
  HYDT_bscd_slurm_launch_procs ({...}/slurm_launch.c:258): demux returned error registering fd
  ...
  main (../../ui/mpich/mpiexec.c:1118): process manager error waiting for completion
  ```

  One known solution: use OpenMPI, and do not overspecify Slurm sbatch or srun options, give only `-n`. A general solution is not known. This was seen with a relatively old AMUSE version, and may be resolved in newer versions.

For OpenMPI specifically:

- **How do I get information about the OpenMPI build on my cluster?**

  The command `ompi_info` reports how a given OpenMPI installation was built, e.g., version, compiler, configure flags, etc.

- **How do I set MCA parameters?**

  https://www.open-mpi.org/faq/?category=tuning#setting-mca-params

- **How do I fix runtime warnings about infiniband ports?**

  ```
  By default, for Open MPI 4.0 and later, infiniband ports on a device
  are not used by default.  The intent is to use UCX for these devices.
  You can override this policy by setting the btl_openib_allow_ib MCA parameter
  to true.
  ...
  WARNING: There was an error initializing an OpenFabrics device.
    Local host:   t035
    Local device: mlx5_0
  ```

  Follow the suggestion and set `btl_openib_allow_ib=1`. The OpenMPI FAQ explains how to set MCA parameters.

5.5 Navigating the source code

- **How do I decipher FLASH setup calls and runtime parameters?**

  Look at `bin/setup_shortcuts.txt` and `object/setup_params`. Appendix C lists the file strucutre for Flash when compiled for the turbulent sphere problem.

- **A runtime parameter X has little or no documentation in `object/setup_params`. What do I do?**

  The file `object/setup_params` will tell you which unit declared the mysterious parameter. Once you figure out what it does, consider adding some documentation (in the FLASH unit's `Config` file) and submitting a pull request to the Torch repository!

- **How do I quickly track down a FLASH error message?**

  In the object directory,

  ```
  grep -I "my error message" *
  ```

  The flag `-I` instructs grep to ignore binary files, greatly speeding up the search.

A Torch component references

This list includes most major components, but is not comprehensive.

- FLASH: user's guide, Fryxell+ 2000

- PARAMESH (in vanilla FLASH): archived v4.1 manual
- AMUSE: documentation, book
- Fervent (radiative transfer): Baczynski+ 2015, Baczynski 2015, Ph.D thesis
- Multigrid (self-gravity): Ricker+ 2008
- BHTree (self-gravity): Wünsch+ 2018
- Sink particles: Federrath+ 2010
- Multiples (N-body): AMUSE book (see Sec. 4.5)
- PeTar (N-body): PeTar, Wang+ 2020b SDAR, Wang+ 2020a
- SeBa (stellar evolution): Portegies Zwart & Verbunt 1996 Toonen, Nelemans & Portegies Zwart 2012
- Stratified box setup: Joung & Mac Low 2006, Ibáñez-Mejía+ 2016

B Notes on build process for specific clusters

B.1 Cartesius

CARTESIUS IS NOW DISCONTINUED - We leave the build instructions here as many of the steps are similar to other systems. Also, while SURF has now brought Cartesius offline, its replacement Snellius has a similar module file structure.

Cartesius is the Dutch national supercomputer managed by the cooperative educational and scientific association SURFsara. Many projects have utilized Cartesius and its up-to-date hard/software are well suited for the many projects involving Torch.

To run Torch on Cartesius, you will need to first request an account on the computer. Logging on to Cartesius is straight forward: `ssh [username]@doornode.surfsara.nl` which will then allow you to select `cartesius` after asking for your password. At this stage, you will be able to use Cartesius in its full capacity but you will not be able to `scp` over your `FLASH` repository until your IP address is whitelisted. This can be accomplished by providing your IP to the Surfsara help center. You will be able to use `git` to bring over the `torch` repository to install everything.

Before we install anything however, you must make sure to use the appropriate software packages. Cartesius operates by use of user-chosen modules to be loaded prior to any process you wish to run. Either by editing your `.bashrc` or creating your own bash script to be run at login, you will need to load the updated module environment, as well as the individual modules required by torch.

```
module load 2019
module load foss/2018b
module load Python/2.7.15-foss-2018b
module load HDF5/1.10.2-foss-2018b
module load h5py/2.8.0-foss-2018b-Python-2.7.15
module load GSL/2.5-iccifort-2018.3.222-GCC-7.3.0-2.30

export MODLOC=/sw/arch/RedHatEnterpriseServer7/EB_production/2019/software
export MPIHOME=$MODLOC/OpenMPI/3.1.1-GCC-7.3.0-2.30

export OMPI_MCA_mpi_warn_on_fork=0
```

At this point, following the installation steps for `Torch` should result in a fully functioning software suite!

Preparing a run. Make sure to edit the `flash.par` file you are using such that the `output_directory` is pointed to your desired directory and make sure your module list script has been executed (an easy check is typing `which mpiexec` in the command line).

Starting a run. Cartesius uses the `slurm` job managing system and so is fairly straightforward to use. Here I have provided my run.sh script that I deliver to Cartesius with `sbatch run.sh`. The script contains parameters read by the slurm system as well as the command you want executed.

```
#!/bin/sh
#SBATCH --job-name=[informative-job-name]
```

```
#SBATCH -n [number of procs]
#SBATCH --time=[day]-[hour]:[min]:[sec]
#SBATCH --mail-user=[your-email@email.com]
#SBATCH --mail-type=[email condition]
mpiexec --mca orte_base_help_aggregate 0 -n 1 python bridge_multiples.py
```

It may take some time before your run makes it through the waitlist. Once it does, there will be some information output in your `simulation` directory where you ran the script. The most useful file is the one named `slurm-#####.out` which is where your screen output is redirected to. I check this periodically to see what simulation time my run has achieved, how many stars I've made, if it is hung up on a process, etc.

B.2 Cartesius - Refactor

CARTESIUS IS NOW DISCONTINUED AND THE REFACTOR BRANCH HAS SINCE BEEN MERGED INTO MAIN - The refactored version of Torch (on git branch `refactor`) uses the latest AMUSE commit on branch `master` which requires Python 3.X as well as FLASH version 4.6.2. The setup for the refactored version of Torch on Cartesius is largely the same as the `master` **(now called** `main`**)** branch as described in this document but with a dew caveats.

Firstly, make sure you download and unzip FLASH4.6.2. This can be done from the same FLASH distribution webpage from which FLASH4.5 or earlier versions can be accessed.

Second, make sure to have the AMUSE `master` branch checked out and is at the most recent commit.

In order to satisfy AMUSE's Python 3.X requirements, we will need to establish a module environment that's a bit different from the one described in Appendix B.1:

```
module load 2019
module load foss/2018b

module load Python/3.6.6-fosscuda-2018b
module load HDF5/1.10.2-foss-2018b
module load h5py/2.10.0-fosscuda-2018b-Python-3.6.6
module load GSL/2.5-iccifort-2018.3.222-GCC-7.3.0-2.30
module load pytest/4.4.0-fosscuda-2018b-Python-3.6.6

export MODLOC=/sw/arch/RedHatEnterpriseServer7/EB_production/2019/software
export MPIHOME=$MODLOC/OpenMPI/3.1.1-GCC-7.3.0-2.30
export OMPI_MCA_mpi_warn_on_fork=0
```

In addition to this slightly altered environment, the Cartesius Python 3.6.6 module does not currently have the `matplotlib` and `docutils` libraries installed. This is okay, we can just install them ourselves in our user environment using `pip install --user matplotlib docutils` after loading the Python/3.6.6-fosscuda-2018b module.

Finally, if running this module environment setup reports any error (such as "XYZ module cannot load due to conflict") try running the same environment setup file again and there should be no issue.

B.3 Snellius

The below modules and setup notes are from Brooke Polak with additions by Sean Lewis. Load the following modules (Summer-Fall 2022):

```
# Load Modules

module load 2021
module load Python/3.9.5-GCCcore-10.3.0
module load h5py/3.2.1-foss-2021a
module load GSL/2.7-GCC-10.3.0
#module load Anaconda3/2021.05
#module load HDF5/1.10.7-gompi-2020a



# Specific Paths
```

```
export MODLOC=/sw/arch/Centos8/EB_production/2021/software
export MPIHOME=$MODLOC/OpenMPI/4.1.1-GCC-10.3.0
export MPI_HOME=$MODLOC/OpenMPI/4.1.1-GCC-10.3.0
export MPI_RUN=$MODLOC/OpenMPI/4.1.1-GCC-10.3.0/bin/mpirun
export PATH=$MPI_HOME/bin:$PATH
export MANPATH=$MPI_HOME/share/man:$MANPATH
export LD_RUN_PATH=$MPI_HOME/lib:$LD_RUN_PATH
export LD_LIBRARY_PATH=$MPI_HOME/lib:$LD_LIBRARY_PATH
export CPATH=$MPI_HOME/include:$CPATH
export HDF5_HOME=$MODLOC/HDF5/1.10.7-gompi-2021a
export HDF5DIR=$MODLOC/HDF5/1.10.7-gompi-2021a/lib
export HDF5INCLUDE=$MODLOC/HDF5/1.10.7-gompi-2021a/include
export HDF5LIB="hdf5_fortran -lhdf5"
export PATH=${HDF5_HOME}/bin:$PATH
export LD_LIBRARY_PATH=${HDF5_HOME}/lib:$LD_LIBRARY_PATH
export LD_RUN_PATH=${HDF5_HOME}/lib:$LD_RUN_PATH
export LIBRARY_PATH=${HDF5_HOME}/lib:$LIBRARY_PATH
export C_INCLUDE_PATH=${HDF5_HOME}/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=${HDF5_HOME}/include:$CPLUS_INCLUDE_PATH

export OMPI_MCA_mpi_warn_on_fork=0

export AMUSE_DIR=$HOME/2023feb-vorch/amuse
export FLASH_DIR=$HOME/2023feb-vorch/FLASH4.6.2
export TORCH_DIR=$HOME/2023feb-vorch/torch

export PYTHONPATH=$TORCH_DIR:$PYTHONPATH
export PYTHONPATH=$TORCH_DIR/src:$PYTHONPATH
export PYTHONPATH=$AMUSE_DIR/test:$PYTHONPATH
export PYTHONPATH=$AMUSE_DIR/src:$PYTHONPATH

export CONDA_PATH={path to your conda env.}

# Activate conda environment
conda activate $CONDA_PATH
export PYTHON=$CONDA_PATHbin/python

MCA_Settings='--mca orte_base_help_aggregate 0 '
UCX_Settings='-x UCX_NET_DEVICES=mlx5_0:1'
```

Now follow the usual quickstart procedure. Here are some fixes that might be needed during that process.

- In `$TORCH_DIR/src/flash/source/Particles/ParticlesMain/ray_pe/MultiSourceSimple/HEALPixModule.F90` Lines 209 and 250: `iand(int(x),y)` i.e. replace the first argument of iand with int(argument)
- In `$FLASH_DIR/bin/setup.py` change `sys.setcheckinterval()` to `sys.setswitchinterval()`
- In `$FLASH_DIR/sites/snellius.surf.nl/Makefile.h` add `-fallow-argument-mismatch` to `FFLAGS_OPT`

Then, `FLASH` can be setup and built.

```
./setup Cube -auto -3d +amuseusm +amusemg +raype +handcnew +amusesinksandstars +amusewinds +enerinj
    +supportppmupwind +pm4dev -maxblocks=100 +cube16 -site={mysitedir}
cd object
make -j
```

And `AMUSE` can be setup and the individual codes built. Note we need to make some under-the-hood changes to the .mk file after `AMUSE` is configured.

```
cd amuse
./configure --with-fftw=$MODLOC/FFTW/3.3.9-gompi-2021a
    --with-hdf5=$MODLOC/HDF5/1.10.7-gompi-2021a/bin/h5c++ --with-gmp=$CONDA_PATH --with-mpfr=$CONDA_PATH
    --with-gsl-prefix=/home/bpolak/TORCH/env/amuse_env
#note: some users found that "./configure" was sufficient without any of the --with flags.

#edit in amuse/config.mk:
```

```
MPIFC=/sw/arch/Centos8/EB_production/2021/software/OpenMPI/4.1.1-GCC-10.3.0/bin/mpifort
FCFLAGS=-g -O2 -fPIC -fallow-argument-mismatch
pip install -e .
make framework
make flash.code
make kepler.code
make ph4.code
make seba.code
make smalln.code
make petar.code
```

B.4 Mendel

The below modules and setup notes are from Eric Andersson.

All python related software was installed with `pip` rather than `conda` (Medndel did not have `conda` at the time). `python venv` was used to create a clean python environment for the installation, see details below.

Load the following libraries (note that FFTW is not available on Mendel as of March 21st, 2023) and export all necessary paths.

```
module load Python/python-3.8.5
module load OpenMPI/openmpi-4.0.4
module load HDF5/hdf5-1.10.1

export MPI_HOME=/usr/local/software/OpenMPI/openmpi-4.0.4
export MPI_RUN=/usr/local/software/OpenMPI/openmpi-4.0.4/bin/mpirun
export PATH=$MPI_HOME/bin:$PATH
export MANPATH=$MPI_HOME/share/man:$MANPATH
export LD_RUN_PATH=$MPI_HOME/lib:$LD_RUN_PATH
export LD_LIBRARY_PATH=$MPI_HOME/lib:$LD_LIBRARY_PATH
export CPATH=$MPI_HOME/include:$CPATH

export HDF5_HOME=/usr/local/software/HDF5/hdf5-1.10.1
export HDF5DIR=/usr/local/software/HDF5/hdf5-1.10.1/lib
export HDF5INCLUDE=/usr/local/software/HDF5/hdf5-1.10.1/include
export HDF5LIB="hdf5_fortran -lhdf5"
export PATH=${HDF5_HOME}/bin:$PATH
export LD_LIBRARY_PATH=${HDF5_HOME}/lib:$LD_LIBRARY_PATH
export LD_RUN_PATH=${HDF5_HOME}/lib:$LD_RUN_PATH
export LIBRARY_PATH=${HDF5_HOME}/lib:$LIBRARY_PATH
export C_INCLUDE_PATH=${HDF5_HOME}/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=${HDF5_HOME}/include:$CPLUS_INCLUDE_PATH
```

Create a clean python environment and install the following packages. Note that the Quick Guide asks for us to install `gmp` and `mpfr`. These are both included in `gmpy2`.

```
mkdir {directory/for/venv/}
cd {directory/for/venv/}
python3 -m venv {env_name}
source {directory/for/venv/env_name}/bin/activate
pip install ipython matplotlib numpy scipy docutils gsl gmpy2 h5py nose mpi4py
```

Follow the Quick Start Guide to set up AMUSE, FLASH and Torch.

In the FLASH Makefile, update the paths to MPI and HDF5 to point to `$MPI_HOME` and `$HDF5_HOME`. All other paths are left blank.

Instead of supplying ./configure with arguments when setting up AMUSE, all relevant paths are updated in `$AMUSE_DIR/config.mk` directly. Note that since `FFTW` is not on Mendel this is one part where the Mendel set-up deviates from the Quick Start Guide. Ignoring `FFTW` is fine.

After this point everything is described in the guide.

B.5 Stampede2

This setup guide is credited to Brooke Polak. **STAMPEDE2 IS NOW DISCONTINUED** - We leave the build instructions here as many of the steps are similar to other systems.

After downloading AMUSE, Torch, and FLASH via git, load the following modules in ∼/.profile

```
module load gcc/9.1.0
module load impi/19.0.9
module load python3/3.8.2
module load hdf5/1.10.4
module load gsl/2.6
module load fftw3
module load mkl
```

And, specify your paths in ∼/.bashrc

```
export AMUSE_DIR=/home1/05402/bp4928/TORCH/amuse
export FLASH_DIR=/home1/05402/bp4928/TORCH/FLASH4.6.2
export TORCH_DIR=/home1/05402/bp4928/TORCH/torch
export MODLOC=/opt/apps
export MPIHOME=/opt/intel/compilers_and_libraries_2020.4.304/linux/mpi/intel64
export MPI_HOME=$MPI_HOME
export MPI_RUN=$MPI_HOME/bin/mpirun
export PATH=$MPI_HOME/bin:$PATH
export MANPATH=$MPI_HOME/share/man:$MANPATH
export LD_RUN_PATH=$MPI_HOME/lib:$LD_RUN_PATH
export LD_LIBRARY_PATH=$MPI_HOME/lib:$LD_LIBRARY_PATH
export CPATH=$MPI_HOME/include:$CPATH
export HDF5_HOME=$MODLOC/gcc9_1/hdf5/1.10.4/x86_64
export HDF5DIR=$HDF5_HOME/lib
export HDF5INCLUDE=$HDF5_HOME/include
export HDF5LIB="hdf5_fortran -lhdf5"
export PATH=${HDF5_HOME}/bin:$PATH
export LD_LIBRARY_PATH=${HDF5_HOME}/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=${TACC_MKL_LIB}:$LD_LIBRARY_PATH
export LD_RUN_PATH=${HDF5_HOME}/lib:$LD_RUN_PATH
export LIBRARY_PATH=${HDF5_HOME}/lib:$LIBRARY_PATH
export LIBRARY_PATH=${TACC_MKL_LIB}:$LIBRARY_PATH
export C_INCLUDE_PATH=${HDF5_HOME}/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=${HDF5_HOME}/include:$CPLUS_INCLUDE_PATH
```

Setup your conda environment following these steps:

```
cd /where/you/want/conda/install
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-py39_4.9.2-Linux-x86_64.sh
source ~/.profile
conda deactivate
conda config --set auto_activate_base false
source ~/.profile
mkdir {your/conda_env/directory}
conda create --prefix {your/conda_env/directory}/{env_name} python=3.9
conda activate {your/conda_env/directory}/{env_name} #give absolute path so conda can find your env
conda install ipython matplotlib numpy scipy docutils gsl gmp h5py mpfr nose
conda install liblapack (?)
conda clean --all
pip install --no-cache-dir mpi4py
```

FLASH setup:

```
cd torch
./install.sh
FLASH SETUP
in $FLASH_DIR/bin/setup.py
```

```
change sys.setcheckinterval() to sys.setswitchinterval()
in $FLASH_DIR/sites
cp -r sites/cartesius.surfsara.nl sites/stampede2.tacc.utexas.edu
in $FLASH_DIR/sites/stampede2.tacc.utexas.edu/Makefile.h set
MPI_PATH   = ${MPI_HOME}
HDF5_PATH  = ${HDF5_HOME}
and replace all instances of mpifort with mpif90
in $FLASH_DIR/sites/Aliases
add:
stampede2.tacc.utexas.edu login1.stampede2.tacc.utexas.edu
stampede2.tacc.utexas.edu login2.stampede2.tacc.utexas.edu
stampede2.tacc.utexas.edu login3.stampede2.tacc.utexas.edu
./setup Cube -auto -3d +amuseusm +amusemg +raype +handcnew +amusesinksandstars +amusewinds +enerinj
    +supportppmupwind +pm4dev +cube16 -maxblocks=50
cd object
make -j
```

AMUSE setup:

```
cd amuse
./configure --with-fftw=$TACC_FFTW3_INC --with-hdf5=$MODLOC/gcc9_1/hdf5/1.10.4/x86_64/bin/h5c++
    --with-gmp=/home1/05402/bp4928/TORCH/env/torch_env
    --with-mpfr=/home1/05402/bp4928/TORCH/env/torch_env
    --with-gsl-prefix=/home1/05402/bp4928/TORCH/env/torch_env
pip install -e .
make framework
make flash.code
make kepler.code
make ph4.code
make seba.code
make smalln.code
--- petar
in amuse/src/amuse/community/petar/Makefile: if commented out, uncomment the following:
add -I flag before {$INCLUDE}
# arch
 CXXFLAGS += -march=core-avx2
 CXXFLAGS += -D INTRINSIC_X86
 CXXFLAGS += -D USE_SIMD
 CXXFLAGS += -D DIV_FIX
make petar.code
```

SLURM submit script:

```
#!/bin/sh
#SBATCH --job-name=test
#SBATCH -N 1
#SBATCH -n 32
#SBATCH -p skx-dev
#SBATCH --time=0:10:00
export FI_PROVIDER=tcp
ibrun -n 1 /home1/05402/bp4928/TORCH/env/torch_env/bin/python torch_user.py
```

B.6 Stampede3

Those installation instructions are from Claude Cournoyer-Cloutier and Brooke Polak, with contributions from Eric Andersson.

After downloading AMUSE and Torch via git, and downloading FLASH from the FLASH website, create a torch.sh file and load the following modules:

```
module load gcc/13.2.0
module load impi
module load python/3.9.18
module load hdf5/1.10.11
```

```
module load gsl/2.8
module load mpfr/4.2.1
module load fftw3/3.3.10
```

Also specify the following paths in torch.sh:

```
export AMUSE_DIR=/home1/10548/cournoyc/TORCH/amuse
export FLASH_DIR=/home1/10548/cournoyc/TORCH/FLASH4.6.2
export TORCH_DIR=/home1/10548/cournoyc/TORCH/torch

export PYTHONPATH=$PYTHONPATH:$TORCH_DIR
export PYTHONPATH=$PYTHONPATH:$TORCH_DIR/src
export PYTHONPATH=$PYTHONPATH:$AMUSE_DIR/test
export PYTHONPATH=$PYTHONPATH:$AMUSE_DIR/src

export MODLOC=/opt/apps

export MPIHOME=/opt/intel/oneapi/mpi/2021.11
export MPI_HOME=$MPIHOME
export MPI_RUN=$MPI_HOME/bin/mpirun
export PATH=$MPI_HOME/bin:$PATH
export MANPATH=$MPI_HOME/share/man:$MANPATH
export LD_RUN_PATH=$MPI_HOME/lib:$LD_RUN_PATH
export LD_LIBRARY_PATH=$MPI_HOME/lib:$LD_LIBRARY_PATH
export CPATH=$MPI_HOME/include:$CPATH

export HDF5_HOME=/opt/apps/gcc13/hdf5/1.10.11
export HDF5DIR=$HDF5_HOME
export HDF5INCLUDE=$TACC_HDF5_INC
export HDF5LIB="lhdf5_fortran -lhdf5"
export PATH=${HDF5_HOME}/bin:$PATH
export LD_LIBRARY_PATH=${HDF5_HOME}/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=${TACC_MKL_LIB}:$LD_LIBRARY_PATH
export LD_RUN_PATH=${HDF5_HOME}/lib:$LD_RUN_PATH
export LIBRARY_PATH=${HDF5_HOME}/lib:$LIBRARY_PATH
export LIBRARY_PATH=${TACC_MKL_LIB}:$LIBRARY_PATH
export C_INCLUDE_PATH=${HDF5_HOME}/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=${HDF5_HOME}/include:$CPLUS_INCLUDE_PATH

### this fixes an error running a job
export MV2_ENABLE_AFFINITY=0
```

You can then create your virtual environment. Using a CONDA environment should also work, but the instructions below use a standard Python virtual environment.

```
cd /home1/10548/cournoyc/TORCH
python -m venv /home1/10548/cournoyc/TORCH/torch_env
source /home1/10548/cournoyc/TORCH/torch_env/bin/activate
pip install --upgrade pip
pip install ipython matplotlib numpy scipy docutils gmp h5py nose pytest
pip install --upgrade setuptools
pip install --no-cache-dir mpi4py
```

Now, you can install and configure Torch. You should start by running

```
cd $TORCH_DIR
./install.sh
```

You can then configure FLASH, by following the instructions below. In `$FLASH_DIR/sites`, run:

```
cp -r sites/cartesius.surfsara.nl sites/stampede3.tacc.utexas.edu
```

Then in $\texttt{\$FLASH}_DIR/sites/stampede3.tacc.utexas.edu/Makefile.h, set$:

Once you have successfully compiled FLASH, you can configure and build AMUSE. Go the AMUSE directory and un the following commands:

```
./configure  # It should find the correct paths on its own
make framework
```

To install the AMUSE flash worker, set the following in amuse/src/amuse/community/flash/Makefile:

```
opt:   FCFLAGS = -O3 -march=native -fallow-argument-mismatch
```

In amuse/config.mk, set:

```
FCFLAGS=-g -O2 -fPIC -fallow-argument-mismatch
```

To install the AMUSE petar work, set the following in amuse/src/amuse/community/petar/Makefile (uncomment if commented out) and add the -I flag beofre $INCLUDE:

```
# arch
CXXFLAGS += -march=core-avx2
CXXFLAGS += -D INTRINSIC_X86
CXXFLAGS += -D USE_SIMD
CXXFLAGS += -D DIV_FIX
```

From there, you can make the worker codes:

```
make flash.code petar.code seba.code kepler.code smalln.code ph4.code
```

SLURM submit script:

```
#!/bin/sh
#SBATCH --job-name=test
#SBATCH -N 1
#SBATCH -n 32
#SBATCH -p skx-dev
#SBATCH --time=0:10:00
export FI_PROVIDER=psm3

source /home1/10548/cournoyc/TORCH/torch_env/bin/activate

ibrun -n 1 python torch_user.py
```

C Flash file structure

Below is the file structure of the Flash code as defined when using ./setup command for the turbulent sphere example in Section ??.

```
 +amuseUSM
       +amuseUnsplit
             += Driver/DriverMain/Unsplit/Couple_AMUSE
                   Driver_computeDt.F90
                   Driver_evolveFlash.F90
       +usm
             += physics/Hydro/HydroMain/unsplit/MHD_StaggeredMesh
                   Config
                   Makefile
                   amr_runtime_parameters.tpl
                   hy_uhd_HLLD.F90
                   hy_uhd_addOhmicHeating.F90
                   hy_uhd_eigenVector.F90
                   hy_uhd_staggeredDivb.F90
                   hy_uhd_addBiermannFluxes.F90
                   hy_uhd_addResistiveFluxes.F90
                   hy_uhd_getElectricFields.F90
                   hy_uhd_unsplit.F90
                   hy_uhd_addHallFluxes.F90
                   hy_uhd_biermannSource.F90
                   hy_uhd_getFluxDeriv.F90
             -= physics/Hydro/HydroMain/split/MHD_8Wave
+amuseMG
       += physics/Gravity/GravityMain/Poisson/Multigrid/Couple_AMUSE
             Config
             Makefile
             Gravity_interface.F90
             Gravity_potentialListOfBlocksPDENonly.F90
             Gravity_getAccelAtPoint.F90
             Gravity_getPotentialAtPoint.F90
             Gravity_potentialListOfBlocks.F90
       -= physics/Gravity/GravityMain/Poisson/BHTree
+rayPE
       += Particles/ParticlesMain/ray_pe/MultiSourceSimple/Active/Sink
             Makefile
             pt_assignRaySink.F90
       += physics/RadTrans/RadTransMain/RayRad/SinksRad
             Config
             Makefile
             RadTrans.F90
             RadTrans_combine2.F90
             RadTrans_computeDt.F90
             calc_ionization.F90
             rt_data.F90
             RadTrans_combine.F90
             RadTrans_combine3.F90
             RadTrans_interface.F90
             get_ionization.F90
             rt_init.F90
+handCnew
       += physics/sourceTerms/Heat/HeatMain/HeatCool/phenHeat/mol_and_dust/solver
             Config
             Makefile
             CoolVars.F90
             Cool_init.F90
             Heat.F90
             HeatCoolInterface.F90
             RadHeat.F90
             heatCool.F90
+amuseSinksAndStars
       += Particles/ParticlesMain/active/Sink/Couple_AMUSE/Couple_AMUSE_Sinks_and_Stars
```

```
            Config
            Makefile
            Particles_addNew.F90
            Particles_data.F90
            Particles_sinkCreateAccrete.F90
            Particles_updateGridVarSinksOnly.F90
            Particles_advance.F90
            Particles_init.F90
            Particles_sort.F90
            Particles_computeDt.F90
            Particles_interface.F90
            Particles_updateGridVarMassiveOnly.F90
            pt_gatherGlobal.F90
            sort.F90
            qsort2D.F90
+amuseWinds
      += Particles/ParticlesMain/active/Sink/Couple_AMUSE/wind
            Config
            Makefile
            Particles_computeDt.F90
            Particles_windData.F90
            Particles_advance.F90
            Particles_wind.F90
            pt_windInterface.F90
            inject_direct.F90
            overlap.F90
            normal_rand.F90
+enerInj
      += Particles/ParticlesMain/energyInjection
            Config
            Makefile
            Particles_energyInjection.F90
            pt_enerInjInterface.F90
            overlap.F90
```