BREAKOUT – PRODUCTIVITY

Argonne NATIONAL LABORATORY

SPECIAL CONCERNS OF LARGE MULTIPHYSICS HPC CODES

August 22, 2016
DOE-COE-PP
Denver, CO

PERFORMANCE PORTABILITY

In the past

- Programming to a common machine model
  – Separation of concerns
- Obtain some fraction of achievable performance through algorithmic and data structure choices
- Limited platform specific optimization
  – The least productive part of software lifecycle
- Trade-offs between component performances
- Trade-offs between programmer productivity and performance

WHAT CAN VENDORS DO

- Continue supporting the features of the language
  - Not to pull the rug from under apps
- If there is a good reason, document
- Enable ways of writing codes that avoid optimization blockers
  - Constructs that can be used to express operations without going into details
- Science libraries
  - How to write some of them, e.g. sparse solvers
- Perhaps use some resources for buffering next stage data
  - Similar to overlap between computation and communication in MPI

REFACTORING CHALLENGES

- Selection of abstractions
  - At what level ?
- Machine model
  - Not specific platforms
- Cost estimation and resource allocation
- Transition plan
- Coexistence of development and production

DATA POINTS WITH ABSTRACTIONS USE

- Good results with RAJA
  - Incremental changes to code
  - On ramp built in
  - Easy to maintain production during transition
- Figuring out how to make changes occur incrementally can be a challenge
  - But needs to happen
- May need more than one iteration
  - Generating a mini-app at first can be useful

TESTING AND VERIFICATION

- Integral and critical part of code modification
  - Especially refactoring
- Designing tests – a critical but underappreciated activity
  - As demanding as experiment design
- Bitwise reproducibility impossible
  - Differentiating between perturbations due to machine precision from a bug
- Retroactively putting in tests in legacy codes
  - Intertwined dependencies
- Coverage – not just lines of code covered, but also interoperability

SOME USEFUL TESTING IDEAS

Ideally building confidence through a pyramid

- Unit tests at the bottom
- Higher granularity tests above
- Integration tests
- System level tests
- Different permutations of capabilities

- Diagnostics – an alternative way of gaining confidence

- Inject bugs into the code to make sure tests work

- Use a matrix to put together tests in a test suite
- Test on multiple platforms

PERFORMANCE TESTING

- Check for performance variation in addition to correctness testing
- Easy to check catastrophic ones, much harder to catch gradual degradation
- Blue gene was much easier than x86

PRODUCTIVITY AWARENESS

- More mature in UK with RSE
- No equivalent in the US yet, but culture is changing
  - Projects such as ECP are helping propagate it
- Maturing as a community
- Equipment design recognized as a critical scientific contribution in its own right
  - Software plays the same role