Fermilab U.S. DEPARTMENT OF ENERGY | Office of Science

SC19 Highlights: Spack / Software Engineering

Chris Green

SCD Projects 2019-12-05

Overview

- Attended several of the many Spack[1]-related events, including two round-tables with opportunity for interaction with core Spack developers.
- $> 1h$ one-to-one with Peter Scheibel (a core Spack developer).
- Software Engineering / QA, including two all-day tutorials.

[1] Package manager & build orchestrator developed for HPC intended to replace our use of UPS and bring consistency to package/distribution build procedures.

Spack—Concretizer

- New solver-based concretizer (dependency tree constraint resolver) due in March:
- Will be *much* faster than current *ad hoc* Python-based concretizer.
- New concretizer will enable many improvements required for Spack to do everything we need.
- Can be evaluated for LArSoft dependency tree: preview available as a *spack solve* command on a feature branch, not yet integrated into Spack proper.

Spack—Other Highlights

- Explicit callout of FNAL contributions to Spack core in BoF.
- Developer discussions, including:
  - Integration of SpackDev[2] into Spack proper as an integrated command.
  - Other improvements desired for SpackDev and our use of Spack for package and release building and management.
- **E4S**[3]: a base collection of scientific software, built and managed using Spack, part of the official ECP[4] effort. Specially-developed (but now generally available) integration with Gitlab CI build pipelines enables quick builds of new software and/or new platforms / architectures. Contents will evolve: soliciting additions to the stack. Investigation required to evaluate usefulness / feasibility of adding HEP-centric software (ROOT, Geant4, *etc.*) *vs* losing fine control of software versions within a stack.

[2] FNAL-originated Spack external command extension for multi-package development.

[3] Extreme-Scale Scientific Software Stack.

[4] Exascale Computing Project.

Software-Engineering—Highlights

- Survey of promising—and free—tools for examining *e.g.* compiler (version, option) and platform-dependence of numeric results, floating point exceptions in (CUDA) GPU code, and optimization of mixed precision calculations for improved speed within precision tolerances. Work with developers will be necessary to remove applicability limitations for some tools that reduce their usefulness to us (code must be a simple Makefile-based project, C++ support).
- Good panel discussion of management of software engineering and SE engineers for computational research (“Research Software Engineers”—RSEs), focused on universities and national labs.
  - Approaches vary, but preponderance of centralized organization with RSEs farmed out to projects as needed, with some chargeback.
  - Generally centralized policies and/or guidelines.
  - Training / knowledge-pooling of RSEs, but also training / outreach to projects.

Software-Engineering—Highlights

- Nearly everyone is incorporating SE/QA at some level: some on-boarding faster than others.
- Increased awareness of the consequences of not doing SE/QA (wasted HPC time, time wasted debugging, reduced agility and ability to evolve software, bad results, retracted papers, destroyed careers (!)).
- Increased expectations of SE/QA in scientific results, manifested in transparency / reproducibility requirements for data $\rightarrow$ results-based papers in some journals (IEEE, ACM-TOMS, ACM-TOMACS. . .) and conferences (SC19).
- Help and advice is available at multiple levels: developer $\rightarrow$ project management $\rightarrow$ lab management—also close by at ANL (Better Scientific Software / IDEAS).
- Much learned, with plenty of ideas and resources for help incorporating / improving SE/QA and convincing others of the need for same: more details in other fora.

Backup slides...

Spack Concretizer Improvements

- Use of installed/cached binary packages from older recipe versions (cope with hash change).
- Explicit (including no) rather than implicit compiler dependence.
- Language standard compliance as a virtual dependency.
- Use of pre-built older versions of packages as installed or cached binary packages over building latest without explicit version specification.
- Better reconciliation of compatible variant specifications for the same dependency in different dependents.
- Better handling of build (*vs* link, run, test) dependencies.

Spack Developer Discussions

- Automatic detection and use of system-installed packages.
- Integration of Spack Environments into SpackDev.
- Expansion of the concept of Spack extension commands to more general Spack additions such as extensible integrated commands (external subcommands), new build systems, new module systems.