Installing ALBANY/LCM and TRILINOS on Fedora 22/23

The ALBANY/LCM team*

November 24, 2015

1 Introduction

This document describes the necessary steps to install ALBANY/LCM and TRILINOS on a machine with Fedora Linux. The procedures described herein were tested using Fedora 22 and Fedora 23. If you want a shortcut obtain the script `install_albany.sh` (stored in `Albany/doc/LCM/install`) and then try

```
./install_albany
```

This will install and build TRILINOS and ALBANY in the current directory. If the script does not complete it will tell you why and with help from this document you will be able to complete the install.

2 Required Packages

The following packages should be installed using the `dnf` command

```
blas
blas-devel
lapack
lapack-devel
openmpi
openmpi-devel
netcdf
netcdf-devel
netcdf-static
netcdf-openmpi
netcdf-openmpi-devel
hdf5
hdf5-devel
hdf5-static
hdf5-openmpi
hdf5-openmpi-devel
boost
boost-devel
boost-static
boost-openmpi
boost-openmpi-devel
matio
matio-devel
cmake
gcc-c++
git
hwloc-libs
hwloc-devel
environment-modules
```

For example, to install the first package you should type

---

*Original version by Julián Rímoli

```
sudo dnf install blas
```

Make sure that all these packages are installed, specially if you create a script to do so. If a package is not installed because of a typo then the compilation will fail.

Optional but strongly recommended packages:

```
clang
clang-devel
gitk
```

3 Repository Setup with GitHub

In a web browser go to www.github.com, create an account and set up ssh public keys. If you require push privilges for ALBANY, email Glen Hansen at gahanse@sandia.gov and let him know that. On the other hand, if you require push privileges for TRILINOS, it is best if you contact the TRILINOS developers directly. Go to www.trilinos.org for more information.

It is strongly recommended that you join the AlbanyLCM Google group to receive commit notices. Go to groups.google.com/forum/#!forum/albanylcm and join the group. You can also browse the source code at github.com/gahansen/Albany.

4 Directory Structure

In your home directory, create a directory with the name LCM:

```
mkdir LCM
```

Change directory to the newly created one:

```
cd LCM
```

Check out the latest version of TRILINOS, which is hosted now on GitHub:

```
git clone git@github.com:trilinos/Trilinos.git Trilinos
```

Finally, check out the latest version of ALBANY:

```
git clone git@github.com:gahansen/Albany.git Albany
```

At this point, the directory structure should look like this:

```
LCM
  |- Albany
  |- Trilinos
```

5 Environment Variables

In ~/.bashrc, the following variables are needed:

```
export LCM_DIR=~/LCM
export MODULEPATH=$LCM_DIR/Albany/doc/LCM/modulefiles
```

The LCM_DIR variable should contain the location of the top-level LCM directory.

6 Installation Scripts

Create symbolic links to the installation scripts inside the directory LCM/Albany/doc/LCM/build to the top-level LCM directory. If you intend to modify these scripts, it is better to copy them. The necessary scripts are:

```
albany-config.sh
build-all.sh
build.sh
env-all.sh
env-single.sh
trilinos-config.sh
```

Once this is done, go to the top-level `LCM` directory, open the `env-all.sh` and `env-single.sh` scripts and make sure they match your environment. If you do not want to tinker with any of this, just change the email addresses for tests reports at the end of `env-single.sh` and make sure all the scripts are executable and read only:

```
cd ~/LCM
chmod 0555 *.sh
```

The `build.sh` and `build-all.sh` scripts perform different actions according to the name with which they are invoked. This is accomplshed by creating symlinks to build.sh and using them to run it. For example:

- clean.sh will delete all traces of the corresponding build and will create a new configuration script based on the corresponding template.

- config.sh will attempt to reconfigure the build.

- build.sh (original name) will build using cmake.

- test.sh will run the cmake tests.

- mail.sh will mail the results of the ctest to the email address configured in env-single.sh.

- symlinks with combinations of the above (e.g. clean-config-build.sh) will perform the specified actions in sequence. See build.sh for valid sequences.

For example, the following symbolic links will create separate commands for clean up, configuring and testing:

```
ln -s build.sh clean.sh
ln -s build.sh config.sh
ln -s build.sh test.sh
```

They could also be combined for convenience:

```
ln -s build.sh clean-config.sh
ln -s build.sh clean-config-build.sh
ln -s build.sh clean-config-build-test.sh
ln -s build.sh config-build.sh
ln -s build.sh config-build-test.sh
```

There is also a script `LCM/Albany/doc/LCM/install/albany-lcm-symlinks.sh` that will create the appropriate symbolic links.

It is recommended that at least the *-config.sh be made read and execute only, as erroneous modifications to the build.sh script may result in their being overwritten.

7 Modules

Modules are used to create different environments for the configuration and compilation of both ALBANY and TRILINOS. To see the available modules that correspond to different thread models, compilers and build types:

```
module avail
```

This results in something like:

```
------------------ /home/amota/LCM/Albany/doc/LCM/modulefiles ------------------
openmp-gcc-debug     pthreads-gcc-release serial-gcc-debug
openmp-gcc-profile   pthreads-gcc-small   serial-gcc-profile
openmp-gcc-release   serial-clang-debug   serial-gcc-release
openmp-gcc-small     serial-clang-profile serial-gcc-small
pthreads-gcc-debug   serial-clang-release
pthreads-gcc-profile serial-clang-small
```

The naming convention for the modules follows the pattern

```
[thread model]-[toolchain]-[build type]
```

The `[thread model]` option refers to the thread parallelism model that the code will use by means of the KOKKOS package in TRILINOS. Currently the sopported models are: `serial` that works for all supported compilers, `openmp` that works with the `GCC` and `Intel` toolchains, `pthreads` that works for all supported compilers, and `cuda` that is only supported for the `GCC` toolchain. The installation and configuration of the Cuda framework is complex. Much more detailed information can be found at `http://developer.nvidia.com/cuda`.

Currently the options for `[toolchain]` are `gcc`, `clang` and `intel` if the Intel compilers are installed, and for `[build type]` are `debug` (includes symbolic information), `release` (optimization enabled), `profile` (symbolic information and optimization enabled for profiling) and `small` (minimizes size of executables). The `clang` toolchain requires installation of the `clang` and `clang-devel` packages.

Build directories are created within the LCM top-level directory and named according to the loaded module and package specified to the build.sh script, e.g.:

```
albany-build-gcc-release
```

In addition, for TRILINOS an install directory similarly named is created at the LCM top-level directory.

8   Configuring and compiling

Assuming that we want to compile with a `serial` thread model using the `gcc` tool chain in `debug` mode, load the appropriate module:

```
module load serial-gcc-debug
```

Now first configure and compile TRILINOS. Within the top-level `LCM` directory type:

```
./config-build.sh trilinos [# processors]
```

For example, if you want to build using 16 processors, type:

```
./config-build.sh trilinos 16
```

Finally, repeat the procedure for ALBANY:

```
./config-build.sh albany [# processors]
```

For example, if you want to build a version of the code using 16 processors, type:

```
./config-build.sh albany 16
```

Note that to compile a version of ALBANY with a specific thread model, toolchain and build type, the corresponding version of TRILINOS must exist.

9   After Initial Setup

The procedure described above configures and compiles the code. From now on, configuration is no longer required so you can rebuild the code after any modification by simply using the `build.sh` script. For example:

```
./build.sh albany 16
```

There are times when it is necessary to reconfigure, for example when adding or deleting files under the `LCM/Albany/src/LCM` directory. This is generally anounced in the commit notices.

Also, note that both TRILINOS and ALBANY are heavily templetized C++ codes. Building the debug version of ALBANY requires large amounts of memory because of the huge size of the symbolic information required for debugging. Thus, if the compiling procedure stalls, try reducing the number of processors.

10   Running and Bebugging LCM

After building ALBANY, you might want to run and/or debug the code. Tools were built in TRILINOS (decomp, epu, etc.) that are necessary for parallel execution. The environment created by loading the appropriate module sets the proper paths so that the executables that correspond to the type of build are accessible.

11   Commiting Changes and Code Style

ALBANY is a simulation code for researchers by researchers. As such, vibrant development of new and exciting capabilities is strongly encouraged. For these reasons, don't be afraid to commit changes to the master git repository. We only ask that you don't break compilation or testing. So please make sure that the tests pass before you commit changes.

In addition, within LCM we strongly encourage you to follow the C++ Google style guide that can be found at `http://google-styleguide.googlecode.com/svn/trunk/cppguide.html`. This style is somewhat different to what is currently used in the rest of ALBANY, but we believe that the Google style is better in that it advocates more style differentiation between the different syntactic elements of C++. This in turn makes reading code easier and helps to avoid coding errors.