Software Testing and Verification

Anshu Dubey (she/her)
Argonne National Laboratory

Software Productivity and Sustainability track @ Argonne Training Program on Extreme-Scale Computing summer school

Contributors: Anshu Dubey (ANL), Patricia Grubel (LANL), Rinku Gupta (ANL), Alicia Klinvex (SNL), Mark C. Miller (LLNL), Jared O'Neal (ANL), David M. Rogers (ORNL), Gregory R. Watson (ORNL)

License, Citation and Acknowledgements

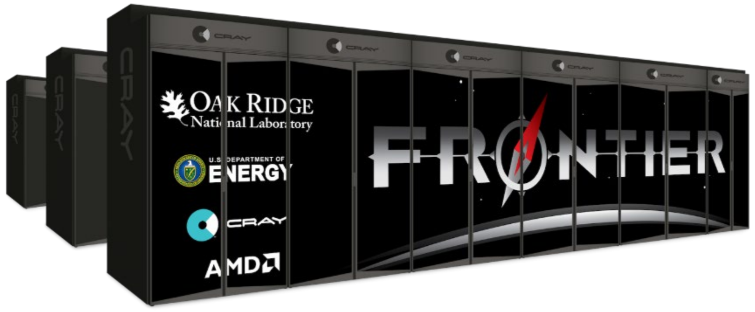
License and Citation

- **The requested citation the overall tutorial is: David E. Bernholdt, Anshu Dubey, Todd Gamblin, Jared O'Neal, and Boyana R. Norris, Software Productivity and Sustainability track, in Argonne Training Program on Extreme-Scale Computing, St. Charles, Illinois, 2022. DOI: 10.6084/m9.figshare.20416215.**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software tutorial, ISC, 2022 …

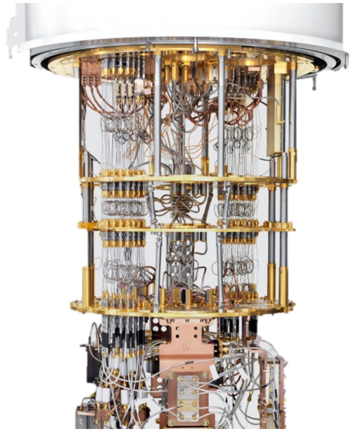
Acknowledgements

- 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 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 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 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.
- This work was performed in part at University of Oregon through a subcontract with Argonne National Laboratory.

Motivation – Testing Practices



- Supercomputer Cycles are Scarce Resources
  - Goal = capture QA details during science runs
- Many people need to have confidence in your results:
  - You
  - Your project lead or boss
  - Your sponsor
  - Your reviewers or referees
  - Your readers
- Testing helps build credibility *without* repeating runs.



IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

What about Verification and Validation?

- Scientific computing and software engineering use different definitions

	Scientific computing	Software engineering
Verification	Confirms the mathematical accuracy and stability of a numerical solution in addition to specifications.	Confirms that the software conforms to its specifications (i.e. requirements.)
Validation	Confirms the physical accuracy of a given model by comparing against experimental data.	Confirms that the software actually meets the customer's needs.

- Validation in scientific computing requires a comparison to the experimental data, whereas in software engineering it is based on customer needs
- Also, for a real problem, there is typically no way to check for correct output given some inputs. Validation is still required however, so an indirect method must be used.

Testing within the software development lifecycle

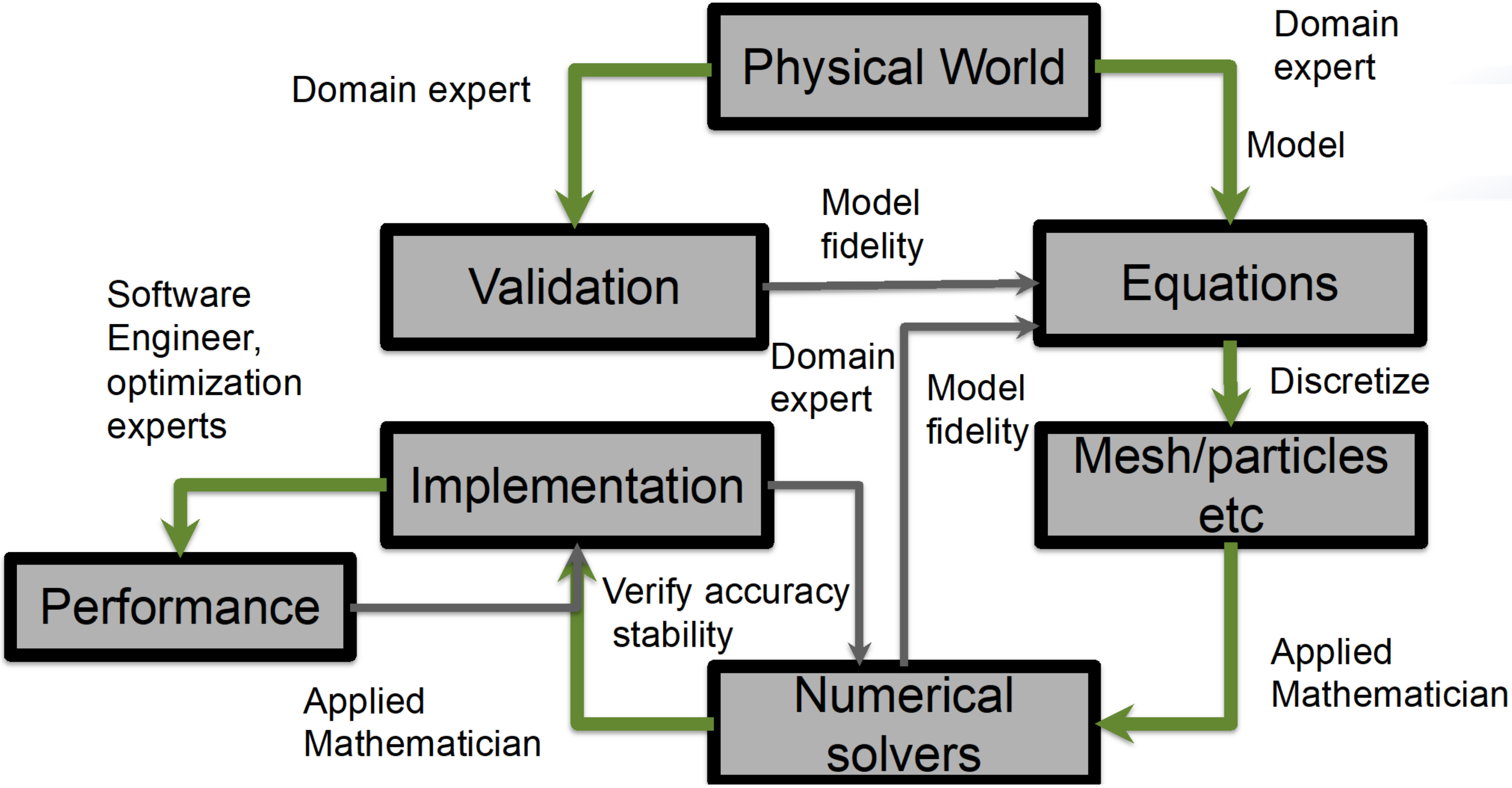
- When should functional tests be provided?
- Ideally before the code is written
  - Also known as test driven development (TDD)
  - Tests then become the specification for the program
- This approach also ensures that thought is given to what it means for the program to be correct, rather than just what the program should do
- Requires:
  - Care in writing tests
  - Frequent running of tests (see our Continuous Integration module)
  - Wide adoption by development team

Steps for test driven development

- Write a single test[1] describing an aspect of the program
- Run the test, which should fail because the feature does not exist
- Write just enough code to make the test pass
- Refactor the code
- Repeat, creating new tests as new functionality is added

[1]In numerical methods there are times when a single test may not suffice

Testing within the software development lifecycle



Developing Tests

We verify correct behavior

How ?

Developing Tests

We verify correct behavior

How ?

We think of ways in which we can tell whether the code is doing what it is supposed to do

Developing Tests

We verify correct behavior

How ?

Compare against a known analytical solution

Compare against a manufactured solution

We think of ways in which we can tell whether the code is doing what it is supposed to do

Developing Tests

We verify correct behavior

How ?

Compare against a known analytical solution

Compare against a manufactured solution

We think of ways in which we can tell whether the code is doing what it is supposed to do

Right there are all the ingredients for building tests

All that is needed is automating directly or indirectly

Developing Tests

We verify correct behavior

How ?

Compare against a known analytical solution

Compare against a manufactured solution

We think of ways in which we can tell whether the code is doing what it is supposed to do

Visualize and inspect output

Right there are all the ingredients for building tests

All that is needed is automating directly or indirectly

Developing Tests

We verify correct behavior

How ?

Compare against a known analytical solution

Compare against a manufactured solution

We think of ways in which we can tell whether the code is doing what it is supposed to do

Visualize and inspect output

Develop diagnostics (indirect ways of verification)

Right there are all the ingredients for building tests

All that is needed is automating directly or indirectly

Developing Tests

We verify correct behavior

How ?

Compare against a known analytical solution

Compare against a manufactured solution

We think of ways in which we can tell whether the code is doing what it is supposed to do

Visualize and inspect output

Develop diagnostics (indirect ways of verification)

Right there are all the ingredients for building tests

All that is needed is automating directly or indirectly

Including these through automation is equally important

Comparison utility
Conserved quantities
Error bars
Statistical analysis

Test Development For a New Code

For every new functionality being added, think about its verification

Simple functions: relation between input and output => unit test

Other functions: build scaffolding

If it has limited dependencies, manufacturing input for known output will give you a self test

If manufacturing input is too difficult, again apply scaffolding

Test Development For a New Code

For every new functionality being added, think about its verification

Simple functions: relation between input and output => unit test

If it has limited dependencies, manufacturing input for known output will give you a self test

Example – Shock Hydrodynamics with Adaptive Mesh Refinement

Components needed

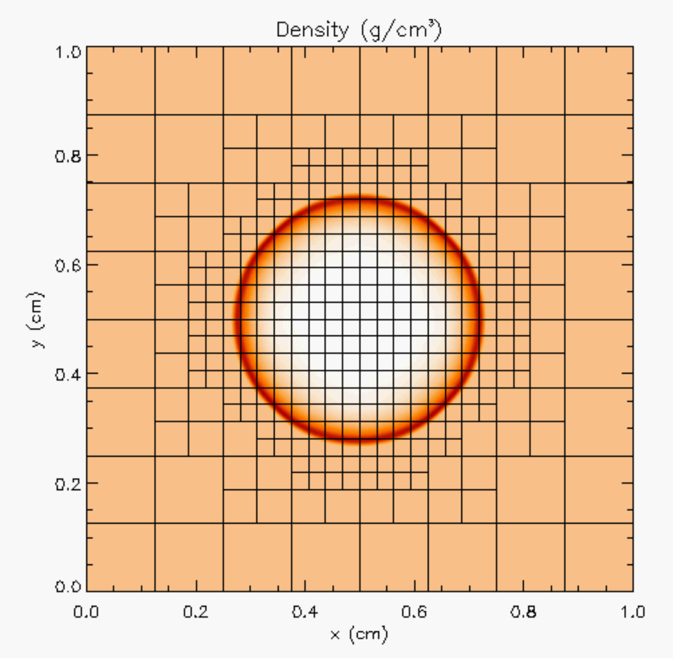
- Mesh
- Hydrodynamics solver
- Equation of state
- Parallelization

Strategy for development
Think of an application with analytical solution

Example – Shock Hydrodynamics with Adaptive Mesh Refinement

Components needed

- Mesh
- Hydrodynamics solver
- Equation of state
- Parallelization



- Sedov blast wave
- High pressure at the center
- Shock moves out in a circle
- Analytical solution for how far the shock has travelled

Strategy for development
Think of an application with analytical solution

Step 1 – Equation of State

- Initialize density and internal energy with known values
- Compute pressure and temperature using EOS
- Next use density and computed pressure as input and compute internal energy and temperature using EOS
- Compare computed values against initialized values

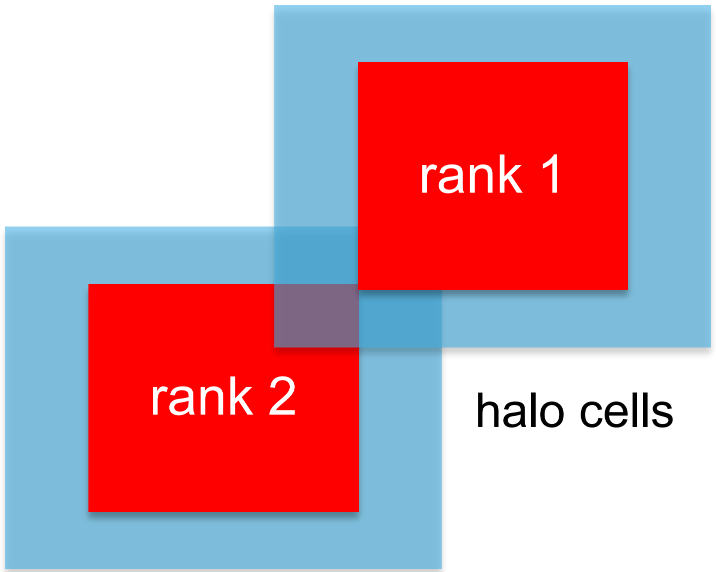
Step 1 – Equation of State

- Initialize density and internal energy with known values
- Compute pressure and temperature using EOS
- Next use density and computed pressure as input and compute internal energy and temperature using EOS
- Compare computed values against initialized values

We have a unit test

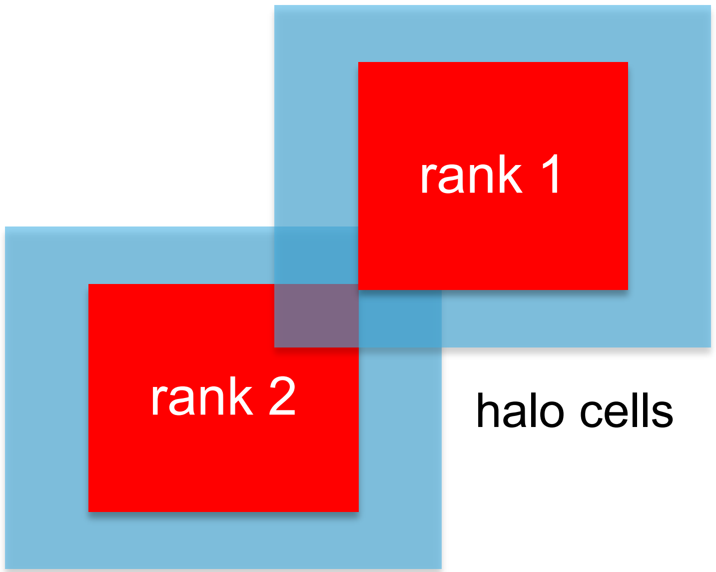
Step 2 – Mesh

- Start with uniform grid
- Domain decomposition for parallelization
  - Halo fill operation
- Initialize the interior (red) with a known function
- Apply halo fill
- Compute values for the halo using the known function
- Compare against filled values



Step 2 – Mesh

- Start with uniform grid
- Domain decomposition for parallelization
  - Halo fill operation
- Initialize the interior (red) with a known function
- Apply halo fill
- Compute values for the halo using the known function
- Compare against filled values



We have another unit test

Step 3 – Hydrodynamics

- Apply initial conditions to the mesh
  - zeroes everywhere except at the center
- Write code for the analytical expression of the distance traveled by the shock
- Do time integration
- At time **T** compare evolved solution against analytical solution

If both mesh and EOS unit test pass, then any failure is in Hydrodynamics

This is the idea behind scaffolding

Step 4: AMR

- The same halo fill unit test for mesh also works for AMR
- Additional functionalities to test are:
  - Fine-coarse boundary resolution
  - Regridding
- Steps in testing
  - Run Sedov with UG
  - Run Sedov with AMR, but no dynamic refinement
    - If failed fault is in flux correction
  - Run Sedov with AMR and dynamic refinement
    - If failed fault is in regridding

Step 4: AMR

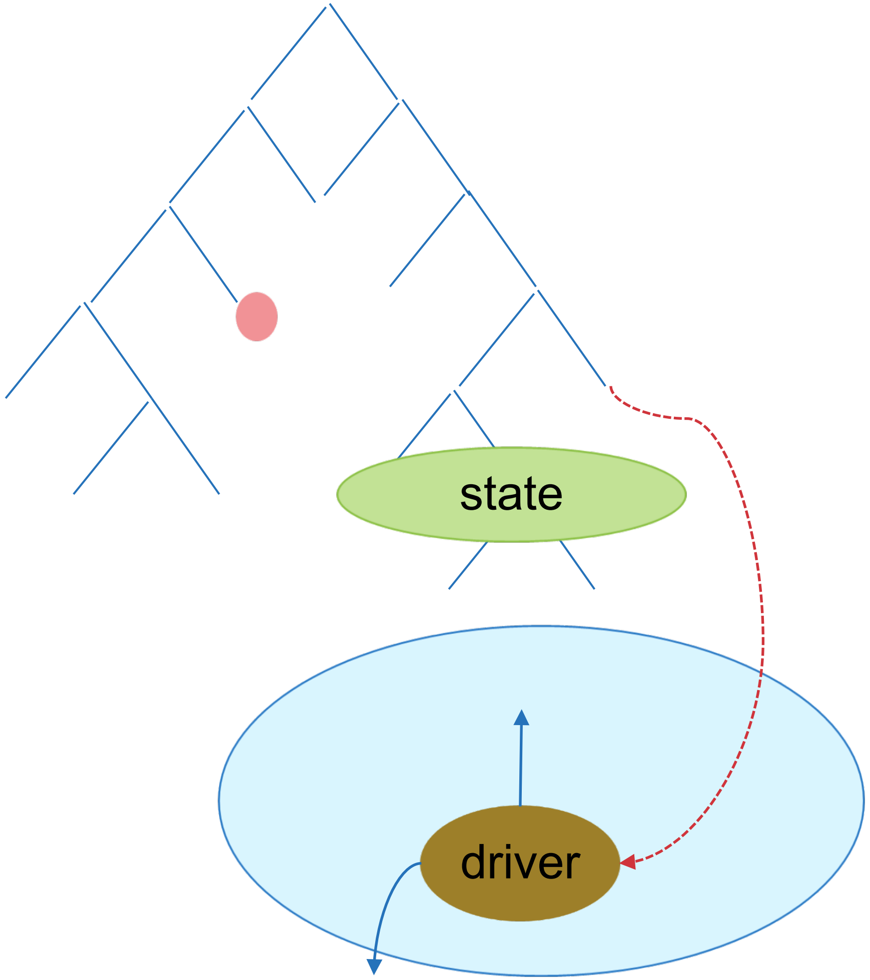
- The same halo fill unit test for mesh also works for AMR
- Additional functionalities to test are:
  - Fine-coarse boundary resolution
  - Regridding
- Steps in testing
  - Run Sedov with UG
  - Run Sedov with AMR, but no dynamic refinement
    - If failed fault is in flux correction
  - Run Sedov with AMR and dynamic refinement
    - If failed fault is in regridding

We have continued to build scaffolding and are able to pinpoint cause of error

Test Development For a Legacy Code

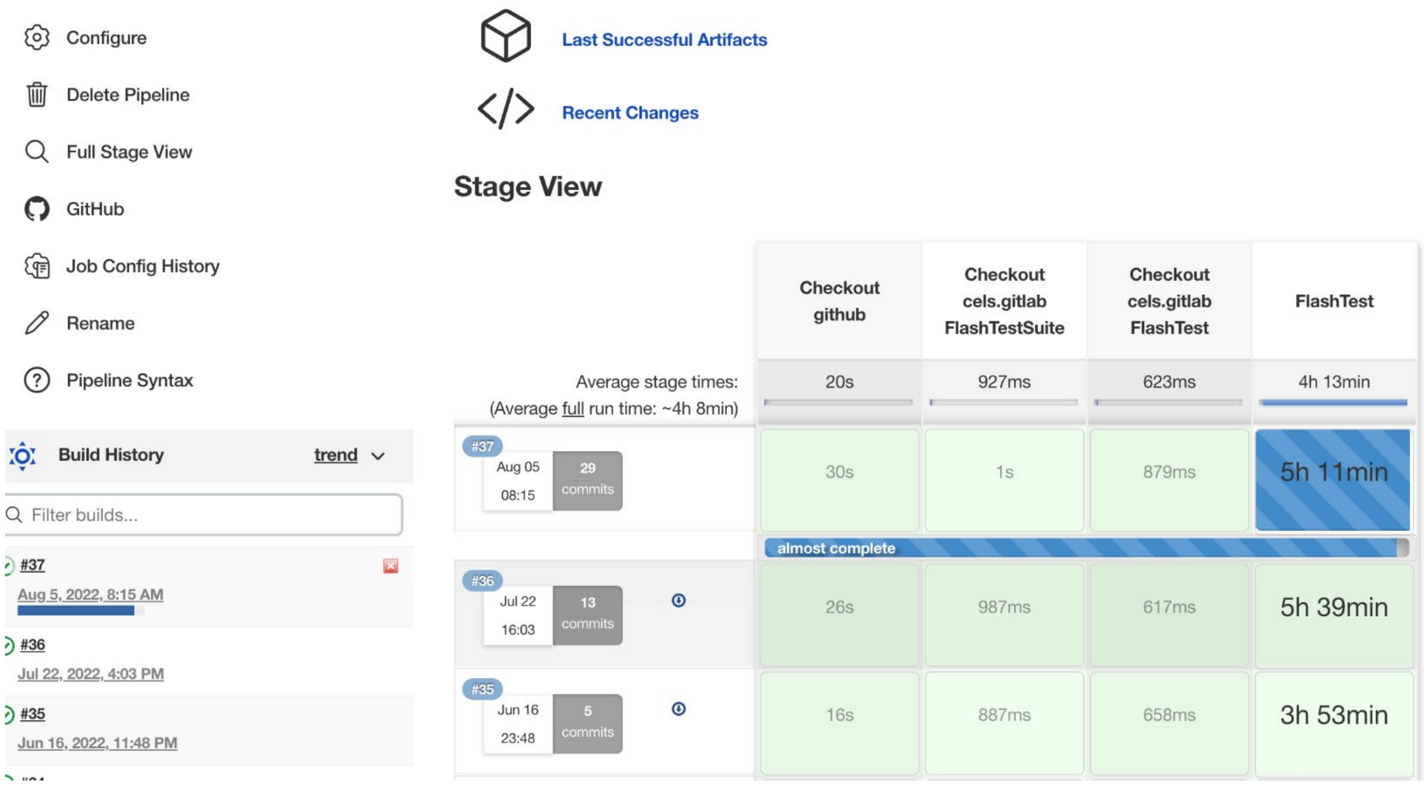
There may not be existing tests

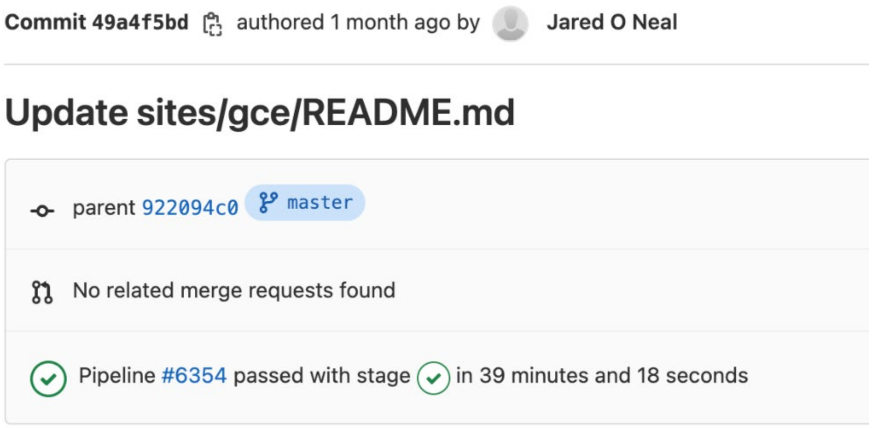
- Isolate a small area of the code
- Dump a useful state snapshot
- Build a test driver
  - Start with only the files in the area
  - Link in dependencies
    - Copy if any customizations needed
- Read in the state snapshot
- Restart from the saved state
- Verify correctness
  - Always inject errors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



How to build your test suite?

- Two "levels"
  - Automated / scheduled testing
    - May be long running
    - Provide comprehensive coverage
  - Continuous integration
    - Quick diagnosis of error





How to build your test suite?

- 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/

How do we determine what 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
  - Necessary for testing, but not sufficient
- 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

Building Test-suite

First line of defense – code coverage tools

- Code coverage tools necessary but not sufficient
- Do not give any information about interoperability

	Hydro	EOS	Gravity	Burn	Particles
AMR	CL	CL		CL	CL
UG	SV	SV			SV
Multigrid	WD	WD	WD	WD	
FFT			PT		

- Map your tests and examples – what do they do?
- Follow the order
  - All unit tests – including full module tests (e.g. CL)
  - Tests sensitive to perturbations (e.g. SV)
  - Most stringent tests for solvers (e.g. WD, PT)
  - Least complex test to cover remaining spots (**Aha!**)

IDEAS productivity ECP EXASCALE COMPUTING PROJECT

Good Rules of Thumb

- Test your tests!
  - Make sure tests fail when they're supposed to!
- Add "regression tests"
  - Ensure that bugs aren't creeping in
- Test regularly
  - Critical when teams are adding code regularly
  - To identify and document where changes to the underlying platform change code behavior/results
- Automate regular testing
  - Inculcate the discipline of monitoring the outcome of regular testing
- Exercise third-party dependencies
- Physics/math based strategies
  - Conserved quantities, symmetries, synthetic operators
  - Eliminate complete dependence on bitwise reproducibility

Summary

- A testing strategy is essential for producing reliable trustworthy software
  - Invest the time needed to thoroughly test your software at all levels
  - Use automation whenever possible
- Different challenges are associated with exploratory, legacy, and composable codes
  - Adapt your strategy to fit your situation.
  - Eventually you will want to be able to verify all components in a code release.
- Don’t get distracted by all the technologies out there – focus on exercising your code.
  - Scaffolding projects can help with mechanics.

Resources

- Oberkampf, W., & Roy, C. (2010). Verification and Validation in Scientific Computing. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511760396
- Michael Feathers. 2004. Working Effectively with Legacy Code. Prentice Hall PTR, USA. ISBN: 9780131177055
- A Dubey, K Weide, D Lee, J Bachan, C Daley, S Olofin… - Ongoing Verification of a Multiphysics Community Code. Software: Practice and Experience, 2015 **https://doi.org/10.1002/spe.2220**