pFUnit
(the slides are available under “Presentation Materials” in the above URL)
Date: April 10, 2019
Presented by: Thomas Clune (NASA)

Q. Will there be a discussion on building pfUnit?

A. Yes

Q. Where do we find the main pFUnit documentation? For example of the asserts annotations.

A. Not documented but there are examples of most if not all features. Some documentation of the old pFUnit.

There is some existing documentation for older versions of pFUnit. Much of it remains relevant, and some of it is still correct. But there are probably just enough incorrect items to be problematic for new users.

Previously I had been using Doxygen to document pFUnit, but am now switching to GitHub wiki pages. Over the next few weeks I expect to produce a fair bit of very simple documentation to get the ball rolling. We can probably get other experienced users to contribute, and if there are specific concerns I can prioritize addressing them in the Wiki first. Open an issue ...

Q. How much validation and verification can one do with short tests? Do you have ways to test errors in parallel implementation/execution?

A. (I at first read “short” in the question to mean short in duration. I now am wondering if it was meant as in a smallish number of lines of code?)

Almost any (sufficiently) small layer of code can be well-tested with a modest number of tests, each of which run in milliseconds. A full test suite that covers a large application could therefore easily take over a minute, but probably well under an hour. (10^6 tests at 1 ms each is ~20 minutes).

Unfortunately, many procedures are large (1000's of lines), and are very difficult to test at all, much less quickly. Other things can slow down tests as well. E.g. file-system access where a file must be created and destroyed for a test can result in slow tests. OTOH, there are generally very few of those tests compared to the other kinds.

I am not sure what to say about validation. In my environment, validation means comparing results between long, high-resolution numerical simulations against large data sets. If the comparison can be expressed as a set of assertions, one could certainly do some of this with a testing framework, but it would not be short. I would not be advocating pFUnit for such things in general.

Testing errors in parallel execution, is sort of the raison d'etre for pFUnit. Note: with regard to short (duration) tests, in the early days, each MPI test ran its own MPI_Init() and MPI_Finalize(), but the overhead for launching MPI even on a laptop was often O(1 sec). I compromised on this issue for the sake of running many tests quickly.

Q. Do you intend to print variable values in context on failures the way pytest does? Many test suites tend not to do that, but it's a really nice feature of pytest and obviates problems just be reading the output without needing to visit with a debugger.

I need to understand more about what pytest does and/or what is desired by the questioner. In general, python has much deeper capabilities for introspection, so I will not be surprised if the desired capability is very difficult/impossible to replicate in pFUnit. But if it useful and only modestly difficult, I'd consider it a fun chanllenge to incorporate.

A. The current error messages are largely following the choices made by jUnit. Not exact of course, but for simpler cases it is nearly identical. (If nothing else it keeps me from going into an infinite loop debating relatively equivalent choices.)

Note that pFUnit is extensible. Some changes in formatting could be accomplished by producing a new subclass of ResultPrinter. (Would also need a command line option to activate it.) With a bit more work, we could allow someone to swap in a different Assert library. But future development in this regard really should focus on hamcrest. Users can then just subclass the existing Matcher and have it format messages in their preferred manner.

Note that users should not rely on precise formatting details for pFUnit generated messages. (pFUnit internal tests do so, of course.) These don't change often, but I'd hate to have a constraint like that get in the way of improvements.

Q. Can you drop into a debugger on test failure?

A. Early group had integrated pFUnit into photran (Eclipse plugin).   This could be added as an optional feature, but I don't know how to go about it.   I'm happy to accept contributions, but may rely on the contributor to maintain such capabilities.

Q. Is there a "add_pfunit_ctest" that works with MPI? Or can you use it with MPI? If so, how? Since ctest needs to know that the test has to be executed with mpirun/mpiexec.

A. Yes, it's in the examples. The "add_pfunit_ctest" has additional options (NPES) so that it recognizes that it has to run with mpirun.   This is less tested so may need some tweaks to work in other environments.

Q.   Can I use asserts with derived types with type-bound compare procedures?

A.   If you define your own == operator, you can do @assertTrue(a == b). Not as useful. You want @assertEqual(a, b) and call some compare function on top of that:

```
if (.not. (a == b)) then
    call throw(a%compare_message(b),  source_location)
Endif
```

There would need to be a convention and then a simple extension to the python preprocessor to use it.   There is already a user contribution in the preprocessor along these lines, but the generated message just names the arguments.   You see something like <a> not equal to <b>. (Which might be sufficient in many cases.) If you want something more in this direction, please open an issue! Need a good diagnostic test when not equal.

Q. Do you require a minimum cmake version? CMake 3.13.4 fails for me with:

```
omsai@xm2:~/src/build-pfunit$ cmake ../pFUnit
...
-- Configuring done
CMake Error at include/add_pfunit_ctest.cmake:51
(add_executable):
  Cannot find source file:
     Test_ExceptionList.F90

  Tried extensions .c .C .c++ .cc .cpp .cxx .cu .m .M .mm
.h .hh .h++ .hm
  .hpp .hxx .in .txx
Call Stack (most recent call first):
  tests/funit-core/CMakeLists.txt:94 (add_pfunit_ctest)


CMake Error at include/add_pfunit_ctest.cmake:51
(add_executable):
  No SOURCES given to target: new_tests.x
Call Stack (most recent call first):
  tests/funit-core/CMakeLists.txt:94 (add_pfunit_ctest)


-- Build files have been written to:
/home/omsai/src/build-pfunit

omsai@xm2:~/src/build-pfunit$ git -C ../pFUnit status
On branch v4.0_beta
Untracked files:
  (use "git add <file>..." to include in what will be
committed)

    source/

nothing added to commit but untracked files present (use
"git add" to track)
omsai@xm2:~/src/build-pfunit$
```

A. I need to learn more about the context. I am using the same version of CMake in my development environment. I have tested with NAG 6.2 in my environment and the Travis tests use gfortran (8.2 or 8.3). Please let me know the OS and compiler version and I'll attempt to replicate. I very much want this to work for everyone and apologize that you experienced the worst possible frustration when trying a new tool.

Q. What is "hamcrest"? What does it mean? What is the origin?

A. See http://hamcrest.org/

Q. fHamcrest looks like a fork in the development. As a new user, would you recommend adopting the fHamcrest syntax

A. Unfortunately, not quite yet.   What is there works, and I had hoped to fill in more before this talk.   Nothing like a deadline to make things happen.   fHamcrest should be very robust already, but it is missing overloads for ranks and kinds which would make it limited for real codes. (But fine for demonstrations.)   There also is a tiny bit of work in the preprocessor to make it recognize @assert_that (and @assertThat).

I hope/expect to finish all of that when I formally release 4.0.   (Probably am pushing the definition of “beta” a bit by committing to adding functionality.   Apologies to the purists.)

So come back in a month, and I expect my answer to be YES - please use fHamcrest and avoid the old @assertEqual.

Q. Can someone please put in a plug for "starring" the pFunit repository on GitHub? I am a Homebrew maintainer and would like to add this to our default formula repository, but this requires sufficient community interest & popularity.

A. I’ve starred it! (Not the presenter)
B. Thanks for the tip. I’ll mention it to other users as well.   (The presenter)

Q. Does the MPI implementation matter? Does it work with all/most implementations?

A. pFUnit actually uses very little MPI. Just some calls to create communicators and to gather messages and such.   The harder part is the variation in how one launches MPI. Through CMake there is now hope that a canonical launch will work everywhere, but colleagues have told me this is not quite true.   Mostly this means we may need to add logic to the add_pfunit_ctest() macro to detect an unusual flavor of MPI.   There is already a check for OpenMPI so that it adds the --oversubscribe flag to allow me to run tests on my laptop.
B. I’ve not tested 4.0_beta with a variety of MPI’s just yet, but that layer has hardly changed.   Certainly I’ve used 3.x with OpenMPI, MVAPICH, IntelMPI, and I think even SGI’s (now HP’s) MPT.   Others have made it work with Cray.

Q. See man 3 abort for 1 way to provide a basic debug support, if core file size limit in shell is non-zero (ulimit -c) then you will get core dump when calling abort(), would be easy to implement as a wrapped c function called from fortran

A. Please open an issue in the repo. And pretty-please provide a pull-request. :-)

Q. Does pFUnit currently support mocking?

A. No. You will see a bit of functionality lurking from an early attempt. I'd love to spend more time on this, but that's not what my day job is about. And there are things that Python can do "magically" on this front that will be very difficult in Fortran. Single-inheritance can also be a source of limitation on this topic.)

An intermediate step will probably be requiring the users to provide a YAML file describing the interface of the layer to be mocked. From that I can generate a Fortran mock that ties into a generic layer within pFUnit. A more advanced solution would be to use a true Fortran parser to derive interface information directly from source code. Not my area of expertise, so this would require an external contribution. (hint, hint)

Even if mocks cannot be conveniently automated for user code, it is probably still worth a manual process for mocking widely-used libraries like MPI. Even mocking a very small subset of MPI could be very useful.

Q. Is there Mock MPI in pFUnit?

A. No. I will soon post an example on the demos page showing how I did this in a limited context. I manually mocked just a few MPI interfaces and exercised them in serial tests. These are illustrative, but are a poor imitation of what is desired. Usually mock frameworks allow the test to express a sequence of "expectations" and then those are verified.

Q. Does pFUnit has a contributor agreement ? Do you accept pull requests?

A. I definitely accept pull requests. I was rather sloppy back when pFUnit was on sourceforge, and would often miss messages for extended periods of time. But I'm now reasonably adept with GitHub, and encourage pull-requests.

Regarding a "contributor agreement", that sounds very legalistic to me. Currently pFUnit is available under the NASA Open Source Agreement (NOSA). With a small effort, I can get the NASA lawyers to provide an Apache 2.0 license, which is more familiar to most.   Contributors must decide for themselves whether their contributions meet the requirements.s