Ten(?) Top Tips for Benchmarking Research Software

Tom Crick
Department of Computing
Cardiff Metropolitan University
Cardiff, UK
Email: tcrick@cardiffmet.ac.uk

Benjamin A Hall
Microsoft Research
Cambridge, UK
Email: benhall@microsoft.com

Samin Ishtiaq
Microsoft Research
Cambridge, UK
Email: samin.ishtiaq@microsoft.com

***Abstract*—Abstract here...**

I. Introduction

Intro...

If we want to be truthful about it, then the Scientific Literature about software tools (CAV Tool papers, Nature Method(?) papers, etc) are not doing Science themselves. We're not saying all the papers are like this, but a typical paper, if you peer beneath the shiny LaTeXand seemingly statistical tables of figures, is this:

1) The reader cannot re-implement the paper's algorithm. Reproducibility is a basic tenet of good Science. Many algorithms described in premier scientific/adademic conferences cannot be re-implemented, often because because the description lacks sufficient detail and reasoing about why something was done one way or another.
2) The tool that the paper describes either doesn't exist for download. Or runs only one one particular platform. Or might run for the author, for a while, but will bot-rot so soon that even the author cannot compile it in two months time.
3) The benchmarks the tool describes are fashioned only for this instance of this time. They might claim to be from the Windows device driver set, but the reality is that they are are stripped down versions of the originals. Stripped down so much as to be useless to anyone but the author vs the referee. It's worst than that really: enough benchmarks are included to beat other tools. The comparisons are never fair. (Neither are other peoples' comparisons against your tool.) If every paper has to be novel, then every benchmark, too, will be novel; there is no historical truth in new, crafted benchmarks.

Things can be much better.

These things present a barrier to scientific software reliability:

- the pressure to "make the discovery" and publish quickly discincentizes careful software development
- software development is hard, but sharing and using others code is relatively easy. Code developing groups may wish to hold onto their code as a competitive advantage, especially if there exist larger competitors who could use the available code to "reverse scoop" and lead into a promising new area opened by someone else. unshared code is untestable ···
- testing new scientific software is difficult, as until the software is complete unit tests may not be available
- some models may be chaotic and influenced by floating point errors (e.g. molecular dynamics), further frustrating testing
- some scientists may not have had any formal/informal training in programming, and even a basic introduction may improve the software http://philipwfowler.wordpress.com/2013/12/19/the-oxford-software-carpentry-boot-camp-one-year-on/

II. Top Tips

N.B. The "rules" should make reference to how this would improve the reusability of software that implemented the c.10 points...

Cite [1]

- scientist education http://f1000research.com/articles/3-62/v1
- the code should have an appropriate open source license (recommendations?)
- the code should be available on github or similar source control/repo?
- the code should include links to papers publishing individual algorithms
- the code should include explicit relationships to other projects on the repo (i.e. B was branched from A)
- you should provide the compiler and build toolchain
- you should provide builds tools/makefiles/ant/etc and build instructions
- you should list/link to all non-standard packages/libraries/APIs
- you should note the hardware and OS used
- you should include the models used for testing/benchmarking

- you could include an interface to the tool and a model in the web version of the paper, to demonstrate how it works i.e. make the paper "executable" eg Ben's retrodiction paper
- main proposal: automatic online dependency/build/traffic light system with github hooks?
- more main proposal: a lineage of the software (history/branches) and a (more limited) lineage of the algorithm its built on
- case studies: CS, systems biology?

III. Conclusion

Conclusions...

References

[1] C. Collbery, T. Proebsting, G. Moraila, A. Shankaran, Z. Shi, and A. M. Warren, "Measuring Reproducibility in Computer Systems Research," Department of Computer Science, University of Arizona, Tech. Rep., 2014, available from: http://reproducibility.cs.arizona.edu/tr.pdf.