Configuring and Customing the Cray Programming Environment on CLE 6.0 Systems

Geir Johansen
Cray Inc.
geir@cray.com

***Abstract*—The Cray CLE 6.0 system will provide a new installation and configuration model for software on Cray XC systems. This paper will focus on how these new processes affect the installation and configuration of the Cray Programming Environment. Topics include an overview of the new installation method, the configuration of login shell scripts to load modulefiles, and new features of the PrgEnv modulefile. The paper will also discuss current processes for porting programming tools and customizing the site's programming environment.**

***Keywords*—Cray Programming Environment**

I. INTRODUCTION

The goal of this paper is to outline the changes to the Cray Programming Environment necessitated by the new CLE 6.0 installation model and to provide current information on customization procedures for the programming environment. The intended audience is site administrators and system consultants responsible for the Cray XC programming environment. While CLE 6.0 introduces new installation methods, the changes to the end user are minimal. Existing Cray customer users should not have to modify how they use the Cray Programming Environment to build, analyze or debug their applications.

The Cray Programming Environment provides a wide range of compilers, libraries and tools. However, customer sites may require additional third party programming environment software. Often third party programming solutions come in the form of open-source software that must be built on the Cray system, so guidelines for porting software to the Cray XC will be discussed. In order to improve integration of third party software with the Cray Programming Environment, the procedures for creating modulefiles and using pkgconfig to interface third party libraries with the CrayPE compiler drivers will be outlined.

II. CRAY PROGRAMMING ENVIRONMENT INSTALLATION IN CLE 6.0

A. Overview

In CLE 6.0, Cray introduces new tools for installation and configuration of Cray XC systems. The Cray Programming Environment (PE) is mainly affected by this change in that the PE software is no longer installed in to a shared-root filesystem, but rather into a PE image root. IMPS , the Image Management and Provisioning System, leverages the use of rpm and zypper instead of the CLE 5 Cray installation method.

The installation of the Programming Environment software into a PE image root is performed on the system's SMW. The PE image root is then pushed to the boot node so that it can be mounted by a group of DVS servers and then mounted to the system's login and compute nodes. One of the advantages of the PE image root model is that the installation is designed to be system and hardware agnostic, so the same PE image root can also be used for other systems, such as eLogin systems or another Cray XC. A feature of IMPS images is that they are easily "cloned". This ability allows the site to test new PE releases, and also makes reverting back to previous PE releases easier.

B. Change in Installation Directory

One notable change in CLE 6.0 is that the Cray Programming Environment software originating from Cray will now be installed in the directory `/opt/cray/pe`, as oppose to `/opt/cray`. This change was done to separate the Cray PE software from the other Cray software components that are installed into other IMPS images under the `/opt/cray` directory. Third party software, such as the Intel compiler and TotalView, will continue to be installed in their respective directories under the /opt directory (i.e `/opt/intel`, `/opt/totalview`).

C. Installation of Cray issued Programming Environment Software

The `craype-installer` remains the tool to install Cray issued Programming Environment software and its interface has essentially not changed for CLE 6.0. The same Cray Developer Toolkit (CDT) ISO file is used to install the Cray Programming Environment software on either a CLE 5 or CLE 6 system. For CLE 6.0 systems, the `install-cdt.yaml` configuration file is edited by the administrator so that the IMAGE_DIRECTORIES variable points to the PE root image. The `craype-installer` is then executed on the SMW in a similar manner as performed under CLE 5.

A feature being added to the `craype-installer` during the CLE 6.0 timeframe is the ability to remove older versions of CDT. The administrator will be able to specify a date and the tool will be able to remove Cray Software Programming environment software components installed before that date unless it satisfies a dependency for a more currently installed component.

D. Installation of Third Party Programming Environment Software

The installation of third party PE software is performed on the SMW by using chroot to access the PE root image. For example, the install procedure for the PGI 15.10.0 compiler is to copy the PGI rpm to the PE image and then execute rpm under chroot:

```
  smw#  cp pgi-15.10.0-*.rpm  <PE root image location>/var/tmp
  smw#  chroot <PE root image location> rpm -ivh /var/tmp/pgi-15.10.0-*.rpm
```

As previously mentioned, an advantage of the new installation methods is that they are system agnostic, however, some vendors installation methods may perform checks on the system being installed and react accordingly. Cray will provide steps on how to resolve this issue and will work with vendors in avoiding this situation. As an example, Appendix A shows the process to install the Intel compiler and create a modulefile for a CLE 6.0 system.

E. Pushing updated PE image root to nodes

Once the installation of programming environment software has been completed, the PE image root is then pushed to the boot node using the following command:

```
 smw# image push --dest boot <PE root image location>
```

To bind the updated PE image root to the system nodes, the ansible-playbook command is executed on the boot node. For example, the following command line binds the updated PE image root on the compute nodes:

```
 SDB# pcmd -r -n ALL_COMPUTE "ansible-playbook /etc/ansible/cray_image_binding.yaml"
```

When this command is invoked, the compute nodes will start the `/opt/cray/pe_postmount_callback.sh` script to bind mount directories from the PE root image to the compute nodes' root.

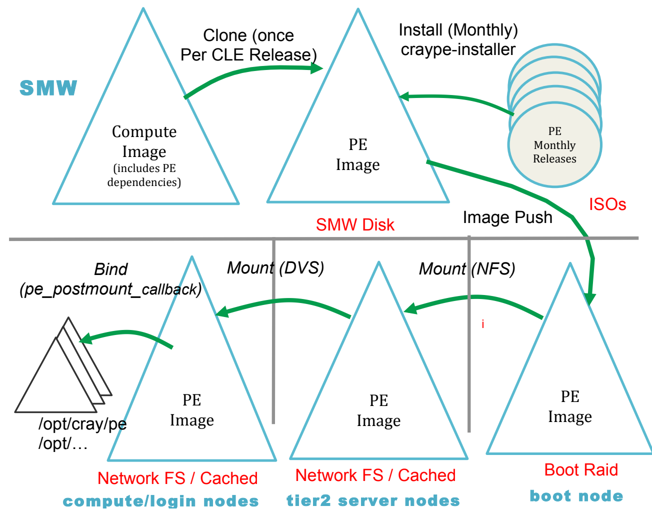

Figure 1. PE Management Diagram

A summary diagram of the Cray Programming Environment installation and system management process is shown in figure 1. The top half of the diagram shows the processes executed on the SMW, while the bottom half represents the processes inititiated by the ansible-playbook command on the boot node.

III. Configuring CLE 6.0 Shell Initialization Files to Load Programming Environment Modulefiles

Typically a Cray XC system will set up the shell's initialization files to load a set of modules when a user logs into the machine. For example, a customer site could choose to load the Cray compiler environment (PrgEnv-cray) when the user logs in to the system, while another site may choose to have the Intel compile environment (PrgEnv-intel) loaded. For CLE 6.0, the administrator needs to update the file `/etc/opt/cray/pe/admin-pe/site-config` to specify the modulefiles that will be loaded when a user logs into the system. This file supports the update of the initialization file scripts for the following shells: bash (sh), csh, tcsh, zsh, ksh (lksh, mksh, pdksh).

The `/etc/opt/cray/pe/admin-pe/site-config` file contains a list of modules commands to be performed by the shell initialization file. Here is an example:

```
$ cat /etc/opt/cray/pe/admin-pe/site-config
# Defines the Programming Environment modules
# that will automatically be loaded.
  module add PrgEnv-cray
  module add atp
  module add cray-mpich
  module add craype-ivybridge
  module add perftools-base
  module add forge
  module add slurm
$
```

In CLE 6.0, the system's shell initialization files are updated by `/opt/cray/pe/bin/setup_shell_rcs.sh`. This script is automatically executed during the execution of the ansible-playbook command.

IV. PrgEnv modulefiles in CLE 6.0

A. PrgEnv modulefiles released in CDT package

The PrgEnv modulefiles (`PrgEnv-cray, PrgEnv-gnu, PrgEnv-intel, PrgEnv-pgi`) will no longer be released with CLE 6.0 software, but will be release with the CDT package. CLE 5.2 systems will continue to use the PrgEnv-* modulefiles released in the CLE 5.2 software.The functionality of the PrgEnv modulefiles remains the same, but there have been a couple of new features added.

B. SITE_MODULE_NAMES Environment Variable

The swapping of a PrgEnv module file preserves the versions of the Cray Programming Environment libraries that are loaded. For example, given the situation of `PrgEnv-cray` and `cray-mpich/7.2.6` modulefiles are loaded, then a "`module swap PrgEnv-cray PrgEnv-intel`" will result in the `cray-mpich/7.2.6` module to continued to be loaded. What actually occurs during the swap of PrgEnv is that the process notes which version of cray-mpich is loaded, then proceeds to unload it, load the new PrgEnv modulefile, and then re-load appropriate version of cray-mpich. This functionality is now extended to local library modulefiles by the use of the SITE_MODULE_NAMES environment variable.

As an example, a site that has a custom built version of the Boost library with a modulefile named '`boost`' can set `SITE_MODULE_NAMES=boost`. When the PrgEnv module file is swapped, the `boost` modulefile will be reloaded to account for any changes in the compiler being used.

```
$ module load cray-netcdf cray-tpsl boost
$ export SITE_MODULE_NAMES=boost
$ module show PrgEnv-gnu 2>&1 | grep swap
module          swap craype/2.5.4
module          swap cray-mpich cray-mpich/7.3.3
module          swap cray-hdf5 cray-hdf5/1.8.16
module          swap cray-tpsl cray-tpsl/16.03.1
module          swap boost boost/1.59.0
$
```

C. cdt modulefiles

CLE 6.0 introduces the `cdt` modulefiles that will instruct the modules command to use software components from a specific Cray Programming Environment CDT release when loading modulefiles. The `cdt` modulefiles effectively changes the default version of a software component modulefile to be the version associated with that specific CDT release. For example, a "`module load cdt/16.1`" and "`module load cray-mpich`" will result in a `cray-mpich/7.3.1`, the version released with CDT 16.1, being loaded. A subsequent "`module swap cdt/16.1 cdt/16.3`" and "`module swap cray-mpich cray-mpich`" will result in `cray-mpich/7.3.2`, the version released with CDT 16.1, being loaded.

D. Module command substring search

In the modules 3.2.10.4 release, the `module avail` command will support the '`-S` *<string>*' option, so that the command will search for any modulefile name that contains *<string>*:

```
$ module avail trilinos
$ module avail -S trilinos

----------- /opt/cray/modulefiles -----------
cray-trilinos/12.2.1.0(default)
$
```

V. Porting Software to Cray Programming Environment

A brief discussion of porting code is provided along with current developments of the Autoconf and CMake utilities.

A. General Guidelines

Many open-source third party programming environment software packages are available for Linux and can be used on Cray systems. Porting open-source to Cray systems is often not difficult, but there are a few common issues that frequently arise when porting software.

1) CPU targeting. The CPU type of the node that is building the software is very often different than the compute node executing the software. Build processes, particular the Autoconf `configure` script, will create and execute small programs on the build system to check the system configuration. To resolve this, it is often necessary to use the `craype-`*<cpu target>* modulefile that matches the CPU of the build system. Once the `configure` script has been executed to create a `makefile`, the user may be able to load the `craype-`*<cpu target>* that matches the compute nodes and then execute the `makefile` to build the software using the appropriate CPU target.

While Cray does not officially support executing the Cray Programming Environment compilers on compute nodes, a user can often eliminate CPU differences between login and compute nodes by building the program using a compute node.

2) Static versus dynamic linking. While many Cray systems default to static linking, the reality is that the default use of static linking in Linux software is rare. Given this situation, it is not uncommon for users to encounter problems

when building a static version of a software package. A quick way to workaround this issue is to build the software on the Cray system using dynamic linking. This can be done by setting the environment variable CRAYPE_LINK_TYPE=dynamic.

3) Use of PrgEnv CrayPE compiler drivers. When using autoconf and CMake utilities to build applications, the common procedure is to set `CC=cc`, `FTN=ftn`, and `CXX=CC`. This will inform the build tools that the CrayPE compiler drivers should be used. Using the compiler drivers will cause the loaded Cray Programming Environment libraries, such as cray-libsci and cray-mpich, to be linked with the program. If the program to be ported does not need any Cray Programming Environment libraries and is not using the Cray compiler, then the CrayPE compiler drivers do not need to be used. Appendix B demonstrates the CMake utility being built without using the CrayPE compiler drivers.

B. Autoconf

GNU Autoconf is a popular tool used for building and installing open-sourced software. The tool has had some CLE related bugs in past versions, but these problems have been resolved in version 2.63, which is the version running on CLE 5.2 (CLE 6.0 has version 2.69).

The Autoconf generated configure scripts performs various tests to check the environment of the system. These tests can include building many small programs that are executed. Given that static linking is the default of many Cray systems and that several large Cray networking libraries are included on the link line, the execution time of the configure script can be much longer than on other systems. One way to resolve this issue is to use the `configure` '`-C`' option to create a `config.cache` file. After this file has been created, subsequent executions using the '`—config-cache`' option will run significantly faster.

C. CMake

Cmake is a popular tool used for building and packaging software. In March 2016, Kitware Inc. released CMake 3.5.0 with a new platform file that increases compatibility with CLE. CMake 3.5.0 is now able to use correct build settings when the Cray Programming Environment PrgEnv modulefiles are loaded by using the following CMake option: `-DCMAKE_SYSTEM_NAME=CrayLinuxEnvironment.`

Appendix B shows the procedures to port CMake 3.5.1 to a Cray XC system. More information about CMake's compatibility with CLE can be found at the following website:

https://cmake.org/cmake/help/v3.5/manual/cmake-toolchains.7.html - cross-compiling-for-the-cray-linux-environment

VI. Using Craypkg-gen to Create Modulefiles and Interface with CrayPE

The Craypkg-gen tool is used to generate a modulefile so that third party programming software can be used in a similar manner as the components of the Cray Programming Environment. For programming libraries, Craypkg-gen has the ability to generate pkg-config (`*.pc`) files that are used to integrate the libraries with the CrayPE compiler drivers. Finally, the Craypkg-gen utility has the functionality to create an RPM file of the software, so that an administrator can install the RPM on a system.

The following example shows the use of Craypkg-gen to create pkg-config files, a modulefile and an RPM for the Boost library built by g++. The following environment variables represent the installation directories:

```
 USER_INSTALL_DIR=<user's install directory
for PE software>
 CLE_INSTALL_DIR=<system wide install
directory for PE software>
```

As an example, the directory `/opt/local` could be used for $CLE_INSTALL_DIR.

A. Creation of pkgconfig files

The pkgconfig/*.pc files contains dependency information for the libraries of the software component. The dependency information includes the header file directories that must be specified on the compilation command line and any specified compiler options that are needed. The pkgconfig file also provides the link option to indicate the dependent libraries that are needed. The Craypkg-gen '`-p`' option is used to create the pkgconfig files:

```
$ module load craypkg-gen
$ craypkg-gen —p $USER_INSTALL_DIR/boost/1.59.0/GNU
```

B. Creation of modulefile

The Craypkg-gen utility can be used to create a modulefile for the software package. The main functionality of a modulefile is to update environment variables, such as update `$PATH` to include the software package's executables or `$MANPATH` to point to any included man pages. For libraries, the modulefile sets the appropriate pkgconfig environment variables so that the CrayPE compiler drivers will include the correct options to find the software header files and libraries. The Craypkg-gen '`-m`' option will create the modulefile:

```
$ craypkg-gen —m $USER_INSTALL_DIR/boost/1.59.0
```

The '`-m`' option also creates a set_default script that will make the associated modulefile the default version that is used by the module command. For this example, the following set_default script was created:

```
$USER_INSTALL/admin-pe/set_default_craypkg/
set_default_boost_1.59.0
```

Executing the generated set_default script will result in a "`module load boost`" loading the `boost/1.59.0` modulefile.

C. Building an RPM

Craypkg-gen provides the functionality to create an RPM. A user may want to create a RPM to give to a system administrator, so that the software package that is installed in the user's local directory can now be installed in a system location for all users. The '`-r`' option is used to create an RPM with the '`—prefix`' option used to specify the final install location:

```
$ craypkg-gen –r$USER_INSTALL_DIR/boost/1.59.0
--prefix=$CLE_INSTALL_DIR
```

The name of the generated RPM will be prefixed by "`craypkg-`". For this example, the name will be: `craypkg-boost-1.59.0-0.x86_64.rpm`.

Appendix C shows information on porting the Boost library for the Cray compiler including the creation of the pkgconfig files, modulefile, and a RPM. Appendix D demonstrates the porting of the Python MPI library mpi4py along with the creation and use of a modulefile for the library.

VII. Future Opportunities

A. Craype-installer

Further work is planned to increase the granularity of installation and uninstallation of the Craype-installer. For example, users will be able to select specific PE software components to install or uninstall of the Cray Developer Toolkit ISO.

B. PrgEnv modulefile

An enhancement to develop a snapshot/restore feature to create user specified functionality of a PrgEnv modulefile is being planned. For example, if a user has a specific versions of libraries that are needed for an application, then the feature will load these libraries when the PrgEnv modulefile is loaded.

VIII. Conclusion

The system installation and configuration methodology introduced in CLE 6.0 is significantly different than previous versions of CLE. The following functionality of the Cray Programming Environment were affected:

- Additional information required for the Craype-installer configuration file
- Change in the installation directory of the Cray Programming Environment
- Configuring shell login 'rc' init scripts to load Cray Programming Environment modulefiles
- Minor changes and enhancements to the PrgEnv modulefiles

Recent enhancements to help port programming software tools and integrate them in the Cray Programming Environment include:

- CMake 3.5.0 feature to provide better compatibility with the Cray Programming Environment.
- Creation of modulefiles for third party software using the Craypkg-gen tool.
- Craypkg-gen's ability to create pkgconfig files that allow libraries to be automatically linked by the CrayPE compiler drivers.

Examples of porting programming environment software have been provided in the appendices.

Acknowledgment

The author thanks the Cray software development team for valuable technical information and consultation. Specifically, I like to thank Ryan Ward, Sean Palmer, Sean Byland, Justin Cook, and Harold Longley.

References

[1] S. Byland and R. Ward, "Custom Product Integration and the Cray Programming Environment," Cray User Group proceedings, May 2015.

[2] H. Longley, "Cray Management System for XC Systems with SMW 8.0/CLE 6.0," Cray Inc. internal presentation, March 2016.

[3] "CMake 3.5.1 Documentation," cmake.org, March 2016

.

APPENDIX A

CLE 6.0 Installation of Intel Compiler

In CLE 6.0 the Programming Environment software is installed on to a PE image root. After downloading the Intel software package, the first step is to copy it to the PE image root:

```
smw # export PECOMPUTE=/var/opt/cray/imps/image_roots/<pe_compute_cle_6.0_imagename>
smw # cp parallel_studio_xe_2016_update1.tgz $PECOMPUTE/var/tmp
```

If the site has an existing license file then this information should also be copied to PE image:

```
smw # cp <Intel license file> $PECOMPUTE/var/tmp
```

Beginning with version 16, the Intel Compiler installation procedure performs checks that require the access to the `/dev` filesystem of a running system. In order to work around this issue the PE image's `/dev` can be bind mounted to the /dev of the SMW system. Also, the compiler license from Intel must be a floating license, as opposed to a node-locked license.:

```
smw # mount --bind /dev $PECOMPUTE/dev
```

Next, perform a chroot to PE image:

```
smw # chroot $PECOMPUTE
```

Unzip and untar the Intel Compiler package.

```
smw # cd /var/tmp
smw # tar xzvf parallel_studio_xe_2016_update1.tgz
```

If the administrator has an existing license for the compiler then it should be installed at this time. For example, if the site has license file that is installed into `/opt/intel/licenses` directory, then this file should be copied over at this time.

The Intel Compiler install script is interactive and requires the administrator to respond to prompts. For installations on Cray systems, Intel suggests the use of '`--ignore-cpu`' and '`--ignore-signature`' options:

```
smw # cd parallel_studio_xe_2016_update1/
smw # ./install.sh  --ignore-cpu --ignore-signature
```

The following messages printed out by the installer can be ignored:

```
df: Warning: cannot read table of mounted file systems: No such file or directory
mount: failed to read mtab: No such file or directory
```

The installation process consists of seven steps.

Step 1 Welcome
The administrator can ignore any prerequisite warnings that occur during this step.

Step 2 License Agreement
Press <space> to read through the license agreement and type '`accept`' when completed.

Step 3 of 7 | Activation
If the license has been installed, the install script should have detected the license and will prompt the user on whether this license

should be used. If a license has not been installed, there are options to install the license at this time or the administrator can select the following option:

I want to evaluate Intel(R) Parallel Studio XE 2016 Update 1 Cluster Edition for Linux or activate later*

Step 4 Intel Software Improvement Program

The default selection is to participate in the Intel Software Improvement Program. This program requires access to the internet, so for sites where compilation is performed on nodes that are not connected to the internet will need to select the 'No' option. Cray suggests selecting 'No' for this option.

Step 5 Options

The first prompt of this step presents a question involving "Configure Cluster Installation". For this prompt, the administrator should select the default option of "Finish configuring installation target".

Subsequent prompts in 'Step 5' includes customization options for selecting components of the Intel compiler package to install. Cray recommends to select the installation of all of the Intel(R) Math Kernel Library (MKL) libraries. It is important that the administrator does not unselect the installation of the poorly named MKL Cluster Support libraries, as this component contains the important Scalapack library. Unless the administrator is positive that a component of the Intel Compiler package will not be used, it is suggested to not remove any components that are installed by default.

Step 6 Installation

This step performs the installation and outputs the progress of the installation.

Step 7 Complete

Outputs final report of the installation.

Creation of the Intel Compiler Modulefile

After installing the Intel Compiler, the administrator needs to create a modulefile for the newly installed release. The following steps will create the modulefile for Intel Compiler 16.0.1.150 release:

```
smw # module load craypkg-gen
smw # craypkg-gen -m /opt/intel/compilers_and_libraries_2016.1.150
Generating a modulefile
/opt/intel/compilers_and_libraries_2016.1.150/linux/bin/compilervars.sh intel64
To make this module accessible please type: module use /opt/modulefiles
smw #
```

The above procedure will create the module file `/opt/modulefiles/intel/16.0.1.150`. Information about creating a modulefile for the Intel compiler can be found in the file:

```
/opt/cray/pe/craypkg-gen/default/intel_example.txt
```

APPENDIX B

Building and Installing CMake for Cray XC

The CMake utility is used by numerous applications as part of their build processing. Beginning with version 3.5.0, the CMake software has modifications to work with in the Cray Programming Environment. The cmake.org website contains the following information:

> Cross compiling for compute nodes in the Cray Linux Environment can be done without needing a separate toolchain file. Specifying `-DCMAKE_SYSTEM_NAME=CrayLinuxEnvironment` on the CMake command line will ensure that the appropriate build settings and search paths are configured. The platform will pull its configuration from the current environment variables and will configure a project to use the compiler wrappers from the Cray Programming Environment's `PrgEnv-*` modules if present and loaded.

More information about using CMake on a Cray XC can be found here:

https://cmake.org/cmake/help/v3.5/manual/cmake-toolchains.7.html - cross-compiling-for-the-cray-linux-environment

The press release announcing CMake's compatibility with Cray Linux Environment can be found here:

http://www.prweb.com/releases/2016/03/prweb13256288.htm

The following demonstrates porting CMake 3.5.1 that was downloaded from cmake.org. Note that the CC and CXX environment variables were intentionally not set, so the build process defaults to directly calling the gcc and g++ compilers.

```
$ tar xzvf cmake-3.5.1.tar.gz
$ cd cmake-3.5.1
$ module load gcc  # load a current version of GCC
$ export PE_INSTALL=<Installation directory for PE tools>
$ mkdir -p $PE_INSTALL/cmake/3.5.1
$ ./configure --prefix=$PE_INSTALL/cmake/3.5.1
$ gmake install
```

A modulefile can be created for CMake by using the Craypkg-gen command

```
$ module load craypkg-gen
$ craypkg-gen -m $PE_INSTALL/cmake/3.5.1
```

The resulting modulefile is placed in the file `$PE_INSTALL/modulefiles/cmake/3.5.0`. It is recommended to edit the modulefile and change the line "`append-path PATH ....`" to "`prepend-path PATH  .....`". This will ensure that if `/usr/bin/cmake` exists, then it will not be used when the `cmake` modulefile is loaded. Here is an example of using the `cmake` modulefile:

```
$ module use $PE_INSTALL/modulefiles
$ module load cmake
$ cmake --version
cmake version 3.5.1

CMake suite maintained and supported by Kitware (kitware.com/cmake).
$
```

A RPM of the resulting CMake package can be created by using the Craypkg-gen '`-r`' option.

```
$ export CLE_PE_INSTALL=<System PE tools directory>
$ craypkg-gen -r $PE_INSTALL/cmake/3.5.1 --prefix=$CLE_PE_INSTALL
```

The resulting RPM that is created will be called `craypkg-cmake-3.5.1-0.x86_64.rpm`.

APPENDIX C

Building and Installing the Boost library for Cray XC

<Caveat: This information is being provided as proof of concept. It has not been tested on a production level system>

The Boost libraries are a set of open-sourced C++ header files and libraries that perform a wide range of functions. Several applications require the use of Boost software as part of their build procedures. The software is available for download at the site boost.org.

The Boost code contains many C++ header and source files to build several libraries, as a result it can take quite a bit of time to build. Before taking the steps to build the libraries, the user may want to verify that the Boost libraries are actually going be used rather than just the Boost header files. The experience of the author of this article has found the majority of applications that require Boost software only use the Boost header files. Thereby the only action required was to download Boost and then provide the application the directory location of the Boost source code.

Since Boost is C++ code, the Boost libraries should be built with the same compiler as the application that intends to use Boost. The following shows the steps used to build Boost 1.59.0 using the Cray compiler (module 'PrgEnv-cray' loaded).

1) Un-tar the Boost software package:

```
$ tar xzvf boost_1_59_0.tar.gz
```

2) Initalize the Boost build script:

```
$ cd boost_1_59_0
$ export CC=cc
$ export CXX=CC
$ export PE_INSTALL=<PE_Installation Directory>
$ ./bootstrap.sh --prefix=$PE_INSTALL/boost/1.59.0/CRAY --without-libraries=python
cflags="-hgnu -h ipa0" cxxflags="-hgnu -hipa0"
```

Note: '`-h ipa0`' was chosen to speed up the build time for the Cray compiler.
The '`--prefix=$PE_INSTALL/boost/1.59.0/CRAY`' option was used to designate an installation location for the Boost header files and libraries.

3) Build the Boost libraries. The following creates static versions of the libraries:

```
$ ./b2 toolset=cray link=static
```

While the following creates dynamic versions of the libraries:

```
$ ./b2 toolset=cray.
```

4) Install Boost header files and libraries into an installation directory.

If the b2 `--prefix` option was specified, then the Boost header files and libraries can be installed into that directory by executing the command:

```
$ ./b2 toolset=cray install
```

With the environment variable `$BOOST_DIR` set to the install directory `$PE_INSTALL/boost/1.59.0/CRAY`, the header files are accessed by the compilation command line specifying the "`-I $BOOST_DIR/include`" option, while the libraries are accessed using the "`-L $BOOST_DIR/lib`" option.

A modulefile can also be created for Boost so that the CrayPE compiler drivers will automatically find the Boost header files and

libraries when the Boost modulefile is loaded. If the Boost library is to be created for other compilers (i.e. GNU, INTEL) then the Craypkg-gen commands should be executed after all the libraries have been created. Here are the steps to create a modulefile for Boost:

A) Set up the pkg-config .pc files:

```
$ module load craypkg-gen
$ craypkg-gen -p $PE_INSTALL/boost/1.59.0
```

B) Create the modulefile:

```
$ craypkg-gen -m $PE_INSTALL/boost/1.59.0
```

If versions of Boost are being built for different compilers, then the resulting modulefile `$PE_INSTALL/modulefiles/boost/1.59.0` needs to be edited to modify `LD_LIBRARY_PATH` to be correct for the possible values of `$loaded_prgenv` (CRAY, GNU, INTEL, PGI). Also, the Cray version of Boost does not build libboost_locale library, so the modulefile needs to be edited to not reference '`boost_locale`' in the `*_PKGCONFIG_LIBS` variables.

To use the Boost header files and libraries, the user would simply need to load the `boost/1.59.0` modulefile and use the CrayPE compiler drivers (`CC, cc, ftn`). The output of the '`CC –c --craype-verbose`' should show an output of a command that specifies the Boost header files directory:

```
$ module swap craype-network-aries craype-network-none
$ module unload cray-mpich cray-libsci
$ module use $PE_INSTALL/modulefiles
$ module load boost
$ cat main.c
int main() { }
$ cc -c -craype-verbose main.c
driver.cc -hcpu=ivybridge -hstatic -D__CRAY_IVYBRIDGE -D__CRAYXT_COMPUTE_LINUX_TARGET
-hnetwork=none -c main.c -Wl,--rpath=/opt/cray/cce/8.5.0/craylibs/x86-64
-hlast_user_arg -nostdinc -ibase-compiler /opt/cray/cce/8.5.0/CC/x86-
64/compiler_include_base -isystem /opt/cray/cce/8.5.0/craylibs/x86-64/include
-I/opt/gcc/4.8.1/snos/lib/gcc/x86_64-suse-linux/4.8.1/include
-I/opt/gcc/4.8.1/snos/lib/gcc/x86_64-suse-linux/4.8.1/include-fixed -isystem
/usr/include -ugcc_base=/opt/gcc/4.8.1/snos -uno_driver_libs
-I$PE_INSTALL/boost/1.59.0/CRAY/include -I/opt/cray/rca/1.0.0-
2.0502.60530.1.62.ari/include -I/opt/cray/cce/8.5.0/craylibs/x86-
64/pkgconfig/..//include -I/opt/cray/krca/1.0.0-2.0502.63139.4.31.ari/include
-I/opt/cray-hss-devel/7.2.0/include
$
```

Assuming that a GCC version of Boost 1.59.0 was also built and installed into $PE_INSTALL/boost/1.59.0/GNU, then a module swap to PrgEnv-gnu will result in the CC compiler driver automatically user the GCC version of the Boost libray:

```
$ module swap PrgEnv-cray PrgEnv-gnu  2>/dev/error
$ module unload cray-libsci
$ cc -c -craype-verbose main.c
gcc -march=corei7-avx -static -D__CRAY_IVYBRIDGE -D__CRAYXT_COMPUTE_LINUX_TARGET
-upthread_mutex_destroy -D__TARGET_LINUX__ -c main.c
-I$PE_INSTALL/boost/1.59.0/GNU/include
$
```

APPENDIX D

Building and Installing the MPI for Python (mpi4py) for Cray XC

<Caveat: This information is being provided as proof of concept. It has not been tested on a production level system>

The Python Software Foundation mpi4py library can be downloaded and built to run python programs that use the MPI library. Here are instructions on how to build the mpi4py library for a Cray XC system. :

1) Download the mpi4py source code. Here is a link to the software:

   https://pypi.python.org/pypi/mpi4py#downloads

2) Untar and unzip the package:

```
$ tar xzvf mpi4py-2.0.0.tar.gz
```

3) Use the GCC environment to build mpi4py

```
$ cd mpi4py-2.0.0
$ module swap PrgEnv-cray PrgEnv-gnu
```

4) Set environment variable CC, so that 'cc' is used as the compiler, then run setup.py to build mpi4py:

```
$ env CC=cc python setup.py build
```

5) Re-execute these link statements of the build of the Python MPI shared libraries, so that 'cc' is used instead of 'gcc to build the libraries. For CLE 6.0 systems (Python 2.7), the commands to execute are:

```
   $ cc -pthread -shared build/temp.linux-x86_64-2.7/src/lib-pmpi/mpe.o -o
build/lib.linux-x86_64-2.7/mpi4py/lib-pmpi/libmpe.so
    $ cc -pthread -shared build/temp.linux-x86_64-2.7/src/lib-pmpi/vt.o -o
build/lib.linux-x86_64-2.7/mpi4py/lib-pmpi/libvt.so
    $ cc -pthread -shared build/temp.linux-x86_64-2.7/src/lib-pmpi/vt-mpi.o -o
build/lib.linux-x86_64-2.7/mpi4py/lib-pmpi/libvt-mpi.so
    $ cc -pthread -shared build/temp.linux-x86_64-2.7/src/lib-pmpi/vt-hyb.o -o
build/lib.linux-x86_64-2.7/mpi4py/lib-pmpi/libvt-hyb.so
    $ cc -pthread -shared build/temp.linux-x86_64-2.7/src/MPI.o -Lbuild/temp.linux-
x86_64-2.7 -ldl -lpython2.7 -o build/lib.linux-x86_64-2.7/mpi4py/MPI.so
    $ cc -pthread -shared build/temp.linux-x86_64-2.7/src/dynload.o -Lbuild/temp.linux-
x86_64-2.7 -ldl -lpython2.7 -o build/lib.linux-x86_64-2.7/mpi4py/dl.so
    $
```

For CLE 5.2 systems (Python 2.6), the commands to execute are:

```
   $ cc -pthread -shared build/temp.linux-x86_64-2.6/src/lib-pmpi/mpe.o -o
build/lib.linux-x86_64-2.6/mpi4py/lib-pmpi/libmpe.so
    $ cc -pthread -shared build/temp.linux-x86_64-2.6/src/lib-pmpi/vt.o -o
build/lib.linux-x86_64-2.6/mpi4py/lib-pmpi/libvt.so
    $ cc -pthread -shared build/temp.linux-x86_64-2.6/src/lib-pmpi/vt-mpi.o -o
build/lib.linux-x86_64-2.6/mpi4py/lib-pmpi/libvt-mpi.so
    $ cc -pthread -shared build/temp.linux-x86_64-2.6/src/lib-pmpi/vt-hyb.o -o
build/lib.linux-x86_64-2.6/mpi4py/lib-pmpi/libvt-hyb.so
    $ cc -pthread -shared build/temp.linux-x86_64-2.6/src/MPI.o -Lbuild/temp.linux-
x86_64-2.7 -ldl -lpython2.6 -o build/lib.linux-x86_64-2.6/mpi4py/MPI.so
    $ cc -pthread -shared build/temp.linux-x86_64-2.6/src/dynload.o -Lbuild/temp.linux-
```

```
x86_64-2.7 -ldl -lpython2.6 -o build/lib.linux-x86_64-2.6/mpi4py/dl.so
    $
```

6) The setup.py 'install' command can be used to install mpi4py into an installation location. To install the files into the directory /opt/local/mpi4py the following command can be executed:

```
$ export PE_INSTALL=<PE Installation Directory>
$ python setup.py install --prefix=$PE_INSTALL/mpi4py/2.0.0
```

7) A modufile for mpi4py can be created by executing the following commands:

```
# craypkg-gen -m $PE_INSTALL/mpi4py/2.0.0
```

The newly created `$PE_INSTALL/modulefiles/mpi4py/2.0.0` modulefile should be edited with the following line added to the file.
CLE 6.0:

```
prepend-path PYTHONPATH $PREFIX/lib64/python2.7/site-packages
```

CLE 5.2:

```
prepend-path PYTHONPATH $PREFIX/lib64/python2.6/site-packages
```

8) Here is a sample test for the mpi4py library:

```
$ cat test.py
from mpi4py import MPI
import os
import glob
COMM = MPI.COMM_WORLD
irank = COMM.Get_rank()
print 'Hello world from rank', irank
$ module use $PE_INSTALL/modulefiles
$ module load mpi4py
$ aprun -n 8 python test.py
Hello world from rank 2
Hello world from rank 4
Hello world from rank 5
Hello world from rank 6
Hello world from rank 7
Hello world from rank 3
Hello world from rank 0
Hello world from rank 1
Application 8209638 resources: utime ~0s, stime ~0s, Rss ~9892, inblocks ~4282,
outblocks ~12
$
```