Testing Procedures for the Space Weather Modeling Framework

Gábor Tóth
Center for Space Environment Modeling
The University of Michigan

April 24, 2024

Contents

1 Description of the Testing Philosophy

The Space Weather Modeling Framework (SWMF) is composed of the core of the framework and the various components modeling the physics domains. Since the components are developed independently, it is neither possible nor desirable to enforce a rigorous and comprehensive testing procedure for each component. As a minimum, however, we require that new components are provided with at least one functionality test before they can be integrated into the SWMF.

On the other hand we must ensure that the core of the SWMF and the key components developed at the Center for Space Environment Modeling (CSEM) are well tested and reliable. We also established some base-line testing for the whole framework involving all the components to verify that the SWMF works as expected for a more or less typical space weather simulation run. This test can be run on different platforms, and it serves as the portability testing for the SWMF.

In summary our testing philosophy follows a layered approach:

- The SWMF core: individual unit testing
- Key components: extensive functionality test suite
- New components: at least one functionality test
- The SWMF as a whole: comprehensive portability tests

The core of the SWMF is tested by unit testing, by the functionality test suite and also by the portability tests. The key components are tested by the functionality test suite and the portability tests. New components are tested by their functionality test and the portability tests. Finally all components and couplers are tested by the portability tests.

2 Description of Unit Testing

The core of the SWMF consists of two layers:

- The super structure: CON/Control, CON/Interface, CON/Stubs
- The infra structure: CON/Library, CON/Coupler, share/, util/

The super structure, as the name suggests, can only be tested together with the components and the infra structure. This complexity can be somewhat reduced if the components are replaced with *stubs*. The stub components do not do any computation, they simply advance the simulation time and use up some CPU time. The stub components are implemented in the CON/Stubs directory.

2.1 Testing CON/Control with CON/Stubs

The SWMF can be compiled with the stub components if the

```
INT_VERSION = Stubs
```

is selected in `Makefile.def`. All component versions can be set to 'Empty'. With this choice the core of the SWMF can be tested without running the components. An example parameter file is provided in

```
Param/PARAM.in.test.stubs
```

To run the test, compile the SWMF with

```
./Config.pl -v=Empty
make -j SWMF INT_VERSION=Stubs
```

then create the run directory and run the test:

```
make rundir
cd run
cp Param/PARAM.in.test.stubs PARAM.in
mpiexec -n 2 SWMF.exe
```

This test is intended for developers only, so the output is relatively complicated. Other than printing to the screen, the test creates several files

```
cd run
ls STDOUT/*.log ??restart_*
```

The stub components can also be used to predict the parallel execution time for various layouts and control parameters. An alternative approach is to use the Scripts/Performance.pl script.

2.2 Testing CON/Library

There is a test for the registration of components. The list and layout of registered components is described by the #COMPONENTMAP command in the PARAM.in file. An example file is provided in CON/Library/src. To test the reading of this file and the various functions provided by CON_world, CON_comp_info and CON_comp_param, run

```
cd CON/Library/src
make test NP=4 NTHREAD=2
```

This test is intended for developers only, so the output is relatively complicated. One can change the PARAM.in file or the number of processors to do more extensive testing.

The other modules in this directory (CON_time and CON_physics) are relatively simple and they do not have a unit tester. These modules are tested in the functionality and portability tests.

2.3 Testing CON/Interface

The CON/Interface directory contains the couplers between the components of the SWMF. These couplers cannot be tested by themselves. The interfaces are tested by the portability tests (see section **??**).

2.4 Testing CON/Coupler

The CON/Coupler directory contains the parallel coupling toolkit of the SWMF. This toolkit is used in some of the component couplers. The unit tester for the coupling toolkit is CON_test_global_message_pass. This module has been used in the past to test the SWMF coupling toolkit. To avoid various problems with the compilers on the SGI Altix machines, the unit tester has been removed recently. The coupling toolkit is tested by the portability tests (see section **??**).

2.5 Testing share/Library

This library is used by the SWMF as well as the stand alone components. It is crucial to thoroughly test all methods provided by this library. To run the unit tests

```
cd share/Library/test
make test
```

Although the output looks rather complex, it is mostly caused by the compiler messages and the verbose information provided by the make program. To get a cleaner output, rerun the tests as

```
make -s tests
```

There should be now very limited output reporting the tests of the various modules and methods. Some of the tests may show small differences relative to the expected results due to round off errors.

2.6 Testing util/TIMING and util/NOMPI

The TIMING utility provides a simple and compiler independent utility to measure the CPU time spent on various parts of a Fortran code. The NOMPI utility allows to compile an MPI parallel F90 code with the NOMPI library instead of the MPI library. The resulting executable can run on a single processor. The NOMPI utility is useful for debugging purposes.

The TIMING can be tested with

```
cd util/TIMING/src
make tests
```

An example output is provided in 'tests.log'. To compare with this, rerun the tests like this

```
make -s tests > tmp.log
diff tmp.log tests.log
```

Note that the timings and the order of the output can vary from test run to test run. Two of the four tests involve the NOMPI library, although only a few of the NOMPI methods are used.

3 Description of Functionality Testing

The functionality tests check the functionality of the software in typical configurations. The SWMF uses a hierarchical test suite. The top level Makefile.test contains various SWMF tests that exercise various subsets of the models running together. Each model inside the SWMF has (or should have) functionality tests that check the model in stand-alone mode. In addition, there are tests for the shared library and some of the utilities. All tests can be executed with

```
make -j test NP=4 NTHREAD=2 >& test_swmf.log
```

where NP sets the number of cores and NTHREAD sets the number OpenMP threads to be used. The tests are typically run from 1 to 8 cores and 1 to 2 threads.

The results of the tests are written into .diff files. An empty .diff file means that the test passed. The results can be collected with

```
make test_check
```

Use

```
make test_help
```

to see a complete list of functionality tests for the SWMF. For individual models or libraries, go into the model directory:

```
cd GM/BATSRUS
make test_help
cd ../../PW/PWOM
make help
cd  ../../share/Library/test
make help
```

The functionality tests are executed on various machines with various compilers and different number of processurs every night. The results are collected to a website, currently available at

```
http://herot.engin.umich.edu/~gtoth/
```