EasyBuild 2.3 on Cray Linux Environment

(with some CSCS specifics)

Petar Forai

Requirements

- Tcl/C Environment Modules `3.2.10` or >= Lmod 5.6.3
- Lmod can consume Cray module files! (Thanks and Kudos to Robert McLay for this!)
- CLE 5.x with `>=craype-2.0` because we only have XC40 systems for developing EB

Current Status as of 2.3

- Some big apps work and most of the machinery is there
- Marked as 'experimental', ie current interfaces and functionality could be changed in future (waiting for feedback from you guys!)
- Needs —**`experimental`** or environment variable `$EASYBUILD_EXPERIMENTAL` defined to get EB to make experimental parts of it available
- Updated status page at

https://github.com/hpcugent/easybuild/wiki/EasyBuild-on-Cray

EasyBuild approach for toolchains

- Toolchain definitions are composition of ‘full stack’ development environments (compilers, MPI, BLAS/ LAPACK, FFTW, etc.)

- Toolchain code in framework exposes things like `$CC, $CXX,$FTN, $CFLAGS,$CXXFLAGS`, etc. when building

- There’s also API in the toolchains to construct compiler call strings - mostly used in complex blocks

- *EB paradigm is to have reproducible software builds and EB archives that by pinning all versions of all of the TC’s dependencies*

Mapping Cray PrgEnv(s) to EasyBuild

- EB offers toolchains for `PrgEnv-{gnu|intel|cray},` they just load PrgEnv-{gnu|intel|cray} in a specific version (same as toolchain), and fftw (*system defaults!*)- which makes this *similar* to other TCs

- Named `Cray{Gnu|Intel|CCE}-<PrgEnv-version> Ie CrayGNU-5.2.40 uses PrgEnv-gnu/5.2.40`

- Install the resulting configs to have the toolchains available for building.

- If a new version of PrgEnv-gnu is provided on the system, create a new EB config for this and install as you would any other piece of software.

```
easyblock = 'Toolchain'

name = 'CrayGNU'
version = '5.2.40'

homepage = '(none)'
description = """Toolchain using Cray compiler wrapper, using PrgEnv-gnu module."""

toolchain = {'name': 'dummy', 'version': 'dummy'}

dependencies = [
    ('PrgEnv-gnu/' + version, EXTERNAL_MODULE),  # also loads cray-libsci, dmi, atp, etc.
    ('fftw/3.3.4.2', EXTERNAL_MODULE),
]

moduleclass = 'toolchain'
```

Working with EB on Cray Environment setup

- On non-Crays EB builds everything from scratch and provide the appropriate modules
- Please note that we rely on the environment to provide the configuration for MPI. Consider this if you run `module purge && eb foo-CrayGNU-5.2.40.eb`!
- Setup scripts for CSCS just runs `module swap PrgEnv-cray PrgEnv-gnu`
- This way the environment is bootstrapped through system defaults, only compiler swapped. MPI and anything else is there already and system module default versions are loaded

Working with EB on Cray External Modules

- External modules supported from 2.1 via meta data description

- Maps software available on system to software available within EasyBuild, useful to integrate Cray provided software in EB (HDF5, third party libraries, etc.)

- Example https://gist.github.com/pforai/d0f25022b7925792f9f7

- CSCS's global EB setup for metadata is **/apps/common/easybuild/cray_external_modules_metadata.cfg**

Working with EB on Cray External Modules Meta Data

- Normally EasyBuild builds everything and generates modules with known and defined structure
- If piece of software exists in EasyBuild then supplying external data via a central config file is the currently supported approach
- Map module name and version into EB software name space
- Example for Cray HDF5 parallel and serial into EB HDF5 build and NetCDF

```
[cray-netcdf/4.3.2]
name = netCDF,netCDF-Fortran
version = 4.3.2, 4.3.2
prefix = NETCDF_DIR

[cray-hdf5/1.8.13]
name = HDF5
version = 1.8.13
prefix = HDF5_DIR

[cray-hdf5-parallel/1.8.13]
name = HDF5
version = 1.8.13
prefix = HDF5_DIR
```

```
dependencies = [
    ('cray-netcdf/4.3.2', EXTERNAL_MODULE),
    ('cray-hdf5-parallel/1.8.13', EXTERNAL_MODULE),
]
```

Working with EB on Cray CPU targeting

- Cray's wrappers accept different backend code generators options/targets through environment variables

- "Cross compilation" works by loading craype-haswell or craype-ivybridge modules or others if machine has different CPUs in login nodes vs backend nodes via the —**optarch=**<target> flag or by setting **EASYBUILD_OPTARCH=<craype-target>**

- This must be specified otherwise EB will refuse to start

Working with EB on Cray Linking

- EasyBuild controls linking by exposing `$CRAY_LINK_TYPE` into the environment while building, otherwise that variable is undefined and default wrapper behaviour applies

- default is to do static linking

- Controlled through toolchain options in EasyBuild configs

```
easyblock = 'ConfigureMake'

name = 'zlib'
version = '1.2.8'


toolchain = {'name': 'CrayGNU', 'version': '5.2.40'}
toolchainopts = {'pic': True, 'dynamic': True}
```

Working with EB on Cray
Other toolchain options

- Get verbose builds by setting verbose TC option

```
…
toolchain = {'name': 'CrayGNU', 'version': '5.2.40'}
toolchainopts = {'pic': True, 'dynamic': True, ‘verbose: True}

…
```

- Enable fine grained multi threading support in MPICH

```
toolchainopts = {'pic': True, 'dynamic': True, 'mpich-mt': True,}
```

Implementation Effort up until 2.3

- Less than 400 SLOC (including external modules support, generic cray support, GNU/Intel/CCE specifics)

- in easybuild/toolchains/craype.py (base class)

- in easybuild/framework/easyconfig/easyconfig.py

- in easybuild/tools/toolchain.py

Potential Caveats with EB

- EB assumes it can execute tests where it is running (can be disabled)
- Executing and/or benchmarking build artefacts (ie numpy for np.dot() needs to run <1s) after the build for sanity checking
- Linking is controlled by setting `$CRAYPE_LINK_TYPE` (no call out to `-shared` or -`dynamic`)

What works

- Some large apps work as expected, like CP2K, GROMACS, WRF, Python with numpy and scipy linked against libsci and cray-mpich for mpi4py (using cray supplied software as dependencies)

- All login nodes software (git, mercurial, Autotools, cURL, Qt4/5, …) that just relies on dummy compiler or

- Simple to medium simple software for BioInformatics, Math libraries, etc.

What is missing

- Most work has gone into getting PrgEnv-gnu, obviously the other PrgEnv- modules will need some more work, mostly in blocks and of course configs
- Boost and some other major dependencies that needs custom blocks
- Generating pkg-config .pc files at end of build to have the wrappers set proper -I and -L -l for 3rd party software (WIP) (workaround available)
- Cleaning ups configs and blocks - see https://github.com/hpcugent/easybuild-easyconfigs/search?utf8=%E2%9C%93&q=CFLAGS (there are assumptions in (some) places that the compilers running accept GCC flags)

Open Questions

- What to do with libsci? EB normally pins all the version of all dependencies. PrgEnv-foo just loads system default for libsci (unlike other some deps in PrgEnv modules)
- Same for GCC!
- On one machine this means that CrayGNU-5.2.25 contains ie libsci 13.0.3 and on another it is 13.0.1
- Current assumption is ‘site knows what it’s doing and is checking for compatibility’
- Should it be pin and if so how to parametrise?
- Provide feedback for EB built software in terms of performance vs your hand compiled stacks (like for large CP2K runs, GROMACs, etc.)?

Open Questions II

- how to correctly deal with MPI: which modules should be loaded? EB Cray toolchain should load an MPI module?Current assumption is ‘site knows what it’s doing and is checking for compatibility’

- Do you “unwrap" the drivers? Do you “unwrap" the drivers? Ie load PrgEnv-gnu/x.y.z but set $CC to gcc?

- Want to approach Cray Inc. at Super Computing 2015 and present this to them. CSC (Olli Pekka Lehto) was approaching them before about EasyBuild.