Software Testing

Rinku Gupta
Argonne National Laboratory

Better Scientific Software Tutorial, SC20, November 2020

License, Citation and Acknowledgements

License and Citation

- This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0).
- **The requested citation the overall tutorial is: David E. Bernholdt, Anshu Dubey, Patricia A. Grubel, Rinku K. Gupta, Better Scientific Software tutorial, in SC ‘20: International Conference for High Performance Computing, Networking, Storage and Analysis, online, 2020. DOI: 10.6084/m9.figshare.12994376**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software tutorial…

Acknowledgements

- Additional contributors include: Mike Heroux, Alicia Klinvex, Mark Miller, Jared O’Neal, Katherine Riley, David Rogers, Deborah Stevens, James Willenbring
- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.
- This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- This work was performed in part at the Lawrence Livermore National Laboratory, which is managed by Lawrence Livermore National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-07NA27344.
- This work was performed in part at the Los Alamos National Laboratory, which is managed by Triad National Security, LLC for the U.S. Department of Energy under Contract No.89233218CNA000001
- This work was performed in part at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525.

Verification

- Code verification uses tests
  - It is much more than a collection of tests
- It is the holistic process through which you ensure that
  - Your implementation shows expected behavior,
  - Your implementation is consistent with your model,
  - Science you are trying to do with the code can be done.

How do verification and validation differ?

- Verification confirms that you have implemented what you meant to
  - Your method does what you wanted it to do
- Validation tells you were right in implementing what you meant to
  - What you wanted your method to do is valid
  - Your model correctly captures the phenomenon you are trying to understand

Stages and types of verification

- During initial code development
  - Accuracy and stability
  - Matching the algorithm to the model
  - Interoperability of algorithms
- In later stages
  - While adding new major capabilities or modifying existing capabilities
  - Ongoing maintenance
  - Preparing for production

Components of Verification

- Testing at various granularity
  - Individual components
  - Interoperability of components
  - Convergence, stability and accuracy
- Validation of individual components
  - Building diagnostics (e.g. ensure conservation of physical quantities)
- Testing practices
  - Error bars
    - Necessary for differentiating between drift and round-off
- Ensuring code and interoperability coverage

Why not always use the most stringent testing?

- Effort spent in devising running and maintaining test suite is a tax on team resources
- When the tax is too high…
  - Team cannot meet code-use objectives
- When is the tax is too low…
  - Necessary oversight not provided
  - Defects in code sneak through
- Evaluate project needs
  - Objectives: expected use of the code
  - Team: size and degree of heterogeneity
  - Lifecycle stage: new or production or refactoring
  - Lifetime: one off or ongoing production
  - Complexity: modules and their interactions

Balance is critical

Good Testing Practices

- Verify Code coverage
- Must have consistent policy on dealing with failed tests
  - Issue tracking
    - How quickly does it need to be fixed?
    - Who is responsible for fixing it?
- Someone should be watching the test suite
- When refactoring or adding new features, run a regression suite before check in
  - Add new regression tests or modify existing ones for the new features
- Code review before releasing test suite is useful
  - Another person may spot issues you didn’t
  - Incredibly cost-effective

How do we determine what other tests are needed?

Code coverage tools

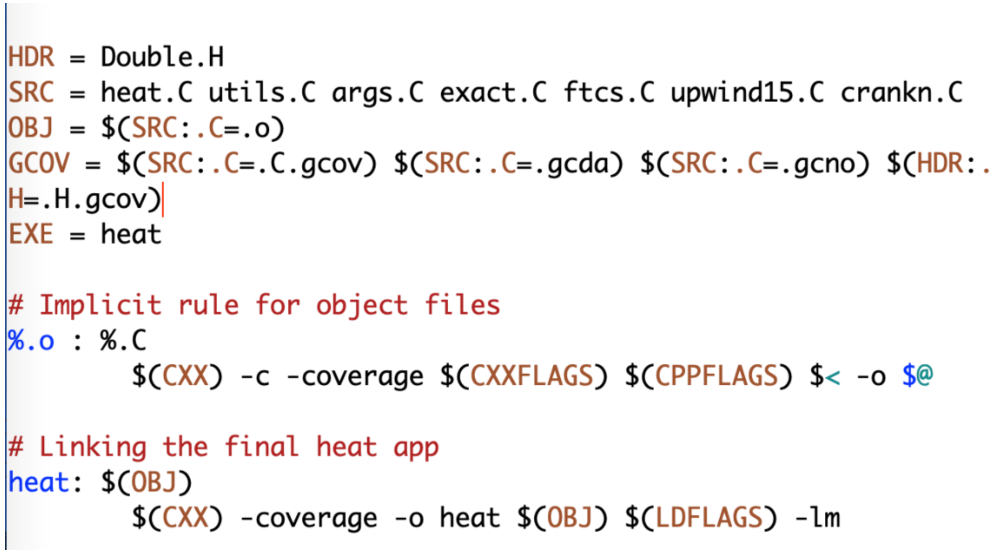
- Expose parts of the code that aren’t being tested
  - gcov - standard utility with the GNU compiler collection suite (we will use it in the next few slides)
  - Compile/link with –coverage & turn off optimization
  - counts the number of times each statement is executed
- gcov also works for C and Fortran
  - Other tools exist for other languages
  - JCov for Java
  - Coverage.py for python
  - Devel::Cover for perl
  - profile for MATLAB
- Lcov
  - a graphical front-end for gcov
  - available at http://ltp.sourceforge.net/coverage/lcov.php
  - Codecov.io in CI module
- Hosted servers (e.g. coveralls, codecov)
- graphical visualization of results
- push results to server through continuous integration server

Interoperability coverage Example Later

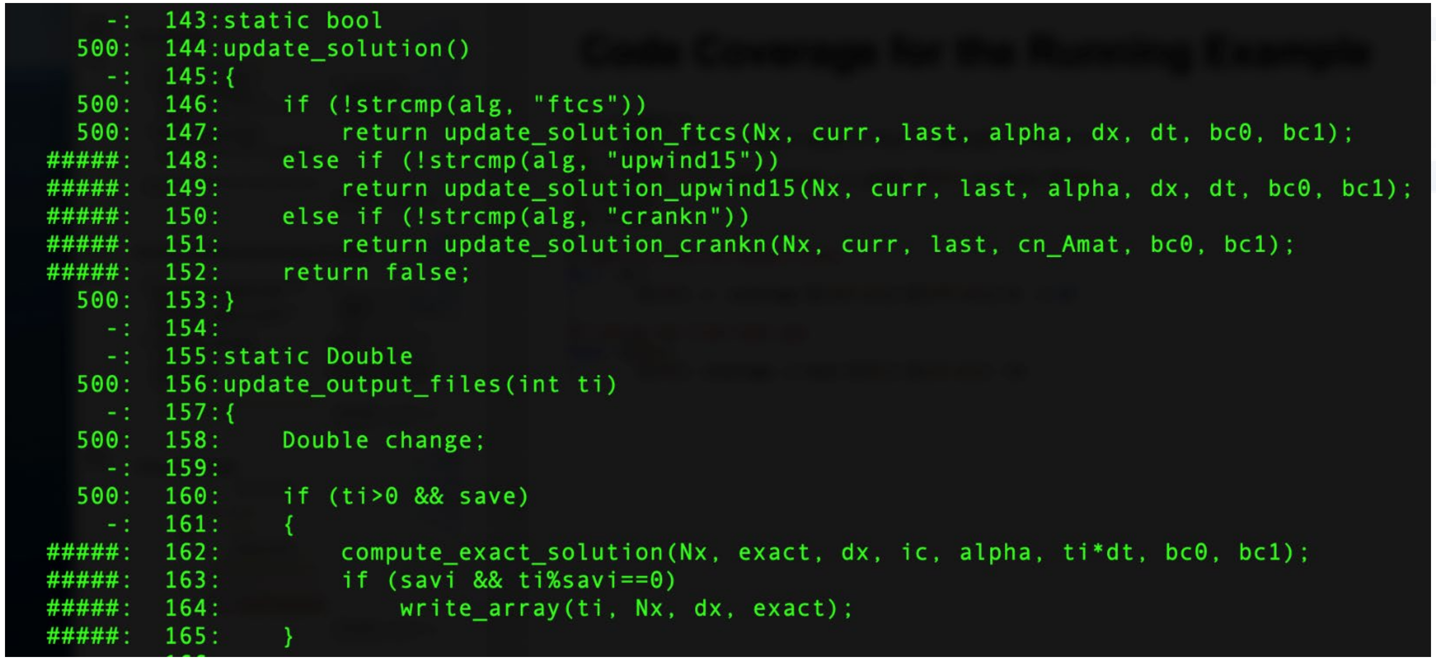
Checking coverage Example

- Example of heat equation
  - Add -coverage as shown below to Makefile
  - Run ./heat runame=“ftcs_results”
  - Run gcov heat.C
  - Examine heat.C.gcov

- A dash indicates non-executable line
- A number indicated the times the line was called
- ##### indicates line wasn’t exercised



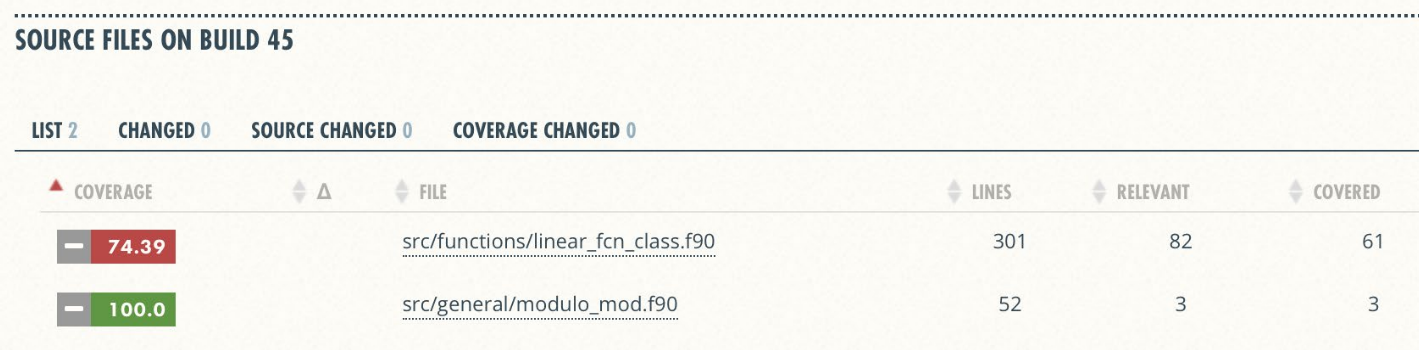
```
HDR = Double.H
SRC = heat.C utils.C args.C exact.C ftcs.C upwind15.C crankn.C
OBJ = $(SRC:.C=.o)
GCOV = $(SRC:.C=.C.gcov) $(SRC:.C=.gcda) $(SRC:.C=.gcno) $(HDR:.
H=.H.gcov)
EXE = heat

# Implicit rule for object files
%.o : %.C
	$(CXX) -c -coverage $(CXXFLAGS) $(CPPFLAGS) $< -o $@

# Linking the final heat app
heat: $(OBJ)
	$(CXX) -coverage -o heat $(OBJ) $(LDFLAGS) -lm
```



```
    -:  143:static bool
  500:  144:update_solution()
    -:  145:{
  500:  146:    if (!strcmp(alg, "ftcs"))
  500:  147:        return update_solution_ftcs(Nx, curr, last, alpha, dx, dt, bc0, bc1);
#####:  148:    else if (!strcmp(alg, "upwind15"))
#####:  149:        return update_solution_upwind15(Nx, curr, last, alpha, dx, dt, bc0, bc1);
#####:  150:    else if (!strcmp(alg, "crankn"))
#####:  151:        return update_solution_crankn(Nx, curr, last, cn_Amat, bc0, bc1);
#####:  152:    return false;
  500:  153:}
    -:  154:
    -:  155:static Double
  500:  156:update_output_files(int ti)
    -:  157:{
  500:  158:    Double change;
    -:  159:
  500:  160:    if (ti>0 && save)
    -:  161:    {
#####:  162:        compute_exact_solution(Nx, exact, dx, ic, alpha, ti*dt, bc0, bc1);
#####:  163:        if (savi && ti%savi==0)
#####:  164:            write_array(ti, Nx, dx, exact);
#####:  165:    }
```

Graphical View of Gcov Output and Tutorials for Code Coverage

Overall Analysis

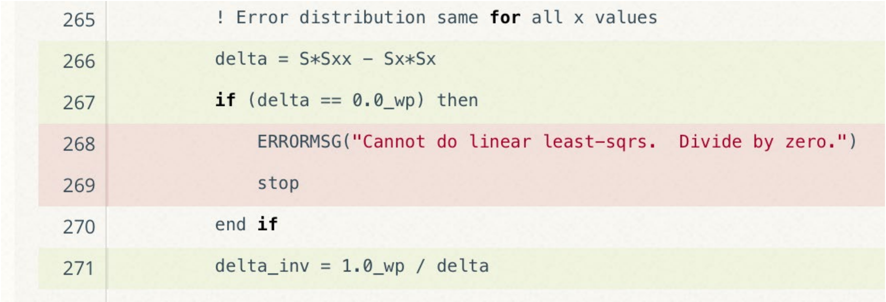


SOURCE FILES ON BUILD 45

LIST 2 CHANGED 0 SOURCE CHANGED 0 COVERAGE CHANGED 0

COVERAGE	Δ	FILE	LINES	RELEVANT	COVERED
− 74.39		src/functions/linear_fcn_class.f90	301	82	61
− 100.0		src/general/modulo_mod.f90	52	3	3

Detailed Analysis



```
! Error distribution same for all x values
delta = S*Sxx - Sx*Sx
if (delta == 0.0_wp) then
    ERRORMSG("Cannot do linear least-sqrs.  Divide by zero.")
    stop
end if
delta_inv = 1.0_wp / delta
```

Online tutorial - https://github.com/amklinv/morpheus

Other example - https://github.com/jrdoneal/infrastructure

How to build your test suite ?

- Two purposes
  - Regression testing
    - May be long running
    - Provide comprehensive coverage
  - Continuous integration
    - Quick diagnosis of error
- A mix of different granularities works well
  - Unit tests for isolating component or sub-component level faults
  - Integration tests with simple to complex configuration and system level
  - Restart tests
- Rules of thumb
  - Simple
  - Enable quick pin-pointing

Useful resources https://ideas-productivity.org/resources/howtos/

Test Development For a New Code

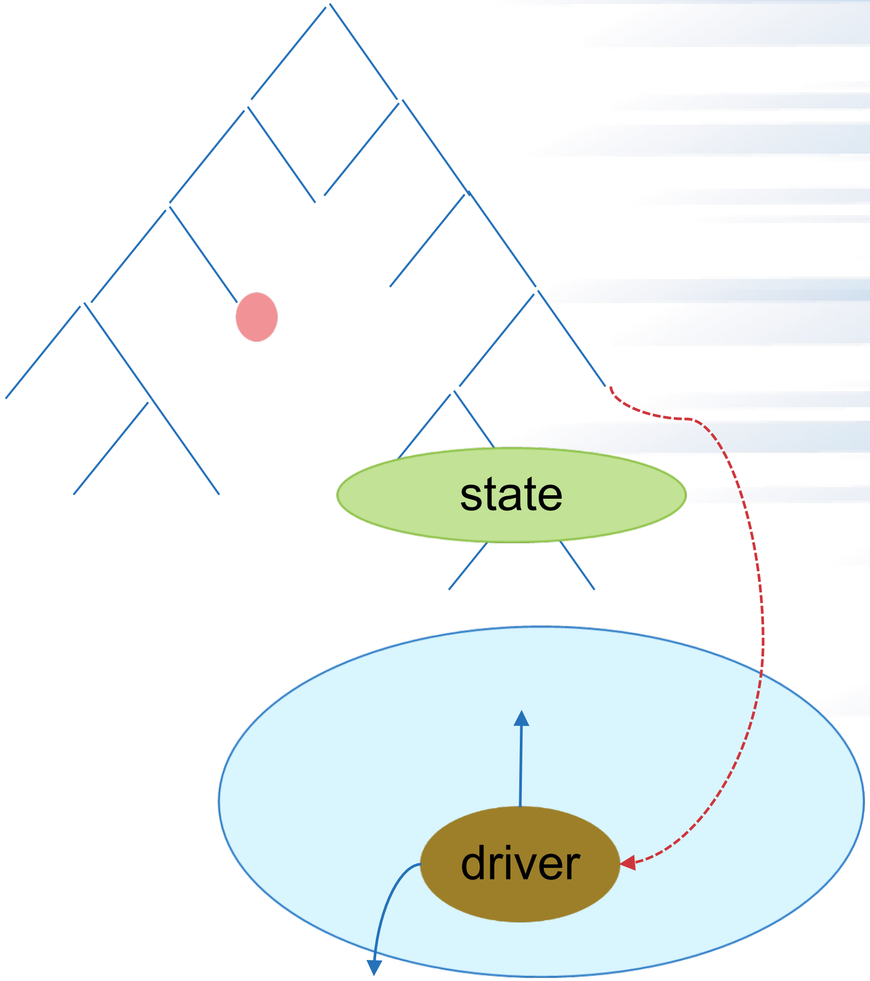
- Development of tests and diagnostics goes hand-in-hand with code development
  - Non-trivial to devise good tests, but extremely important
  - Compare against simpler analytical or semi-analytical solutions
  - Build granularity into testing
  - Use scaffolding ideas to build confidence
  - Always inject errors to verify that the test is working

Detailed example in the next presentation

Test Development For a Legacy Code

There may not be existing tests

- Isolate a small area of the code
- Dump a useful state snapshot
- Build a test driver
  - Start with only the files in the area
  - Link in dependencies
    - Copy if any customizations needed
- Read in the state snapshot
- Restart from the saved state
- Verify correctness
  - Always inject errors to verify that the test is working


state
driver