Software Testing

Anshu Dubey
Argonne National Laboratory

Software Productivity Track, ATPESC 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, Mark C. Miller, Katherine M. Riley, and James M. Willenbring, Software Productivity Track, in Argonne Training Program for Extreme Scale Computing (ATPESC), August 2020, online. DOI: 10.6084/m9.figshare.12719834**
- Individual modules may be cited as *Speaker, Module Title*, in Software Productivity Track…

Acknowledgements

- Additional contributors include: Patricia Grubel, Rinku Gupta, Mike Heroux, Alicia Klinvex, Jared O'Neal, David Rogers, Deborah Stevens
- 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 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.

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

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

Verification Challenges

- Functionality coverage
- Particularly true of codes that allow composability in their configuration
- Codes may incorporate some legacy components
  - Its own set of challenges
    - No existing tests at any granularity
- Examples – multiphysics application codes that support multiple domains

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

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/

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

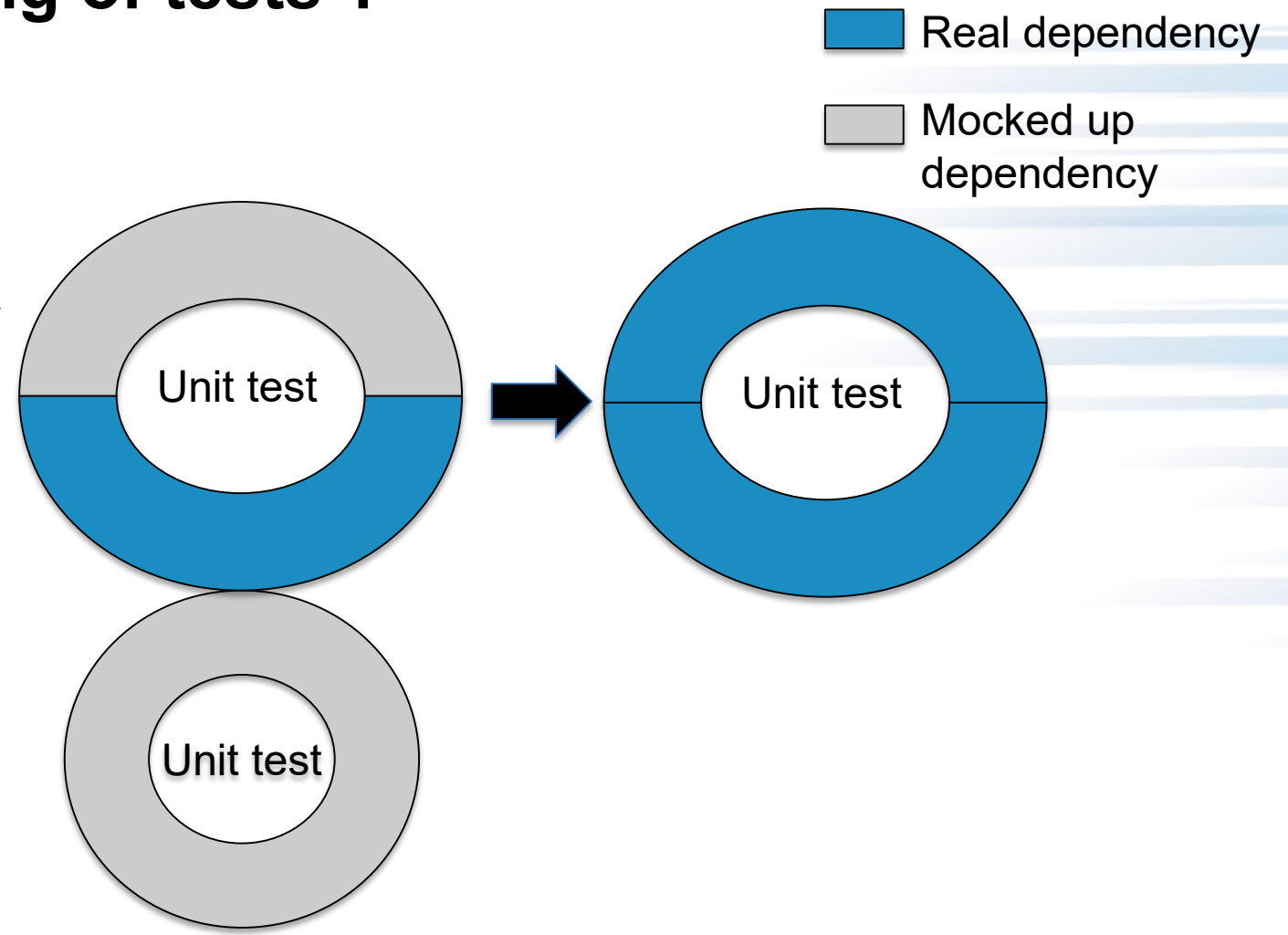
Balance is critical

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

How do you build a scaffolding of tests ?

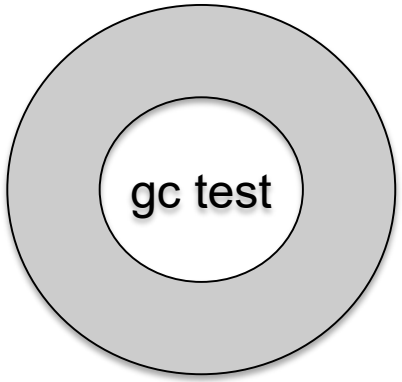
- Approach the problem sideways
  - Components can be exercised against known simpler applications
  - Same applies to combination of components
- Build a scaffolding of verification tests to gain confidence

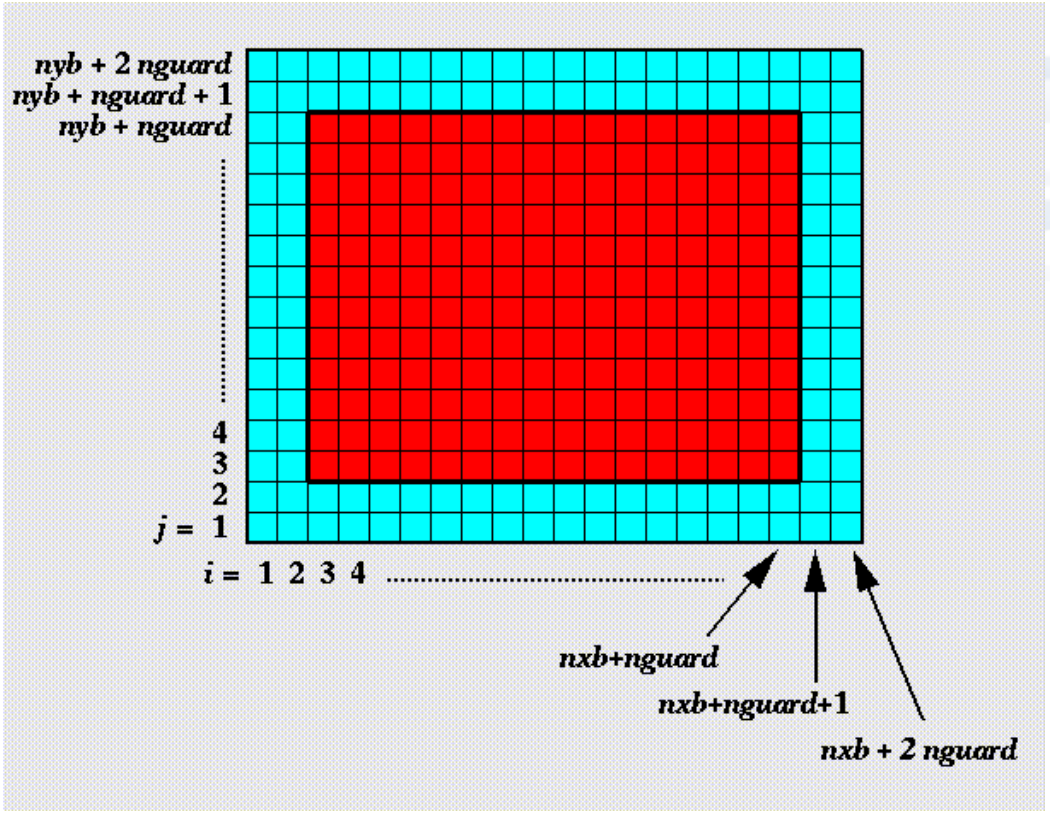


Scaffolding Example from FLASH

Unit test for Grid

- Verification of guard/ghost/halo cell fill
- Use two variables A & B
- Initialize A in all cells and B only in the interior cells (red)
- Apply guard cell fill to B

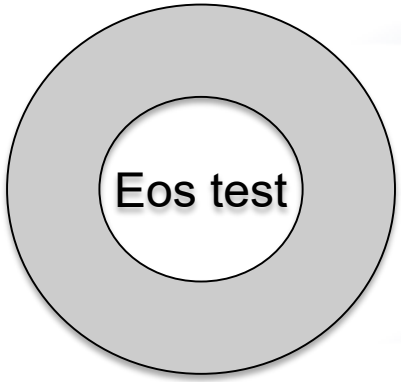




IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Scaffolding Example from FLASH



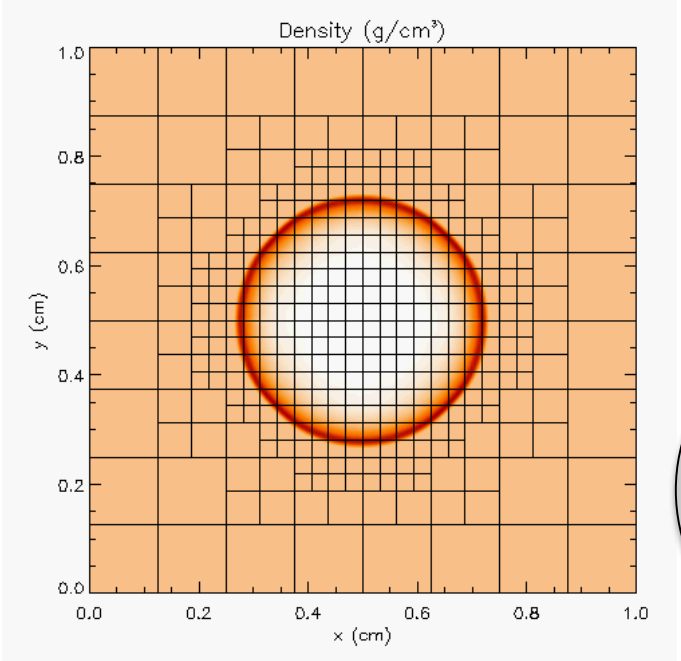
Unit test for Equation of State (EOS)

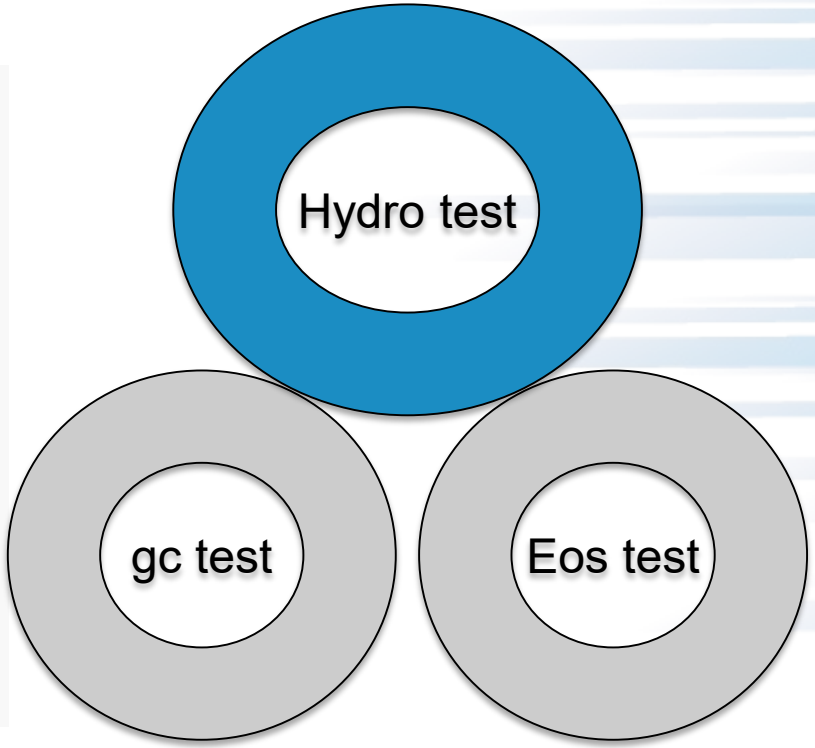
- Three modes for invoking EOS
  - MODE1: Pressure and density as input, internal energy and temperature as output
  - MODE2: Internal energy and density as input temperature and pressure as output
  - MODE3: Temperature and density as input pressure and internal energy as output
- Use initial conditions from a known problem, initialize pressure and density
- Apply EOS in MODE1
- Using internal energy generated in the previous step apply EOS in MODE2
- Using temperature generated in the previous step apply EOS in MODE3
- At the end all variables should be consistent within tolerance

Scaffolding Example from FLASH

Unit test for Hydrodynamics

- Sedov blast wave
- High pressure at the center
- Shock moves out spherically
- FLASH with AMR and hydro
- Known analytical solution





Though it exercises mesh, hydro and eos, if mesh and eos are verified first, then this test verifies hydro

More testing needed for Grid using AMR
Flux correction and regridding

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Scaffolding Example from FLASH

For AMR, correct behavior of flux conservation and regridding should also be verified.

Reason about correctness for testing Flux correction and regridding

IF Guardcell fill and EOS unit tests passed

- Run Hydro without AMR
  - If failed fault is in Hydro
- Run Hydro with AMR, but no dynamic refinement
  - If failed fault is in flux correction
- Run Hydro with AMR and dynamic refinement
  - If failed fault is in regridding

Exercise: Devise a sequence of tests for heat_app.c to provide similar coverage

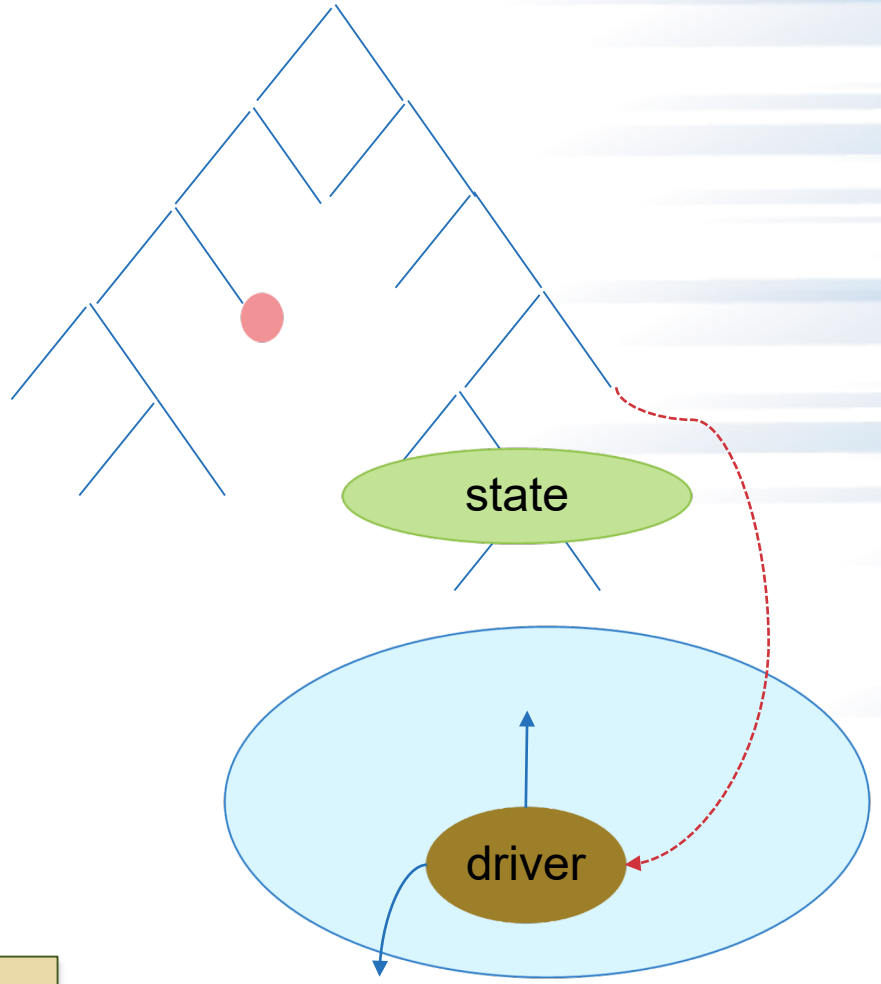
Test Development For a Legacy Code

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
- Verify correctness
  - Always inject errors to verify that the test is working

Example from E3SM

- 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



Exercise: From heat_app.c develop a test for l2_norm function using this method

Test Selection

First line of defense – code coverage tools (demo in refactoring module)

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

- Build a functionality matrix
  - Physics along rows
  - Infrastructure along columns
  - Alternative implementations, dimensions, geometry
- Mark <i,j> if test covers corresponding features, and is a valid combination
- Follow the order
  - All unit tests – including full module tests
  - Tests representing ongoing productions
  - Tests sensitive to perturbations
  - Most stringent tests for solvers
  - Least complex test to cover remaining spots

Example

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

- A test on the same row indicates interoperability between corresponding physics
- Similar logic would apply to tests on the same column for infrastructure
- More goes on, but this is the primary methodology

Tests	Symbol
Sedov	SV
Cellular	CL
Poisson	PT
White Dwarf	WD

Good Testing Practices

- 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

TAKEAWAYS

- YOUR VERIFICATION AND TESTING REGIME SHOULD MEET YOUR PROJECT NEEDS
- NO NEED TO GO OVERBOARD BUT MAKE SURE THAT YOU HAVE CONFIDENCE IN THE CORRECT BEHAVIOR OF YOUR CODE
- DEVISE TESTS TO ENABLE QUICK PINPOINTING OF ERRORS
- MAKE SURE THAT YOUR TESTS FAIL WHEN THEY SHOULD

…….QUESTIONS ?