General set up on Stampede2/TACC

1. Download and install Anaconda for your personal python setup on Stampede2:
    - **`wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-x86_64.sh`**
    - **`bash Anaconda3-2020.11-Linux-x86_64.sh`**
    - An alternative to anaconda is Intel Python, which can be faster for certain numpy array-crunching operations

2. Edit the file **.bashrc** in your home directory on Stampede2, adding the following lines in the indicated block for customizations:
    - **`umask 022`**
    - **`ulimit -s unlimited`**
    - **`module load TACC intel impi hdf5 gsl fftw2`**
    - **`export PATH=$HOME/anaconda3/bin:$PATH`**
    - **`export PYTHONPATH=$HOME/anaconda3/lib/python3.8/site-packages`**
        - This is needed because for some reason loading the module impi overwrites PYTHONPATH to TACC's Intel Python environment, whose packages will override anything you have on your setup

3. Run **`source ~/.bashrc`** to update your bash settings to include the stuff you just added - this only needs to be done whenever you modify your .bashrc

Getting and compiling GIZMO

1. You should not compile and run GIZMO in your **`$HOME`** directory. Go to your work/scratch directory: **`cd $WORK`** (note that **`$WORK`** has finite storage space, ~1TB, so it should fit all but the largest simulations. Alternatively, use **$SCRATCH1. IMPORTANT: `$SCRATCH1`** gets periodically purged, any file that is not accessed in the last 2 weeks might be deleted.)

2. Checkout the gizmo repo: **git clone** https://bitbucket.org/phopkins/gizmo-public.git (you can also use **`hg clone`** but note that Mercurial support is ending on Bitbucket)
3. Enter the gizmo directory (**`cd gizmo_public`**)
4. Open the file **`Makefile.systype`** and make sure that the line **`SYSTYPE="Stampede2"`** is un-commented (and the others commented out). Note that pulling a newer code version might overwrite this, so it is recommended to check after each git pull.
5. Create the file Config.sh and enter the list of compiler flags you want in it. For test problems, the appropriate flags are stored in the commented out region at the top of the *.param files for the problem you want to run.
6. Build the code: **`make`** (if recompiling, precede with a **`make clean`**)

You should now have the compiled GIZMO binary file in the gizmo directory.

Running GIZMO

You should never run GIZMO on the login nodes — it is just not good citizenship.

1. You can run gizmo interactively by doing in the terminal:

```
idev -A <allocation code> -m <how many minutes you want to use in a node>
```

At the prompt you can run on a single core using:

```
./GIZMO <filename>.params
```

Or you can run on multiple cores with:

```
mpirun -np 16 ./GIZMO <filename>.params
```

2. You can run GIZMO by submitting a batch job (see HW5 script), i.e., to call GIZMO in this script do (note you don't need to specify the number of processors in the call):

```
ibrun ./GIZMO <filename>.params
```

3. There are two ways to restart from a checkpoint (see documentation). To start from restart files (these are usually created by the run):

```
ibrun ./GIZMO <filename>.params 1
```

Alternatively, restart from a given snapshot file:

```
ibrun ./GIZMO <filename>.params 2
```

Where you must change the name of the initial condition file in the params file to match the snapshot you want to start from.

If you do not add the additional number, GIZMO will start the run from t=0.