CRAY
THE SUPERCOMPUTER COMPANY

Cray XT Programming Environment's Implementation of Dynamic Shared Libraries

Geir Johansen

May 6, 2009

Goals of the Presentation

- Present how dynamic share libraries are implemented in the Cray XT Programming Environment
  - How to build a dynamically linked executable
  - How to execute a dynamically linked executable
  - How to run an executable not built using the Cray XT Programming Environment
- The Cray XT system design used for supporting dynamic libraries is briefly discussed, but not in great detail

Outline

- Advantages/disadvantages of dynamic shared libraries
- Brief description of the Cray XT system implementation of dynamic shared libraries
- Building an executable with dynamic shared libraries
- Executing a dynamically linked program
- Executing a program not built with the Cray XT Programming Environment
- Creating a dynamic shared library

Advantages of Dynamic Shared Libraries

- Significantly increase the number of applications that can run of the Cray XT
  - Many ISVs only release executables
- Dynamically linked utilities can run on compute nodes
  - Python
  - Compilers
- Libraries can be upgraded without rebuilding applications
  - Linux provides method for backward compatibility

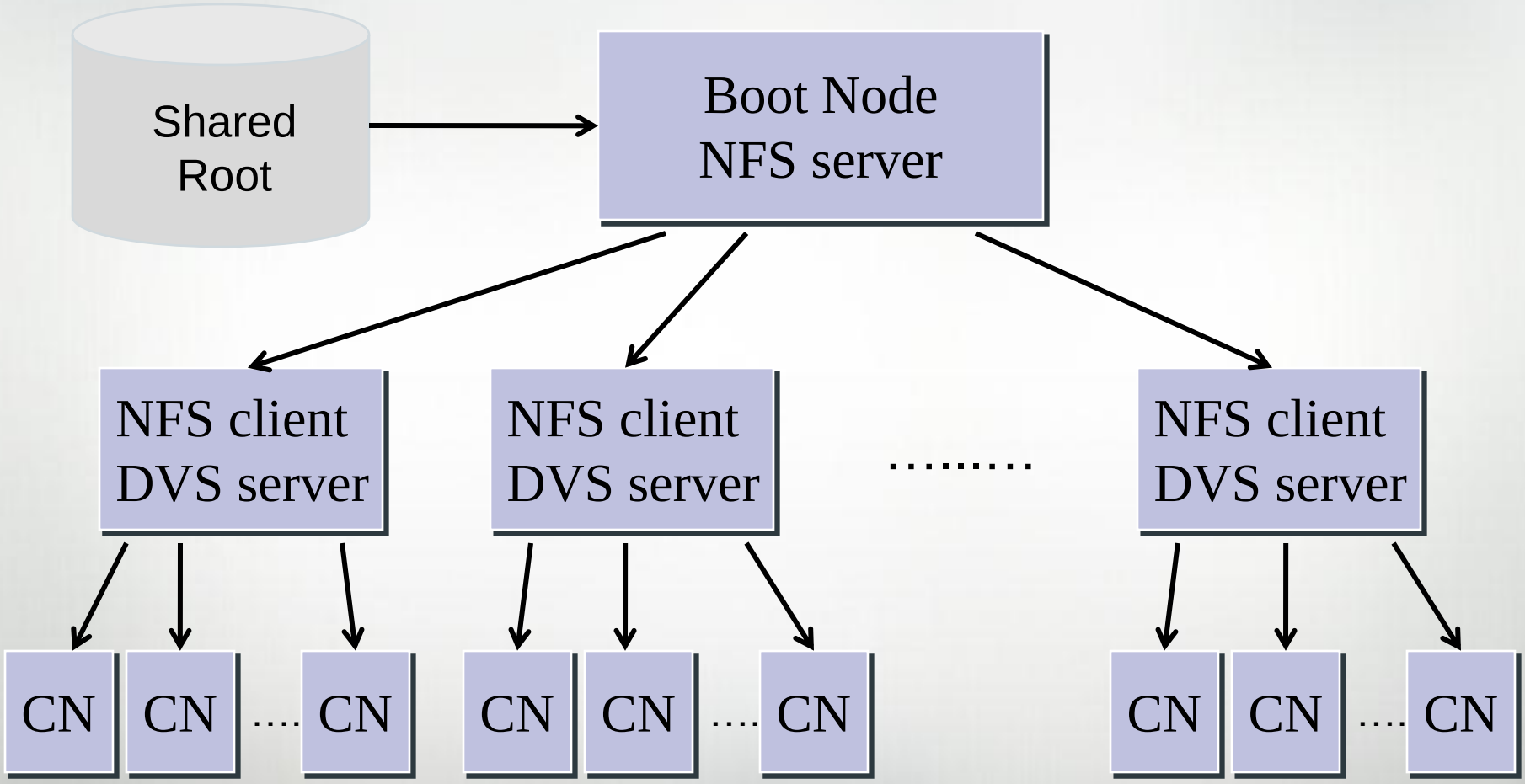
Advantages of Dynamic Shared Libraries

- Dynamic shared libraries are shared in physical memory across processes
  - More important as the number of cores increases
- Medium memory model is supported
  - Allows data sections to be larger than 2 gigabytes
- Use of dynamic linking routines (*dlopen*) supported
- Support of more tools, such as Valgrind
- Dynamically linked executables are smaller in size

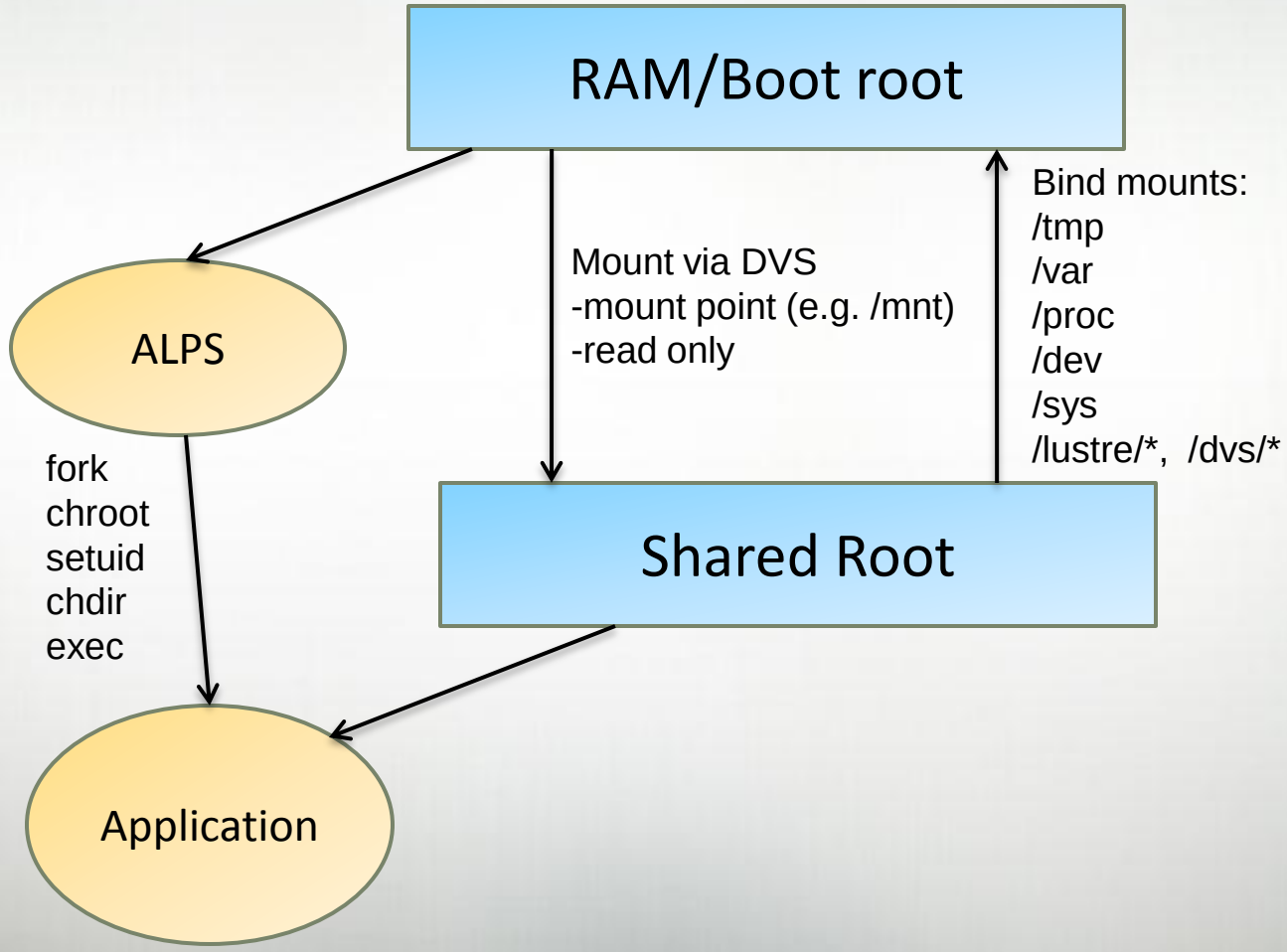
Disadvantages of Dynamic Shared Libraries

- Performance versus statically linked codes
  - Linking process performed on every run
  - Looking up symbols in dynamic library is less efficient
  - Potential “jitter” from loading a library
- CrayPat 5.0 analysis tool does not currently support dynamic shared libraries
  - Feature is planned for a future release
- Statically linked executables are guaranteed to run the same library code across all systems.

Cray XT DVS Shared Root Solution



Compute Node Application Startup



Goal of the Cray PE DSO Implementation

- Support both the existing Cray Programming Environment model and the Linux model of handling libraries
  - Continue to allow library versions to be chosen at build time
  - Provide accessibility to many library versions
  - Provide specific Programming Environment libraries for each compiler.
  - Allow library versions to be easily chosen at execution time
  - Simple process for running non-Cray XT built executables on the Cray XT.

Building a Dynamically Linked Executable

- Same process is used for all Cray XT Programming Environments: PGI, Pathscale, GCC
- The compiler scripts in xt-asyncpe 2.5 support:
  - Building a dynamically linked executable
  - Building a dynamic shared library
- Use of modulefiles determine your build environment
- Cray Compiler Environment (CCE) 7.1 will not support dynamic libraries
  - Feature is planned for a future release

Cray XT Library Releases Supporting DSOs

- **MPT 3.2**
- **LibSci 10.3.4**
- **FFTW 3.2.1**
- **ACML 4.0**
- **PETSc 3.0.0.2**
- **Hdf5_netcdf 1.2** - HDF5 1.8.2, netCDF 4.0.0
  - Parallel HDF5 and netCDF Parallel HDF5 do not provide a build for dynamic libraries
- **Libfast_mv 1.0.3**

Linking a Dynamically Linked Executable

- Compiler script ***–dynamic*** option:
  - **$ ftn –dynamic hello_world.f**
- The ***–dynamic*** option uses the **ld *–rpath*** option to add the exact location of the libraries to the program's ELF header
- The ***–static*** option can be used to explicitly specify a static build
  - Default behavior is static, but could change in the future
- The environment variable **XTPE_LINK_TYPE** can be set to **dynamic** or **static** to create a default setting for the shell
- **ldd** utility shows the shared libraries required by the executable
- **readelf –d a.out** shows the libraries used to build the executable

Sample ldd output

$ ldd a.out

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)

$

Dynamic Shared Library Search Order

- The dynamic linker searches for a library in the following order:
  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*

Running a Dynamically Linked Executable

- *** The Cray Programming Environment modulefiles initialize the LD_LIBRARY_PATH environment variable ***
- To execute using the build libraries, unload PrgEnv module and/or unset LD_LIBRARY_PATH
- To choose runtime library versions, load appropriate PrgEnv module
  - Need to know compiler used to build the application. If built with PGI, then *module load PrgEnv-pgi*
  - For batch jobs, the PrgEnv modulefile needs to be loaded in the batch script, or the batch job needs to be submitted with the *qsub –V* option
- Dynamic linker environment variables
  - **LD_BIND_NOW** - Load all dynamic libraries at startup
  - **LD_DEBUG** -Set LD_DEBUG=help for a list of options

Case 1: Using the build libraries

$ **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

$

Case 2: Configuring library versions at runtime

$ **module load Base-Opts PrgEnv -pgi**

$ **module swap xt-mpt xt-mpt/3.2.0**

$ **echo $LD_LIBRARY_PATH**

/opt/xt-pe/2.1.50HD_PS06/lib:/opt/mpt/3.2.0.1/xt/mpich2-pgi/lib:/opt/mpt/3.2.0.1/xt/util/lib:/opt/mpt/3.2.0.1/xt/pmi/lib:/opt/mpt/3.2.0.1/xt/sma/lib:/opt/xt-libsci/10.3.4/pgi/lib:/opt/pgi/8.0.5/linux86-64/8.0/libso:/opt/pgi/8.0.5/linux86-64/8.0/lib:/opt/xt-os/2.1.50HD_PS06/lib:/opt/xt-libc/2.1.50HD_PS06/amd64/lib y/xt-asyncpe/2.5.4/bin/ftn: INFO: linux target is being used

$ **aprun -n 1 ./hello_world**

MPI VERSION : CRAY MPICH2 XT version 3.2.0-pre (ANL base 1.0.6)

BUILD INFO : Built Mon Apr 6 18:24:16 2009 (svn rev 7351)

Hello World from rank 0

Application 2466917 resources: utime 0, stime 0

$

Running an ISV executable

- Determine the build compiler, then load the appropriate PrgEnv modulefile
  - Fortran and C++ libraries are compiler specific
  - For Fortran, need to find the compiler's run time libraries
    - PGI example: *libpgftnrtl.so*
    - GCC gfortran example: *libgfortran.so*
    - Pathscale example: *libpathfortran.so*
- If a batch job, either the modulefile needs to be loaded in the batch script or the job must be submitted with *qsub –V* option
- Set the LD_LIBRARY_PATH to point to any dynamic libraries included with the application.

Real life ISV Examples

- **CD-adapco STAR-CD** and **Exa Powerflow** have been executed on a Cray XT using executables provided by the vendor.
- Both of the applications were built with MPICH1, so both of the vendors needed to provide a special version of the application built with a MPICH2 library.
- The DVS shared root solution had the benefit of fully supporting the getpw routines (*getpwuid*, *getpwnam*) that were used by the applications.

Building a Dynamic Shared Library

- Program can contain both dynamic and shared libraries.
  - Don't need to convert existing application libraries
- Cray XT compilers support both the *–fpic* and *–fPIC* options.
- Compiler scripts support the *–shared* option for creating a dynamic shared libraries
- Library should be placed in a compute node accessible directory (i.e. lustre).
- Set LD_LIBRARY_PATH to point to the library directory at runtime, or use *ld –rpath* option to store library directory in executable

Building a Dynamic Shared Library

$ **cc -c -fpic test.c**

/opt/cray/xt-asyncpe/2.5.4/bin/cc: INFO: linux target is being used

$ **cc -shared test.o -Wl,-soname=libtest.so.0 -o libtest.so.0.0**

/opt/cray/xt-asyncpe/2.5.4/bin/cc: INFO: linux target is being used

$ **/sbin/ldconfig -n .**

$ **ln -s libtest.so.0.0 libtest.so**

$ **cc –dynamic –L<library-directory> -ltest –Wl,-rpath=<library-directory> main.c**

/opt/cray/xt-asyncpe/2.5.4/bin/cc: INFO: linux target is being used

$

Differences from Linux model support of DSOs

- Libraries not placed in one directory location
  - Cray PE libraries have a version for each supported compiler. For example, there is a separate *libmpich.so* for the PGI, Pathscale, and GCC compilers.
  - Linux usually only keeps the latest major version of each library; Cray makes multiple versions available
- The library versions used during the build process are preserved in the executable's ELF header
- The use of modulefiles provides an easy interface to modify the LD_LIBRARY_PATH environment variable

Open Opportunities

- Measure the performance differences of statically linked and dynamically linked codes.
- Provide default versions of Programming Environment libraries
  - Possibility that the GCC version of the *libmpich.so* library is sufficient to use with the majority of applications.
  - Use of the ***/etc/ld.so.conf*** file and ***ldconfig*** command to point to Cray XT Programming Environment dynamic shared libraries.
- Procedure for supporting user and application provided dynamic shared libraries (i.e. */usr/local/lib*)
- Support of common Linux cluster dynamic shared libraries (MPICH1, HP-MPI)

Conclusion

- The Cray XT Programming Environment implementation of dynamic shared libraries provides support for building and executing dynamically linked codes. It also has the following features:
  - Allows the user to run the executable with the build time programming environment libraries
  - Supports specific programming environment libraries for each of the compilers
  - Provides a method to easily choose a version of the programming environment libraries at runtime


CRAY
THE SUPERCOMPUTER COMPANY