Testing & continuous integration updates in UFS

Alex Richert, Lynker/NOAA EMC

Marshall Ward, NOAA GFDL

Tuesday, September 9, 2025

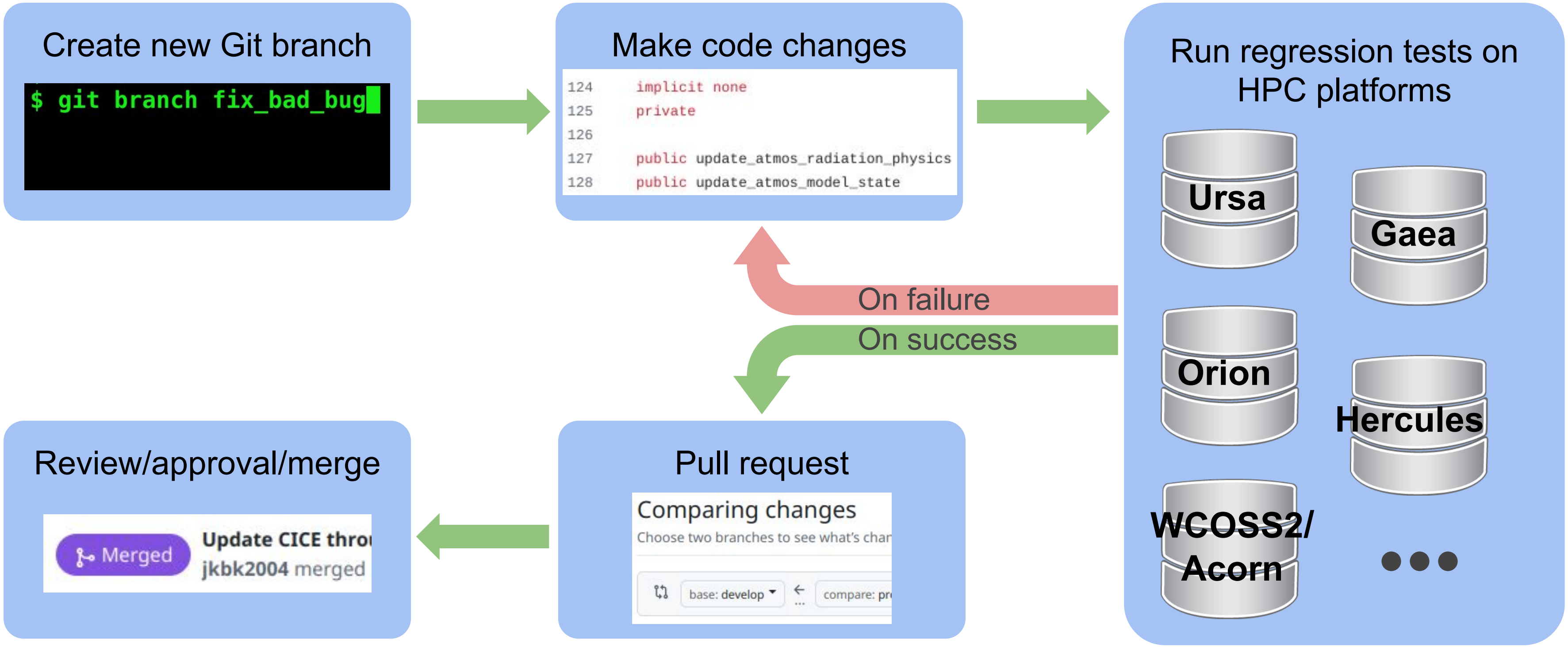
UIFCW25

A UFS Collaboration Powered by EPIC

Overview

- The role of **testing in UFS**
- The value of **unit testing**
- Unit testing successes and challenges in **MOM6**
- Tools of unit testing: **CTest** & **GitHub**-based continuous integration (CI)

UFSWM code changes are end-to-end tested on HPC platforms



Debugging a large code base is a challenge

- UFSWM regression tests take time and effort to run on the tier-1 platforms
- A bug inside a submodule may manifest in non-obvious ways
- The culprit code is not always the point of failure
    - *Example: Numerical inaccuracies that cause RT output checks to fail*

Unit testing emphasizes testing individual code components

- Each unit – typically a function or subroutine – is tested in isolation
- The largest single effort in deploying it is the developing initial infrastructure
- Unit testing **boosts overall code health** and **reduces technical debt**
- Unit testing is *not* a replacement for end-to-end regression testing, good coding practices, and code reviews
- Compared with end-to-end testing, unit testing allows us to **find bugs faster** and **at the source**

Unit testing in the MOM6 ocean model

Models require multiple forms of unit testing

- **Infrastructure code with a clear outcome**

  Was my input forcing opened and read correctly?

- **Numerical equivalence**

  Does my diffusivity match the expected result?

- **Property testing**

  Is my new grid monotonically increasing?

UIFCW25
A UFS Collaboration Powered by EPIC

Infrastructure Testing

Unit test design follows standard principles when the outcome is well-defined:

- `call test_open_param_file()`
- `remove_spaces(" 1 2 3") == "123"`

Numerical tests may have specific well-defined results:

- `cuberoot(1.) == 1.`
- `cuberoot(0.125) == 0.5`

Other tests may utilize a consistency test:

- $|x - \mathrm{cuberoot}(x)^3| < 2\times10^{-15}|x|$

UIFCW25
A UFS Collaboration Powered by EPIC

Tolerance testing: Equation of State

Seawater density *ρ(T,S,p)* is computed using one of several *equations of state*.

MOM6 unit tests rely on traditional **tolerance tests** to verify derivatives of *ρ*:

$$\left| \frac{\partial^2 \rho}{\partial T^2} - \left(\frac{\rho(T + \Delta T) + \rho(T - \Delta T) - 2\rho(T)}{\Delta T^2} \right) \right| < 100\varepsilon$$

UIFCW25
A UFS Collaboration Powered by EPIC

Example: Error in Wright EOS

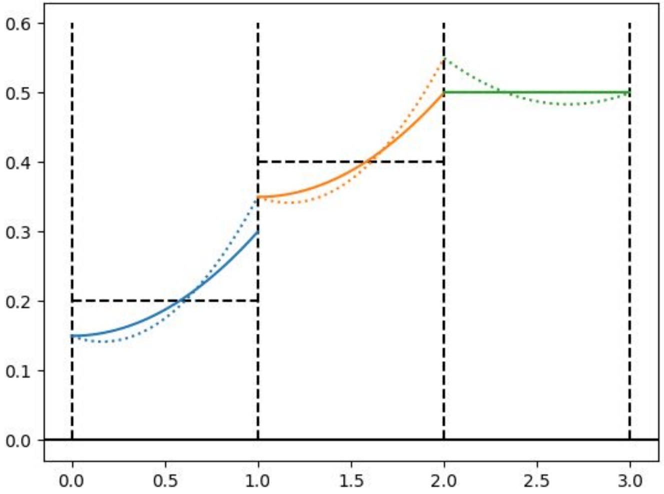
```
The values of WRIGHT drho_dT_dT disagree.
   -1.1402818751119731E-02 and  -7.2448371213340561E-03
   differ by  -4.15798163E-03 ( -4.46E-01), tol=  3.11848682E-04
```

```
z0 = T*(b1 + b5*S + T*(b2 + b3*T))
z1 = (b0 + pressure + b4*S + z0)
z3 = (b1 + b5*S + T*(2.*b2 +2.*b3*T))
z4 = (c0 + c4*S + T*(c1 + c5*S + T*(c2 + c3*T)))
…
drho_dt_dt = (z3*z6 - z1*(2.*c2 + 6.*c3*T + a1*z5) + (2.*b2 + 2.*2.*b3*T)*z4
- z5*z8)/z2_2
  - (2.*(z6 + z9*z5 + a1*z1)*(z3*z4 - z1*z8))/z2_3
```

Should have been a three!

UIFCW25
A UFS Collaboration Powered by EPIC

Property Testing: Remapping



Fields must be remapped to dynamically changing vertical coordinates.

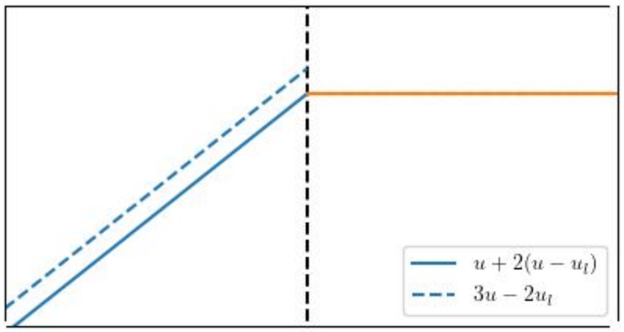
Mappings must be **monotonic**:

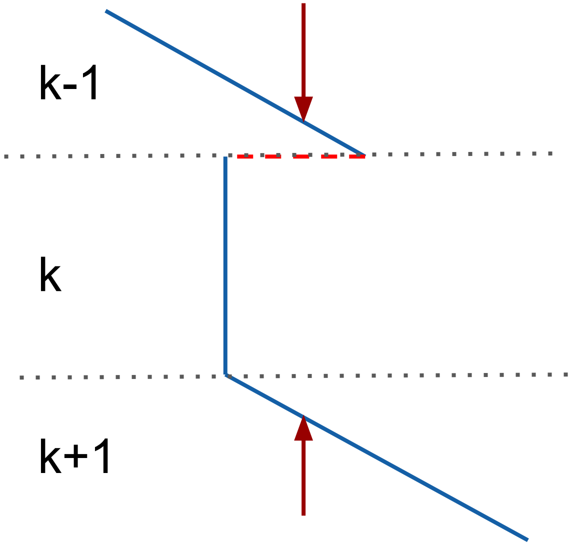
- Interior polynomials cannot have extrema
- Edge values can be *discontinuous* as long as they are also monotonic

Strategy: Apply *random input* to search for property violations.

A UFS Collaboration Powered by EPIC

Example: floating-point remap error





Mathematically equivalent u_r, but $3u-2u_1$ breaks monotonicity!
Second form $u + 2(u - u_l)$ more accurately preserves residuals.

Even a minimal error can disrupt layer search methods, such as when mapping density to a particular layer.

UIFCW25
A UFS Collaboration Powered by EPIC

Dimensional unit testing

Scaling variables in proportion to dimensions provides additional verification.

- Consider any equation in the model:

$$u_{n+1} = u_n + \Delta t \times F$$

- Apply a rescaling that is consistent with their dimensions:
    - $u' \rightarrow (L / T)\, u$
    - $\Delta t' \rightarrow (T)\, \Delta t$
    - $F' \rightarrow (L / T^2)\, F$

- Unit tests must also pass in these rescaled units.

(And if L and T are powers of 2, then results are *bit-reproducible*.)

UIFCW25 A UFS Collaboration Powered by EPIC

Example: Double diffusion bug

A double-diffusive correction to the vertical diffusivity was implemented as

```
Kd_lay(i,j,k-1) = Kd_lay(i,j,k-1) + 0.5**KS_extra(i,K)
```

Should be multiply!

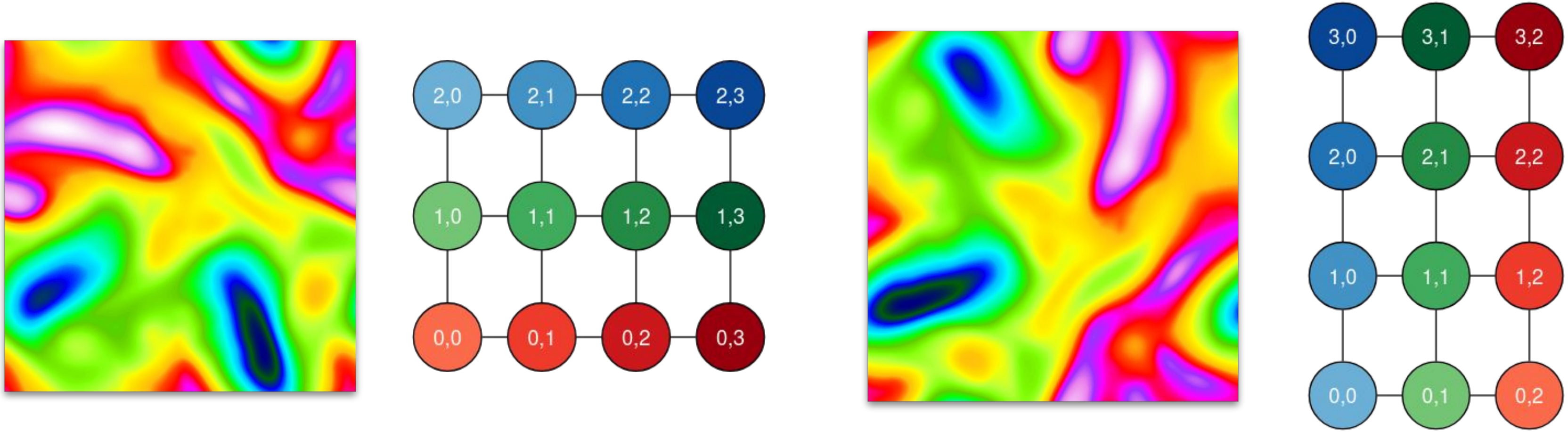
After a dimension rescaling

```
Kd_lay → (L²/T) Kd_Lay
Ks_extra → (L²/T) Ks_extra
```

the error was quickly detected and resolved.

UIFCW25
A UFS Collaboration Powered by EPIC

Rotational Unit Testing



MOM6 can rotate the problem in memory and verify that any directional components are consistent.

UIFCW25
A UFS Collaboration Powered by EPIC

Example: Thickness diffusivity

Rotational testing revealed a sign inconsistency in the thickness diffusivity work:

```
subroutine thickness_diffusivity_full()
  ! …
  Work_u(I,j) = Work_u(I,j) - dWork(I,j)
  ! …
  Work_v(I,j) = Work_v(I,j) + dWork(I,j)
```

Testing cannot determine the sign, but can reveal a physically inconsistent result.

Limitations of MOM6 unit testing

- Testing is **self-consistent**, but not necessarily **truth**
  - Based on fundamental physical principles
  - Platform independent design
  - Numerical testing must consider floating point error

- Strongly modular design, but limited scalability
  - Every module has a custom test framework
  - Difficult for new contributors to provide their own unit tests

The greatest impediments to unit testing in MOM6 are simple, rapid test design and CI integration.

UIFCW25
A UFS Collaboration Powered by EPIC

Determining ground truth outputs for complex code is hard

Partial solution 1:

Organize code into smaller, simpler pieces

Partial solution 2:

Test known edge cases

Partial solution 3:

Verify certain output value **properties** (e.g., enforce "all values in array X are positive")

Partial solution 4:

Use a **golden master** code version to establish baseline results, especially for legacy code (e.g., NCEPLIBS)

CTest provides testing within the CMake framework

CTest **defines** tests as shell commands:

- target (production) executables,
- wrapper scripts, or
- test-specific executables linked to a production library,

where success depends on return code.

CTest tests are simple to **run**:

```
$ cmake ...
$ make
$ make test
```

GitHub Actions allows us to automate unit testing workflows

GitHub Actions workflow:
Defined as a YAML file, typically running shell code

```yaml
on:
  push:
    branches:
    - develop

jobs:
  cdash:
    runs-on: ubuntu-latest

    env:
      FC: gfortran
      CC: gcc
      CDASH_TOKEN: ${{ secrets.CDASH_TOKEN }}

    steps:

    - name: Cache test files
      uses: actions/cache@v4
      with:
        path: testfiles.tgz
        key: NCEPLIBS-ip-test-files-1

    - name: Install dependencies & handle test files
      run: |
        sudo apt-get install libopenblas-dev
```

Developer proposes changes:
Unit test workflows must run and pass

Build caching:

- UFS dependencies take more time to build (45-60+ minutes) than models
- The Spack package manager works neatly with GitHub's caching facilities
- <u>Reusing dependencies means less time waiting on CI runs</u>

For further reading, see github.com/NOAA-EMC/NCEPLIBS/wiki/

UIFCW25
A UFS Collaboration Powered by EPIC

Demo: Adding new, unit-tested code

- Make my code change
- Code my unit test
- Add my unit test in **CTest**
- Create PR, run updated unit test suite through **GitHub Actions**

UIFCW25
A UFS Collaboration Powered by EPIC

$ _

```fortran
!&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
!---------------------------------------------------------------------
!
    subroutine post_finalize(post_gribversion)
!
!     revision history:
!     Jul 2019 Jun Wang: finalize post step
!
      use grib2_module, only : grib_info_finalize
!
      character(*),intent(in) :: post_gribversion
!
      IF(trim(post_gribversion)=='grib2') then
         ! call  grib_info_finalize()
      ENDIF
!
      call de_allocate
!
    end subroutine post_finalize
!
!---------------------------------------------------------------------
!&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
!---------------------------------------------------------------------
!
    subroutine my_new_procedure(my_input)

      character(20), intent(inout) :: my_input

      my_input = trim(my_input) // "_suffix"

    end subroutine my_new_procedure
```

$ _

```fortran
program test_my_new_procedure

    implicit none

    character(20) :: my_input

    my_input = "teststring"

    call my_new_procedure(my_input)

    print*, "Got output: ", trim(my_input)

    if (my_input .ne. "teststring_suffix") then
        stop 1
    endif

end program test_my_new_procedure
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"tests/test_my_new_procedure.F90" [New] 17L, 299B written                    17,33         All
```

$ _

```
find_package(PIO REQUIRED COMPONENTS C Fortran)

# Stage test data
execute_process(COMMAND cmake -E create_symlink
  "${CMAKE_CURRENT_SOURCE_DIR}/data"
  "${CMAKE_CURRENT_BINARY_DIR}/data"
  )

function(add_fv3atm_mpi_test TESTNAME)
  add_executable(${TESTNAME} ${TESTNAME}.F90)
  target_link_libraries(${TESTNAME} PRIVATE ufsatm_fv3 MPI::MPI_Fortran PIO::PIO_C)
  add_test(${TESTNAME} ${MPIEXEC_EXECUTABLE} -n 2 ${CMAKE_CURRENT_BINARY_DIR}/${TESTNAME})
endfunction()

function(add_fv3atm_serial_test TESTNAME)
  add_executable(${TESTNAME} ${TESTNAME}.F90)
  target_link_libraries(${TESTNAME} PRIVATE ufsatm_fv3)
  add_test(${TESTNAME} ${CMAKE_CURRENT_BINARY_DIR}/${TESTNAME})
endfunction()

foreach(testname test_atmos_model test_fv3_cap test_module_wrt_grid_comp)
  add_fv3atm_mpi_test(${testname})
endforeach()

foreach(testname test_post_nems_routines test_my_new_procedure)
  add_fv3atm_serial_test(${testname})
endforeach()
~
~
~
"tests/CMakeLists.txt" 34L, 1061B written                          32,62         Bot
```

Summary

Jobs

build_spack (12, mpich)

build_spack (12, openmpi)

Run details

Usage

Workflow file

build_spack (12, openmpi)

Started 17s ago

Search logs

checkout-fv3atm 15s

```
81  Submodule 'upp' (https://github.com/NOAA-EMC/UPP) registered for path 'upp'
82  Cloning into '/home/runner/work/fv3atm/fv3atm/fv3atm/ccpp/framework'...
83  Cloning into '/home/runner/work/fv3atm/fv3atm/fv3atm/ccpp/physics'...
84  Cloning into '/home/runner/work/fv3atm/fv3atm/fv3atm/fv3/atmos_cubed_sphere'...
85  Cloning into '/home/runner/work/fv3atm/fv3atm/fv3atm/mpas/MPAS-Model'...
93  Cloning into '/home/runner/work/fv3atm/fv3atm/fv3atm/upp'...
94  From https://github.com/NCAR/ccpp-framework
95   * branch            11359cb04a420fc87e4cf0f035f4d1215ab24488 -> FETCH_HEAD
96  Submodule path 'ccpp/framework': checked out '11359cb04a420fc87e4cf0f035f4d1215ab24488'
97  Submodule path 'ccpp/physics': checked out '2c4dbd1fabbea6c332eea2ba97a15bd80a55e630'
98  Submodule 'physics/MP/TEMPO/TEMPO' (https://github.com/NCAR/TEMPO) registered for path 'ccpp/physics/
physics/MP/TEMPO/TEMPO'
99  Submodule 'physics/Radiation/RRTMGP/rte-rrtmgp' (https://github.com/earth-system-radiation/rte-rrtmgp)
registered for path 'ccpp/physics/physics/Radiation/RRTMGP/rte-rrtmgp'
100 Cloning into '/home/runner/work/fv3atm/fv3atm/fv3atm/ccpp/physics/physics/MP/TEMPO/TEMPO'...
101 Cloning into '/home/runner/work/fv3atm/fv3atm/fv3atm/ccpp/physics/physics/Radiation/RRTMGP/rte-rrtmgp'...
102 From https://github.com/NCAR/TEMPO
103  * branch            c62efd27caa26f660edf24232f33f154e608b77a -> FETCH_HEAD
104 Submodule path 'ccpp/physics/physics/MP/TEMPO/TEMPO': checked out 'c62efd27caa26f660edf24232f33f154e608b77a'
```

install-cmake

cache-spack

spack-install

Post checkout-fv3atm

Summary

Jobs

- build_spack (12, mpich)
- build_spack (12, openmpi)
- build_fv3atm (-D32BIT=ON, 12, mpi...
- build_fv3atm (-D32BIT=ON, 12, ope...
- **build_fv3atm (-D32BIT=OFF, 12, m...**
- build_fv3atm (-D32BIT=OFF, 12, ope...

Run details

Usage

Workflow file

build_fv3atm (-D32BIT=OFF, 12, mpich)

succeeded 5 minutes ago in 30m 59s

Search logs

build-fv3atm 30m 23s

```
[100%] Building Fortran object tests/CMakeFiles/test_my_new_procedure.dir/test_my_new_procedure.F90.o
[100%] Built target test_module_wrt_grid_comp
[100%] Linking Fortran executable test_my_new_procedure
[100%] Built target test_my_new_procedure
```

run-tests 0s

```
▸ Run cd $GITHUB_WORKSPACE/build
Test project /home/runner/work/fv3atm/fv3atm/build
    Start 1: test_atmos_model
    Start 2: test_fv3_cap
1/5 Test #1: test_atmos_model .................   Passed    0.08 sec
    Start 3: test_module_wrt_grid_comp
2/5 Test #2: test_fv3_cap .....................   Passed    0.08 sec
    Start 4: test_post_nems_routines
3/5 Test #3: test_module_wrt_grid_comp ........   Passed    0.02 sec
4/5 Test #4: test_post_nems_routines ..........   Passed    0.02 sec
    Start 5: test_my_new_procedure
5/5 Test #5: test_my_new_procedure ............   Passed    0.00 sec
100% tests passed, 0 tests failed out of 5
Total Test time (real) =   0.11 sec
```

get-test-coverage 0s

upload-test-coverage 0s

debug-artifacts 0s

Post checkout-fv3atm 0s

Conclusions

- Unit testing represents an opportunity to accelerate development and improve code health in UFS applications
- NCEPLIBS, fv3atm, and MOM6 are examples of opportunities & challenges for applying unit testing in UFS
- CMake & GitHub are well suited for incorporating and automating unit testing

alexander.richert@noaa.gov
marshall.ward@noaa.gov

UIFCW25
A UFS Collaboration Powered by EPIC