Custom Integration and CrayPE

Sean Byland, Cray Inc.
seanb@cray.com

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

Agenda

- Background information
- CrayPE 2 design goals
- CrayPE 2 implementation
- Simple product integration
- Product integration with multiple builds
- Questions

CrayPE Compiling Drivers

- **Command line interface for compiling and linking**
  - cc – C compilations
  - ftn – Fortran compilation
  - CC – C++ compilation
- **Creates compile/link line**
- **Resolves link dependencies**

Craype 2.0 Design Goals

- The front-end needs to be product- and version-agnostic and to require no explicit version tracking.
- A system that could be learned and used easily.
- A system that would easily integrate into both Cray and standard GNU/Linux systems.
- A compiling front-end that works programmatically.
- Increased performance
- A system that would allow the library stack to be used independently and treated as a separate abstraction layer.
- Improvements in error detecting and reporting.

Product Agnostic Drivers

- Product integration methods should be made generic whenever possible
- Driver updates should never be needed for product support
- All needed driver actions should be configured by files within a product

Link/Compile Line Complexity Reduction

- Duplicate flags should be removed
- Libraries should be ordered correctly
  - This eliminates the need for duplicate libraries and some other linker commands
- Linker library lookup should be relied on when possible
- Dynamic libraries shouldn't explicitly link their dependencies
- Users should be able to isolate libraries that they that are using

Custom Integration

- Anyone should be able to build and integrate their own product.
- Standard integrations methods should be supported whenever possible
- Anyone should be able to integrate a product of any supported complexity
- User-integrated products and Cray products should share the same driver interface (for seamless integration and lowered maintenance)

Add Flag/Library Bypassing (library stack/compiler front-end abstraction)

- **Users should be able to conveniently:**
  - View generated compiling flags
  - View generated linking flags
  - View flags needed by one product and it's dependencies
  - Prevent the driver from inserting flags
  - Pass custom flags to the compiler front-end

Remove Module Interdependencies

- Modules loading modules can leave an inconsistent state
- Leaves users with modules they didn't load
- Will result in every library of a parent module being linked in when only a subset is needed (over-linking).
- Libraries for all products should be accessible with or without the modules being loaded

Our Solution

1. Standard conventions for products to define their own compatibly

2. Modules to set up the environment

3. pkg-config files to define compiling flags/linking flags and dependencies

What CrayPE Provides

- **Standard conventions for complexity beyond pkg-config's scope including:**
  - Multiple product (library) versions
  - Multiple programming environments
  - Multiple compiler version
    - Including compiler compatibility matching
  - Products optimized for multiple architectures
    - Include CPU targeting compatibility fallback

pkg-config

- **A unified interface for querying installed libraries**
- **Provides details for compiling and linking a program to a library:**
  - Parameters for compilers
  - Parameters for linkers
  - Library dependencies
- **Library information contained in metadata *.pc files**
  - Located in a “pkgconfig” directory in the library directory

Why pkg-config ?

- **Pkg-config automatically:**
  - Resolves dependencies
  - Checks versions
  - Handles statics/dynamic library ordering
  - Removes duplicate flags
- **Many third-party products already have pc files**
- **Pkg-config is supported by some third-party builds**
- **pkg-config is a standard utility and therefore is familiar to the Linux community**

pc file example

```
prefix=/usr/local
includedir=${prefix}/include
libdir=${exec_prefix}/lib

Name: foo
Description: The foo library
Version: 1.0.0
Cflags: -I${includedir}/foo
Libs: -L${libdir} –lfoo
Requires: bar >= 1.0.0
```

Basic CrayPE 2.0 Product Integration Conventions

- **Used by pkg-config**
  - PKG_CONFIG_PATH – An environment variable containing paths that pkg-config searches for pc files

- **Used by the driver:**
  - PE_PKGCONFIG_PRODUCTS – List of product's that have product-specific environment variables
  - ${PRODUCT_NAME}_PKGCONFIG_LIBS - A list of top-level pkg-config files (applies to all drivers, cc, CC and ftn)
  - ${PRODUCT_NAME}_C_PKGCONFIG_LIBS - A colon-separated list of top-level C specific pkg-config files
  - ${PRODUCT_NAME}_FORTRAN_PKGCONFIG_LIBS - A colon-separated list of top-level fortran specific pkg-config files
  - ${PRODUCT_NAME}_CXX_PKGCONFIG_LIBS - A colon-separated list of top-level C++ specific pkg-config files

Product Integration with a single library directory

1. Make pc files with the appropriate linking and compile flags and dependencies

2. In the modulefile add the directory path containing pc files to PKG_CONFIG_PATH

3. Add the pc files to the *_PKGCONFIG_LIBS list for the programming languages needed by the product

Example

```
seanb@pe03:~> module show atp
----------- -------------------------------------------------
/opt/cray/modulefiles/atp/1.6.4.1:

prepend-path     PATH /opt/cray/atp/1.6.4.1/bin
prepend-path     MANPATH /opt/cray/atp/1.6.4.1/man
append-path      PKG_CONFIG_PATH /opt/cray/atp/1.6.4.1/lib/pkgconfig
prepend-path     PE_PKGCONFIG_PRODUCTS atp
prepend-path     atp_PKGCONFIG_LIBS AtpSigHandler
prepend-path     atp_PKGCONFIG_LIBS AtpSigHCommData
-------------------------------------------------------------------
```

```
seanb@pe03:~> pkg-config --libs --cflags AtpSigHandler,AtpSigHCommData
 -Wl,--undefined=_ATP_Data_Globals -Wl,--undefined=__atpHandlerInstall -L/opt/cray/atp/1.6.4.1/lib -lAtpSigHandler -
lAtpSigHCommData
```

Product integration with PrgEnv specific libraries

1. Make pc files with the appropriate linking and compile flags and dependencies for each PrgEnv

2. Add the pc files to the *_PKGCONFIG_LIBS list that should be referenced by the driver

3. For each PrgEnv append the PrgEnv specific path to:
   - PE_${PRGENV}_FIXED_PKGCONFIG_PATH

Example

```
:~> module show hdf5
prepend-path    PE_PKGCONFIG_PRODUCTS hdf5
prepend-path    hdf5_PKGCONFIG_LIBS hdf5_hl:hdf5
prepend-path    hdf5_CPP_PKGCONFIG_LIBS hdf5_cpp:hdf5_hl_cpp
prepend-path    hdf5_FORTRAN_PKGCONFIG_LIBS hdf5hl_fortran:hdf5_fortran
setenv          hdf5_FIXED_PRGENV GNU INTEL PGI CRAY
prepend-path    PE_GNU_FIXED_PKGCONFIG_PATH /tmp/hdf5/1.8.11/GNU/lib/pkgconfig
prepend-path    PE_INTEL_FIXED_PKGCONFIG_PATH /tmp/hdf5/1.8.11/INTEL/lib/
pkgconfig
prepend-path    PE_PGI_FIXED_PKGCONFIG_PATH /tmp/hdf5/1.8.11/PGI/lib/pkgconfig
prepend-path    PE_CRAY_FIXED_PKGCONFIG_PATH /tmp/hdf5/1.8.11/CRAY/lib/pkgconfig
prepend-path    LD_LIBRARY_PATH /tmp/hdf5/1.8.11/CRAY/lib
prepend-path    PATH /tmp/hdf5/1.8.11/bin
```

Products integration with multiple compiler or CPU built libraries

- Make pc files with the appropriate linking and compile flags and dependencies for each build
- Add the pc files to the *_PKGCONFIG_LIBS list that should be referenced by the driver
- Define:
  - Add the product to PE_PKGCONFIG_PRODUCTS
- Set "PE_${NAME}_VOLATILE_PKGCONFIG_PATH" with keywords where the path can change e.g.:
  - /opt/cray/product/version/@PRGENV@/@PE_PRODUCT_DIR@/lib/pkgconfig
- Define compiler versions and CPUs e.g:
  - PE_${PRODUCT}_DIR_${PRGENV}_${CPU} "directory_name"

Volatile Paths

- **${PRODUCT_NAME}_DIR** - can be used for defining conditionally name directories
- **${PRODUCT_NAME}_PKGCONFIG_VARIABLES** - can be used for conditionally defining variable's for pkg-config
- **${PRODUCT_NAME}_GENCOMPILERS** – can be used to indicate that the driver should select from the given list of compilers
- **${PRODUCT_NAME}_TARGETS** – can be used to indicate the CPUs that the product was built with and is compatible with.

Scientific Libraries Example

```
seanb@pe03:~> module show cray-libsci
-------------------------------------------------------------------
/opt/cray/modulefiles/cray-libsci/12.1.01:
conflict                xt-libsci
setenv                  CRAY_LIBSCI_VERSION 12.1.01
setenv                  CRAY_LIBSCI_PREFIX_DIR /opt/cray/libsci/12.1.01/CRAY/81/ivybridge
prepend-path            CRAY_LD_LIBRARY_PATH /opt/cray/libsci/12.1.01/CRAY/81/ivybridge/lib
prepend-path            MANPATH /opt/cray/libsci/12.1.01/man
prepend-path            PE_PKGCONFIG_PRODUCTS PE_LIBSCI
setenv                  PE_LIBSCI_GENCOMPILERS_CRAY_x86_64 81
setenv                  PE_LIBSCI_GENCOMPILERS_CRAY_interlagos 81
setenv                  PE_LIBSCI_GENCOMPILERS_CRAY_sandybridge 81
setenv                  PE_LIBSCI_GENCOMPILERS_GNU_x86_64 47 48
setenv                  PE_LIBSCI_GENCOMPILERS_GNU_interlagos 47 48
setenv                  PE_LIBSCI_GENCOMPILERS_GNU_sandybridge 47 48
setenv                  PE_LIBSCI_GENCOMPILERS_INTEL_x86_64 130
setenv                  PE_LIBSCI_GENCOMPILERS_INTEL_interlagos 130
setenv                  PE_LIBSCI_GENCOMPILERS_INTEL_sandybridge 130
setenv                  PE_LIBSCI_GENCOMPILERS_PGI_x86_64 121
setenv                  PE_LIBSCI_GENCOMPILERS_PGI_interlagos 121
setenv                  PE_LIBSCI_VOLATILE_PKGCONFIG_PATH /opt/cray/libsci/12.1.01/@PRGENV@/@PE_LIBSCI_GENCOMPILERS@/
@PE_LIBSCI_TARGET@/lib/pkgconfig
prepend-path            PE_LIBSCI_REQUIRED_PRODUCTS PE_MPICH:PE_FFTW
prepend-path            PE_PKGCONFIG_LIBS sci_mp:sci
prepend-path            PE_CXX_PKGCONFIG_LIBS scicpp
```

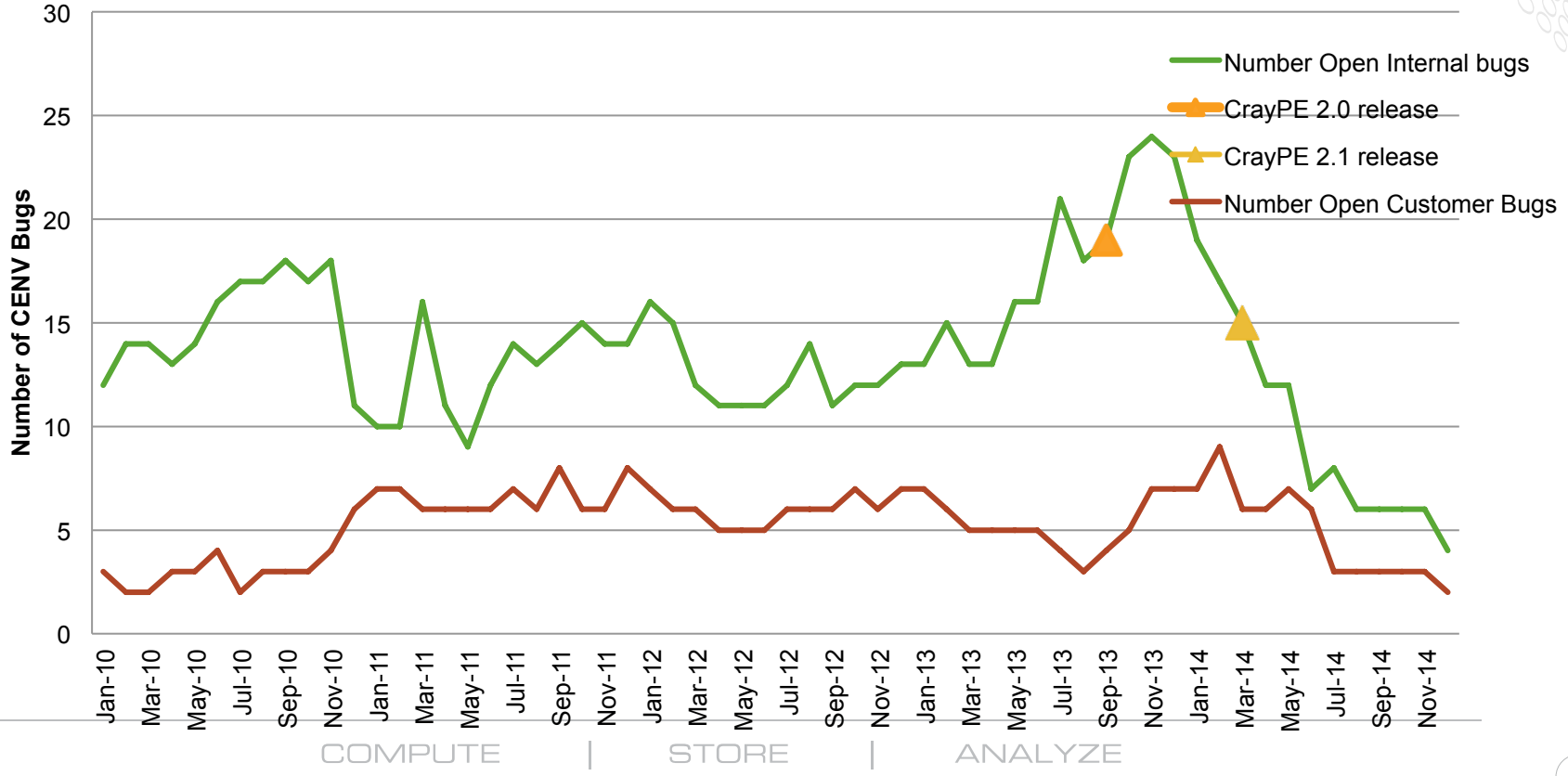
Craypkg-gen

- **Tool for simplifying product integration**
- **Creates *.pc files:**
  - craypkg-gen –p *prefix/product-name/product-version*
- **Creates modulefiles**
  - craypkg-gen –m *prefix/product-name/product-version*

Miscellaneous

- **To prevent the necessity for module interdependencies:**
  - pc files list library dependencies
  - A script for a default product can be placed in:
    /opt/cray/admin-pe/pkgconfig_default_files/
    to set up a PKG_CONFIG_PATH for any default product (so pkg-config can find libraries for all default products
- **To create a better layer of abstraction:**
  - Libraries, compiler flags, pc files or a processed pkg-config path can viewed:
    cc --cray-print-opts=libs|cflags|pkg_config_path|pcfiles|all (all being default)
  - Link/compile creation can be bypassed with:
    cc --cray-bypass-pkgconfig
- **To increase error message usefulness:**
  - Using pkg-config's built in version checking:
  - Detecting & printing compiler/library version mismatches

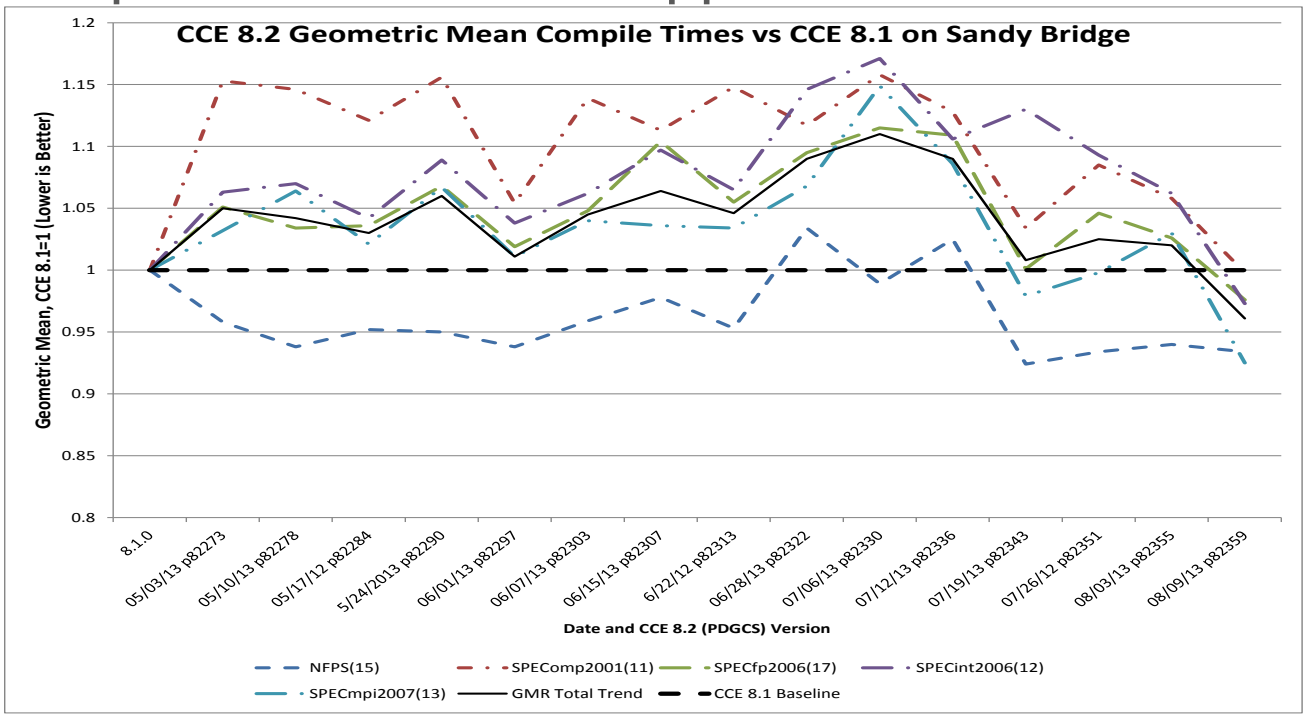
CENV Bugs Over Time



COMPUTE | STORE | ANALYZE

Performance

- Link/compile line creation dropped from 200ms to 15ms



Questions ?

Sean Byland
seanb@cray.com

Additional resources: man craype_api, man craypkg-gen, cc –help
craypkg-gen examples: /opt/cray/craypkg-gen/default/doc