What's new in Spack?
(the slides are available under "Presentation Materials" in the above URL)
Date: July 17, 2020
Presented by: Todd Gamblin (Lawrence Livermore National Laboratory)

Q. In the package.py file you showed, sometimes it's useful to introspect the **spec** object during one of the installation phases when developing a new package. Is there a way to drop into a python debugger pdb when installing the package, or would one have to go the route of using spack shell?

A. Not currently a way to do this because of how output is routed to the build. You can use spack build-env to drop into the build environment and debug things. See https://spack.readthedocs.io/en/latest/packaging_guide.html#cmd-spack-build-env.

Q: Spack recently added public spack mirror where spack can install from buildcache, which arch, compiler/MPI does it support? Can we contribute back to the public spack mirror? Does spack intend to add compilers in buildcache?

A. Current public spack mirror is only for source code. Every tarball and patches and resources are in the mirror. E4s has a binary mirror, targeted at particular containers, which we'll eventually merge. If you have e4s container and spack version, you can use that. See e4s.io. Working toward rolling release of binaries for certain architectures and compilers. E.g. centos for x86, with and without AVX512, support for ARM, GPUs. e4s builds with mpich because it's compatible with most of their vendor systems. Bind mount mpi would (usually) need mpich. Goals is to use more MPIs.

Q.If I need to pass options down to the cmake that builds my package, how do I do that?

A. spack edit <package name>
(Todd showed kripke package.py from the presentation) You can override the **cmake_args** definition: you expose the "knobs" in your package that you care about and pass them on to cmake. Grep around through the packages to see ways to pass arguments.

Q. Is there, or could there be a flag to NOT build the latest version of all dependencies for a particular spack package, but instead use existing (older) spack packages that meet the dependency requirement (without having to explicitly list each concrete dependency in the install command)

A. The new concretizer will try to reuse. Not in the current concretizer.

Q. Is there a plan to include other build systems (Meson, Bazel)?

A. Todd: spack doesn't constrain your build system. There are subclasses for special support for that particular build system.
Meson is supported: https://spack.readthedocs.io/en/latest/build_systems/mesonpackage.html

Bazel is also supported (with bazel as a build only dependency as there no Bazel Build system in spack as of now), see tensorflow for example :
https://github.com/spack/spack/blob/develop/var/spack/repos/builtin/packages/py-tensorflow/package.py
Bazel likes to have control over the order of includes (instead of just -I) which used to cause problems in spack, but this was fixed in https://github.com/spack/spack/pull/16077

Q. Does it make sense to have integration between other package managers? Conan? Hunter? NixOS? (by reading their configuration files, in case this would increase adoption of Spack for new packages)

A. We have thought about this. Definitely for pip. Pain to maintain all the python versions in spack! Could have wrappers around language specific package managers. Nix tries to accomplish a similar goals as spack (but not as parameterized— see https://discourse.nixos.org/t/concept-use-any-package-version/6515/31): translation would be awkward because we would have to parse their functions. Makes more sense to adapt language specific package managers. On the "far out" road map, not in the near term.

Q. How do you tell what options a package support?

A. spack info <package name>
Todd: Also could see https://spack.readthedocs.io/en/latest/package_list.html to see the package list and descriptions.

Q. Can I somehow know the progress of the build? (e.g. "6/10 dependencies built")

A. Spack will output the status of each build along the lines of installing package : Fetching,Staging, Build, Installed.
Todd: We don't tell you what's already installed yet. Could use:
spack spec -I <thing to build>

Q. Is there any desire for, or plan to implement, CI tests for packages before they are merged into the main repo?

A. Yes! That's what we're doing for the e4s pipeline right now. There is a given spack template can be build a million different ways and we would need to choose a small canonical subset of them. We are currently building the default (spack install foo) but will expand the CI figurations we try.

Q. Can I somehow ask to build a package with the same dependencies another package was previously built with? (e.g. use the same MPI)

A. spack install hdf5 ^mpich/abc123 # hash of existing mpich

A.Use “concretize:together” in an environment.

A. You may want to use spack.yaml for those types of concretization preferences.
https://spack.readthedocs.io/en/latest/build_settings.html#concretization-preferences
You could specify what to use for your mpi provider, for example, compiler, default flags to use for a given package or dependency, etc.

Q. Will 'spack external find' find packages already installed by Spack already in a prior 'spack install' command? Spack will build new versions of packages like CMake, Ninja, Python, and Perl, even if they are already installed. For example, we only need one copy of cmake-3.17.2 built with say gcc-8.3.0 but Spack will build CMake over and over again for every other compiler you want to install downstream packages for.

A. Todd: related to earlier question about relating to existing installs.
You can set constraints by ^cmake@3.17.2 per package or for all packages in the config files for now. Currently, spack prefers the newest release of a package (unless specified otherwise) which causes it to update cmake every time a new release comes out. The new concretizer is supposed to fix some of the current behavior by more aggressively re-using available software.

Q. Is there a way to keep the source code of a package?

A. You can do spack install --keep-stage … # still in /tmp so volatile
Instead use spack install --source … # and it will keep it around in
$pkg_install_prefix/share/spack/source

Q. Is there any automated testing testing of E4S with CUDA?

A. Will get tested with CUDA, AMD GPUs and Intel GPUs?

Q. What are the differences between Spack and Guix? Is there a comparative list available somewhere I can refer to?
https://discourse.nixos.org/t/concept-use-any-package-version/6515/31

Spack focuses on allowing you to build any configuration and flexibility. nix and guix don’t have packages with all the parameters spack has, also they require root to build. Spack packages are in python; spack does concretization (dependency resolution); nix/guix do not. Spack has more extensive compiler/arch/virtual dependency support. Spack binaries are relocatable, do not require root.

A. Todd: Guix is the GNU version of Nix and is in scheme. Difference is spack packages are templated. In those systems, you end up having to hack packages a lot more; more cumbersome to swap a compiler in a Nix build or build with a different parameter. In spack you're writing Templates in python. Spack focuses on combinatorial versions and Guix and Nix focus more on reproducibility.
Outdated but here is an older one :
https://archive.fosdem.org/2018/schedule/event/installing_software_for_scientists/attachments/slides/2437/export/events/attachments/installing_software_for_scientists/slides/2437/20180204_installing_software_for_scientists.pdf

Q. How does spack stacks and concretize=together interact to force consistency of the packages being installed (like a collection of libraries that all use the same MPI)?

A. Todd: Complicated! Concretize together is not for stacks, for regular environments. Like single prefix environment in regular OS = concretize together. Ensures the same dependencies and fails in certain cases. Spack stack does NOT concretize together because it needs to build with different MPIs and compilers and so on. (Really nice such thing as spack stacks — they're just combinatorial environments, but if you do a matrix and you also concretize together you'll get a conflict because the matrix specs can't be concretized together)

Q. What is the reasoning behind defaulting to using spack's own source mirror for source code downloads instead of using it as a fallback

A.just trying to be good citizens and provide bandwidth for our tool. Other distros do this. Also mirror is likely more reliable and will potentially support CloudFront or other CDN, so it'll be quite fast.

Q. Are all the heuristics used in the Spack case for the new concretizer deterministic?

A.yes, with optimization. Forcing an optimal result means you're guaranteed that you get most recent/most preferred version, most preferred MPI possible, etc. provided we weight all attributes properly there will be one "best" result.

Q. In the atmospheric sciences modeling community, it's somewhat common for Fortran based models to require recompiling files when changing any model parameters (CESM, CAMx, etc). Which means instead if those types of software were to be supported by spack, instead of installing a package, spack would need to create an environment for the Fortran model code to be recompiled each time a parameter is changed. Is this type of scenario supported? Things like environments seem to have the right idea, but one would really need the build environment also supported: things like RPATHs that I think only behave well inside the package's python-based build environment and are not exported by spack module?

PS: If this is too long of a question I could post on the mailing list instead.

PPS: I'm not a fluent Fortran programmer and don't understand the technical reason why changing parameters requires recompilation instead of loading parameters from a file. Maybe someone with more experience in Fortran land might be able to comment?

A. Todd: Answered in the presentation. Spack environments are virtualenvs for everything. Unifies the idea of manifests and lock files like cargo / bundler with the notion of an environment. Could make 2 environments with different versions of the root code with different values for some variant if the parameters require a recompile. Dependencies of code will be shared and symlinked into the env to not waste space. Only the app would be duplicated.

Regarding recompiling Fortran due to parameter change, I think it's just a matter of how that Fortran program was written. One can also write the Fortran code to load parameters from a file.
^ Ok thanks! I'll need to ask upstream if they can make that change.
^ can also patch this into the code in the spack package

Q. Considering the example given for language features as dependencies, e.g.: cxx@2017, since compilers claim to be c++17 compliant while they still aren't (e.g.: libc++ still misses some c++17 bits) do you consider managing specific toolchain differences about language level support?

A. Todd: We can do that. In spack, packages can provide more than 1 virtual dependency. Can have a package that provides C++ lambdas and a more fine grained spec of what virtuals you need instead of broad language levels. Could also have language levels if fully compliant and more fine grained if only a subset of features is supported.

Q. Thank you for the presentation Todd. I am new in HPC environment in general and spack also. What learning path will you recommend to be able to create new packages that are not already in spack?

A. Todd: Checkout the first slide of the presentation; tutorial in 2 weeks. The other thing to do is just try making a package. Look at the packaging guide in the docs. Quite long! The best way to get feedback, even if it's not working, is to submit a pull request in spack. People will comment and give you feedback. Better to put wrong answer on the internet to get an answer (Cunningham's law https://en.wikipedia.org/wiki/Ward_Cunningham#Ideas_and_inventions)

Also join spack slack — typically 50-60 people on at any given time, willing to answer questions. Visit https://spackpm.herokuapp.com for an automatic invitation

Q. How should packages be handled where dependencies change frequently? The conditions on the dependencies in the package.py can become pretty messy very fast. Also, what are plans if packages transition to a different build system?

A. Todd: Good question that we don't have good answers to! LLVM and GCC loop over their valid configurations. Have your package implicitly enumerate over how they build - see those packages in spack.

Q. Are there plans to support multiple download urls depending on variants/spec features? I.e. binaries for redhat vs ubuntu vs mac or Spark with and without Hadoop support.

A. Todd: For spack binaries we can do this. Spack already handles installing binaries for different configurations that way. Currently don't have a "when" clause directive for versions (so no way to have per-arch tarballs). Can't download certain packages for certain architectures. Will probably be added in once we have the bandwidth to handle it (after new concretizer).

Q. Is there already a Linux distribution that uses Spack as their main package manager?

A. Todd: Let's make one and call it SpackOS! xD
Don't think it would be our default mode.
Would be like Gentoo, building everything from the kernel up. Interesting.
(Lev in chat also suggested instead of SpackOS, Sparch Linux!).

Once we have compilers as dependencies can start thinking about building "distroless" apps and container images from a spack-built glibc up.