1. Build MAPL - Follow "<https://github.com/GEOS-ESM/MAPL/wiki/Building-and-Testing-MAPL-as-a-standalone>" but add "git checkout develop" before the "mkdir build" step.

You may want to build on a batch node.

In your install/bin directory, you will have the executable ExtDataDriver.x

2. Test runs to mimic GEOS GCM

Two groups: Checkpoint followed by Restart

History followed by Extdata

Checkpoint: Writes data

Restart: Reads data written by Checkpoint

History: Writes data

Extdata: Reads data written by History

3. /discover/nobackup/aoloso/for\_Lucas contains two directories corresponding to the two groups i.e. checkpoint\_restart and history\_extdata.

Each directory contains sample setups for the tests.

checkpoint\_restart:

AGCM1.rc, CAP1.rc, and HISTORY1.rc are for checkpointing

AGCM2.rc, CAP2.rc, and HISTORY2.rc are for restart

Please note HISTORY?.rc is empty and extdata is not referenced at all - focus is on checkpoint and restart

history-extdata:

Similar arrangement except HISTORY1.rc contains data for the history part and extdata.yaml contains data for the ExtData part.

How to run (don't forget to always source g5\_modules):

checkpoint:

1. Comment out CAP2.rc in CAP.rc (insert # at the beginning of the line)
2. Total number of MPI processes (hence CPU cores) needed is the product of NX and NY in AGCM?.rc. The example provided requires 24 cores i.e. 2\*12.
3. Simply do "mpirun -np <num\_procs> <full\_path\_to\_ExtDataDriver.x>
4. Successful run produces file root\_import\_checkpoint.
5. Profiler provides time it takes to write data

restart:

1. Uncomment CAP2.rc and comment out CAP1.rc
2. Run the same was as before (i.e. as in step iii for checkpoint)
3. Profiler provides time it takes to read data

4. Benchmark the following resolutions:

Root.LM: 91

Root.IM\_WORLD: 720

Root.LM: 91

Root.IM\_WORLD: 1440

Root.LM: 91

Root.IM\_WORLD: 2880

Root.LM: 91

Root.IM\_WORLD: 5760

Root.LM: 181

Root.IM\_WORLD: 2880

Root.LM: 181

Root.IM\_WORLD: 5760

Please note that as you increase resolution, you need to increase correspondingly the number of total CPU cores (hence nodes). Typically, when you double the horizontal resolution, you may need to quadruple the number of nodes. Also note that the parameter NY must always be a multiple of 6 due to the fact that GEOS is on cubed-sphere grid.

Also please note to change Root.GRIDNAME AGCM?.rc files to reflect the horizontal resolution i.e. PE<IM\_WORLD>x<IM\_WORLD\*6>-CF. For example For IM\_WORLD=720, it will be PE720x4320-CF.

5. Variant of checkpoint and restart to also benchmark:

As resolution increases, for memory reasons, in real life, we split the data by cubed-sphere face such that instead of the entire grid written to or read from a single file, the data is split into six parts, each part corresponding to a face of the cubed-sphere. To achieve this, there are three additional parameters added to the AGCM?.rc files. For checkpoint (AGCM1.rc), the parameters are:

NUM\_READERS: 6

NUM\_WRITERS: 6

WRITE\_RESTART\_BY\_FACE: YES

For restart (AGCM2.rc), the parameters are

NUM\_READERS: 6

NUM\_WRITERS: 6

READ\_RESTART\_BY\_FACE: YES

You can play with NUM\_READERS and NUM\_WRITERS to see their effects on timing. They are both set to 6 in the example. Of course, you need to uncomment them to test this feature.

6. Follow the same process described for checkpoint-restart for history-extdata.

The variant described in 5 for checkpoint-restart does not apply to history-extdata.

However, history-extdata does have its own variant where we use what we call the o-server (that runs on a separate MPI communicator to asynchronously write data such that communication is overlapped with data dump). To benchmark this variant:

i. Request excess nodes more than you need for the model

ii. Supply arguments to ExtDataDriver.x.

To demonstrate this for our example case, we know we need 24 cores to run the model. That fits in one Skylake node, for example. To exercise the o-server, we will then need at least two nodes, one for the model, one for the o-server. The run command will look like:

mpirun -np 80 <full\_path\_to\_ExtDataDriver.x> --npes\_model 24 --nodes\_ouput\_server 1 --oserver\_type multigroup --npes\_backend\_pernode 16

In this example, we have a total of 80 cores (2 Skylake nodes). We are using only 24 cores for the model since NX\*NY=2\*12=24, one exclusive node for the oserver, and 16 backend processes to handle data writing. Please note that this is a very small case. For the higher resolutions, the number of nodes for the model will be much larger than the number of nodes for the o-server. For example, one might have over 200 nodes for the model and only 8 nodes for the o-server with each o-server node having 2 backend processes.

Try to benchmark with the asynchronous o-server. Play with the --nodes\_ouput\_server and --npes\_backend\_pernode parameters.