Cray XT Programming Environment's Implementation of Dynamic Shared Libraries

Geir Johansen, *Cray Inc.*
Barb Mauzy, *Cray Inc*.

ABSTRACT: *The Cray XT Programming Environment will support the implementation of dynamic shared libraries in future Cray XT CLE releases. The Cray XT implementation will provide the flexibility to allow specific library versions to be chosen at both link and run times. The implementation of dynamic shared libraries allows dynamically linked executables built with software other than the Cray Programming Environment to be run on the Cray XT.*

KEYWORDS: Cray XT, Programming Environment

1.0 Introduction

The goal of this paper is provide application analysts a description of Cray XT Programming Environment's implementation of dynamic shared libraries. Topics covered include the building of dynamic shared library programs, configuring the version of dynamic shared libraries to be used at execution time, and the execution of a dynamically linked program not built with the Cray XT Programming Environment. While a brief description of the Cray XT system implementation of dynamic shared libraries will be given, the Cray XT system architecture and system administration issues of dynamic shared libraries are not the focus of this paper. An important caveat to this paper is that the Cray XT implementation of dynamic shared libraries has not been released and some of the information provided is likely to change.

2.0 Motivation for Supporting Dynamic Shared Libraries

The implementation of dynamic shared libraries on the Cray XT provides users with a more comprehensive application runtime environment. The support of this feature will allow more types of applications to be executed on the Cray XT. This section will describe the advantages of using dynamic shared libraries along with some of its disadvantages.

2.1 Advantages of Dynamic Libraries

The main benefit of supporting dynamic shared libraries is that it significantly increases the number of applications that can execute on the Cray XT. Several independent software vendors (ISVs) only provide dynamically linked executables of their software, so the support of dynamic shared libraries will now allow many of these applications to run on the Cray XT. With dynamic shared library support, applications do not need to be built with the Cray XT Programming Environment to execute on the Cray XT.

In addition to dynamically linked applications, the support of dynamic shared libraries allows for dynamically linked utilities to be executed on Cray XT compute nodes. For example, the Python interpreter can now be executed on Cray XT compute nodes. This is important as Python becomes an increasingly used tool for parallel computing. Other utilities that can now be executed on Cray XT compute nodes are the compilers. A user can now use the Cray XT compute nodes to decrease the time to compile large codes.

Dynamic shared libraries have an advantage over static libraries in that they are *shared* libraries. A shared library will have only one copy of its read-only information in physical memory that can be shared among multiple processes. This feature potentially reduces the memory allocated for a program on a compute node. This advantage becomes more important as the number of available cores increases.

Another advantage of dynamically linked executables is that the system libraries can be upgraded and take effect without the need to relink applications. The ability to install library changes without having to rebuild a program is a significant advantage in quickly addressing problems, such as a security issue with a library code. The experiences of system administrators have shown that codes using dynamic shared libraries are more resilient to system upgrades than codes using static libraries [1].

The support of dynamic linking allows codes that use dynamic linking routines, such as *dlopen*, to execute on a Cray XT. Dynamic linking allows users to add functionality to a program without access to the source code. Pluggable authentication modules (PAM) are an example of the use of dynamic linking.

A benefit of the Cray XT implementing dynamic shared libraries is that the medium memory model can now be supported on the Cray XT. The AMD Opteron medium memory model feature allows executables to address a statically allocated memory space greater than 2 gigabytes. The medium memory model is enabled by using the compilers' *-mcmodel=medium* option. A statically linked executable can only support the medium memory model if and only if all the libraries it uses were also compiled with the medium memory model option. The Cray XT Programming Environment does not provide static versions of libraries built with the medium memory model. Dynamic shared libraries are not required to be built with the *-mcmodel=medium* option to support this feature, so this feature can be used in applications that are dynamically linked.

The implementation of dynamic shared libraries will allow tools that assume the use of dynamic executables to be able to be used on the Cray XT. An example would be the Valgrind suite of debugging and analysis tools.

Finally, an advantage of dynamically linked executables is that they are smaller in size than statically linked executables, so they will use less disk space.

2.2 Disadvantages of Dynamic Libraries

The main disadvantage of dynamic shared libraries is the runtime performance costs of dynamic linking. Every time the program is executed it has to perform a large part of its linking process. The lookup of symbols in a dynamic shared library is much less efficient than in static libraries. The loading of a dynamic shared library during an application's execution may result in a "jitter" effect where a single process holds up the forward progress of other processes of the application while it is loading a library.

The use of dynamically linked executables has a significant disadvantage in that they are not supported by the CrayPat 5.0 performance tool. Support of analyzing dynamically linked executables is a planned feature that will be implemented in a future release of CrayPat.

An advantage of statically linked executables is that they are guaranteed to run the same library code across all systems. This helps prevents the behavior of an application changing when a system is upgraded. This feature of static libraries is helpful for sites that have a certification process for their applications.

3.0 Cray XT DVS Shared Root Solution

The Cray XT implementation of dynamic shared libraries will use Cray Data Virtualization Service (DVS) to allow compute nodes to access a shared root filesystem that contains the dynamic shared libraries. The DVS server will access the shared root from the system's boot node. Multiple DVS servers will run on dedicated Cray XT nodes that will be used to serve a DVS client running on the compute nodes. Using multiple DVS servers will provide scalability and also failover capability. Figure 1 shows a diagram of the DVS shared root solution.

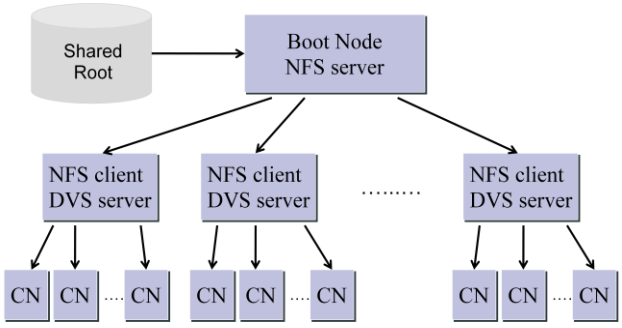

Figure 1. DVS Shared Root Solution

The shared root is read-only mounted via DVS to a mount point on the compute node (e.g. /mnt). The parts of the root filesystem required to be writable, such as /tmp and /proc, are bind mounted from the compute node's RAM/boot filesystem An application's access to the shared root is coordinated by ALPS during the launch of the application. When the *aprun* command is initiated, ALPS will perform the following functions to launch the application on the compute node:

1. *fork*
2. *chroot* to the shared root filesystem
3. *setgid*, *setuid*
4. *chdir* to the launch directory
5. *exec* the application

Figure 2 shows an illustration of the launch of an application that uses the shared root.

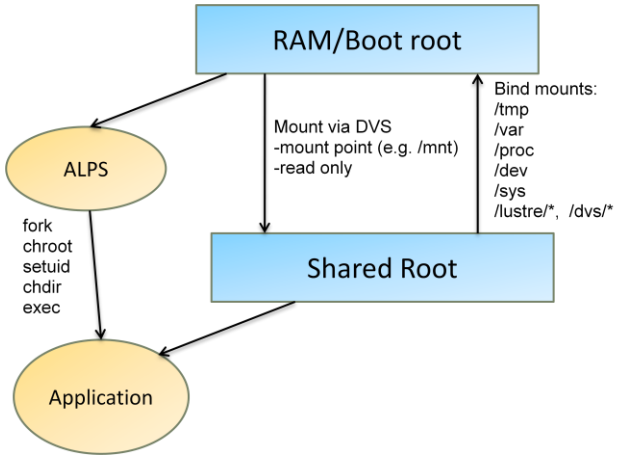

Figure 2. Startup of an Application

4.0 Implementation of Dynamic Shared Libraries

The Cray XT Programming Environment implementation of dynamic shared libraries will support the building and executing of dynamically linked code. It will also provide attributes of the current Cray XT Programming Environment. Specifically, Cray XT users will continue to be able to choose from several versions of programming environment libraries for each of the compiler environments.

4.1 Building a Dynamically Linked Executable

Creating a dynamically linked executable is very similar to the existing Cray XT compilation process. The use of programming environment *modulefile*s initializes the build environment, which is used by compiler scripts to link in the appropriate versions of the libraries. The compiler scripts provided in the **xt-asyncpe 2.5** product will support the building of dynamically linked executables and the building of dynamic shared libraries. The same options for dynamic linking are used for all the compiler programming environments supported by the Cray XT. Note that Cray Compiler Environment (CCE) versions 7.0 and 7.1 will not support the building of dynamically linked code. This feature is planned for a future release of CCE.

The following list shows the versions of Cray XT Programming Environment components that provide dynamic shared versions of their libraries:

- **MPT 3.2** – Cray XT implementation of MPICH2 and SHMEM libraries
- **LibSci 10.3.4** – Cray Scientific libraries

- **FFTW 3.2.1 –** MIT FFTW library
- **ACML 4.0** – AMD Core Math Library
- **PETSc 3.0.0.2** – ANL PETSc library
- **hdf5_netcdf 1.2** – HDF5 1.8.2 and netCDF 4.0.0 dynamic shared libraries are provided. The Parallel HDF5 and netCDF Parallel HDF5 builds do not provide support for dynamic libraries.
- **libfast_mv 1.0.3** - Fast_mv high performance math library

The Cray XT compiler scripts use the *–dynamic* compiler option to specify that dynamic shared libraries should be used. When performing a dynamic link, the compiler scripts will use the *ld –rpath* option for each of the programming environment libraries. The *–rpath* option adds the library's directory to the executable's ELF header, which allows these specific libraries to be used at runtime. The following example shows a Fortran program being built with dynamic shared libraries:

```
$ cat hello_world.f90
program hello_world
  implicit none
  include "mpif.h"
  integer :: rank, ierr
  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD,  &
                                rank, ierr)
  print *, 'Hello World from rank ', rank
  call MPI_FINALIZE(ierr)
end program hello_world

$  ftn -dynamic hello_world.f90 –o hello_world
/opt/cray/xt-asyncpe/2.5/bin/ftn: INFO: linux target is being used
$
```

The compiler scripts also support the *–static* option to explicitly specify the use of static libraries. The use of static libraries is the default behavior of the compiler scripts; however, it is possible that this will change in future releases of CLE. The environment variable XTPE_LINK_TYPE can be set to 'dynamic' or 'static' to create a default setting for the user's environment.

The *ldd* utility is a useful tool to print out the shared libraries required by an executable. The output shows the dynamic shared libraries that a program will use when executed. Any "file not found" messages indicate libraries that could cause execution failures and will need path definition using the LD_LIBRARY_PATH environment variable. Appendix A shows the output of *ldd* on the Fortran program example.

The dynamic shared library information contained in the executable's ELF header can be shown by using the *readelf* utility. The *readelf –d* option shows the programming environment library directories placed in the ELF header at link time. Appendix B shows the output of *readelf –d* on the Fortran program example.

4.2 Executing a Dynamically Linked Executable

The process of executing a dynamically linked executable on a Cray XT is similar to executing a statically linked executable. A program built with dynamic shared libraries provides the opportunity for the user to choose which versions of the dynamic libraries that will be used at run time. To better understand the execution of dynamically linked code, it is important for the user to be aware of the search order the dynamic linker uses when loading a dynamic shared library:

1) The value of the environment variable LD_LIBRARY_PATH
2) The value of the DT_RUNPATH dynamic section of the executable. This is set by the linker with the use of the *ld –rpath* option.
3) The contents of the cache file */etc/ld.so.cache*.
4) The default path of */lib*, and then */usr/lib*.

A critical feature of the Cray XT Programming Environment is that the programming environment modulefiles modify the LD_LIBRARY_PATH environment variable. The loading of a programming environment modulefile will prepend the appropriate library directory to LD_LIBRARY_PATH. The use of modulefiles provides the user with an interface for choosing the programming environment libraries at runtime. Conversely, in order to ensure that the build time libraries are executed; the programming environment modulefiles need to be unloaded, or the environment variable LD_LIBRARY_PATH needs to be unset.

The following shows the Fortran example being executed with the libraries used at build time. The command *module purge* unloads all modulefiles and consequently clears out the value of the LD_LIBRARY_PATH environment variable. The program output shows that the build time version of the MPICH2 library (3.1.2) is being used:

```
$ module swap xt-mpt xt-mpt/3.1.2
$ ftn –dynamic hello_world.f90 -o hello_world
/opt/cray/xt-asyncpe/2.5.4/bin/ftn: INFO: linux target is being used
$ module purge
$ echo $LD_LIBRARY_PATH

$ export MPICH_VERSION_DISPLAY=1
$ aprun -n 1 ./hello_world
MPI VERSION : CRAY MPICH2 XT version 3.1.2 (ANL base 1.0.6)
BUILD INFO  : Built Mon Mar 16 10:55:48 2009 (svn rev 7308)
 Hello World from rank           0
```

Application 2466916 resources: utime 0, stime 0
$

When using the programming environment modulefiles to configure the run time versions of the libraries, it is important that the modulefiles that are loaded match the compiler that was used to build the program. For example, if the Pathscale compiler was used to build an application, then a *module load PrgEnv-pathscale* must be performed when setting the environment to execute the application. For programs that are run in batch jobs, the appropriate programming environment modulefile must be loaded in the batch script, or the *qsub –V* option needs to be used to pass the user's environment variables to the batch job. The following shows how modulefiles can be used to change the Fortran example from using the build time MPICH2 library (3.1.2) to a newer run time MPICH2 library (3.2.0):

$ module load Base-Opts PrgEnv-pgi
$ module swap xt-mpt xt-mpt/3.2.0 ←Updates env. variable LD_LIBRARY_PATH
$ export MPICH_VERSION_DISPLAY=1
$ aprun -n 1 ./hello_world
MPI VERSION : CRAY MPICH2 XT version 3.2.0 (ANL base 1.0.6)
BUILD INFO : Built Tue Apr 7 13:24:56 2009 (svn rev 7351)
Hello World from rank 0
Application 2452158 resources: utime 0, stime 0
$

The dynamic linker supports several environment options to alter behavior or provide information of a program's dynamic linking. One useful environment variable is LD_BIND_NOW, which instructs the dynamic linker to load all the dynamic libraries at startup. Another helpful environment variable is LD_DEBUG, which displays information about the dynamic linking process during the execution of a program. Setting LD_DEBUG=help provides a list of the types of information that can be displayed.

4.3 Executing an ISV code

One of the main goals of supporting dynamic shared libraries is to run ISV applications that are only released as executables. When running an ISV application, it is important to know the build compiler, so that the appropriate programming environment modules can be loaded. Programming environment libraries written in Fortran and C++ are compiler specific, so they must match the application's compile time environment, Also, Fortran codes need to be able to find the appropriate Fortran run time libraries; such as *libpgftnrtl.so* for PGI codes, *libgfortran.so* for GCC gfortran codes, and *libpathfortran.so* for Pathscale codes.

The application may come with its own dynamic shared libraries that are needed by the application. If so, these libraries can be placed in a compute node readable location, such as a Lustre filesystem. When executing the application, the LD_LIBRARY_PATH environment variable needs to be modified to point to the location of the application's dynamic shared libraries.

The applications CD-adapco STAR-CD and Exa Powerflow have been executed on a Cray XT using executables provided by the vendors. Both of the applications were built using an MPICH1 library, which is not compatible with the MPICH2 library that runs on the Cray XT. In both cases, the vendors needed to provide special versions of the applications built with a MPICH2 library. Once the vendors provided the applications linked with a MPICH2 library, the applications were able to successfully run on the Cray XT.

The DVS shared root solution has the side benefit for the Cray XT of now fully supporting the "getpw" routines (i.e. *getpwuid*, *getpwnam*). Prior to the shared root solution, the getpw routines only got the limited */etc/passwd* information found on the compute node RAM root filesystem.

4.4 Building a dynamic shared library

The Cray XT compiler scripts support the building of dynamic shared libraries. The scripts pass the *–shared* option to the linker to indicate that a dynamic shared library is being built. It is important to clarify that both static and dynamic libraries can be used in the same executable. A user program that links in dynamic versions of Cray XT programming environment libraries does not have to link in dynamic shared versions of any libraries that the program creates. Existing applications do not have to convert their application's libraries to be dynamic shared libraries in order to dynamically link with the programming environment dynamic shared libraries.

Dynamic shared libraries that are created by a user need be placed in directory that can be read by compute nodes, such as a Lustre filesystem. The user may elect to use *–rpath* to store the library's directory location in the application's ELF header, or require that the environment variable LD_LIBRARY_PATH be modified to point to the location of the library. The following shows an example of building a dynamic shared library on the Cray XT. Note that all the compilers of the Cray XT will accept either *–fpic* or *-fPIC* as options to build position independent code that is required for dynamic shared libraries.

$ **cat test.f**

```
      subroutine shared_lib_test
      print *,'In libtest version 0.0'
      endsubroutine
```

$ **ftn -c -fpic test.f**
/opt/cray/xt-asyncpe/2.5.4/bin/ftn: INFO: linux target is being used
$ **ftn –shared -Wl,-soname=libtest.so.0 -o libtest.so.0.0 test.o**
/opt/cray/xt-asyncpe/2.5.4/bin/ftn: INFO: linux target is being used
$ **/sbin/ldconfig –n .**
$ **ln -s libtest.so.0.0 libtest.so**
$ **cat main.f**

```
program test
call shared_lib_test()
endprogram
```

$ **ftn –dynamic -L<*library-directory*> -ltest -Wl,-rpath=<*library-directory*> main.f**
/opt/cray/xt-asyncpe/2.5.4/bin/ftn: INFO: linux target is being used
$ **aprun –n 1 ./a.out**
In libtest version 0.0
$

4.4 Differences from Linux Implementation

The Cray XT Programming Environment implementation of dynamic shared libraries differs than the usual method dynamic shared libraries are typically installed on Linux systems. The Linux model is that dynamic shared libraries are installed in only a handful of directories and that only the latest version of a library's major release is kept on the system. One reason that the Cray XT uses separate directories for each version of the programming environment libraries is that separate versions are required for each of the compilers supported on the Cray XT. For example, a separate Cray Scientific Library *libsci.so* is required for the PGI, Pathscale, and GCC compilers. Also, Cray XT users are accustomed to having multiple version levels of programming environment libraries available on the system, so a library is not automatically removed when a newer version of that library is installed on the system.

The Cray XT Programming Environment also differs in its use of the LD_LIBARY_PATH and the *–rpath* option. These features of the dynamic linker are not commonly used by applications running on Linux. The Cray XT Programming Environment uses these features in order to provide the flexibility of either using the build time libraries or being able to easily choose other versions to use as the run time libraries.

5.0 Future Opportunities

The support of dynamic shared libraries on the Cray XT is in the initial part of its development, so there are several issues that can be further investigated to enhance the use of dynamic shared libraries with the Cray XT programming environment.

5.1 Performance of Dynamically Linked Applications

It is generally accepted that statically linked applications will perform slightly better than dynamically linked applications [1][3]. The performance differences of dynamically linked code versus statically linked code needs to be measured and investigated, so that guidance can be provided to the user on how to optimize the execution of their application. Also, there are strategies on the optimization of dynamic shared libraries that may be able to be applied to the Cray XT programming environment libraries [2][4].

5.2 Default versions of the Cray XT Programming Environment Libraries

The current implementation uses the application's ELF header or the value of LD_LIBRARY_PATH to provide the location of the programming environment dynamic shared libraries. Further testing can determine the viability of providing default versions of programming environment libraries in a single directory location. For example, it may be possible that the GCC version of the MPICH2 library *libmpich.so* is sufficient to use with the majority of applications, even those applications not compiled with GCC. If so, this version of *libmpich.so* could be moved to a default directory location and this directory name would be added to the file */etc/ld.so.conf*. This action would allow applications that require the *libmpich.so* library to execute without having the directory of the library in the applications ELF header or included in the LD_LIBRARY_PATH environment variable.

The */etc/ld.so.conf* file could also be used to add the location of the compiler's run time libraries. For example, when the latest PGI compiler is installed, the installation process would add the location of its run time libraries (i.e. *libpgftnrtl.so*) to */etc/ld.so.conf*, and then proceed to run *ldconfig* to initialize */etc/ld.so.cache*. Many Cray XT sites have a testing period before making a compiler version the default on the system, so the installation procedure would need to account for this case.

5.3 User and Application Provided Dynamic Shared Libraries

Users and applications will want to install dynamic libraries so that they can be used by other users. One solution is to use a directory, such as */usr/local/lib64*, for these types of libraries. The procedure used to solve this issue will need to be determined and documented.

5.3 Support of Common Linux Cluster Dynamic Shared Libraries

The experience of running ISV provided application executables has shown that additional libraries may need to be supported to further increase the number of available ISV applications that can run on the Cray XT. Both the STAR-CD and Powerflow applications required a

MPICH1 library. A possible solution is to provide header files and libraries that map the MPICH1 functions to the MPICH2 functions currently available in the Cray XT Programming Environment. The HP-MPI library is another library that would be helpful to make available on the Cray XT.

6.0 Conclusion

The Cray XT Programming Environment implementation of dynamic shared libraries provides support for building and executing dynamically linked codes. As part of it implementation, it also provides the following features:

- Capability to run the executable with the build time versions of the programming environment libraries.
- Supports compiler specific programming environment libraries that can be configured at run time.
- Provides an interface to easily choose specific versions of the programming environment libraries to use at run time.

The Cray XT Programming Environment implementation of dynamic shared libraries is in its early stages of its development process and will continue to evolve to enhance the Cray XT application runtime environment.

Acknowledgments

The authors thank Cray software development and the Cray performance team for valuable technical information and consultation. Specifically, we would like to thank Wayne Ohlrich, Ting-Ting Zhu, and David Whitaker.

About the Authors

Geir Johansen works in Software Product Support, Cray Inc. He is responsible for support of Cray Programming Environment software for the Cray X1, Cray XT, Cray X2, and Cray XMT platforms. He can be reached at Cray Inc., 1340 Mendota Heights Road, Mendota Heights, MN 55120, USA; Email: geir@cray.com

Barb Mauzy works in Software Development for Cray Inc. She develops compiler drivers and other parts of the Cray Programming Environment interconnect for all Cray platforms. She can be reached at Cray Inc., 1340 Mendota Heights Road, Mendota Heights, MN 55120, USA; Email: bam@cray.com

References

[1] Levine, John R.. Linkers and Loaders. Morgan-Kauffman, 1999.

[2] Drepper, Ulrich. "How to Write Shared Libraries", http://people.redhat.com/drepper/dsohowto.pdf , 2006.

[3] Wheeler, David A.. Program Library HOWTO, http://tldp.org/HOWTO/Program-Library-HOWTO/index.html , 2003.

[4] Dynamic Library Programming Topics. http://developer.apple.com/documentation/DeveloperTools/Conceptual/DynamicLibraries/Dynamic_Libraries.pdf , Apple Inc., 2009.

Appendix A

Example Output of *ldd* Utility

$ ldd hello_world

```
    libportals.so.1 => /opt/cray/portals/2.2-1.0000.16519.93.1/lib64/libportals.so.1 (0x00002b3e36273000)
    libcr.so.0 => /opt/cray/blcr/0.7.3-1.0000.191.9.3/lib64/libcr.so.0 (0x00002b3e3637c000)
    libdl.so.2 => /lib64/libdl.so.2 (0x00002b3e3649d000)
    libpthread.so.0 => /lib64/libpthread.so.0 (0x00002b3e365a1000)
    libsci.so => /opt/xt-libsci/10.3.3/pgi/snos64/lib/libsci.so (0x00002b3e366b8000)
    libsma.so => /opt/mpt/3.1.2/xt/sma/lib/libsma.so (0x00002b3e4786e000)
    libmpichf90.so.1.1 => /opt/mpt/3.1.2/xt/mpich2-pgi/lib/libmpichf90.so.1.1 (0x00002b3e47997000)
    libmpich.so.1.1 => /opt/mpt/3.1.2/xt/mpich2-pgi/lib/libmpich.so.1.1 (0x00002b3e47a99000)
    librt.so.1 => /lib64/librt.so.1 (0x00002b3e47d49000)
    libpmi.so => /opt/mpt/3.1.2/xt/pmi/lib/libpmi.so (0x00002b3e47e52000)
    libalpslli.so.0 => /opt/mpt/3.1.2/xt/util/lib/libalpslli.so.0 (0x00002b3e47f65000)
    libalpsutil.so.0 => /opt/mpt/3.1.2/xt/util/lib/libalpsutil.so.0 (0x00002b3e48068000)
    libm.so.6 => /lib64/libm.so.6 (0x00002b3e4816b000)
    libc.so.6 => /lib64/libc.so.6 (0x00002b3e482c0000)
    /lib64/ld-linux-x86-64.so.2 (0x00002b3e36157000)
$
```

Appendix B

Example Output of *readelf –d*

$ readelf -d hello_world

Dynamic section at offset 0x49078 contains 34 entries:

Tag	Type	Name/Value
0x0000000000000001	(NEEDED)	Shared library: [libsci.so]
0x0000000000000001	(NEEDED)	Shared library: [libfftw3.so.3]
0x0000000000000001	(NEEDED)	Shared library: [libfftw3f.so.3]
0x0000000000000001	(NEEDED)	Shared library: [libsma.so]
0x0000000000000001	(NEEDED)	Shared library: [libmpichf90.so.1.1]
0x0000000000000001	(NEEDED)	Shared library: [libmpich.so.1.1]
0x0000000000000001	(NEEDED)	Shared library: [librt.so.1]
0x0000000000000001	(NEEDED)	Shared library: [libpmi.so]
0x0000000000000001	(NEEDED)	Shared library: [libalpslli.so.0]
0x0000000000000001	(NEEDED)	Shared library: [libalpsutil.so.0]
0x0000000000000001	(NEEDED)	Shared library: [libpthread.so.0]
0x0000000000000001	(NEEDED)	Shared library: [libm.so.6]
0x0000000000000001	(NEEDED)	Shared library: [libc.so.6]
0x000000000000000f	(RPATH)	Library rpath: [/opt/xt-libsci/10.3.4/pgi/lib:/opt/xt-libsci/10.3.4/pgi/snos64/lib:/opt/fftw/3.2.1/lib:/opt/mpt/3.2.0.1/xt/sma/lib:/opt/mpt/3.2.0.1/xt/util/lib:/opt/mpt/3.2.0.1/xt/pmi/lib:/opt/pgi/8.0.5/linux86-64/8.0-5/lib]
0x000000000000001d	(RUNPATH)	Library runpath: [/opt/xt-libsci/10.3.4/pgi/lib:/opt/xt-libsci/10.3.4/pgi/snos64/lib:/opt/fftw/3.2.1/lib:/opt/mpt/3.2.0.1/xt/sma/lib:/opt/mpt/3.2.0.1/xt/util/lib:/opt/mpt/3.2.0.1/xt/pmi/lib:/opt/pgi/8.0.5/linux86-64/8.0-5/lib]
0x000000000000000c	(INIT)	0x403490
0x000000000000000d	(FINI)	0x43b784
0x0000000000000004	(HASH)	0x400258
0x0000000000000005	(STRTAB)	0x401df8
0x0000000000000006	(SYMTAB)	0x4008f8
0x000000000000000a	(STRSZ)	2804 (bytes)
0x000000000000000b	(SYMENT)	24 (bytes)
0x0000000000000015	(DEBUG)	0x0
0x0000000000000003	(PLTGOT)	0x5492f8

0x0000000000000002 (PLTRELSZ) 2256 (bytes)
0x0000000000000014 (PLTREL) RELA
0x0000000000000017 (JMPREL) 0x402bc0
0x0000000000000007 (RELA) 0x402b30
0x0000000000000008 (RELASZ) 144 (bytes)
0x0000000000000009 (RELAENT) 24 (bytes)
0x000000006ffffffe (VERNEED) 0x402ab0
0x000000006fffffff (VERNEEDNUM) 4
0x000000006ffffff0 (VERSYM) 0x4028ec
0x0000000000000000 (NULL) 0x0