Comparing Binaries Between Cray Linux Environment (CLE) Systems, Standalone Whiteboxes, and ESLogin Nodes

Published Date 11/20/09

Abstract

Users have shown an interest in understanding all potential differences between a binary generated on a Cray XT login node and a binary generated on either a standalone SuSE Linux whitebox or an ESlogin node installed with the Cray Application Developer's Environment and its supplement for standalone machines. This paper documents likely, known sources of differences between these binaries and how to detect and avoid them.

Table of Contents

Introduction

Users have shown an interest in understanding all potential differences between a binary generated on a Cray XT login node and a binary generated on either a standalone SuSE Linux whitebox or an ESlogin node installed with the Cray Application Developer's Environment and its supplement for standalone machines. This paper documents likely, known sources of differences between these binaries and how to detect and avoid them.

Types of Binary Differences

The most common source of binary differences we have observed in-house is caused by library version mismatches between the XT and the standalone machine. It is vitally important to keep the software on both machines up-to-date with each other if you want comparable binaries. We have made our security patches available specifically for the xt-sysroot package in order to ensure that the SLES software versions exactly match between your XT and your standalone development machine. Use the `module list` command on each machine to compare library versions before you decide that the software on the standalone machine is not capable of producing nearly identical binaries. The wrong modulefile might be loaded or set as default on one of the two systems. Differing symbol versions in the two libraries would explain the difference in binaries.

Even when the library versions and security patch levels exactly match between two systems, there are other potential sources of binary mismatch. Within the standard ELF headers, every binary contains a date-and-time stamp that almost never matches any other binary perfectly. Due to this timestamp field, you can almost never produce a perfect byte-for-byte match using separate link steps on two different machines. Therefore, we consider this to be an acceptable difference.

Also, the locations of dynamic libraries can differ when linking libraries dynamically using our new dynamic shared library support. You can observe this by executing the `ldd` command on the affected binary. On a standalone machine, dynamic links first search in the /opt/cray/xt-sysroot path for dynamic library files that normally would be located in /usr/lib64, /usr/lib, /lib64, and /lib. This allows Cray to provide an alternate set of dynamic shared objects which perfectly matches with that OS version level of XT CLE. When executing on a Cray XT machine, these binaries would need to satisfy those libraries dependencies using their usual location rather than the xt-sysroot path embedded in the binary. With a properly set LD_LIBRARY_PATH, this should usually not be a problem, but it is an area of concern to be aware of.

Evaluating Binary Differences

Discovering and diagnosing a substantial binary difference requires tools that provide more information than printing an `ls -l` file listing and glancing at the binary size or running the `md5sum` or `cksum` utilities. For one thing, the observed file size can vary based upon the underlying filesystem. More importantly, these methods do not explain the reasons for the binary difference in a useful manner. Explained below are some alternative choices of tools for understanding the cause of a binary difference.

First, check the load map to make sure that all libraries were loaded from the proper locations on each machine. To do this, execute your application's compile step with added command line options to pass the "-M" option to the linker: "-Wl,-M". The compiler should then send a load map to standard output that you can save and compare by redirecting standard output to a file. This load map file shows which library on the host machine is being used to satisfy each symbol in the program. On a whitebox or ESlogin node, the xt-sysroot path provides a mirror of the standard libraries on the XT login node, so most libraries in the loadmap should be obtained from the xt-sysroot path.

Second, the `readelf` utility can parse the binary into useful, easy-to-comprehend sections that can be used for comparison purposes. Using the "-a" or all option, the produced output shows the lengths and names of each section, header, and symbol in the application binary file along with other useful information. By performing a graphical `diff` of the two readelf outputs, you can quickly discover which symbols and sections differ. If the symbols differ in size, there is a library mismatch.

For dynamically-linked binaries, the `ldd` command can be helpful to show which library paths are likely to be used as the runtime link search path. The `ldd` listing uses LD_LIBRARY_PATH, rpath information from the a.out ELF header, and ld_conf settings in that order to locate a runtime path. Missing path references are listed by the tool and must be accounted for by either loading a module or supplying an LD_LIBRARY_PATH.

Finally, you can disassemble the binary with a disassembler utility such as diStorm64 or udis86. Often, the disassembled output is more difficult to read than a loadmap or readelf file because the standard operating system code used to control program startup and termination are encoded into machine code without helpful variable or symbol names to clue in the user to what each code segment accomplishes. Library symbols are not easy to distinguish without their symbol names. However, the disassembler output helps to understand how significant the binary differences are in terms of actual code differences.

Setting PGI Core Runtime Libraries

During the linking phase with each compiler, a few standard GCC object files are linked into the binary, called the runtime core libraries. Many compilers within the SLES10-based Programming Environment use the GCC 4.1.2 version of these files. They are provided with the GCC 4.1.2 compiler package that is already identical to what is installed on your XT, so they do not normally require any special configuration to avoid binary differences. However, the Portland Group compiler sets its own location for these files that should be overridden by hand using a 'siterc' configuration file.

In addition to runtime core libraries, PGI compilers use the SuSE gcc runtime at /usr/lib64/gcc for linking during a build. For SLES10-based systems, this is GCC 4.1.2. Defining this PGI path in a 'siterc' file to use the xt-sysroot path ensures that a standalone machine behaves like a service node during compilation and linking.

If there are differences between the standalone system and the Cray XT with respect to the runtime core libraries or SuSE gcc runtime, then it is critical to create a 'siterc' file for PGI compilations. This file must be placed in a

PGI bin directory that is used exclusively for compiling for the XT. Below is an example of a PGI 'siterc' file which uses your currently loaded xt-sysroot path:

```
## siterc ......
## Forces use of /opt/cray/xt-sysroot directory instead of /usr/lib64.
## This only works when xt-sysroot is loaded. When xt-sysroot is
## not loaded, sysroot_gcc_ver is unset and the GCCDIR path is not correct.

variable sysroot_dir is environment(SYSROOT_DIR);
variable sysroot_gcc_ver is environment(SYSROOT_GCC_VER);

set GCCDIR=$sysroot_dir/usr/lib64/gcc/x86_64-suse-linux/$sysroot_gcc_ver;
set DEFSTDOBJDIR=$sysroot_dir/usr/lib64;

set LCRTI=$STDOBJDIR/crti.o;
set LCRTN=$STDOBJDIR/crtn.o;
set LCRT1=$STDOBJDIR/crt1.o;

set LCRTE=$GCCDIR/crtend.o;
set LCRTB=$GCCDIR/crtbegin.o;
```

Conclusion

Although there have been known differences observed in the past between the login node and standalone machines, Cray is working hard to minimize all differences that come from our released library packages. We have created a merged Cray Application Developer's Environment release for both XT and standalone systems so that library symbols should not differ between an XT and non-XT system running the same version of CADE, CADES, and licensed products. We have also created an xt-sysroot package for standalone machines for the purpose of providing an identical library environment to the installed version of XT CLE. While we occasionally spotted minor differences during development, we have corrected most that we have found. We are working hard at Cray to replicate the XT library environment completely and accurately, so that all library path and operating system differences are fully accounted for.