HW#5: High-Performance Computing

1. **Hello TACC :)**

a) *Account:* Use your EID to make a TACC account if you don't have one already. Visit `https://portal.tacc.utexas.edu` and click on 'Create a TACC Account'. When you complete this step, email me your username and I will add you to the class allocation (the class allocation project number is *ast381*).

b) *Documentation:* Familiarize yourself with the documentation on how to set up your environment, compile and run jobs on Stampede2: `https://portal.tacc.utexas.edu/user-guides/stampede2`. Note you do not need to install all dependencies from scratch into your home directory – instead you may load the pre-installed modules. See what is available typing 'module avail' at the command line.

2. **Computing time, please!**

In this problem you will evaluate the scaling and parallel performance of a parallel code on STAMPEDE2 and write subsections of a computing allocation request. If you are using a parallel code for your research you can use that code for this homework. Otherwise, use HYPERION to complete the exercises below. If you have more than one project allocation on TACC, you can add `#SBATCH -A ast381` to your submission script to charge our class allocation for the computing time you use. See notes below on submitting jobs.

a) *Scaling and Performance:* Install your parallel code of choice on Stampede2 and get it running. Set up and run a weak and a strong scaling test. Make plots of efficiency versus number of processors for each of your tests with the results from 1 to at least 64 cores. Include at least 5 points in each of your scaling relationships. To obtain consistent results, make sure your jobs run on the same type of processor – on *Stampede2* that means either KNL or SKX processors.

b) *Code Profiling (optional):* When a code doesn't perform well (and even if it does!) it is useful to figure out where the code spends its time. A variety of profiling tools have been developed to "time" code performance during execution. Two of these, *VTune* and *gprof*, are available on TACC machines (see documentation). Pick one of these and profile a code's parallel performance. This will require compiling in debug mode. (For Hyperion that means adding `enable_debug = "yes"` in the configure file.) Where is the code spending most of its time? How does this change for different numbers of processors and/or work loads? Are the results reasonable, i.e., do you see a way to improve the performance?

c) *Describing Performance:* Using the Simon et al. code performance document as an example, write a summary of your tests and the results. Evaluate whether the results are "good" or not. (Note: if you are working with HYPERION, the Robitaille (2011) HYPERION method paper includes some discussion and results for how HYPERION scales, which are a useful point of reference.) Comment on any features in the plots. If you find your code doesn't scale "well," you may want to experiment with the problem setup, e.g., in terms of the size and work per core.)

d) *Production Science:* An important component of a computing time request is the estimate and justification for the amount of requested computing time and memory storage. Formulate a substantive science project using HYPERION or your research code. The project should require a significant amount of HPC resources (e.g., $> 10^3$ hours) but not so much that it is infeasible to complete during a typical proposal cycle (1 year, plus allowing for other users to also use the machine!). Using the example XSEDE proposals (`https://portal.xsede.org/allocations/research`), your results from above, and our class notes as guides, write a request for computing resources at TACC. The request should include a brief description of the proposed problem, outline the runs to be carried out and present a quantitative estimate of how much time and memory the calculations will require. *The intent here is not to write a complete 10-page proposal (much of which is science background) but to instead focus on designing a computing study, describing its execution and justifying computing resources (e.g., 1-2 pages).*

3. **AST381 Journal Club.**

Find a recent journal article that relates to computational astrophysics and material covered in this class. Submit the article and arXiv link. I will review and approve it.

You will present your article (15 min) to the class on one of the following dates: April 21, 26, 28 or May 5. Your presentation should not simply summarize the article and its results. Instead you should focus on the numerical method and think critically about the computational aspects of the research. Be prepared to discuss: How does this article relate to the class? What numerical methods are used? What are the strengths and weaknesses of the numerical approach with respect to the specific astrophysics problem? Is the simulation resolution sufficient for the problem and are the calculations converged? Are the results robust – why or why not? If you were a referee would you request additional information or calculations?

Addendum: Notes on submitting jobs

There are two ways to submit jobs:

1. You can run in 'real time' by submitting an interactive job. The command looks like:
```
idev -p normal -N 2 -n 64 -m 30
```
which starts a job in the normal queue that will run on n cores distributed between N nodes for a wallclock time of 30 minutes. Once the session starts you can run your job like:
```
mpirun -n 64 hyperion input output > jobinfo
```
.

Note your *Hyperion* command should select the correct executable, but you can manually call the MPI version, e.g., `hyperion_sph_mpi`.

2. You can also submit a bunch of runs at once to the compute nodes. Then you will need to make a script with the appropriate settings:

```
#!/bin/sh

#SBATCH -N 1 # Number of nodes

#SBATCH -n 68 # Number of cores

#SBATCH -p normal # Queue

#SBATCH -J hyperion # Job name

#SBATCH -o hyperion.o%j # Name of stdout output file

#SBATCH -e hyperion.e%j # Name of stderr error file

#SBATCH -A ast381

#SBATCH -t 02:00:00 # time in hours:minutes:sec

ibrun ./hyperion input output
```

Then submit the script by typing `sbatch myscript` at the command line.