Verification and Evaluating Project Needs

Presented at
Better Scientific Software tutorial

ECP 2nd Annual Meeting, Knoxville, Tennessee

Anshu Dubey

Argonne National Laboratory

EXASCALE COMPUTING PROJECT

National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

IDEAS productivity

License, citation and acknowledgements

- Requested citation: Anshu Dubey, Verification and Evaluating Project Needs, tutorial, in Exascale Computing Project 2nd Annual Meeting, Knoxville, Tennessee, 2018. DOI: TBA.

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.

Scientific Software Verification

Challenges specific to scientific software

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.

CSE verification challenges

- Floating point issues
  - Different results
    - On different platforms and runs
    - Ill-conditioning can magnify these small differences
      - Final solution may be different
      - Number of iterations may be different
- Unit testing
  - Sometimes producing meaningful testable behavior too dependent upon other parts of the code
- Definitions don’t always fit

CSE verification challenges

- Integration testing may have hierarchy too
- Particularly true of codes that allow composability in their configuration
- Codes may incorporate some legacy components
  - Its own set of challenges
    - No existing tests of any granularities
- Examples – multiphysics application codes that support multiple domains

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

Stages and types of verification

- If refactoring
  - Ensuring that behavior remains consistent and expected
- All stages have a mix of automation and human-intervention

Note that the stages apply to the whole code as well as its components

Test Development

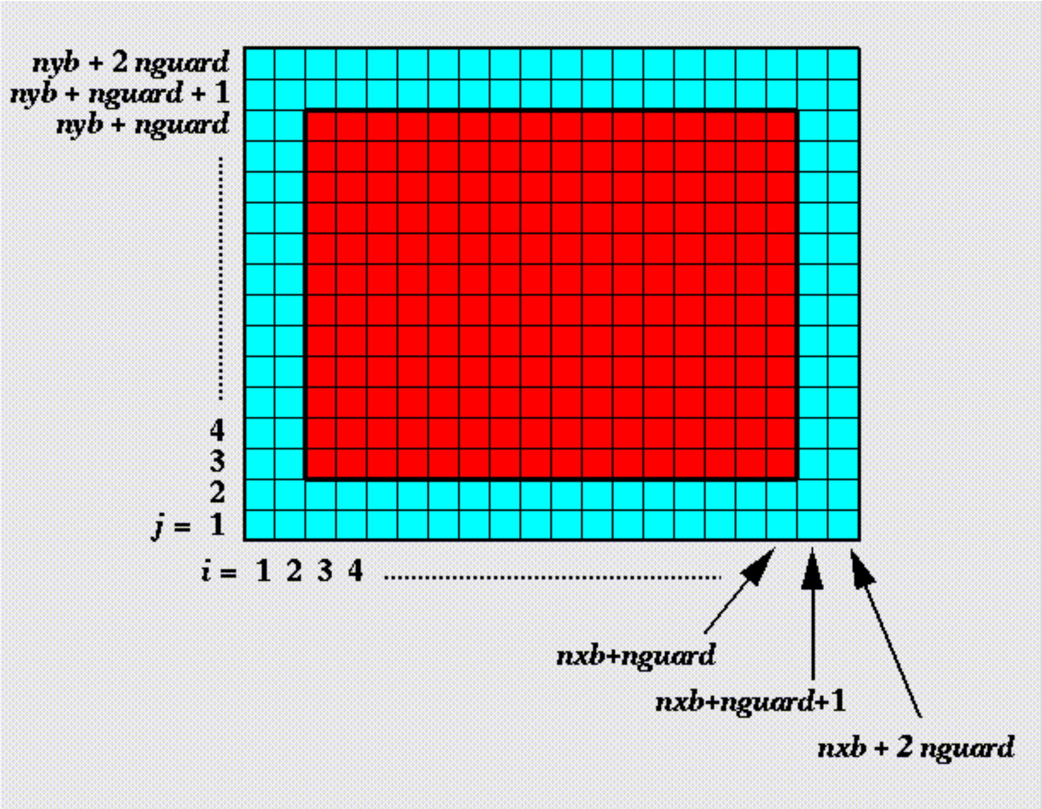
- Development of tests and diagnostics goes hand-in-hand with code development
  - Non-trivial to devise good tests, but extremely important
  - Compare against simpler analytical or semi-analytical solutions
    - They can also form a basis for unit testing
- In addition to testing for “correct” behavior, also test for stability, convergence, or other such desirable characteristics
- Many of these tests go into the test-suite

Example from Flash

- Grid ghost cell fill
    - Use some function to initialize domain
    - Two variables, in one only interior cells initialized, in the other ghost cells also initialized
    - Run ghost cell fill on the first variable – now both should be identical within known tolerance
  - Use redundant mechanisms

Against manufactured solution

- Verification of guard cell fill
- Use two variables A & B
- Initialize A including guard cells and B excluding them
- Apply guard cell fill to B



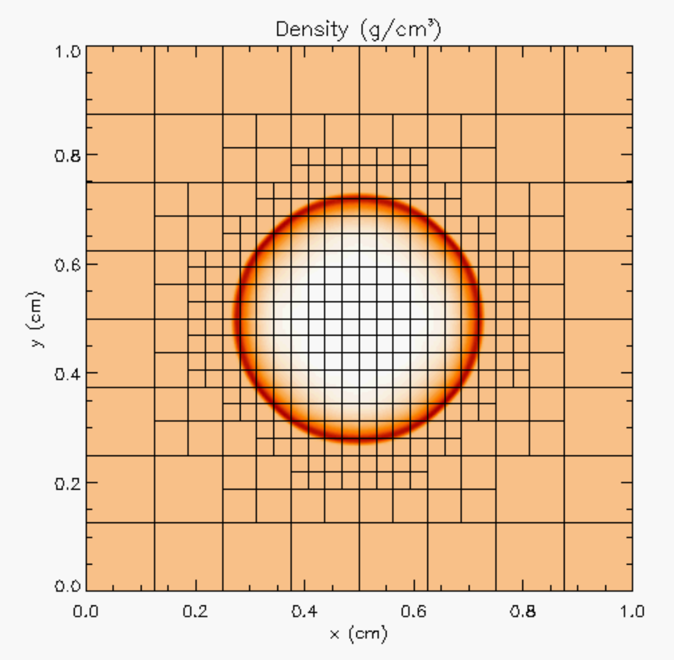
Example from Flash

- EOS
  – Use initial conditions from a known problem
  – Apply EOS in two different modes – at the end all variables should be consistent within tolerance

- Hydrodynamics
  – Sedov blast problem has a known analytical solution
  – Runs with UG and AMR

Against analytical solution

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



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

Building confidence

- First two unit tests are stand-alone
- The third test depends on Grid and Eos
  - Not all of Grid functionality it uses is unit tested
    - Flux correction in AMR
- If Grid and Eos tests passed and Hydro failed
  - If UG version failed then fault is in hydro
  - If UG passed and AMR failed the fault is likely in flux correction

How to evaluate project needs

And devise a testing regime

Why not always use the most stringent testing?

- Effort spent in devising tests and testing regime are 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

Evaluating 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

Commonalities

- Unit testing is always good
  - It is never sufficient
- Verification of expected behavior
- Understanding the range of validity and applicability is always important
  - Especially for individual solvers

Development phase – adding on

- Few more steps when adding new components to existing code
  - Know the existing components it interacts with
  - Verify its interoperability with those components
  - Verify that it does not inadvertently break some unconnected part of the code
- May need addition of tests not just for the new component but also for some of the old components
  - This part is often overlooked to the detriment of the overall verification

Selection of tests

- Important to aim for 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

Full paper Dubey et al 2015

Approach

- Build a matrix
  - Physics along rows
  - Infrastructure along columns
  - Alternative implementations, dimensions, geometry
- Mark <i,j> if test covers corresponding features
- 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		

Tests	Symbol
Sedov	SV
Cellular	CL
Poisson	PT
White Dwarf	WD

- 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

Challenges with legacy codes

Checking for coverage

- Legacy codes can have many gotchas
  - Dead code
  - Redundant branches
- Interactions between sections of the code may be unknown
- Can be difficult to differentiate between just bad code, or bad code for a good reason
  - Nested conditionals

Code coverage tools are of limited help

An Approach

- 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

Methodology developed for the ACME project, proving to be very useful

Agenda

Time	Topic	Speaker
1:30pm-2:15pm	Why effective software practices are essential for CSE projects	Anshu Dubey, ANL
2:15pm-2:45pm	Better (small) scientific software teams	Michael A. Heroux, SNL
2:45pm-3:00pm	Improving Reproducibility Through Better Software Practices	Michael A. Heroux, SNL
3:00pm-3:30pm	Break	
3:30pm-4:15pm	Testing HPC Scientific Software: Introduction	Jared O'Neal, ANL
4:15pm-4:45pm	Verification, and Evaluating Project Testing Needs	Anshu Dubey, ANL
4:45am-5:00pm	Code Coverage and CI	Jared O'Neal, ANL