This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
Introduction

The intent of this guide is to ease the transition from CHAOS 3 to CHAOS 4. It covers many of the issues and questions brought up by those who ported their codes to CHAOS 4 during its initial implementation.

CHAOS 4 enables the rollout of many Lustre and system management enhancements that should make Lustre and the clusters more stable and easier to upgrade and allow Livermore Computing (LC) to more easily track security enhancements as they are released. Most of the enhancements, while benefiting everyone, are fairly invisible to the users. The enhancements and changes that are apparent to the users are a new /usr/local organization, newer versions of most software, the move of the X11 libraries to /usr/lib64, and an older default Python version (2.4.3). CHAOS 4 is based on Red Hat Enterprise Linux 5. Information specific to both Yana and Hopi is also included in this updated edition of the porting guide.

Rebuilding Required for CHAOS 4

Most CHAOS 3 executables will not work on CHAOS 4, and it is also likely that Makefiles and build systems will have to be tweaked slightly. We strongly recommend rebuilding everything under CHAOS 4 to ensure compatibility. CHAOS 4 brings two new SYS_TYPES—chaos_4_x86_64 ib and chaos_4_x86_64—to help you manage your executables.

The method by which /usr/local, /usr/local/tools, /usr/local/bin, and dotkits are being managed and organized has changed significantly for CHAOS 4. There will now be a separate /usr/local for each cluster, thus allowing LC to roll out /usr/local changes a cluster at a time rather than affecting all clusters at once. In order to manage this effectively, we are packaging most of the software in /usr/local, which has resulted in a complete reorganization of /usr/local and thus may require path tweaks in your Makefiles and build systems for libraries and compiler executables. Hints on how to tweak your Makefiles in response to these changes are this guide’s primary purpose.

Compiler Changes

The default version of the various compilers advances with the release of CHAOS 4:

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Path</th>
<th>Default Version</th>
<th>Previous Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU</td>
<td>gcc/g++/gfortran</td>
<td>4.1.2</td>
<td>3.4.4</td>
</tr>
<tr>
<td>Intel</td>
<td>icc/icpc/ifort</td>
<td>9.1.052</td>
<td>9.1.033</td>
</tr>
<tr>
<td>PGI</td>
<td>pgcc/pgCC/pgf77/pgf90/pgf95/pgfhpf</td>
<td>7.0.6</td>
<td>6.23</td>
</tr>
<tr>
<td>Pathscale</td>
<td>pathcc/pathCC/pathf90</td>
<td>3.1</td>
<td>2.9.99</td>
</tr>
</tbody>
</table>
The default version of the compilers (and other software) can be determined by running the command `dpkg-defaults`. In the output, the version with the * (asterisk) in front of it indicates the current default version.

All compiler versions supported are now installed in `/usr/local/bin` with an embedded version in their name (i.e., `/usr/local/bin/mpiicc-10.1.011`). By using a named compiler version, you can prevent dotkits and default version updates from changing the compiler your code is built with. We have removed `/usr/local/tools/compilers` under CHAOS 4 because it is no longer necessary. We have also switched to full version numbers (10.1.011 instead of 10.1) in order to allow us to install new minor compiler versions without potentially affecting current users.

All compiler versions supported are also now available via dotkits (e.g., `use ic-10.1.011`). By using dotkits, you can change the version of the compiler picked up when you invoke just the compiler name (i.e., `mpiicc` or `icc` will invoke icc version 10.1.011 after executing `use ic-10.1.011`). To find the versions available for a compiler, we recommend running:

```
use -l <compiler>
```

where `-l` is a lowercase L. That is `use -l icc` for the installed versions of icc (e.g., ic-9.1.052, ic-10.0.025, and ic-10.1.011 at the time this was written). The compilers we have installed are from GNU (gcc/g++/g77/gfortran), Intel (icc/icpc/ifort), PGI (pgcc/pgCC/pgf77/pgf90/pgf95/pgfhpf), and Pathscale (pathcc/pathCC/pathf90).

We recommend you use the same compiler on CHAOS 4 as you did on CHAOS 3. If you used Intel 9.1 on CHAOS 3, we recommend you stay with Intel 9.1 on CHAOS 4 unless you are using the `-zero` option, as described in “Known Intel Compiler Issues” below. For the Pathscale, PGI, and GNU compilers, you may have to go to a newer compiler version (some of the older versions are not compatible with CHAOS 4).

### Known Intel Compiler Issues

For Intel compilers 10.0 and greater (10.0.025 and 10.1.011), it is recommended that the compiler flag `-nolib_inline` be added in order to get consistent results with Intel 9.1 compiled code. This compiler flag prevents the inlining of intrinsic functions, which we have found to cause numeric differences in several codes.

If you need to use the `-zero` flag with ifort, we recommend using version 10.1.0.11 under CHAOS 4 rather than the current 9.1 or 10.0 compilers. We have found that the compiler flag `-zero` can cause an error of the form “multiple definition of 'var$485'” while linking. We are working with Intel to resolve this issue.

### Known TotalView Issues

TotalView may hang during start-up or rerun of a threaded code, for example, `srun`. If this happens, `^C totalview` and `scancel` the job step, then start a new TotalView session.
Man Pages for Specific Compiler Versions

In order to get a man page specific to a compiler version (e.g., Intel 10.1.011), you either must use the package with that compiler in it (i.e., use ic-10.1.011) first before doing the man commands (e.g., man ifort), or you need to run man on <compiler>-<version> (e.g., man ifort-10.1.011). Otherwise, you will get the man page for the “default” version of each compiler, which may have different compiler options than other versions.

Library Location Changes

Most of the user local packages will be located in /usr/local/tools/[package_name]-[version]. This should make it easier to determine what software is installed on a machine and what versions are available. The utility dlocate can be used to quickly find files in these packages (e.g., dlocate mkl.so lists the packages and directories that the mkl library is installed in). Of course, findentry is still available and is very useful for finding libraries and missing symbols (see “Finding Symbols and Libraries with Findentry”).

Commonly asked about library locations that have changed in CHAOS 4:

<table>
<thead>
<tr>
<th>Library</th>
<th>New Location</th>
<th>Old Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKL</td>
<td>/usr/local/tools/mkl-8.1.1.004/lib, etc.</td>
<td>/usr/local/intel/mkl*)</td>
</tr>
<tr>
<td>ACML</td>
<td>/usr/local/tools/acml-ifort/lib, etc.</td>
<td>/usr/local/intel/acml, etc.</td>
</tr>
<tr>
<td>MPI</td>
<td>/usr/local/tools/mvapich-intel/lib, etc.</td>
<td>/usr/lib/mpi</td>
</tr>
<tr>
<td>X11</td>
<td>/usr/lib64</td>
<td>/usr/X11R6/lib64</td>
</tr>
</tbody>
</table>

On Linux you often also need to specify a run-time path (rpath) for shared libraries. This can be done automatically with an LC-specific command -Wl,--auto_rpath added to your link line. The manual/portable way to specify an rpath is -Wl,-rpath,<libpath> added to the link line. For example, for the MKL, in addition to adding -L/usr/local/tools/mkl-8.1.1.004/lib -lmkl -lpthread to the link line, you also need to add -Wl,-rpath,/usr/local/tools/mkl-8.1.1.004/lib to the link line to tell it where to find the mkl shared objects at run time (or you can add -Wl,--auto_rpath to have the linker do this for you).

For more details, see “Verifying Correct RPATHs for Shared Libraries.”

Python Version Changes

The default version of Python on CHAOS 4 is 2.4.3 (it was 2.5 on CHAOS 3). This change is necessary to avoid breaking Red Hat Python scripts and tools that depend on finding Python 2.4.3 first in PATH (such as “meld”). Python 2.5 will remain accessible in /usr/local/bin/python2.5 and also through the /usr/apps/python and /usr/apps/python2.5 directories (as well as the python25 dotkit, use python25).
**MPI Changes**

CHAOS 4 includes a new MPI (099_nodreg) and MPI shared library organization that will allow the same executable to run on IB cluster machines (i.e., Zeus, Rhea) and standalone nodes (i.e., Yana, Hopi), once everything is upgraded to CHAOS 4 and uses Moab.

We strongly recommend using our MPI wrapper script to build your code, but we realize that some codes do not for various reasons. If you do not use the MPI wrappers scripts but want to know what we add (which changes with every MPI update), run the MPI wrapper with `-show`. For example, `/usr/local/bin/mpiicc-9.1.052 -show` outputs the compile and link commands needed by MPI:

**Build Line:**

```
icc-9.1.052 -i-dynamic -Wl,-rpath,/usr/local/tools/mvapich-intel/lib/shared
-DUSE_STDARG -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_UNISTD_H=1
-DHAVE_STDARG_H=1 -DUSE_STDARG=1 -DMALLOC_RET_VOID=1
-I/usr/local/tools/mvapich-intel/include
-L/usr/local/tools/mvapich-intel/lib/shared -L/usr/lib64
-L/usr/local/tools/mvapich-intel/lib -lmpich
```

**Note:** If your environment or script sets the environment variable MPICH_ROOT, this will affect the `-show` output and the behavior of our MPI compiler wrappers, possibly giving you unexpected results. We have changed the location of all the MPI libraries and we recommend unsetting MPICH_ROOT to get the new default location for your MPI libraries.

**Porting Hints**

The remaining information, while not new to CHAOS 4, is often very useful when porting to a new OS version.

**Finding Symbols and Libraries with Findentry**

LC’s `findentry` is often useful when porting to a new system. To find a library, run `findentry with -l libname`:

```
> findentry -l X11
<snip>
Mar 22 2007 /usr/lib64/libX11.so
also known as: /usr/lib64/libX11.so.6.2.0
also known as: /usr/lib64/libX11.so.6
```

**Note:** If you omit the space between `-l` and `libname` (a common issue), `findentry` will not find anything.
To find which libraries have a symbol, put the symbol name (without the `-l`) on the command line:

```
> findentry fast_lock
<snip>
entries found:
/usr/local/tools/icc-10.0.025/lib/libintlc.so.5:__fast_lock
/usr/local/tools/icc-10.0.025/lib/libirc.so:__fast_lock
/usr/local/tools/icc-10.0.025/lib/libirc.a[imalloc.o]:__fast_lock
/usr/local/tools/icc-10.1.011/lib/libintlc.so.5:__fast_lock
/usr/local/tools/icc-10.1.011/lib/libirc.so:__fast_lock
```

**Mixing C and FORTRAN**

If you are linking C/C++ and FORTRAN code together and need to explicitly specify the FORTRAN (or C/C++) libraries on the link line, we recommend compiling a “hello world” program in that language (i.e., FORTRAN) with `-v` to see what the compiler links in. For example:

```
> ifort-10.1.011 -v hello.f
<snip>
ld /usr/lib/gcc/x86_64-redhat-linux/4.1.2/../../../../lib64/crt1.o
/usr/lib/gcc/x86_64-redhat-linux/4.1.2/../../../../lib64/crti.o
/usr/lib/gcc/x86_64-redhat-linux/4.1.2/crtbegin.o --eh-frame-hdr
 -dynamic-linker /lib64/ld-linux-x86-64.so.2 -o a.out
/usr/local/tools/ifort-10.1.011/lib/for_main.o -rpath
/usr/local/tools/ifort-10.1.011/lib /var/tmp/gyllen/ifortWOp937.o
-L/usr/local/tools/ifort-10.1.011/lib
-L/usr/lib/gcc/x86_64-redhat-linux/4.1.2/
-L/usr/lib/gcc/x86_64-redhat-linux/4.1.2/../../../../lib64 -Bstatic -lifport
-lifcore -lifm -lsvml -Bdynamic -lm -Bstatic -lipgo -lirc -Bdynamic -lc
-lgcc_s -lgcc -Bstatic -lirc -Bdynamic -ldl -lc
/usr/lib/gcc/x86_64-redhat-linux/4.1.2/crtend.o
/usr/lib/gcc/x86_64-redhat-linux/4.1.2/../../../../lib64/crtn.o
```

From the `-v` output above, it appears that

```
```

and

```
-Bstatic -lifport -lifcore -lifm -lsvml -Bdynamic
```

are a good starting point for linking an Intel 10.1 compiled FORTRAN library to a C/C++ program. The `-rpath` command is needed to pick the right library at run time, and we need to add `-Wl,` and a comma to the command to get it through the compiler (see “Verifying Correct RPATHs for Shared Libraries” for more details). The `-Bstatic ... -Bdynamic` options indicate that the Intel FORTRAN libraries should be linked in statically, and the trailing `-Bdynamic` allows the other libraries to be shared libraries.
Thus, an Intel 10.1.011 link line with a C++ main and a FORTRAN subroutine would look like:

```
icpc-10.1.011 -Wl,-rpath,/usr/local/tools/ifort-10.1.011/lib
-L/usr/local/tools/ifort-10.1.011/lib -o mixed_hello c++_main.o
fortran_hello.o -Bstatic -lfiport -lifcore -llmf -lsvml -Bdynamic
```

If the above link line generated errors about unresolved externals, one would need to add other libraries, etc., from the link line shown with `-v`, like `-Bstatic -lipgo -lirc -Bdynamic` and `-Bstatic -lirc_s -Bdynamic` until those errors were resolved. In this case, `icpc-10.1.011` already added these options (as seen by running `icpc-10.1.011 -v hello.cc` and looking at the link line).

**Verifying Correct RPATHs for Shared Libraries**

The common use of shared libraries on Linux provides many benefits, but if the application developer is not careful, they can also be a source of vexing problems. The most common shared library problems are: (1) not finding the shared libraries at run time (preventing the application from running at all), and (2) the much worse case of silently picking up different (and possibly incompatible) shared libraries at run time. This section recommends ways to ensure that your application finds and uses the expected shared libraries.

These shared library problems can occur more often on LC systems than on stand-alone Linux systems because LC often installs many different versions of the same compiler or library in order to give users the exact version they require. Although Linux provides methods for differentiating shared library versions, many of these compilers and libraries do not use this technology. As a result, on LC systems, there can be several shared libraries with exactly the same name that are actually different from, and possibly incompatible with, each other.

In order to make shared library version errors as visible as possible (i.e., dying at startup versus just silently getting the wrong library), LC intentionally put no LC-specific paths in the default search path for shared libraries (e.g., in `/etc/ld.so.conf`). Our compilers and MPI wrappers have been modified to automatically include the appropriate rpaths (run-time paths) for those shared objects the compilers or MPI automatically include. For all other shared libraries that your code links in that are not in `/usr/lib64`, you probably need to specify an rpath for them.

Rpaths may be specified on the link line explicitly with one or more `-Wl, -rpath,<path>` arguments or you can use an LC-specific linker option `-Wl, --auto_rpath` to help with this. If you specify `-Wl, --auto_rpath` on your link line, all the `-L<path>` commands on the link line will automatically be added to your rpath, which is typically what is needed to access the proper shared library. It should be noted that the use of `-Wl, --auto_rpath` will encode all `-L` paths into your rpath, which may include paths LC does not control (such as `/usr/gapps`). *(Note: The `-Wl,` part of all these commands tells the compiler to pass the commands to the linker without interpretation, and the “,” after `-rpath` is replaced with a space.)*
If your rpaths are not set properly, at run time you may get an error of the form:

```
./mixed_hello: error while loading shared libraries: libifport.so.5: cannot open shared object file: No such file or directory
```

Although LD_LIBRARY_PATH can be used to specify where to search for shared objects, we strongly recommend encoding the paths you need into the executable instead, either by adding `-Wl,--auto_rpath` to your link line or by explicitly specifying paths with `-Wl,-rpath,<path>`. By encoding the rpaths into the executable, you ensure that the executable will work as expected regardless of how LD_LIBRARY_PATH is set.

The RPATHs for an existing executable can be queried with

```
readelf -a <your_exe> | grep RPATH.
```

For example:

```
> readelf -a ./mixed_hello | grep RPATH
0x000000000000000f (RPATH) Library rpath:
[/usr/local/tools/icc-10.1.011/lib:/usr/local/tools/ifort-10.1.011/lib]
```

The SHARED LIBRARIES requested by an executable (or .so file) can be queried with

```
readelf -a <your_exe> | grep NEEDED.
```

For example:

```
> readelf -a ./mixed_hello | grep NEEDED
0x0000000000000001 (NEEDED) Shared library: [libm.so.6]
0x0000000000000001 (NEEDED) Shared library: [libstdc++.so.6]
0x0000000000000001 (NEEDED) Shared library: [libgcc_s.so.1]
0x0000000000000001 (NEEDED) Shared library: [libc.so.6]
0x0000000000000001 (NEEDED) Shared library: [libdl.so.2]
```

The ACTUAL SHARED LIBRARIES used by your executable can be queried with

```
ldd <your_exe>
```

This list usually is longer than the one above because shared libraries can pull in other shared libraries.

For example:

```
> ldd ./mixed_hello
libm.so.6 => /lib64/libm.so.6 (0x00002aaaaacc6000)
libstdc++.so.6 => /usr/lib64/libstdc++.so.6 (0x00002aaaaaf49000)
libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00002aaaab249000)
libc.so.6 => /lib64/libc.so.6 (0x00002aaab458000)
libdl.so.2 => /lib64/libdl.so.2 (0x00002aaab7a8000)
/lib64/ld-linux-x86-64.so.2 (0x00002aaaaaabb000)
```

If your rpath is not set properly, `ldd` will print put messages of the form:
The `ldd` output is useful in determining that your application can find all its shared libraries and that it is picking up the versions you expect.

**Yana and Hopi Specifics**

On Yana and Hopi, CHAOS 4 combined with Moab brings MPI application binary compatibility across all the Opteron clusters through the deployment of compatible cluster-specific MPI shared libraries. This means MPI applications built on Žeus will run on Yana, and conversely, MPI applications built on Yana will run on Žeus. (Custom MPI link lines may require some tweaking for MPI compatibility.)

Although `mpirun` may still be used, `srun` is now the recommended method for running MPI applications on Yana and Hopi (thus matching the method used on other Opteron clusters, such as Zeus and Rhea). In addition, the use of `srun` allows pdebug nodes to be offered on Yana and Hopi in order to allow interactive debugging of MPI jobs. Yana will start with one pdebug node so that LC can evaluate its usefulness to Yana’s users.

**New MPI Vendor on Yana and Hopi**

With the update to CHAOS 4 on Yana and Hopi, the shared-memory MVAPICH-0.9.9 will replace MPICH-1.2.7 as the default MPI. This change allows MPI applications built on one of the CHAOS 4 InfiniBand-enabled Opteron clusters (Zeus, Atlas, Minos, Rhea) and linked to the default MPI’s shared library to run without modification on Yana and Hopi and vice versa. Users may no longer, therefore, need a separate application build for Yana/Hopi from Zeus/Rhea/etc. after the CHAOS 4 upgrade.

If you are rebuilding your MPI application on Yana or Hopi, be aware that rebuilding all code is required because the MPI header files between the two MPI vendors are not compatible.

**Known MPI Compatibility Issues between Serial and InfiniBand Clusters**

Using MPI compiler wrappers (such as mpiicc, mpipathcc, etc.) with the default MPI version is the best way to guarantee compatibility between serial clusters (Yana, Hopi) and Infiniband clusters (Zeus, Rhea). Cross-cluster compatibility for non-standard MPI versions may be possible but most likely will not be implemented unless specifically requested via the LC Hotline.

The most common cause of serial cluster MPI incompatibility is custom MPI link lines that link either to static MPI libraries or to InfiniBand-specific shared libraries, such as ibverbs or ibumad. To fix these MPI incompatibilities, make sure the MPI library directory ends with “/lib/shared” and not “/lib” and that `-libverbs` and `-libumad` are not specified. For example, to link to icc’s shared-library version of MPI, the following link should be used:

```
```
See Section “8c. Using MPI without MPI Wrapper Scripts” in /usr/local/docs/chaos4_x86_64.basics for more details on this topic.

**Use srun Instead of mpirun on Yana and Hopi**

Because SLURM is now available, MPI jobs can be launched on Yana and Hopi just as on Zeus and Atlas. In fact, while `mpirun` is still available, `srun` is now the recommended method to launch MPI jobs on Yana and Hopi. With `srun`, use `-n #` to specify number of MPI tasks (instead of `-np #` with `mpirun`). For example, to launch mpiBench with two MPI tasks, use:

```
srun -n 2 ./mpiBench
```

In batch scripts, the `srun -n #` option can be omitted when using all the requested processors for MPI. For example, if `-np 4` is used in a psub script, `srun ./mpiBench` will run on all four processors allocated by the batch script.

For interactive debugging of MPI applications using the pdebug pool (i.e., outside of a batch script), pass `-p pdebug` to `srun`. For example, to run a two-processor MPI job (mpiBench) in the pdebug pool, use:

```
srun -p pdebug -n 2 ./mpiBench
```

Also, TotalView sessions may be launched in the same way as on other clusters in the pdebug pool:

```
totalview srun -a -p pdebug -n 2 ./mpiBench
```

Currently `srun` cannot be used to run MPI jobs on the login node, so `mpirun` must be used in this case. LC is working to restore the ability to run serial MPI jobs on Yana without mpirun (or srun) and will release this feature as soon as it is fully implemented (early July 2008).

More details on using `srun` (and Moab) on Yana and Hopi can be found online in “Using Moab on LC’s Yana and Hopi Clusters” at https://computing.llnl.gov/jobs/moab/yanahopiMPI.pdf.

**Using Dotkits**

In CHAOS 4, many software items are packaged using Dotkit. If you do not find the necessary tool or software version, you may need to use a Dotkit to add it to your path.

To find the appropriate Dotkit, use the `use -l` command for a list of all Dotkit packages and `use <package name>` to use one of the Dotkit packages. The Dotkit will update your current login session to put the specified tools into your default path.

If you want to add the use of a Dotkit to your login files (i.e., `.cshrc`, `.profile`, etc.) it is very important to use the quiet mode, e.g., `use -q`, otherwise, it may interfere with the execution of other commands, such as `xemacs` and `scp`. (Outputting text to the screen in your login files in general will also cause the same problems.)
Detailed Dotkit usage information is available at https://computing.llnl.gov/?set=jobs&page=dotkit and in Section “15. Dotkit on the LC Systems” in /user/local/docs/chaos4_x86_64.basics.