Spack Package Manager

Andy Turner

a.turner@epcc.ed.ac.uk

epcc

THE UNIVERSITY OF EDINBURGH

Reusing this material

cirrus

epcc

THE UNIVERSITY OF EDINBURGH

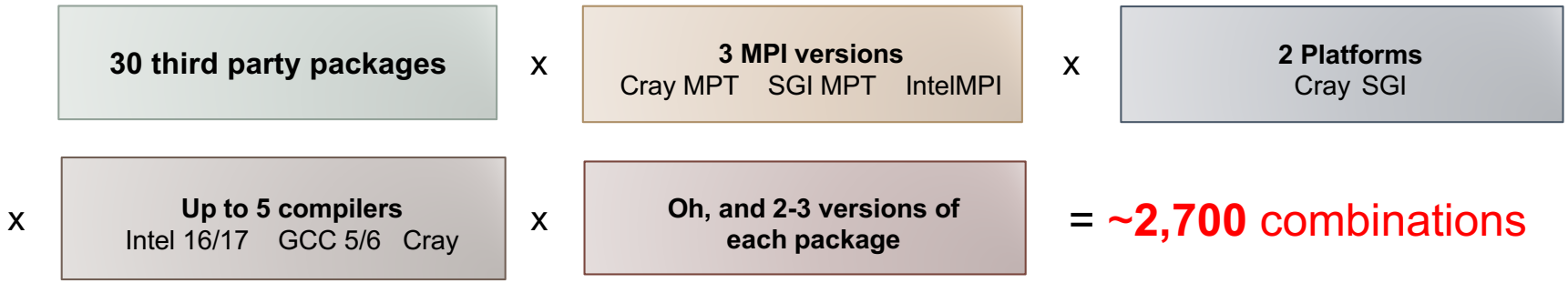
www.epcc.ed.ac.uk

www.cirrus.ac.uk

Thanks to Todd Gamblin, LLNL for much of the technical material

HPC package management issues

- Not much standardization in HPC: every machine/app has a different software stack
  - Usually source compilation is required
- Sites share unique hardware among teams with *very* different requirements
  - May want to experiment with many exotic architectures, compilers, MPI versions
  - All of this is necessary to get the best ***performance***
- Example environment for EPCC national services:



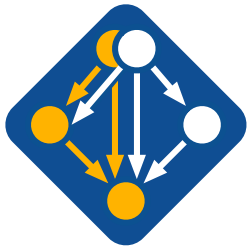
We want an easy way to quickly sample the space and to build configurations on demand

cirrus

epcc | THE UNIVERSITY OF EDINBURGH

Existing tools

- Do not really support combinatorial versioning…
- Traditional binary package managers
  - RPM, yum, APT, yast, etc.
  - Designed to manage a single stack for standard hardware
  - Install *one* version of each package in a single prefix (`/usr`).
  - Seamless upgrades to a *stable, well tested* stack
- Port systems
  - BSD Ports, portage, Macports, Homebrew, Gentoo, etc.
  - Minimal support for builds parameterised by compilers, dependency versions.
- Virtual Machines and Linux Containers
  - Containers allow users to build environments for different applications.
  - Does not solve the build problem (someone has to build the image!)
  - Performance, security, and upgrade issues prevent widespread HPC deployment – maybe a route past this with Singularity…



Spack approach

- *Simple* package management based on source compilation
  - Support for multiple compilers and versions
- Automatically resolve (and build) dependencies
  - Reuse already installed dependencies
  - Support pre-installed dependencies (e.g. vendor MPI, MKL)
- Capture compilation instructions in natural way
  - Open Source to share with community
- Integration with various environment module solutions
- Focus on simplicity of implementation and use

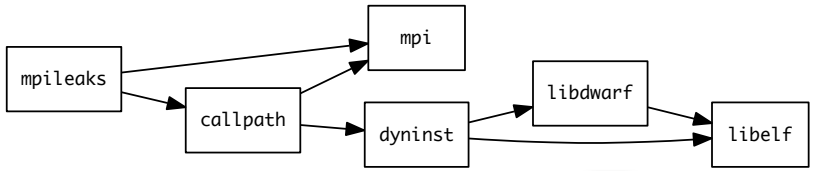
cirrus

epcc

THE UNIVERSITY OF EDINBURGH

Spack – handling combinations

Dependency DAG



Installation Layout

Hash

```
spack/opt/
    linux-x86_64/
        gcc-6.2.0/
            mpileaks-1.1-
0f54bf34cadk/
        intel-17.0.2/
            hdf5-1.8.15-lkf14aq3nqiz/
    bgq/
        xl-12.1/
            hdf5-1-8.16-fqb3a15abrwx/
    ...
```

- Each unique dependency graph is a unique ***configuration***.
- Each configuration installed in a unique directory.
  - Configurations of the same package can coexist.
- **Hash** of entire directed acyclic graph (DAG) is appended to each prefix.
- Installed packages automatically find dependencies
  - Spack embeds RPATHs in binaries.
  - No need to use modules or set LD_LIBRARY_PATH
  - Things work *the way you built them*

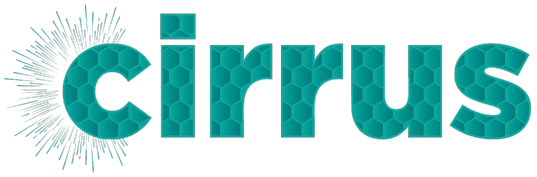
cirrus

epcc | THE UNIVERSITY OF EDINBURGH

spec syntax to describe configurations

```
$ spack install gromacs                          unconstrained
$ spack install gromacs@2016.3                   @ custom version
$ spack install gromacs@2016.3 %gcc@6.2.0        % custom compiler
$ spack install gromacs@2016.3 %gcc@6.2.0 +double   +/- build option
$ spack install gromacs@2016.3 %intel@17 target=knl   target=<cpu target>
```

- Each expression is a ***spec*** for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customise install on the command line!
- Syntax abstracts details in the common case
  - Makes parameterisation by version, compiler, and options easy when necessary

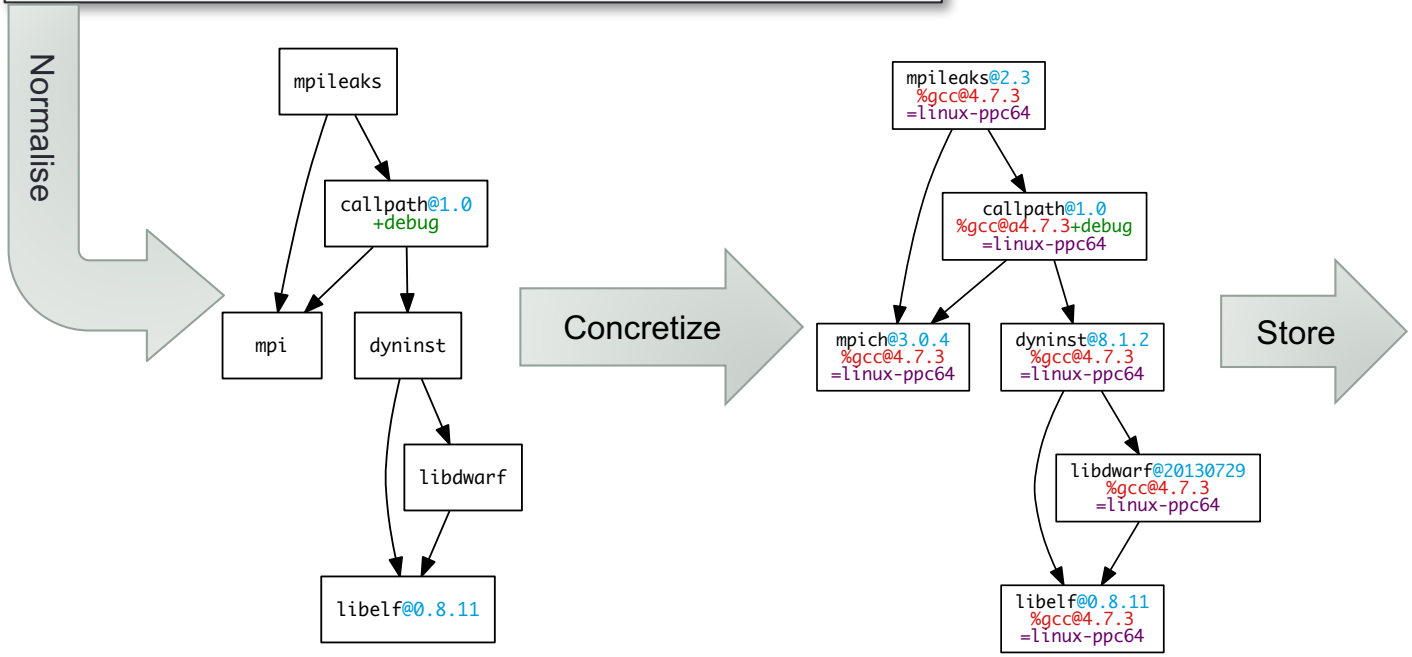


Concretization

- Fills in missing configuration details when user is not explicit

```
mpileaks ^callpath@1.0+debug ^libelf@0.8.11
```

User input: *abstract* spec with some constraints



Abstract, normalised spec with some dependencies.

Concrete spec is fully constrained and can be passed to install.

spec.yaml

```
spec:
- mpileaks:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies:
      adept-utils: kszrtkpbzac3ss2ixcjkcorlaybnptp4
      callpath: bah5f4h4d2n47mgycej2mtrnrivvxy77
      mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
    hash: 33hjjhxi7p6gyzn5ptgyes7sghyprujh
    variants: {}
    version: '1.0'
- adept-utils:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies:
      boost: teesjv7ehpe5ksspjim5dk43a7qnowlq
      mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
    hash: kszrtkpbzac3ss2ixcjkcorlaybnptp4
    variants: {}
    version: 1.0.1
- boost:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies: {}
    hash: teesjv7ehpe5ksspjim5dk43a7qnowlq
    variants: {}
    version: 1.59.0
...
```

Detailed provenance is stored with the installed package

cirrus

epcc | THE UNIVERSITY OF EDINBURGH

Spack packages are simple Python scripts.

- Scripts closely mirror the commands you would usually type on the command line
- Easy to understand what is going on and modify

```
from spack import *

class Dyninst(Package):
    """API for dynamic binary instrumentation."""

    homepage = "https://paradyn.org"

    version('8.2.1', 'abf60b7faabe7a2e', url="http://www.paradyn.org/release8.2/DyninstAPI-8.2.1.tgz")
    version('8.1.2', 'bf03b33375afa66f', url="http://www.paradyn.org/release8.1.2/DyninstAPI-8.1.2.tgz")
    version('8.1.1', 'd1a04e995b7aa709', url="http://www.paradyn.org/release8.1/DyninstAPI-8.1.1.tgz")

    depends_on("libelf")
    depends_on("libdwarf")
    depends_on("boost@1.42:")

    def install(self, spec, prefix):
        libelf = spec['libelf'].prefix
        libdwarf = spec['libdwarf'].prefix

        with working_dir('spack-build', create=True):
            cmake('..',
                  '-DBoost_INCLUDE_DIR=%s' % spec['boost'].prefix.include,
                  '-DBoost_LIBRARY_DIR=%s' % spec['boost'].prefix.lib,
                  '-DBoost_NO_SYSTEM_PATHS=TRUE'
                  *std_cmake_args)
            make()
            make("install")

    @when('@:8.1')
    def install(self, spec, prefix):
        configure("--prefix=" + prefix)
        make()
        make("install")
```

Metadata

Versions and URLs

Dependencies

Patches, variants (not shown)

Commands for installation

Access build config through the *spec* parameter.

cirrus

epcc

THE UNIVERSITY OF EDINBURGH

Demo!

Spack and Easybuild

- Similar tools but underlying philosophy slightly different
  - Easybuild focus on *reproducibility* of binaries – loses some simplicity
  - Spack focus on *simplicty* of use – loses full reproducibility
- Both widely used across HPC centres worldwide
- Both have active developer communities
- Both support a wide range of software
- Suspect that personal preference will decide which one is more useful for you
  - Probably the one that most matches your team mentality…
  - …in our case the simplicity of Spack and particularly the package scripts made our minds up after playing with both of them

Python and R

- Both have their own package dependency chain
  - Can have their own package managers: e.g. conda, pip
- On HPC systems some of modules/packages have to be compiled
  - Usually parallel/performance modules
  - Cannot usually be managed using tools such as conda
- Often the distro Python/R and modules are too out of date
- Spack supports source installation of Python/R and many common scientific modules
  - Can also be linked into controlled environments to ensure people only access the correct packages for that installation

Conclusions

- Spack has proven very useful to us on Cirrus
- Simplified our central installation of packages
  - Easy stuff just installs with single command
  - Some tweaking needed for some packages
  - Some packages still compiled and supported outwith Spack…
  - …but the dependency chain is often mostly met using Spack
- Allows us to focus on more difficult installations
  - Less time spent resolving source compiled dependency chains
- Next steps:
  - Get Spack up and running on ARCHER
  - Explore Python and R support