Reproducing ECCOv4r3 on TACC machines

An T. Nguyen, Tim Smith, Ou Wang

May 23, 2019

1 File System

1.1 community directory

The directory **/work/projects/aci/ECCO/community/**
should be accessible to ALL members of the ECCO School 2019 (those who have registered for TACC access). This is where we will host all the ECCOv4r3 data required to rerun the set up. In the following we will refer to this directory as **communitydir**.

There is an example directory set up by An Nguyen, along with example bash scripts and extra files needed for the TACC machine stampede2, so that users can browse through and compare while trying to reproduce the ECCOv4r3 run:
$communitydir/**atnguyen_example/** (extra_namelists_tiles/ , *.bash)

1.2 ∼/.bashrc

A bashrc template has been created for new users at: $communitydir/computing/env/bashrc_stampede2.
To set up environment, libraries, and modules, copy this file to your home directory and source it:

```
cp $communitydir/computing/env/bashrc_stampede2 ~/.bashrc
source ~/.bashrc
```

Optional: Prior to sourcing the ∼/.bashrc file, you can define several environment variables to designate some directories that will be extensively used later, by editing the ∼/.bashrc file and adding these lines to the bottom of the file.

```
export communitydir=/work/projects/aci/ECCO/community/
export binarydir=$communitydir/ECCO/ECCOv4/Release3/input.ecco_v4r3
export WORKINGDIR=$WORK/
export basedir=$WORKINGDIR/MITgcm/verification/release3
```

Then, save the ∼/.bashrc file and re-source it.

2 Reproducing the ECCOv4r3 run

General instructions to download the code, compile, and rerun the ECCOv4r3 set up are available at (Note that MAC "Safari" browser can **not** access this file, so you need to use a different browser to open it.):

`ftp://ecco.jpl.nasa.gov/Version4/Release3/doc/v4r3_reproduction_howto.pdf`

You can refer to the above PDF document for further clarification when needed. However, the instruction provided in this tutorial is sufficient for users to successfully download, compile, and run the forward and adjoint ECCOv4r3 solution without needing any external information. The compilation of the MITgcm code is machine dependent. In this section, we provide instructions for how to compile and run on the TACC supercomputer stampede2, using Skylake nodes (48-cpus per node, 192GB memory per node, 4GB/cpu). Prior to downloading the code, create a working directory. For all users we recommend you set your $WORKINGDIR to be the same as your own $WORK directory. Refer to Section 1.2 for how to do this.

1. Downloading the MITgcm and ECCOv4r3 code:

   ```
   cd $WORKINGDIR
   export CVSROOT=‘:pserver:cvsanon@mitgcm.org:/u/gcmpack’
   cvs login
    (enter the CVS password: cvsanon)
   cvs co -P -D "2017-04-27 8:00" MITgcm_code

   cd MITgcm
   mkdir -p verification/release3
   cd verification/release3
   cvs co -P MITgcm_contrib/ecco_utils/ecco_v4_release3_devel/code
   mv MITgcm_contrib/ecco_utils/ecco_v4_release3_devel/code .
   \rm -rf MITgcm_contrib
   ```

   Note: make sure to type single quotes for ‘:pserver:cvsanon@mitgcm.org:/u/gcmpack’ and double quotes in “2017-04-27 8:00”; copy and paste from a pdf file may not work properly.

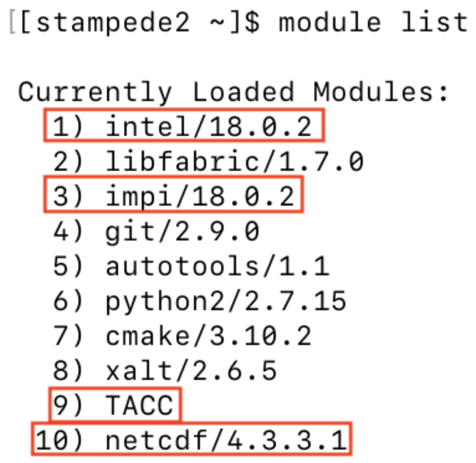
   After downloading the code, run this command to define your basedir and go into that directory:

   ```
   basedir=$WORKINGDIR/MITgcm/verification/release3
   cd $basedir
   ```

2. Run the following command to set up the binary directory:

   ```
   binarydir=$communitydir/ECCO/ECCOv4/Release3/input.ecco_v4r3
   ```

   This is where all the required inputs for ECCOv4r3 resides; more on this section later, for now, go to step 3
   $binarydir/input_ecco/
   $binarydir/input_forcing/
   $binarydir/input_init/
   $binarydir/input_init/NAMELIST/
   $binarydir/input_init/error_weight/ctrl_weight/
   $binarydir/input_init/error_weight/data_error/
   $binarydir/xx/
   $binarydir/profiles_30x30/



```
[[stampede2 ~]$ module list

 Currently Loaded Modules:
   1) intel/18.0.2
   2) libfabric/1.7.0
   3) impi/18.0.2
   4) git/2.9.0
   5) autotools/1.1
   6) python2/2.7.15
   7) cmake/3.10.2
   8) xalt/2.6.5
   9) TACC
  10) netcdf/4.3.3.1
```

Figure 1: Important modules highlighted in red that are needed to successfully compile the ECCOv4r3 code on stampede2.

3. Copy $communitydir/atnguyen_example/extra_namelists_tiles/SIZE.h_* to the code directory:

   ```
   cp $communitydir/atnguyen_example/extra_namelists_tiles/SIZE.h_* $basedir/code
   ```

   Depending on your resource availability (e.g., number of CPUs and nodes you have access to on stampede2), replace the file **SIZE.h** in the **code** directory with either SIZE.h_45x45x48 for 48-CPUs or SIZE.h_30x30x96 for 96-CPUs or SIZE.h_15x30x187 for 187-CPUs.

   The more CPUs you use the faster the model will run. However, each student has been allocated 48 cores so if you want to compile the model to run on 96 or 187 cores, please make teams of two or four people, respectively.

   For example, to compile the model to use 96 cores use the following commands:

   ```
   cd code
   cp -p SIZE.h_30x30x96 SIZE.h
   ```

4. Compile the **forward** code.

   (a) Prior to compiling, check what modules are loaded by typing the command:

   ```
   module list
   ```

   and compare your list with that shown in Fig. 1. If any of the modules highlighted in red in the list is not loaded, you should re-check your ∼/.bashrc to load modules correctly and re-source the file. See Section 1.2 for how to do this.

   (b) The **optfile** you will need for compiling is shown below, just for your knowledge, we will use it in the next step
   $communitydir/computing/optfiles/linux_amd64_ifort+mpi_stampede2_skx

(c) Compile the **forward** code with the appropriate optfile.

```
cd $basedir
mkdir build
cd build
../../../tools/genmake2 -mods=../code -mpi \
    -optfile=$communitydir/computing/optfiles/linux_amd64_ifort+mpi_stampede2_skx
make depend
make all
mv mitgcmuv mitgcmuv_30x30x96 #(if SIZE.h_30x30x96 was chosen)
```

NOTE: The last step above:

```
mv mitgcmuv mitgcmuv_30x30x96
```

renames your **mitgcmuv** executable to **mitgcmuv_30x30x96**. The "30x30x96" indicates the size of the tiles (30x30) and the number of CPUs (96) that will be used to run the model (as determined by the SIZE.h you selected). If, for example, you selected **SIZE.h_45x45x48** then you would rename your **mitgcmuv** as:

```
mv mitgcmuv mitgcmuv_45x45x48
```

NOTE: **Recompiling**: If something goes wrong in the **make depend** or **make all** steps, stay in the build directory and do the following to recompile:

```
make CLEAN
make makefile
make depend
make all
mv mitgcmuv mitgcmuv_30x30x96 #(for 96 CPUs)
```

5. Copy the namelists.

```
cp -r $binarydir/input_init/NAMELIST $basedir/my_namelists
chmod -R u+w $basedir/my_namelists
cp $communitydir/atnguyen_example/extra_namelists_tiles/data* $basedir/my_namelists
cd $basedir/my_namelists/
cp data_exch2_30x30x96 data.exch2 #(if SIZE.h_30x30x96 was chosen)
```

Go to the my_namelists directory,

```
cd $basedir/my_namelists
```

In MITgcm jargon, "namelists" are text files where we specify model run-time parameters (parameters that we can change without having to recompile the model). Example run-time model parameters that are included in these MITgcm namelist files include (namelist file in parentheses):

(a) model calendar start time (data.cal)

(b) length of model time step (data)

(c) number of time steps to run (data)

(d) model initial condition filenames (data)

(e) model atmospheric boundary condition filenames (data.exf)

(f) list of model fields to output to disk and frequency of output (data.diagnostics)

(g) which model packages or submodels to use (data.pkg)

(h) observations for model-data misfit cost function (data.ecco)

(i) model control parameter adjustments (data.ctrl)

The first time you run the model we suggest that you change two things: (1) the number of model timesteps in the namelist file **data** to 59 and (2) which **data.diagnostics** namelist file to use from the default to **data.diagnostics.3day**. **data.diagnostics.3day** instructs the model to output the daily-average of two fields, *ETAN* and *THETA*.

Change 1: *nTimesteps* :

```
nTimesteps = 59
```

Change 2: *data.diagnostics*:

```
rm data.diagnostics          # delete default
cp data.diagnostics.3day data.diagnostics
```

6. The job script: Copy the script below to your $basedir directory,

```
cd $basedir
cp -p $communitydir/atnguyen_example/script_r3_stampede2.bash .
```

Use the above script as an example, edit it and make necessary changes. In particular:
Set the name for the job in the batch queue (-J) and the out (-o) and error (-e) files

```
#SBATCH -J eccov4r3
#SBATCH -o eccov4r3.%j.out
#SBATCH -e eccov4r3.%j.err
```

Specify the queue to which the job is submitted (-p) and the maximum running time allowed for the job (-t)

```
#SBATCH -p skx-normal
#SBATCH -t 8:00:00
```

Specify the number of nodes requested (-N) and the number of cpus (-n). For the 30x30x96 example size, choose:

```
#SBATCH -N  2
#SBATCH -n 96
```

Set the mail-user to your own email address and the type of email you want to receive

```
#SBATCH --mail-user=YOUR_EMAIL@XYZ.com
#SBATCH --mail-type=begin
#SBATCH --mail-type=end
```

Note: in the first block of the script file, lines starting with a double ## before the word SBATCH are commented out, whereas lines starting with single # are active. In the rest of the file, lines starting with single # are comment lines.

Make sure that the modules section in the bash script file is as follows:

```
#--- 0.load modules ------
module purge
module load intel/18.0.2
module load impi/18.0.2
module load TACC
module load netcdf/4.3.3.1
```

Make sure the dimensions and number of processors are set correctly to match the dimensions used to compile the code set in SIZE.h (e.g. for the choice of 30x30x96 as in the example above for SIZE.h)

```
nprocs=96
snx=30
sny=30
```

The "use_optim" flag is only relevant for much later application (See Section 6). For now, it should be set to the default value of zero:

```
use_optim=0
```

The iteration number should be the default value of 59, which corresponds to the ECCOv4r3 solution:

```
iter=59
```

Set the appropriate directories to run the job and write output, in particular set the appropriate basedir (yours) and exedir (to write output). The directory exedir should be on your /scratch directory, where large output files can be stored, not on your $WORKINGDIR, where the code and script files are kept.

```
#--- 2.set dir ------------
basedir=YOUR_BASEDIR #(should be set to $WORKINGDIR/MITgcm/verification/release3)
builddir=$basedir/build${forwadj}
codedir=$basedir/code
inputdir=$basedir/my_namelists
binarydir=/work/projects/aci/ECCO/community/ECCO/ECCOv4/Release3/input.ecco_v4r3
scratchdir=/scratch/YOUR_TACC_NUMBER/YOUR_USERNAME/(YOUR_DIRECTORY)
exedir=$scratchdir/run${whichexp}_it${ext2}${forwadj}_np${nprocs}
```

7. Submit the job,

```
sbatch script_r3_stampede2.bash
```

Table 1 lists a few useful commands to submit and monitor your jobs.

command	explanation
sbatch script_r3_stampede2.bash	submit the run script
squeue -u yourusername	check the status of your job
scancel yourjobid	kill your job

Table 1: Useful commands for batch jobs. More information can be found in the TACC user guide page for the job scheduler https://portal.tacc.utexas.edu/user-guides/stampede2#slurm-job-scheduler

After you submit your job script, and if the job enters the queue properly, two log files will be written to $basedir: eccov4r3.JOBIDNUMBER.err and eccov4r3.JOBIDNUMBER.out, that are generated by the job scheduler. If/when your job has been successfully finished, the last line in eccov4r3.JOBIDNUMBER.err is "NORMAL END", and the output of "squeue -u yourusername" will not include the job JOBIDNUMBER.

3 Results of Forward Run

3.1 Expected result

When a run is finished, one should expect to see the following files in their run directory:

```
STDOUT.XXXX : monitored mean/max/min of model state variables .
STDERR.XXXX : any warnings
diags/*.{data,meta} : outputs of the model state in binary format.
profiles/* : output from pkg/profiles
m_*.{data,meta} : output from pkg/ecco
xx* : output from pkg/ctrl
```

3.2 Clean up the run directory

The run directory will have a lot of files. An Nguyen has made three scripts available for tidying up the directory to help users navigate the outputs easier. These scripts are at:

$communitydir/atnguyen_example/script_cleanup_rundir/
run_cleanup : consolidate files into separate directories, list binary and forcing inputs for book-keeping
run_cleanup_tar: tar the large input and grid binaries
run_cleanup_rm: remove files that have already be tarred and zipped.

One can copy these scripts to the run dir, make them executable and run them using these commands:

```
chmod u+x run_cleanup
chmod u+x run_cleanup_tar
./run_cleanup
./run_cleanup_tar

(chmod u+x run_cleanup_rm)
(./run_cleanup_rm)
```

Be careful of the last command run_cleanup_rm, because it is removing links and files. Make sure you have tarred them correctly first prior to executing this command.

Example output:
$communitydir/atnguyen_example/MITgcm/verification/release3/run_r3_stampede2_it0059_np96
$communitydir/atnguyen_example/MITgcm/verification/release3/run_r3_stampede2_it0059_np48

NOTE: If you are using **data.diagnostics.3day** or **data.diagnostics.clean** then we suggest using the Python script **reorganize_diags.py** written by Ou Wang. This routine moves each diagnostic output variable into its own directory in 'run_dir/diags/'. From the run directory do:

```
cp $communitydir/Hands_on/reproducingv4r3/reorganize_diags.py .
python reorganize_diags.py
```

4 Instruction for the adjoint run

4.1 taf license

For all students and instructors: if you already have a taf licence at your institute, e.g., MIT or JPL or UT-Austin, copy your key over to stampede2 . If you do not yet have a license, you should do the following:

```
cp $communitydir/bin/taf_keys/taf* ~/.ssh/
cd ~/.ssh
chmod  taf
chmod go-rx taf
```

Then edit and add this line to your ∼/.bashrc:

```
export PATH=$PATH:/work/projects/aci/ECCO/community/bin
```

and resource the ∼/.bashrc file. Once you have resourced, staf should now be accessible. You can check:

```
source ~/.bashrc
which staf (should get a valid path to staf)
staf -test (should get "Your access to the TAF server is enabled.")
```

4.2 tamc.h

The adjoint run requires knowledge of the node's memory in order to balance between what can be stored in memory versus what should be written to files. This is the reccommended parameters relevant for memory on stampede2 in code/tamc.h:

```
integer nchklev_1
parameter( nchklev_1 = 12 )
integer nchklev_2
parameter( nchklev_2 = 120 )
integer nchklev_3
parameter( nchklev_3 = 150 )
```

4.3 Compiling

```
cd $basedir
cd code
cp SIZE.h_30x30x96 SIZE.h
cd ../
mkdir build_ad
cd build_ad
../../../tools/genmake2 -mods=../code -mpi \
-optfile=$communitydir/computing/optfiles/linux_amd64_ifort+mpi_stampede2_skx
make depend
make adtaf
make adall
mv mitgcmuv_ad mitgcmuv_ad_30x30x96
cd ..
```

4.4 Results

An Nguyen's example:
$communitydir/atnguyen_example/MITgcm/verification/release3/
  scripts_r3_stampede2.bash (change variable forward="" to forwadj="_ad")
  build_ad
  my_namelists (change "nTimesteps" in **data** to run shorter, e.g, 2-yr or 3-mo)
  run_r3_stampede2_it0059_ad_np96 (2-yr adjoint, using 96 cpus, took 6hrs to finish)

5 Wall clock time:

Forward:
48-cpus, 1 node: 6.67 min per model month
96-cpus, 2–4 nodes: 1.67 min per model month
187-cpus, 4–8 nodes: 1.25min per model month

Adjoint:
48-cpus, 1-node: The eccov4r3 set up with all its costs and controls will not run, even for a 3-month duration, because the nodes' memory (192GB, with about 180GB available for use) can not accommodate the load. It will require at least 2-nodes. However, if the number of costs is reduced to 1 (e.g., SST), and nchklev_1 is changed to 2 (minimal stored in memory, heavy I/O), the adjoint can run on 1 single node using all 48-cpus. Timing: ∼3hr30min for a 3-mo adjoint run for iter59, 5hr for 3-mo adjoint run for iter60 (17min to run the forward and cost calculation, 4hr23min to run the adjoint, memory usage: 162GB out of available 182GB in adjoint mode).
96-cpus, 4-nodes: 6hr to run a 2-yr adjoint **with** profiles; 1hr to run 3-mo adjoint with profiles.
187-cpus, 8-nodes: 3hr40min to run a 2-yr adjoint without profiles.

6 Optimization

This section gives instruction on how to do the "optimization", which involves running the adjoint over two iterations (e.g., iter59 and iter60), with the calculation of the control adjustments $\delta\{ctrl\}$ being done offline after iter59 and for use in iter60 to reduce the cost (model minus data misfit). More information can be found elsewhere, and here we will only provide the exact instruction for how to get it to work on stampede2 on TACC.

6.1 Getting needed parameters and files

At the end of the adjoint run at iter59, you should get two files in the run dir:

```
ecco_ctrl_MIT_CE_000.opt0059 (control vector to run iter59)
ecco_cost_MIT_CE_000.opt0059 (sensitivity of the cost to the control vector for iter59)
```

In addition, two more parameters are needed an can be obtained from the STDOUT.0000 file using the following commands:

```
grep 'ctrl-wet 13' STDOUT.0000 (length of the control vector, e.g., 16909590)
grep 'global fc' STDOUT.0000 (total cost for iter59, e.g., 0.5141D+5)
```

6.2 Downloading and compiling the line-search code

The line-search code will use the information from Section 6.1 to calculate the changes in the control space ($\delta ctrl$) corresponding to a percentage of cost reduction we want between the two iterations. The code is accessible at:

```
https://github.com/mjlosch/optim_m1qn3
```

To download and compile the code, do the following:

```
cd $WORKINGDIR/MITgcm/
git clone https://github.com/mjlosch/optim_m1qn3.git
cd optim_m1qn3/src
cp $communitydir/atnguyen_example/MITgcm/optim_m1qn3/src/Makefile ./Makefile
```

where the last command copies over An Nguyen's Makefile, which has the correct setting for stampede2. Edit the Makefile and make the following changes specific to your run and adjoint build_ad directories:

Line 13:

```
MAKEDEPEND=$WORKINGDIR/MITgcm/tools/xmakedepend
```

Line 44:

```
-I../../verification/release3/build_ad (or where your build_ad is)
```

Line 54: change it to the number you obtained from the grep 'ctrl-wet 13' above,

```
-DMAX_INDEPEND=16909590         \
```

Save the file, then do:

```
make clean
make depend
make
```

The result should be an executable file named optim.x .

6.3 Preparing the OPTIM directory

Now execute the following set of commands

```
cd $basedir
mkdir OPTIM
cd OPTIM
cp $communitydir/atnguyen_example/MITgcm/verification/release3/OPTIM/data* ./
cp $communitydir/atnguyen_example/MITgcm/verification/release3/OPTIM/script_optim.bash ./
```

Next, copy the results from your adjoint run directory (e.g., iter59) to here:

```
cp $YOURRUNDIR/ecco_c*.opt0059 ./
```

And copy the executable optim.x to here:

```
cp $WORKINGDIR/MITgcm/optim_m1qn3/src/optim.x ./
```

Edit data.optim, change the following lines:
Line 3: Should be changed to the last iteration you've run, e.g., 59:

```
optimcycle=59,
```

Line 7: *fmin* is the cost you're aiming for. Calculate this using the result from the grep 'global fc' above, and let's say you aim to reduce that number by 5%, so the result should be: 0.5141D+5 * 0.95 = .488D+5, enter this for *fmin*:

```
 fmin = .488D+5,
```

Now, edit the job script script_optim.bash, make changes as follows:
Line 10: change to your email address
Line 22: point to your basedir

```
basedir=$WORKINGDIR/MITgcm/verification/release3/
```

Save the file, and submit the job:

```
sbatch script_optim.bash
```

If successful, you should get in the OPTIM directory the file

```
ecco_ctrl_MIT_CE_000.opt0060 (the control vector adjustments for the next iteration)
```

Now, if one repeats the steps outlined in Section 4 for iter60, one expects a ∼5% reduction in the cost as reported in costfunction0060. Here is an example from An Nguyen's runs:

```
grep fc $communitydir/atnguyen_example/MITgcm/verification/release3/run_r3_stampede2_it00*_ad_np48/costfunction*

run_r3_stampede2_it0059_ad_np48/costfunction0059: fc =   51406.8061573238       0.0000000E+00
run_r3_stampede2_it0060_ad_np48/costfunction0060: fc =   46967.6424613923       0.0000000E+00
```

The cost has reduced from 51406.806 in iter59 to 46967.642 in iter60, which is a 8.6% reduction (more than the 5% we asked for in the OPTIM dir).