Community Codes and Good Software Techniques

Scientific codes are complex

Introduction to the next two days

Katherine Riley
August 13, 2014
Scientific applications are complex

- Physics/Domain Problem
- Applied Mathematics
- Computer Science
- I/O
- Verification
- Validation
- Software Engineering

Using the largest computer systems pushes the boundaries of all of these
Scientific applications are complex

- Physics/Domain Problem
- Applied Mathematics
- Computer Science
- I/O
- Verification
- Validation
- Software Engineering

Using the largest computer systems pushes the boundaries of all of these
ATPESC Material Covered so far

- Architecture
- Programming Models
  - MPI, OpenMP, Acceleratos/OpenACC
  - Chapel, Charm++, UPC, ADLB, etc
- Numerical Algorithms
  - Libraries, toolkits, etc
- Tools & Performance
- Visualizing & Analyzing Data
- I/O & Data

Combine your science to have the primary building blocks of your application code
Simple – right?
Putting all this to use

- Architecture
- Programming Models
  - MPI, OpenMP, Acceleratos/OpenACC
  - Chapel, Charm++, UPC, ADLB, etc
- Numerical Algorithms
  - Libraries, toolkits, etc
- Tools & Performance
- Visualizing & Analyzing Data
- I/O & Data

Software Practice
Scientific Process
Producing Domain Science
Putting all this to use

- Architecture
- Programming Models
  - MPI, OpenMP, Acceleratos/OpenACC
  - Chapel, Charm++, UPC, ADLB, etc
- Numerical Algorithms
  - Libraries, toolkits, etc
- Tools & Performance
- Visualizing & Analyzing Data
- I/O & Data

Producing Domain Science

Software Practice

Scientific Process

It is getting friendlier Perhaps because it has to
Individual components and the whole picture

- Architecture
- Programming Models
  - MPI, OpenMP, Accelerators/OpenACC
  - Chapel, Charm++, UPC, ADLB, etc
- Numerical Algorithms
  - Libraries, toolkits, etc
- Tools & Performance
- Visualizing & Analyzing Data
- I/O

Concepts
- Software Practices
- Scientific Process
- Portability
- Extensibility
- Performance
- Provenance
- Resilience
- Reproducibility
- Verification and Validation
- And more...

Your code will live longer than you think it will.
## Software Engineering and HPC
### Efficiency vs. Other Quality Metrics

<table>
<thead>
<tr>
<th>How focusing on the factor below affects the factor to the right</th>
<th>Correctness</th>
<th>Usability</th>
<th>Efficiency</th>
<th>Reliability</th>
<th>Integrity</th>
<th>Adaptability</th>
<th>Accuracy</th>
<th>Robustness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>↑</td>
<td></td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td></td>
<td>↑</td>
<td>↓</td>
</tr>
<tr>
<td>Usability</td>
<td></td>
<td>↑</td>
<td></td>
<td>↑</td>
<td>↑</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td>↓</td>
<td></td>
<td></td>
<td></td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>↑</td>
<td></td>
<td></td>
<td>↑</td>
<td>↑</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td></td>
<td></td>
<td></td>
<td>↑</td>
<td></td>
<td>↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adaptability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td>↑</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robustness</td>
<td>↓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source:
*Code Complete*
Steve McConnell

![Source: Sandia National Laboratories](image)
Selective Slice of Good Scientific Practice

- **Error**
  - Numerical Sensitivity
  - Machine rounding
  - Reproducibility

- **Verification**
  - All parts of the code keep giving what you expect
  - Reproducibility

- **Validation**
  - Compare to experiments
  - Reproducibility
  - Reproducibility of results
    - Exact code used
    - Documented
    - Method transparency
    - Data availability
    - Coding Standards

Experimentalists have a strong culture of reproducing results. Computational science needs to get there. It is trying.

**Scientist's Nightmare: Software Problem Leads to Five Retractions**
Greg Miller
Some first steps for good Scientific Process

- **Error**
  - Applied math (numerical analysis)

- **Verification**
  - Unit testing
  - Regular testing - on scale of development speed

- **Validation**
  - Prove it represents the real world
  - Very science driven

- **Reproducibility**
  - Version tag code used for simulations
  - Clear documentation on code - even publish it
  - Data provenance
  - Data archiving
  - Understand & document workflow
  - Agree & Document coding standards

“Scientific Data is an open-access, peer-reviewed publication for descriptions of scientifically valuable datasets. Our primary article-type, the Data Descriptor, is designed to make your data more discoverable, interpretable and reusable.”
Okay - don’t run away

- Scientific code is complex
- A lot of concepts
  - Some you know already
  - Some addressed at ATPESC
  - Some brand new
- It is overwhelming if you are trying to consider 100 things at once before working on codes
- Don’t
  - Start with what works for you or your team but know that as the scale of science grow, the big picture becomes more crucial
  - If adopting an existing code, consider more
How to start the Software Process

- Decide on crucial data structures
  - Informed by science, architectures, future
  - How much will you share?
  - Data flow through functionality

- Architecture of code
  - Functional abstractions
  - *Parallelism abstractions*
  - Data ownership clear
  - Interplay between architecture and performance
  - Coding Standards

- Understand workflow
Why is Software Process Important

- Modern scientific computing is no longer a solo effort
  - Should not be a solo effort
  - Most interesting modeling questions that could be simulated by the heroic individual programming scientist have already been investigated
  - “Productivity language” that are meant to alleviate the complexity of programming high performance software have not delivered yet
  - Thus, coding is complicated and requires division of roles and responsibilities.
- Working together on a common code is difficult unless there is a software process
Software Process Components

For All Codes
- Code Repository
- Build Process
- Code Architecture
- Coding Standards
- Verification Process
- Maintenance (Support) Practices

Publicly Distributed
- Distribution Policies
- Contribution Policies
- Attribution Policies
Many flavors of code and trade-offs

- Blackbox use of existing code
- Alteration of/collaboration on existing code
- Use of libraries
- Use of a framework
- Development of new code

- Trade offs include (not complete)
  - Speed to science
  - Features
  - Control of methods & accuracy
  - Complexity of use
  - Validation
  - Verification
Building a Scientific Code

Domain component interfaces
- Data mediator interactions
- Hierarchical organization
- Multiscale/multiphysics coupling

Native code & Data objects
- Single use code
- Coordinated component use
- Application specific

Shared data objects
- Meshes
- Matrices

Library Interfaces
- Data transformation
- Parameter config

Documentation
- Source markup
- Embedded examples

Testing Content
- Unit Tests
- Glue Testing

Build Content
- Rules
- Parameters

Programming Model & Languages

Libraries
- Solvers

Frameworks & tools

SW Engineering
- Productivity tools
- Models, processes

Adapted a slide from Mike Heroux, SNL
Considerations

Some of the technical considerations

- Choosing your tools, codes, etc
  - Libraries
  - Frameworks
  - Open source code
  - Community code
  - Closed or commercial code
- Writing the code
  - Data structures
  - Data structures
  - Data structures from storage to memory to cache and back to storage (locality)
  - Parallelization of work and data
  - Languages

Everything else

- Development
  - Availability where and when you need them
  - Sustained support
  - Feature support
- The future of the code
  - HPC is the land of low level languages
  - HPC is the land of some bleeding
- Flexibility to replace libraries
- Flexibility to adapt to architectures
- ..
Obstacles for Reusing Code

- Using externally developed software seen as risk
  - Can be hard to learn
  - May not not be what you need
  - May not be what you *think* you need
  - Upgrades of external software can be risky
    - Backward compatible?
    - Regression in capability?
  - Support model may not be sufficient
  - Long term commitment may be missing

- What can reduce the risk of depending on external software?
  - Use strong software engineering processes and practices
    - high quality, low defects, frequent releases, regulated backward compatibility, ...
  - 10-30 year commitment
  - Develop self-sustaining software
Models for developing scientific codes

- Open source community developed codes
  - Always available, any contribution open source code
  - Central controls of code development
  - Closed non-commercial codes
  - Commercial code
- Speed of change
- Key design ideas
  - Scientific mission - scientists involved
  - Always capable of science
  - Portability - range of platform scale very beneficial
  - Documented
  - Clear design
  - Prove the code
Self Sustaining Software

- **Open-source**: The software has a sufficiently loose open-source license allowing the source code to be arbitrarily modified and used and reused in a variety of contexts (including unrestricted usage in commercial codes).

- **Core domain distillation document**: The software is accompanied with a short focused high-level document describing the purpose of the software and its core domain model.

- **Exceptionally well testing**: The current functionality of the software and its behavior is rigorously defined and protected with strong automated unit and verification tests.

- **Clean structure and code**: The internal code structure and interfaces are clean and consistent.

- **Minimal controlled internal and external dependencies**: The software has well structured internal dependencies and minimal external upstream software dependencies and those dependencies are carefully managed.

- **Properties apply recursively to upstream software**: All of the dependent external upstream software are also themselves self-sustaining software.

- **All properties are preserved under maintenance**: All maintenance of the software preserves all of these properties of self-sustaining software (by applying Agile/Emergent Design and Continuous Refactoring and other good Lean/Agile software development practices).
Side note on legacy codes

Productivity in science fundamentally depends on productivity in software

- Grand-Challenge Science
- Complex Legacy Applications
- Effective Use of HPC
- Computational Science
- Extreme-scale Computational Software
- Computational Science Expertise
- Software Engineering Expertise
- Productive Collaborations
- Research Need: Software Productivity for Extreme-scale Science

- People disagree
- I think this is wrong
- But not totally wrong
- DOE has an investment to maintain

From a set of DOE workshops on HPC productivity
Consider the HPC ecosystem

- Developing code exclusively for a small cluster is not the same as developing code for HPC
- You *can* develop HPC code that will work well on your cluster and your laptop
- In HPC, the trade-off with design and performance is omnipresent
- Have reached a complexity point that code reuse & design is very important
- All your lessons from software engineering do **not** apply

Quick glimpse of some stats on Mira applications.
100% MPI, 65% threaded

<table>
<thead>
<tr>
<th>Languages</th>
<th>Code Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Open 52%</td>
</tr>
<tr>
<td>F</td>
<td>Closed 26%</td>
</tr>
<tr>
<td>Charm++</td>
<td>Fuzzy 22%</td>
</tr>
<tr>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
Over the next two days

- Impact of Community Codes on Astrophysics - Anshu Dubey
- Climate and Community Codes - Rob Jacob
- Portable Performant Scientific Code - Hal Finkel
- Quantum Monte Carlo and Electronic Structure - Anouar Benali
- Organizing the USQCD - Rich Brower
- Software Engineering Practices - Aron Ahmadia & Chris Kees
- Modern Features of a Production Scientific Code - Martin Berzins
- Workflows - Mike Wilde
- Data Provenance - David Koop
Goals

- Show you the approach and effectiveness of code cooperation in a variety of domains
- Illustrate some of the challenges of those approaches
  - Sociological & Technical
- Expose some of the processes around developing large, production scientific codes
- Ensure you know the importance and specifics of the scientific process

- We are not trying to teach you these codes
- We are passing on experience
- A lot* of people have spent a lot of time thinking about maintain codes that use the largest systems in the world

*A lot normalized for the size of the HPC community. Especially the largest scales.
Questions