MPAS Testing and Machine-specific Instructions

Overview

1. MPAS website: https://mpas-dev.github.io/
2. MPAS-Ocean User's Guide includes a quick start guide and description of all flags and variables.
3. Test cases for latest release version:
    a. https://mpas-dev.github.io/ocean/releases.html
    b. https://mpas-dev.github.io/ocean/release_6.0/release_6.0.html

COMPASS

Configuration Of Model for Prediction Across Scales Setups (COMPASS) is an automated system to set up test cases that match the current repository. All namelists and streams files begin with the default generated from the Registry.xml file, and only the changes relevant to the particular test case are altered in those files.

To begin, load the compass conda package for your particular machine. For example

```
# on LANL IC:
source /usr/projects/climate/SHARED_CLIMATE/anaconda_envs
/load_latest_compass.sh
# at NERSC:
source /global/cfs/cdirs/e3sm/software/anaconda_envs/load_latest_compass.sh
```

To install your own compass environment, see https://github.com/MPAS-Dev/MPAS-Model/blob/ocean/develop/testing_and_setup/compass/README_ocean.md

You will also need to `git clone` an MPAS repo. It is usually best to start in the ocean/develop branch.

Setting up a test case

If you are new to MPAS-Ocean, it is easiest to download a prepared test case. To see all available test cases you can make yourself in compass, start in the ocean/develop branch.:

`cd testing_and_setup/compass`

`./list_testcases.py`

and you get output like this:

`69: -o ocean -c global_ocean -r QU240 -t init`

`70: -o ocean -c global_ocean -r QU240 -t performance_test`

To set up a particular test case, you can either use the full sequence of flags:

```
./setup_testcase.py \
  --config_file general.config.ocean \
  --work_dir $WORKDIR \
  --model_runtime runtime_definitions/mpirun.xml \
  -o ocean -c global_ocean -r QU240 -t init
```

or you can replace the last line with the simple shortcut: `-n 69.`

Here `$WORKDIR` is a path, usually to your scratch space. For example,

`--work_dir /lustre/scratch4/turquoise/mpeterse/runs/191210_test_new_branch`

and `general.config.ocean` is a file that specifies directory and file paths. You can either add paths to the repo file in that directory, or you can use these files, which use my paths:

- LANL IC: general.config.ocean_turq
- NERSC: general.config.ocean_cori

You should change the MPAS repo to your directories to test your own code.

The `--model_runtime` is either `srun` or `mpirun`, depending which modules you loaded.

Running a test case

After compiling the code and setting up a test case, you can log into an interactive node (see machine instructions below) and then

```
cd $WORKDIR
cd ocean/global_ocean/QU240/init
./run.py
```

Note the sequence of subdirectories is the same as the flags used to set up the case.

In order to run a bit-for-bit test with a previous case, use `-b $PREVIOUS_WORKDIR`.

Regression suites

We have assembles suites of test cases for code regressions and bit-for-bit testing. They are here:

`ls testing_and_setup/compass/ocean/regression_suites/`

`land_ice_fluxes.xml light.xml nightly.xml rpe_tests.xml`

You can set up a regression as follows:

```
cd testing_and_setup/compass/
./manage_regression_suite.py -s \
   --config_file general.config.ocean \
   -t ocean/regression_suites/nightly.xml \
   --model_runtime runtime_definitions/mpirun.xml \
   --work_dir $WORKDIR
```

where the details are the same as for setting up a case. You can use the same `general.config.ocean` file and use `-b $PREVIOUS_WORKDIR` for bit-for-bit comparison of the output with a previous nightly regression suite.

To run the regression suite, log into an interactive node, load your modules, and

```
cd $WORKDIR
./nightly_ocean_test_suite.py
```

Machine-Specific Instructions

The simplest way to set up a new repo is:

```
git clone git@github.com:MPAS-Dev/MPAS.git your_new_branch
cd your_new_branch
git checkout -b your_new_branch origin/ocean/develop
```

Note that for ocean development, it is best to branch from ocean/develop. You can also make a fork on github and then

```
git add remote my_alias git@github.com:my_github_username/MPAS.git
git fetch my_alias
```

Slurm job queueing

Most systems now use slurm. Here are some basic commands:

```
salloc -N 1 -t 2:0:0 # interactive job (see machine specific versions below)
sbatch script # submit a script
squeue # show all jobs
squeue -u $my_moniker # show only your jobs
scancel jobID # cancel a job
```

Also see:

- Introduction to Slurm at LANL
- Basic Slurm Guide for LANL HPC Users
- Slurm Command Summary
- Slurm: Running Jobs on HPC Platforms
- example of batch scripts: https://hpc.lanl.gov/slurm#batch_scripts

LANL Institutional Computing

See https://int.lanl.gov/hpc/institutional-computing/index.shtml

DST Calendar: http://hpccalendar.lanl.gov/

Machine specifications: grizzly badger turquoise network

login: `ssh -t $my_moniker@wtrw.lanl.gov ssh gr-fe` or ba-fe

File locations:

- small home directory, for start-up scripts only: `/users/$my_moniker`
- home directory, backed up: `/usr/projects/climate/$my_moniker`
- scratch space, not backed up: `/lustre/scratch3/turquoise/$my_moniker` or scratch4

Check compute time:

- `sacctmgr list assoc user=$my_moniker format=Cluster,Account%18,Partition,QOS%45`
- Which is my default account? `sacctmgr list user $my_moniker`
- `sshare -a | head -2; sshare -a | grep $ACCOUNT | head -1`
- `sreport -t Hours cluster AccountUtilizationByUser start=2019-12-02 | grep $ACCOUNT`
- check job priority: `sshare -a | head -2; sshare -a | grep $ACCOUNT`
- https://hpcinfo.lanl.gov
- https://hpcstats.lanl.gov

Check disk usage:

- your home space: `chkhome`
- total disk usage in Petabytes: `df -BP |head -n 1; df -BP|grep climate; df -BP |grep scratch`

Archiving

- see https://hpc.lanl.gov/turquoise_archive
- archive front end: `ssh -t $my_moniker@wtrw.lanl.gov ssh ar-tn`
- storage available at: `cd /archive/<project_name>`
- you can just copy files directly into here for a particular project.

git and compass environment, for all LANL IC machines:

```
module load git
module use /usr/projects/climate/SHARED_CLIMATE/modulefiles/all/
module unload python
source /usr/projects/climate/SHARED_CLIMATE/anaconda_envs
/load_latest_compass.sh
```

LANL uses slurm. To obtain an interactive node:

```
salloc -N 1 -t 2:0:0 --qos=interactive
```

use `--account=ACCOUNT_NAME` to change to a particular account.

grizzly, gnu

```
module load gcc/5.3.0
module load openmpi/1.10.5 netcdf/4.4.1 parallel-netcdf/1.5.0 pio/1.7.2
make gfortran CORE=ocean
```

Hint: you can put the following line in your bashrc:

alias mlgnu='module purge; module load git; module use /usr/projects/climate/SHARED_CLIMATE/modulefiles/all/; module load gcc/5.3.0 openmpi/1.10.5 netcdf/4.4.1 parallel-netcdf/1.5.0 pio/1.7.2; module unload python; source /usr/projects/climate/SHARED_CLIMATE/anaconda_envs/load_latest_compass.sh; echo "loading modules anaconda, gnu, openmpi, netcdf, pnetcdf, pio for grizzly"

grizzly, intel 17

```
module load intel/17.0.1
module load openmpi/1.10.5 netcdf/4.4.1 parallel-netcdf/1.5.0 pio/1.7.2
make ifort CORE=ocean
```

grizzly, intel 19

Source this shell script:

```
#!/usr/bin/env bash

echo "Loading modules for grizzly intel 19"
module purge
source /usr/projects/climate/SHARED_CLIMATE/anaconda_envs
/load_latest_compass.sh
module use /usr/projects/climate/SHARED_CLIMATE/modulefiles/all/
module load friendly-testing
module load intel/19.0.4 intel-mpi/2019.4 hdf5-parallel/1.8.16 pnetcdf/1.
11.2 netcdf-h5parallel/4.7.3 mkl/2019.0.4 scorpio/pio2/1.10.1
export I_MPI_CC=icc
export I_MPI_CXX=icpc
export I_MPI_F77=ifort
export I_MPI_F90=ifort
```

badger, gnu

```
module use /usr/projects/climate/SHARED_CLIMATE/modulefiles/spack-lmod
/linux-rhel7-x86_64

# IC mods
module load gcc/6.4.0
module load openmpi/2.1.2
module load cmake/3.12.1
module load mkl

# spack mods
module load openmpi/2.1.2-bheb4xe/gcc/6.4.0/netcdf/4.4.1.1-zei2j6r
module load openmpi/2.1.2-bheb4xe/gcc/6.4.0/netcdf-fortran/4.4.4-v6vwmxs
module load openmpi/2.1.2-bheb4xe/gcc/6.4.0/parallel-netcdf/1.8.0-2qwcdbn
module load openmpi/2.1.2-bheb4xe/gcc/6.4.0/pio/1.10.0-ljj73au

export NETCDF=/usr/projects/climate/SHARED_CLIMATE/software/badger/spack-
install/linux-rhel7-x86_64/gcc-6.4.0/netcdf-fortran-4.4.4-
v6vwmxsv33t7pmulojlijwdbikrvmwkc
export PNETCDF=/usr/projects/climate/SHARED_CLIMATE/software/badger/spack-
install/linux-rhel7-x86_64/gcc-6.4.0/parallel-netcdf-1.8.0-
2qwcdbnjcq5pnkoqpx2s7um3s7ffo3xd
export PIO=/usr/projects/climate/SHARED_CLIMATE/software/badger/spack-
install/linux-rhel7-x86_64/gcc-6.4.0/pio-1.10.0-
ljj73au6ctgkwmh3gbd4mleljsumijys/

make gfortran CORE=ocean
```

NERSC

login: `ssh $my_moniker@cori.nersc.gov`

compass environment:

```
source /global/cfs/cdirs/e3sm/software/anaconda_envs/load_latest_compass.sh
```

interactive login:

```
# for Haswell:
salloc --partition=debug --nodes=1 --time=30:00 -C haswell

# for KNL:
salloc --partition=debug --nodes=1 --time=30:00 -C knl
```

Compute time:

- Check hours of compute usage at https://nim.nersc.gov/

File system:

- Overview: https://docs.nersc.gov/filesystems/
- home directory: `/global/homes/$my_moniker`
- scratch directory: `/global/cscratch1/sd/$my_moniker`
- Check your individual disk usage with `myquota`
- Check the group disk usage with `prjquota projectID`, i.e. `prjquota m2833` or `prjquota acme`

Archive:

- NERSC uses HPSS with the commands hsi and htar
- overview: https://docs.nersc.gov/filesystems/archive/
- E3SM uses zstash B04. zstash: HPSS long-term archiving tool

cori, gnu

```
module switch PrgEnv-intel PrgEnv-gnu
module load cray-netcdf-hdf5parallel
module load cray-parallel-netcdf/1.11.1.0
module load papi
export PIO=/global/u2/h/hgkang/my_programs/ParallelIO_1.9.23/build/pio

make gnu-nersc CORE=ocean
```

cori, intel

```
module rm intel
module load intel/18.0.1.163
module load cray-mpich/7.7.6
module load cray-hdf5-parallel/1.10.2.0
module load cray-netcdf-hdf5parallel/4.6.1.3
module load cray-parallel-netcdf/1.8.1.4
export PIO_VERSION=1.10.1
export PIO=/global/homes/m/mpeterse/libraries/pio-${PIO_VERSION}-intel

make intel-nersc CORE=ocean
```

PIO on cori

We have already compiled PIO on cori, and paths are given in the previous instructions. If you need to compile it yourself, you can do that as follows (thanks @ Xylar Asay-Davis for instructions).

```
#!/bin/bash

export PIO_VERSION=1.10.1

rm -rf ParallelIO pio-${PIO_VERSION}

git clone git@github.com:NCAR/ParallelIO.git
cd ParallelIO
git checkout pio$PIO_VERSION

cd pio

export PIOSRC=`pwd`
git clone git@github.com:PARALLELIO/genf90.git bin
git clone git@github.com:CESM-Development/CMake_Fortran_utils.git cmake
cd ../..

# Purge environment:
module rm PrgEnv-cray
module rm PrgEnv-gnu
module rm PrgEnv-intel

module load PrgEnv-intel/6.0.5
module rm intel
module load intel/18.0.1.163

module rm craype
module load craype/2.5.18

module rm pmi
module load pmi/5.0.14
```

```
module rm cray-netcdf
module rm cray-netcdf-hdf5parallel
module rm cray-parallel-netcdf
module rm cray-hdf5-parallel
module rm cray-hdf5

module rm cray-mpich
module load cray-mpich/7.7.6

# Load netcdf and pnetcdf modules
module load cray-hdf5-parallel/1.10.2.0
module load cray-netcdf-hdf5parallel/4.6.1.3
module load cray-parallel-netcdf/1.8.1.4

export NETCDF=$NETCDF_DIR
export PNETCDF=$PARALLEL_NETCDF_DIR
export PHDF5=$HDF5_DIR
export MPIROOT=$MPICH_DIR

export FC=ftn
export CC=cc
mkdir pio-${PIO_VERSION}
cd pio-${PIO_VERSION}
cmake -D NETCDF_C_DIR=$NETCDF -D NETCDF_Fortran_DIR=$NETCDF \
   -D PNETCDF_DIR=$PNETCDF -D CMAKE_VERBOSE_MAKEFILE=1 $PIOSRC
make

DEST=$HOME/libraries/pio-${PIO_VERSION}-intel
rm -rf $DEST
mkdir -p $DEST
cp *.a *.h *.mod $DEST
```

Jupyter notebook on remote data

You can run Jupyter notebooks on NERSC with direct access to scratch data as follows:

```
ssh -Y -L 8844:localhost:8844 MONIKER@cori.nersc.gov
jupyter notebook --no-browser --port 8844
# in local browser, go to:
http://localhost:8844/
```

Note that on NERSC, you can also use their Jupyter server (Jupyter.nersc.gov), it's really nice and grabs a compute node for you automatically on logon. You'll need to create a python kernel from e3sm-unified following these steps (taken from https://docs.nersc.gov/connect/jupyter/). After creating the kernel, you just go to "Change Kernel" in the Jupyter notebook and you're ready to go.

You can use one of our default Python 2, Python 3, or R kernels. If you have a Conda environment, depending on how it is installed, it may just show up in the list of kernels you can use. If not, use the following procedure to enable a custom kernel based on a Conda environment. Let's start by assuming you are a user with username `user` who wants to create a Conda environment on Cori and use it from Jupyter.

```
cori$ module load python
cori$ conda create -n myenv python=3.7 ipykernel <further-packages-to-
install>
<... installation messages ...>
cori$ source activate myenv
cori$ python -m ipykernel install --user --name myenv --display-name MyEnv
Installed kernelspec myenv in /global/u1/u/user/.local/share/jupyter/kernels
/myenv
cori$
```

Be sure to specify what version of Python interpreter you want installed. This will create and install a JSON file called a "kernel spec" in `kernel.json` at the path described in the install command output.

```
{
    "argv": [
        "/global/homes/u/user/.conda/envs/myenv/bin/python",
        "-m",
        "ipykernel_launcher",
        "-f",
        "{connection_file}"
    ],
    "display_name": "MyEnv",
    "language": "python"
}
```

Anvil/Blues

intel on anvil

First, you might want to build PIO (see below), or use the one from @ Xylar Asay-Davis referenced here:

```
source /lcrc/soft/climate/e3sm-unified/load_latest_compass.sh

module purge
module load cmake/3.14.2-gvwazz3 intel/17.0.0-pwabdn2 \
    intel-mkl/2017.1.132-6qy7y5f netcdf/4.4.1-tckdgwl netcdf-cxx/4.2-
3qkutvv \
    netcdf-fortran/4.4.4-urmb6ss mvapich2/2.2-verbs-qwuab3b \
    parallel-netcdf/1.7.0-lbykqph

export NETCDF=/blues/gpfs/software/centos7/spack-latest/opt/spack/linux-
centos7-x86_64/intel-17.0.0/netcdf-4.4.1-tckdgwl
export NETCDFF=/blues/gpfs/software/centos7/spack-latest/opt/spack/linux-
centos7-x86_64/intel-17.0.0/netcdf-fortran-4.4.4-urmb6ss
export PNETCDF=/blues/gpfs/software/centos7/spack-latest/opt/spack/linux-
centos7-x86_64/intel-17.0.0/parallel-netcdf-1.7.0-lbykqph
export PIO=/home/xylar/libraries/pio-1.10.1-intel

make ifort CORE=ocean
```

PIO on anvil

@ Xylar Asay-Davis: If you need to compile it yourself, you can do that as follows:

```
#!/bin/bash

export PIO_VERSION=1.10.1

rm -rf ParallelIO pio-${PIO_VERSION}

git clone git@github.com:NCAR/ParallelIO.git
cd ParallelIO
git checkout pio$PIO_VERSION

cd pio

export PIOSRC=`pwd`
git clone git@github.com:PARALLELIO/genf90.git bin
git clone git@github.com:CESM-Development/CMake_Fortran_utils.git cmake
cd ../..

module purge
module load cmake/3.14.2-gvwazz3 intel/17.0.0-pwabdn2 \
    intel-mkl/2017.1.132-6qy7y5f netcdf/4.4.1-tckdgwl netcdf-cxx/4.2-
3qkutvv \
    netcdf-fortran/4.4.4-urmb6ss mvapich2/2.2-verbs-qwuab3b \
    parallel-netcdf/1.7.0-lbykqph

export NETCDF_C_PATH=/blues/gpfs/software/centos7/spack-latest/opt/spack
/linux-centos7-x86_64/intel-17.0.0/netcdf-4.4.1-tckdgwl
export NETCDF_FORTRAN_PATH=/blues/gpfs/software/centos7/spack-latest/opt
/spack/linux-centos7-x86_64/intel-17.0.0/netcdf-fortran-4.4.4-urmb6ss
export PNETCDF_PATH=/blues/gpfs/software/centos7/spack-latest/opt/spack
/linux-centos7-x86_64/intel-17.0.0/parallel-netcdf-1.7.0-lbykqph

export MPIROOT=$I_MPI_ROOT

export FC=ifort
export CC=icc
mkdir pio-${PIO_VERSION}
cd pio-${PIO_VERSION}
cmake -D NETCDF_C_DIR=$NETCDF -D NETCDF_Fortran_DIR=$NETCDF \
   -D PNETCDF_DIR=$PNETCDF -D CMAKE_VERBOSE_MAKEFILE=1 $PIOSRC
make

DEST=$HOME/libraries/pio-${PIO_VERSION}-intel
rm -rf $DEST
mkdir -p $DEST
cp *.a *.h *.mod $DEST
```

Personal OSX Machine

Hyun-Gyu Kang: This personal approach worked for me (macOS Catalina 10.15).

- Required: Homebrew (https://brew.sh)

Installation of MPAS dependencies except PIO

```
git clone https://github.com/pwolfram/homebrew-mpas.git
cd homebrew-mpas
vi install.sh
#    Comment out line 19 (#brew install pwolfram/mpas/pio --build-from-
source)
chmod 700 install.sh
./install.sh
cd ..
```

PIO-1.9.23 installation with some modifications

```
wget https://github.com/NCAR/ParallelIO/archive/pio1_9_23.tar.gz
tar xzvf pio1_9_23.tar.gz
cd ParallelIO-pio1_9_23
cd pio
git clone https://github.com/PARALLELIO/genf90.git bin
git clone https://github.com/CESM-Development/CMake_Fortran_utils.git cmake
vi pio_types.F90
#      Go to line 309
#             Change 'nf_max_var_dims' to '6'  (i.e., PIO_MAX_VAR_DIMS = 6)
#      Go to line 328
#             Change 'nf_max_var_dims' to '6'
cd ..
mkdir build
cd build
# Set shell environmental variables (for BASH)
export FC=mpif90
export CC=mpicc
cmake ../
make
# PIO libs and includes will be installed in ParallelIO-pio1_9_23/build/pio
cd ../../
```

MPAS-O installation

```
git clone https://github.com/MPAS-Dev/MPAS-Model.git
cd MPAS-Model

# Set shell environmental variables (for BASH)
export PIO="PATH_TO_PIO_INSTALL"
# example:  export PIO="/Users/3hk/test/ParallelIO-pio1_9_23/build/pio"
export NETCDF="PATH_TO_NETCDF_INSTALL"
# example:  export NETCDF="/usr/local/Cellar/netcdf/4.6.3_1"
export NETCDFF="PATH_TO_NETCDFF_INSTALL"
# example:  export NETCDFF="/usr/local/Cellar/netcdf/4.6.3_1"
export PNETCDF="PATH_TO_PNETCDF_INSTALL"
# example:  export PNETCDF="/usr/local/Cellar/parallel-netcdf/1.7.0_2"
make gfortran CORE=ocean
# or
make gfortran-clang CORE=ocean
```

Personal Linux Machine

@ Xylar Asay-Davis: This approach worked for me under Ubuntu 18.04

@ Luke Van Roekel (Unlicensed) : Also worked for me under Max OS X 10.14.6

Installation of MPAS dependencies including SCORPIO and the compass conda environment

First, I run the following script in an empty directory that I can delete later:

```
#!/bin/bash
set -e

export PNETCDF_VERSION=1.12.0
export SCORPIO_VERSION=1.1.1

# modify this to fit your system
export CONDA_PATH=/home/xylar/miniconda3

source ${CONDA_PATH}/etc/profile.d/conda.sh

conda create -y -n mpas -c e3sm python=3.8 "compass=0.1.6=mpi_mpich*" \
    netcdf-fortran mpich fortran-compiler cxx-compiler c-compiler m4 git
cmake

conda activate mpas

# modify this
export PREFIX="${CONDA_PATH}/envs/mpas"

export MPICC=mpicc
```

```
export MPICXX=mpicxx
export MPIF77=mpifort
export MPIF90=mpifort
export LDFLAGS="-L${PREFIX}/lib"

rm -rf pnetcdf-${PNETCDF_VERSION}*

wget https://parallel-netcdf.github.io/Release/pnetcdf-${PNETCDF_VERSION}.
tar.gz

tar xvf pnetcdf-${PNETCDF_VERSION}.tar.gz
cd pnetcdf-${PNETCDF_VERSION}

./configure --prefix=${PREFIX}
make
make install

cd ..

rm -rf scorpio*

git clone git@github.com:E3SM-Project/scorpio.git
cd scorpio
git checkout scorpio-v$SCORPIO_VERSION

mkdir build
cd build
CC=mpicc FC=mpifort cmake -DCMAKE_INSTALL_PREFIX=$PREFIX \
    -DPIO_ENABLE_TIMING=OFF -DNetCDF_Fortran_PATH=$PREFIX \
    -DPnetCDF_Fortran_PATH=$PREFIX -DNetCDF_C_PATH=$PREFIX \
    -DPnetCDF_C_PATH=$PREFIX ..

make
make install

cd ../..
```

Setup before compiling/running

Then, when I want to build or run MPAS-Ocean, I source a file containing:

```
conda activate mpas
# Modify this path to point to your mpas conda environment
export PREFIX="/home/xylar/miniconda3/envs/mpas"
# this step might not be needed
export MPAS_EXTERNAL_LIBS="-L${PREFIX}/lib -lnetcdff"
export NETCDF=${PREFIX}
export PNETCDF=${PREFIX}
export PIO=${PREFIX}
# change to one of the other cores as needed
export CORE=ocean
export AUTOCLEAN=true
```

Other Machines

This page replaces the page: MPAS stand alone , which has some old configurations for Titan and Theta. The old ALCC ocean/ice shelf interaction table of contents links to useful instructions for E3SM, analysis, and specific machines, but now may be out of date.

Library stack installation

Grizzly

Installation of PIO follows from the following pre-existing module files:

```
module purge
module load friendly-testing
module load intel/19.0.4 intel-mpi/2019.4 hdf5-parallel/1.8.16 pnetcdf/1.
11.2 netcdf-h5parallel/4.7.3 mkl/2019.0.4
# note the following MPAS-O assumed location variables
export NETCDF=/usr/projects/hpcsoft/toss3/grizzly/netcdf/4.7.3_intel-19.0.4
_intel-mpi-2019.4_hdf5-1.8.16/
export PNETCDF=/usr/projects/hpcsoft/toss3/grizzly/pnetcdf/1.11.2_intel-
19.0.4_intel-mpi-2019.4_hdf5-1.8.16/
```

Note, DO NOT use openmpi/3.1.5 as there is a bug (RMIO https://github.com/MPAS-Dev/MPAS-Model/issues/576).

PIO2 from https://github.com/E3SM-Project/scorpio was used, specifically tag `scorpio-v1.1.0` with the following build command (note use of intel compilers):

```
CC=mpiicc FC=mpiifort cmake -DCMAKE_INSTALL_PREFIX=/usr/projects/climate
/SHARED_CLIMATE/software/grizzly/pio/1.10.1/intel-19.0.4/intel-mpi-2019.4
/netcdf-4.7.3-parallel-netcdf-1.11.2/ -DPIO_ENABLE_TIMING=OFF -
DNetCDF_Fortran_PATH=/usr/projects/hpcsoft/toss3/grizzly/netcdf/4.7.3_intel-
19.0.4_intel-mpi-2019.4_hdf5-1.8.16 -DPnetCDF_Fortran_PATH=/usr/projects
/hpcsoft/toss3/grizzly/netcdf/4.7.3_intel-19.0.4_intel-mpi-2019.4_hdf5-
1.8.16 -DNetCDF_C_PATH=/usr/projects/hpcsoft/toss3/grizzly/netcdf/4.7.3
_intel-19.0.4_intel-mpi-2019.4_hdf5-1.8.16  -DPnetCDF_C_PATH=/usr/projects
/hpcsoft/toss3/grizzly/pnetcdf/1.11.2_intel-19.0.4_intel-mpi-2019.4_hdf5-
1.8.16 ..
```

build with `make` and install with `make install`. Installation here implies

```
export PIO=/usr/projects/climate/SHARED_CLIMATE/software/grizzly/pio/1.10.1/intel-19.0.4/intel-mpi-2019.4/netcdf-4.7.3-parallel-netcdf-1.11.2/
```

as needed for the build.

References

- https://github.com/MPAS-Dev/MPAS-Model/issues/576
- https://github.com/E3SM-Project/scorpio/issues/308

MPAS within E3SM

E3SM has its own documentation, including tutorials and a quick start guide. Here we provide some abbreviated instructions relevant to MPAS components. The simplest way to get started with E3SM is

```
git clone --recursive git@github.com:E3SM-Project/E3SM.git
```

The MPAS source code is a submodule within E3SM, and is contained in the subdirectory `components/mpas-source`. The `--recursive` flag downloads MPAS code into that directory from the MPAS repository. If you cd into that directory, you can edit that code directly and use git commands on that repo locally.

Pre-packaged E3SM simulation: create_test

You can run a pre-packaged test case like this:

```
cd cime/scripts
./create_test PET_Ln9.T62_oQU240.GMPAS-IAF
```

where `./create_test --help` will show you all the options. Here `PET` is a thread test, `Ln9` is for nine steps, `T62` and `oQU240` are the atmosphere and ocean resolution, and `GMPAS-IAF` is a G-case comp set. You can, for example, specify the machine, compiler, queue, walltime with

```
./create_test PET_Ln9.T62_oQU240.GMPAS-IAF \
  --queue debug \
  --walltime 00:30:00 \
  --machine cori-knl \
  --compiler intel
```

etc. if you don't want the defaults. Note that `query_config` and `query_testlists` will list all the options for machines, resolutions, comp sets, etc.

Custom E3SM simulation: create_newcase

For more control over your simulation, use these four commands in sequence: `create_newcase`, `case.setup`, `case.build`, `case.submit`, where the later three reside in the case directory. See the E3SM documentation for more details. The `create_test` command in the previous section rolls these four together.

Here is an example of using these four commands, with bash variables to simplify the arguments.

```
cd $E3SM_REPO/cime/scripts
export E3SM_CASE=run_name
export CASE_ROOT=/global/homes/m/mpeterse/e3sm_cases

./create_newcase \
  -case $CASE_ROOT/$E3SM_CASE \
  -compiler gnu \
  -mach theta \
  -project OceanClimate_2 \
  -compset GMPAS-IAF-ISMF \
  -res T62_oRRS30to10v3wLI
```

where `$E3SM_REPO` is the path to your repository, `$E3SM_CASE` is the name of this particular simulation, and `$CASE_ROOT` is the path to your case directories on this particular machine.

To customize the processor layout, use the `xmlchange` command in the case directory, or edit the `env_mach_pes.xml` directly. For example,

```
cd $CASE_ROOT/$E3SM_CASE
./xmlchange -file env_mach_pes.xml -id NTASKS_OCN -val 8192
./xmlchange -file env_mach_pes.xml -id ROOTPE_OCN -val 8192
./xmlchange -file env_mach_pes.xml -id NTASKS_ICE -val 4096
```

The remaining steps are

```
cd $CASE_ROOT/$E3SM_CASE
./case.setup
./case.build
./case.submit
```

At any time, you can change options for the next submission in `env_run.xml`. For example, to alter the duration, and restart write frequency

```
./xmlchange -file env_run.xml -id STOP_OPTION -val nmonths
./xmlchange -file env_run.xml -id STOP_N -val 2
./xmlchange -file env_run.xml -id REST_OPTION -val nmonths
./xmlchange -file env_run.xml -id REST_N -val 1
```

After the first run, you can set the next submission to continue from the restart, and auto-resubmit, with

```
./xmlchange -file env_run.xml -id CONTINUE_RUN -val TRUE
./xmlchange -file env_run.xml -id RESUBMIT -val 4
```

E3SM Pull Requests involving MPAS code

The workflow for any contributor to alter MPAS code within E3SM is as follows:

1. If you are working on a certain CORE (ocean, seaice, landice) make a branch of `MPAS-Dev/MPAS-Model:CORE/develop` and make your code alterations.
2. Push that branch to your fork.
3. Make a pull request back to the`CORE/develop` branch of the `MPAS-Dev/MPAS-Model` repo.

A CORE developer is then responsible for the following steps:

1. Review the PR.
2. Merge into `MPAS-Dev/MPAS-Model:CORE/develop`
3. When the E3SM repo is available, merge `CORE/develop` into `e3sm/develop`.
4. In an E3SM repo, create a new branch from `E3SM-Project/E3SM:master`, and

```
cd components/mpas-source
git fetch
git reset --hard origin/e3sm/develop
git log --graph --oneline # and you should see your recent merge
cd ..
git add mpas-source
git commit -am "Update mpas-source: add detail"
```

5. Change any corresponding E3SM code. These files are often altered for defaults in
   namelists: `components/mpas-ocean/bld/namelist_files/namelist_defaults_mpaso.xml`
   streams file: `components/mpas-ocean/cime_config/buildnml`
   for the ocean core. See similar files for other cores.
6. Commit remaining changes.
7. Run a series of automated tests. A typical sequence on cori might include:

```
cd cime/scripts
./create_test PET_Ln9.T62_oQU240.GMPAS-IAF.cori-knl_intel -q debug --
walltime 00:30:00
./create_test PEM_Ln9.T62_oQU240.GMPAS-IAF.cori-haswell_gnu -q debug --
walltime 00:30:00
./create_test PET_Ln3.T62_oEC60to30v3wLI.GMPAS-DIB-IAF-ISMF.cori-
haswell_intel -q debug --walltime 00:30:00
./create_test PET_Ln9.ne30_oECv3_ICG.A_WCYCL1850S.cori-knl_gnu -q debug
--walltime 00:30:00 --force-procs 1024
./create_test PEM_Ln9.ne30_oECv3_ICG.A_WCYCL1850S.cori-knl_intel -q
debug --walltime 00:30:00 --force-procs 1024
./create_test SMS_D.T62_oQU120_ais20.MPAS_LISIO_TEST.cori-knl_intel -q
debug --walltime 00:30:00
./create_test SMS_D.T62_oQU120_ais20.MPAS_LISIO_TEST.cori-knl_gnu -q
debug --walltime 00:30:00
```

This does not cover all possible combinations, but provides bit-for-bit threading (PET) and decomposition (PEM) tests on a variety of compsets, resolutions, compilers (intel/gnu) and machines (knl/haswell). The last two are a smoke test with debug on (SMS_D), and `LISIO` runs the MALI core, while the others do not. See instructions on this page, two sections up. To see the results, copy the path from the output line
`Creating test directory /PATH/TO/TEST/DIR`
and look for PASS versus FAIL with
`cat /PATH/TO/TEST/DIR/TestStatus`

8. Push branch to `E3SM-Project/E3SM` repo. Branch names typically are formatted as `username/CORE/topic`, i.e. `mark-petersen/ocean/best_change_ever`.
9. Make a pull request from your new branch to `E3SM-Project/E3SM:master`.
10. An E3SM hub will review and merge. This is Jon Wolfe for MPAS changes. First the PR is merged to `E3SM-Project/E3SM:next` and tested with the E3SM nightly regression suite. If it passes, the PR is merged into `E3SM-Project/E3SM:master`.