Marine UFO Hackathon (May 7-18, 2018)

Location

NCWCP Building, 5830 UNIVERSITY RESEARCH CT, COLLEGE PARK, MD 20740

- Atrium 4552-4553: May 7, 9-11, 14, 17-18
- Atrium 2552-2553: May 8, 15-16

If you do not have a CAC card, please contact Sandra Claar or gvernier.

First day, May 7th

For those of you without a CAC card/access to the building inside the security gates: We will meet at the main entrance of the NCWCP building between 8:50-9:00 am.

Marine-UFO-code-sprint.pdf

Status of UFO repository

- sea-ice fraction: Will be used as a template/example for the development of **linear** UFO's.
- sea-ice thickness: Will be used as a template/example for the development of **nonlinear** UFO's
- CRTM based
- Radio sonde
- The Thermodynamic Equation of Seawater - 2010 is included in the ufo repository

Prerequisite & Homework

In a nutshell

- You will need a laptop with Singularity installed.
- If you decide to run/build outside of singularity, there will be some support for those of you who have access to Theia (contact Rahul Mahajan or Stylianos Flampouris) and Discover (contact Dan Holdaway).
- Access to the JEDI repositories and singularity image: contact Yannick Tremolet and ask for access to the JEDI bundle.
- Compiling issues sorted before the start of the hackathon.
- Basic knowledge of git so we can all work together efficiently.
- We will use ZenHub to manage the hackathon
- Basic understanding of cmake and ctest.
- Observation files for the UFO you want to work on.
- Interpolated model fields needed for the UFO you will be implementing. We will try to provide a prototype JEDI encapsulation of MOM6-SIS2.
- `wiki from Nov 2017 UFO hackathon`

Build and test the soca-bundle: **TO DO before May 7**

Start by making a directory in which to put all of your hackathon code and build.

```
$ mkdir marine-ufo
$ cd marine-ufo
```

Clone the soca-bundle

```
$ git clone -b develop https://github.com/JCSDA/soca-bundle.git
```

What will be in the bundle (after issuing ecbuild in the build process):

- IODA: Interface for Observation Data Access. Currently only contains a nice netcdf wrapper, provided by the GMAO.
- CRTM: Community Radiative Transfer Model.
- OOPS: Object Oriented Prediction System.
- UFO: Unified Forward Operator.
- SOCA: JEDI encapsulation of MOM6-SIS2

On Theia

Building

Start by making a directory in which to put all of your hackathon code and build.

```
$ cd ./soca-bundle/scripts/
$ ./build_socab.sh
```

If the configuration and built went OK, things should lokk like this

```
$ cd ..
$ tree -L 1
.
 build
 CMakeLists.txt
 crtm
 eckit
 fckit
 ioda
 LICENSE
 oops
 README.md
 scripts
 soca
 ufo
```

Testing

git-lfs is not installed on Theia, you will need to manually link to the files in:

/scratch4/NCEPDEV/ocean/scrub/Guillaume.Vernieres/JEDI/soca/data/latest/

Get an interactive session

```
$ qsub -I -l procs=16 -l walltime=08:00:00 -A marine-cpu -N soca-test
$ cd ./build
```

Load the modules

```
$ source ../scripts/jedi_env_theia.sh
$ module list

Currently Loaded Modules:
  1) intel/18.1.163   2) impi/5.1.2.150   3) hdf5/1.8.14   4) netcdf/4.3.0   5) cmake/3.9.0   6) boost/1.64-
intel-18.1.163   7) eigen/3.3.4
```

Start the unit testing

```
$ ctest
```

In Singularity containers

Building

1. Install singularity. See http://singularity.lbl.gov/ or Install Singularity on Mac OS X
2. cd into the marine-ufo directory
3. Download the image

   Checkout the latest singularity image

   ```
   $ singularity pull shub://JCSDA/singularity
   ```

4. Start the singularity container

   Start the singularity container

   ```
   $ singularity shell -e /path/to/singularity/image/JCSDA-singularity-master-latest.simg
   ```

5. Download the pre-built mom6-sis2-fms libraries:
   a. create a directory /1-level-above-soca-bundle/mom6-sis2-fms
   b. Download the two tarballs and mom6 executable, untar in the above directory
   c. Your directory structure should look like this:

   Start the singularity container

   ```
   .
    JCSDA-singularity-master-latest.simg
    mom6-sis2-fms
       include
       libfms.a
       libmom6.a
       marine_include.tar
       marinelibs.tar
       MOM6
    soca-bundle
        CMakeLists.txt
        LICENSE
        README.md
        scripts
   ```

6. cd into scripts and run: ./build_socab_sing.sh
7. The above should have cloned all the repositories and build all the applications. Your directory structure should now look like:

Start the singularity container

```
.
 JCSDA-singularity-master-latest.simg
 mom6-sis2-fms
    include
    libfms.a
    libmom6.a
    marine_include.tar
    marinelibs.tar
    MOM6
 soca-bundle
     build
     CMakeLists.txt
     crtm
     eckit
     fckit
     ioda
     LICENSE
     oops
     README.md
     scripts
     soca
     ufo
```

Testing

cd into ./build and run the ctest

Current status (04/30/2018) of the unit testing under singularity:

Start the unit testing

```
gvernier:test$ ctest
Test project /home/gvernier/Sandboxes/soca/soca-bundle-test/build/soca/test
      Start  1: test_soca_truth
 1/17 Test  #1: test_soca_truth .................   Passed    2.81 sec
      Start  2: test_soca_forecast
 2/17 Test  #2: test_soca_forecast ..............   Passed    3.09 sec
      Start  3: test_soca_makeobs3d
 3/17 Test  #3: test_soca_makeobs3d .............   Passed   71.89 sec
      Start  4: test_soca_geometry
 4/17 Test  #4: test_soca_geometry ..............   Passed    0.31 sec
      Start  5: test_soca_linearmodel
 5/17 Test  #5: test_soca_linearmodel ...........   Passed   51.90 sec
      Start  6: test_soca_state
 6/17 Test  #6: test_soca_state .................***Exception: SegFault  0.70 sec
      Start  7: test_soca_modelaux
 7/17 Test  #7: test_soca_modelaux ..............   Passed    0.19 sec
      Start  8: test_soca_model
 8/17 Test  #8: test_soca_model .................   Passed    1.67 sec
      Start  9: test_soca_increment
 9/17 Test  #9: test_soca_increment .............   Passed    2.23 sec
      Start 10: test_soca_errorcovariance
10/17 Test #10: test_soca_errorcovariance ........***Exception: SegFault  3.02 sec
      Start 11: test_soca_localization
11/17 Test #11: test_soca_localization ...........   Passed    0.27 sec
      Start 12: test_soca_obserrorcov
12/17 Test #12: test_soca_obserrorcov ............   Passed    0.08 sec
      Start 13: test_soca_hofx
13/17 Test #13: test_soca_hofx ...................   Passed   69.12 sec
      Start 14: test_soca_3dvar
14/17 Test #14: test_soca_3dvar ..................   Passed  112.80 sec
      Start 15: test_soca_3dvarfgat
15/17 Test #15: test_soca_3dvarfgat ..............***Failed  206.50 sec
      Start 16: test_soca_4denvar
16/17 Test #16: test_soca_4denvar ................***Failed    0.32 sec
      Start 17: test_soca_dirac
17/17 Test #17: test_soca_dirac ..................***Failed    0.28 sec

71% tests passed, 5 tests failed out of 17

Label Time Summary:
boost         =  60.37 sec (9 tests)
executable    =  60.37 sec (9 tests)
script        = 466.80 sec (8 tests)
soca          = 527.18 sec (17 tests)

Total Test time (real) = 527.20 sec

The following tests FAILED:
          6 - test_soca_state (SEGFAULT)
         10 - test_soca_errorcovariance (SEGFAULT)
         15 - test_soca_3dvarfgat (Failed)
         16 - test_soca_4denvar (Failed)
         17 - test_soca_dirac (Failed)
Errors while running CTest
```

Possible JEDI-Model interface available for the Hackathon

Instead of using pre-interpolated model fields at observation locations, we **might** be able to use the following model/JEDI interfaces:

- MOM6/SIS2 (soca, provided in the bundle)
- FV3GFS
- WW3

Scope of the Hackathon: Implement marine related Unified Forward Operators

Each UFO will require the development of the **nonlinear**, **linearized** and **adjoint** operators.

Tentative to do list for the hackathon (in no particular order):

- Nonlinear mapping of insitu temperature and salinity as a function of temperature (potential or conservative) and salinity(practical/absolute).
- Jacobians of GSW-TEOS10
- TLM/AD of insitu temperature and salinity
- Readers for Argo, CTD, XBT, arrays, moorings …
- I/O software for bufr, netcdf, grib2, ...
- Diurnal interface UFO
- Interface to vertical interpolation functions (forward and adjoint)
- Drifters
- Ocean color
- Altimeters (e.g. L2 ADT or SSHA)
- Along track L2P SST and SSS from GHRSST
- Develop code to test if an observation is inside of the domain
- Add your favorite observation type ...
- DOCUMENT

Tutorial: How to add a UFO

The ufo repository is part of the JEDI bundle and should be under ./code/jedi-bundle/ufo/

Most (all) of your work will be saved under ~/code/jedi-bundle/ufo/src/ufo/marine/

content of marine ufo directory

```
.
 gsw                # Equation of state of sea-water <------ OBSOLETE
 obsop
 seaicefraction     # Template for linear UFO
 seaicethickness    # Template for nonlinear UFO
 stericheight
```

Branching/Merging

... more on that later.

NOW IS A GOOD TIME TO CREATE YOUR OWN BRANCH: Branch out of feature/marine and call it something like feature/marine-myufo.

Step-by-step how to implement a UFO

MOVED TO JEDI-DOCS

This tutorial example is based on a simple, yet, nonlinear observing operator for sea-ice thickness. If your UFO is linear, use the sea-ice fraction UFO as a template.

The sea-ice thickness operator takes as input sea-ice fraction and thickness of each sea-ice categories, interpolated at the observation locations, and outputs the weighted average thickness.

Nonlinear: $h(c_1,...,c_{N_c}, h_1,...,h_{N_c}) = \sum\limits_{n=1}^{n=N_c} c_n h_n$
Linearized around c^{traj}, h^{traj}: $\delta h(\delta c_1,...,\delta c_{N_c}, \delta h_1,...,\delta h_{N_c}) = \sum\limits_{n=1}^{n=N_c} (c_n^{traj} \delta h_n + h_n^{traj} \delta c_n)$

Adjoint $ \begin{align} \delta \hat{c_n} = \delta \hat{c_n} + h_n^{traj} \delta h \\ \delta \hat{h_n} = \delta \hat{h_n} + c_n^{traj} \delta h \end{align} $ $n=1,..., N_c$

Copy & edit the following files and rename them appropriately:

1. Class definition from OOPS base classes:
    a. ./src/ufo/marine/seaicethickness/ObsSeaIceThickness.h
    b. ./src/ufo/marine/seaicethickness/ObsSeaIceThicknessTLAD.h
2. Fortran/C++ binding:
    a. ./src/ufo/marine/seaicethickness/ObsSpace.SeaIceThickness.interface.F90
    b. ./src/ufo/marine/seaicethickness/ObsSeaIceThickness.interface.F90
3. Low level fortran implementation of the classes' methods (where most of your work will be):

    a. Observation operators:
        i. ./src/ufo/marine/seaicethickness/ufo_seaicethick_mod.F90
    b. Observation
        i. ./src/ufo/marine/seaicethickness/ufo_obs_seaicethick_mod.F90
4. Edit ./src/ufo/Fortran.hto finish defining the c-binding
5. Add the obs operators (NL, TLM & AD) to:
    a. ./src/ufo/instantiateLinearObsOpFactory.h
    b. ./src/ufo/instantiateObsOperatorFactory.h
6. Edit ./src/ufo/ObsSpace.cc
7. Implement the unit tests:
    a. Edit ./test/CMakeLists.txt
    b. Add or edit the appropriate .json file in ./test/testinput/