Installing compilers on a Linux system

September 7, 2010

All Linux distributions will provide the option of installing C, C++, and Fortran compilers from the Gnu Compiler Collection (GCC). Commercial compilers from Intel, the Portland Group, and others are also available for a price, but we'll concentrate on the GCC compilers which are open source and available for free.

When working with "bleeding edge" software it's a good idea to be able to build with different compiler versions. It's possible to maintain multiple compiler versions on a single computer, but you need to set things up carefully to avoid accidental inconsistencies between compilers and libraries.

1 Installing the distribution's default compiler

This part is easy: Use your distribution's package manager to install the compilers `gcc`, `g++`, and `gfortran`. If your machine is 64 bit, install support for both 64 and 32 bit. You should also install the development tools `make` and `m4` and the `gdb` and `ddd` debuggers.

Most distributions will put the compiler's executable in `/usr/bin`. You can locate the compiler by running

```
which gcc
```

which will respond by giving the full pathname of the compiler. No matter how many other compilers you install, you can always run the default compiler by giving the full pathname, *e.g.*, `/usr/bin/gcc` instead of `gcc`.

2 Installing an alternate compiler

2.1 Prerequisites

Check the GCC installation documentation to see what other software must be installed before building a compiler. Currently, the prerequisites are the MPFR, GMP, and MPC multiple-precision libraries.

If MPFR and GMP are not already installed, you'll need to do so yourself. Get the source code. You'll install into `/usr/local`. To get the dependencies right, build GMP first, then MPFR, then MPC.

See the online installation information for MPFR, GMP, and MPC if you need further information.

2.2 Installing the compiler

If you're maintaining multiple compilers on your system, you need to install the compiler someplace where it won't conflict with other compilers. Suppose you are working with GCC version *x.y.z.* I recommend installing it in the directory

```
/usr/local/gcc-x.y.z
```

Specification of the installation directory is done with the `--prefix` option. From the build directory, do:

```
../gcc-x.y.z/configure --prefix=/usr/local/gcc-x.y.z --enable-languages=c,c++,fortran
  --with-mpfr=/usr/local --with-gmp=/usr/local --with-mpc=/usr/local
```

```
make
```

```
sudo make install
```

2.3 Post-installation configuration

2.3.1 Set your path to find the compiler

Edit your `.bashrc` or `.tcshrc` file to put the installation directory first in your path. Source the file to invoke the new path immediately.

2.3.2 Setting the runtime library path

A program built with shared libraries will load them at runtime. To do so, it needs to find them; it looks for them in the runtime library path, often called the **rpath**. You'll need to set the rpath for your new compiler.

It's important to understand that each compiler you install on your system will need its own rpath. With the `LD_LIBRARY_PATH` environment variable it is possible to set an runtime library path to be used by all executables. **Do NOT use LD_LIBRARY_PATH**. Doing so can cause mismatch between libraries and executables.

The syntax for setting the rpath varies from compiler to compiler. With the GCC compilers, the rpath is set in a `specs` file. The following bash script will create a `specs` file in the correct location and splice the correct rpath into it.

```
#!/bin/bash
# Patch the compiler specs file to set the search path for shared libraries
# You will need write permissions in the gcc installation directory, so
# you will probably need to run this as root.

rpathbase=`which gcc | sed 's|bin/gcc||'`
specpath=`gcc --print-file libgcc.a | sed 's|/libgcc.a||'`

if [ -d ${rpathbase}/lib64 ]; then
    rpath="${rpathbase}lib;${rpathbase}lib64"
else
    rpath="${rpathbase}lib"
fi

echo "rpath will be: " $rpath
echo "writing to " ${specpath}"/specs"

gcc -dumpspecs | sed '/\*link:/{
N
s|\*link:\n|\*link:\n%{!rpath: -rpath '$rpath'} |
}' > ${specpath}/specs
```

3 Checklist for installation of an alternate compiler

1. Make sure you are ready to build an alternate compiler
   (a) If you didn't install the default compiler for your distribution, do so now using your distribution's package manager.

(b) Check that `make` and `m4` have been installed. If not, do so now using your distribution's package manager.

(c) Check that you've installed GMP, MPFR, and MPC in `/usr/local`. If not:

i. Check the GCC website's installation instructions for the minimum required versions of GMP, MPFR, and MPC. Get the source tarballs from the project website.

ii. Configure GMP with `--prefix=/usr/local`, then make and sudo make install.

iii. Configure MPFR with the arguments `--prefix=/usr/local --with-gmp=/usr/local`, then make and sudo make install.

iv. Configure MPC with the argument
`--prefix=/usr/local --with-gmp=/usr/local --with-mpfr=/usr/local`
then make and sudo make install.

2. Get and unpack the source for your alternate compiler. I'll refer to the alternate as GCC x.y.z.

3. Make a build directory.

4. From within the build directory, run configure with arguments
`--prefix=/usr/local/gcc-x.y.z --with-gmp=/usr/local --with-mpfr=/usr/local --with-mpc=/usr/local --enable-languages=c,c++,fortran`

5. Run make, then sudo make install

6. Put `/usr/local/gcc-x.y.z` first in your path.

7. Run (with sudo) the script `patch-compiler-specs.sh` to set the RPATH for the new compiler.

8. Test that you can build and run "Hello, World" in C, C++, and Fortran.

4 Troubleshooting

4.1 Which compiler am I running?

Even when you've installed everything correctly, it's possible to get confused about which compiler you're running (particularly when running it indirectly through a wrapper such as `mpicc`.) Two tools can help: the `which` commmand and GCC's `--version` option. Running `which gcc` (or `which gfortran`, or `which g++`) will give the full pathname of the compiler. The `--version` option prints the version number; it can pass through a wrapper such as `mpicc` to tell which `gcc` is being invoked by `mpicc`.

Here are some situations in which things can go wrong:

- If you've installed an alternate `gcc` but not `gfortran`, the shell will find the default `gfortran` and use it along with your alternate `gcc`. This inconsistency will cause build failures at best, runtime failures at worst.
- If you're not careful when specifying the compilers to be used when building MPI, the MPI compiler wrappers `mpicc`, `mpif77`, and `mpicxx` can wrap inconsistent compilers. Use the `--version` flag to check.