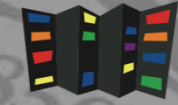


Best practices for installing application software

Martin Čuma, Anita Orendt
Center for High Performance Computing
University of Utah
m.cuma@utah.edu, *anita.orendt@utah.edu*



- Before installation considerations
- Community codes
- Commercial codes and licensing
- Building for multiple architectures
- Automatic building
- Application management
- Python and R



1. Download the talk slides

http://home.chpc.utah.edu/~mcuma/chpc/Codes_RMACC17.pdf

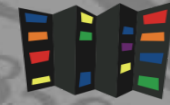
2. Get an user/password paper slip

3. Using terminal application (Mac terminal, PuTTY, GIT Shell)

- `ssh userxx@linuxclass.chpc.utah.edu`

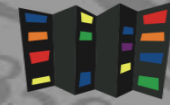
4. Make sure you can load the Python and R module

- `module load python; module load R`



- Community programs
 - Free (sort of), written by scientists and engineers
 - Varying level of support and stability
 - There may be support on commercial basis
- Commercial programs
 - Sold as a product, have usage restrictions and/or licensing
 - Generally offer support and stability

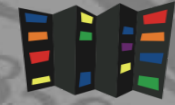
- Community programs
 - Numerical libraries (OpenBLAS, FFTW)
 - Simulation programs (NAMD, NWChem, WRF, OpenFoam)
 - Visualization programs (VisIt, Paraview)
- Commercial programs
 - Numerical libraries (MKL, IMSL)
 - Numerical analysis (Matlab, IDL, Mathematica)
 - Chemistry/material science simulation (Gaussian, Schroedinger)
 - Engineering simulation/CAE (Ansys, Abaqus, COMSOL, StarCCM+)



- Supported OS
 - A necessity for binaries (even on Linux)
 - Less strict for builds from source but helpful
- Compilers
 - Most sources build with GNU, may get better performance with commercial compilers (Intel, PGI)
- Software prerequisites (libraries the given code depends on)
 - Additional system packages (e.g. rpms on RedHat/CentOS)
 - Hand built libraries (e.g. MPI, FFTW, ...)

- Single user system
 - Often have root, install themselves (or use --prefix)
- Multi user system
 - Commonly used programs – user support installs
 - Uncommon or experimental programs – steer users to install themselves
- Special case – Python or R packages
 - Include common packages to the build (numpy, SciPy,...)
 - Instruct users to install themselves and use PYTHONPATH, RLIBS, etc.

- Local system
 - Some system path (standard /usr/..., /opt) or user's home
- Network file system
 - Applications file system (e.g. NFS) mounted on all servers
 - Need to use --prefix or other during installation
 - No need for root
 - Specific branch for each architecture (x86, power), and potentially OS version (CentOS6, 7)

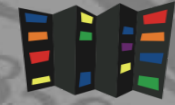


COMMUNITY CODES

- Binaries
 - Many packages supply binaries for the given OS (CentOS), use them, especially if they use graphics
- Build from source
 - several configuration/build systems
 - GNU autoconf (configure/make)
 - CMAKE
 - Scons
 - Need to include dependencies if any

- Get the source
 - Ask the researcher, colleagues, or do web search
- Find out how to build it
 - Untar and look for `configure`, `cmake` files, etc
 - Read the documentation
 - Do web search
 - Beware of configuration options (`configure --help`)
- Decide what compiler and dependencies to use
 - GNU for basic builds, Intel for better optimizations

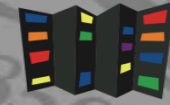
1. Make a directory called "zlib" and cd to it.
2. Download and untar the zlib library with the following :
 - `wget http://zlib.net/zlib-1.2.8.tar.gz`
 - `tar -xzf zlib-1.2.8.tar.gz`
3. Configure zlib so that it installs in the current directory (zlib) and not the source directory (zlib-1.2.8).
 - `./configure --prefix=$HOME/zlib` (as an example)
4. Compile using `make` and then `make install`.
5. Check to see if the library was installed properly in `zlib/lib` (the files `libz.so`, `libz.a` should exist).
6. See other configure options, `./configure --help`



COMMERCIAL CODES

- Pay and use w/o license manager (but enforcing license)
 - VASP, Gaussian
- License manager
 - Flexera FlexNet (formerly FlexLM) – used by most
 - Extension to FlexNet (Ansys), other license tool (RLM, own provenience)
- License server setup
 - Best external server, running one license daemon per Imgrd server program
 - Good candidate for VM as long as file system traffic is low
- External license servers
 - NAT to access cluster private network
 - Troubleshoot connectivity issues / firewall (`lmutil` `lmstat`, etc)

- Modify makefile and build
 - VASP, Gaussian
- Installers (text or GUI)
 - Mostly straightforward installation
 - Pay attention to where to enter license information
 - Enter license.dat or license server info in the installer
 - Copy license.dat to directory with the program
 - Most FlexNet licenses have environment variable to specify license info, e.g. `MLM_LICENSE_FILE=12345@mylicense.u.edu`
 - If use 3 redundant servers, license must be specified by env. var.

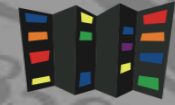


BUILDING FOR MULTIPLE ARCHITECTURES

- Most institutions run several generations of CPU and network
 - May have significant performance implications
 - E.g. CPU vectorization instructions can quadruple FLOPS going from SSE4.2 to AVX2 CPUs (3 tic-toc CPU architecture generations)
- What to do about it?
 - Build for lowest common denominator
 - Potentially significant performance implications
 - Build separate optimized executable for each architecture
 - Need to keep track of what executable to run where
 - **Build single executable using multi-architecture options**

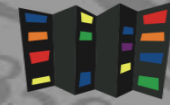
- Some compilers allow to build multiple versions of objects (functions) into a single executable
 - Intel calls this “automatic CPU dispatch”
 - Compiler flag `-axCORE-AVX2,AVX,SSE4.2`
 - PGI calls this “unified binary”
 - Compiler flag `--tp=nehalem,sandybridge,haswell`
- For multiple network types – use MPI that support multiple network channels
 - Most MPIs these days do – MPICH, OpenMPI, Intel MPI
 - Network interface selected at runtime, usually via environment var.

- Link with optimized libraries
 - Some vendors (Intel MKL) provide these
 - Build yourself
- Build your application with the appropriate compiler flags/MPIs
- For details see
<https://www.chpc.utah.edu/documentation/software/single-executable.php>

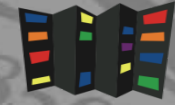


AUTOMATIC BUILDING

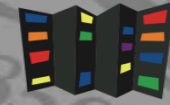
- Occasional builds can be done manually
 - keep old configure files/scripts
- Repetitive builds can be scripted
 - MPIs, file libraries (NetCDF, HDF), FFTW
- Use build automation tools
 - Some localized to a HPC center (Maali, Smithy, HeLMOD)
 - Wider community – EasyBuild, Spack



- Automatic build and installation of (scientific) programs
- Flexible and configurable (build recipes)
- Automatic dependency resolution
- Module file generation, logging, archiving
- Good documentation, increasing community acceptance
- Relatively simple to set up and use when using defaults
- Due to its flexibility, more complicated to customize
- Probably best deployed as a fresh build-out

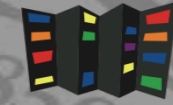


- Less complex than EasyBuild
- Simpler customization over command line
 - Dependencies, versions
- Uses RPATH for dependencies
 - Lower risk of dependency conflicts
- May be easier to use for incremental deployment

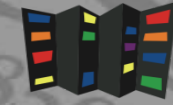


APPLICATION MANAGEMENT

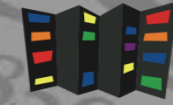
- Location of the programs
 - Usually mounted file server
 - Every site has different directory structure
- Presenting programs to the users
 - Shell init scripts
 - Not flexible, need to log out to reset environment
 - Environment modules



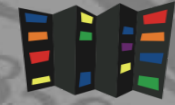
- Things to keep in mind when designing directory structure
 - Hierarchy/dependence of applications (Compiler – MPI)
 - Source, build and installation preferably in unique location
- Some sites choose hierarchical structure
 - Can lead to deep directory structure with a lot of empty/non-existing directories



- Separate directories for source, build, installation
 - srcdir, builddir, installdir
 - Only pristine source in srcdir – allows for reuse when building with different compilers, MPIs, configure options, etc
- Subdirectories as package/version
 - E.g. srcdir/mpich/3.2
- Hierarchy denoted with extensions to directory names
 - E.g. built with PGI compilers, installdir/mpich/3.2p
- We generally don't worry too much about compiler/MPI version as they tend to be fairly backwards compatible
 - Exceptions treated via module dependencies and specific directory names



- Allow user to load and unload program settings
 - TCL based modules part of CentOS distro
 - Lmod from TACC
- Lmod advantages
 - 3 level hierarchy of modules (compiler – MPI – application)
 - Usability enhancements (`ml`, `+`/`-`, `save`)
 - Site customization options
 - E.g. implementation to limit module loading to certain groups (licensees)



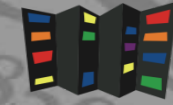
PYTHON AND R PACKAGES

- Many packages (libraries) that extend the basic functionality
 - Some are well developed and used
 - Some less so
- CHPC's old approach
 - Build R and Python from source
 - R still a must for acceleration/threading with Intel stack
 - Python now not so much thanks to Anaconda
 - Install most of what users ask for
- This eventually got out of hand with the number of packages needed, especially when upgrading Python or R.

- Install the common packages
 - Python – NumPy+friends, ...
 - R – BioConductor, dplyr, ...
- Have users to install other things
 - Most packages are simple enough for that
 - Good documentation how to do that is the key

<https://www.chpc.utah.edu/documentation/software/python3-venv3.php>

<https://www.chpc.utah.edu/documentation/software/r-language.php#rpkg>
- Occasionally need to help or install more difficult packages



- PYTHONPATH and --prefix
 - User needs to remember to put --prefix during installation and set PYTHONPATH environment variable
- Python Virtual Environment
 - Install own packages in user space
 - Use all packages installed in our main distribution

- Create Python Virtual Environment:

```
module load python/3.5.2
```

```
pyvenv --system-site-packages ~/VENV3.5.2
```

```
module unload python/3.5.2
```

- --system-site-packages allows use of packages from the main Python distribution

- Activate PVE:

```
source ~/VENV3.5.2/bin/activate #for bash shell
```

```
source ~/VENV3.5.2/bin/activate.csh #for tcsh shell
```

```
which python
```

```
~/VENV3.5.2/bin/python
```

- Deactivate PVE:

```
deactivate
```

- Remove PVE:

```
rm -r ~/
```

- PIP with custom Python
 - Easy to install
 - Must be careful about package dependencies – may mangle existing packages
- PIP commands

```
python -m pip install package # install package
python -m pip list             # List ALL the packages which are
installed in the main distribution or the Virtual Environment
python -m pip show numpy       # Show information about the NumPy module
python -m pip search math      # Searches for packages containing the
'math' string
python -m pip help             # Show all the options
```

- Manual installation that avoids checking for dependencies
 - By default installed in the PVE directory
 - Otherwise use `-prefix` flag, and use `PYTHONPATH`
- setuptools commands

```
cd ~
```

```
wget
```

```
https://pypi.python.org/packages/ab/8f/e0b437e55d0a067cc11d80737b88d60f975a362b13d77a3e38226278cc9d/chempy-0.5.1.tar.gz
```

```
tar -zxvf chempy-0.5.1.tar.gz
```

```
cd chempy-0.5.1
```

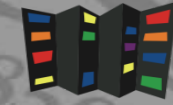
```
python setup.py build
```

```
python setup.py install
```

```
cd .. ; rm -R chempy-0.5.1
```

- Manual installation that avoids checking for dependencies
 - By default installed in the PVE directory
 - Otherwise use `-prefix` flag, and use `PYTHONPATH`
- setuptools commands

```
cd ~; wget https://pypi.python.org/packages/source/n/netCDF4/netCDF4-1.1.9.tar.gz
tar -zxvf netCDF4-1.1.9.tar.gz; cd netCDF4-1.1.9
export HDF5_DIR=/uufs/chpc.utah.edu/sys/installdir/hdf5/1.8.14/
export NETCDF4_DIR=/uufs/chpc.utah.edu/sys/installdir/netcdf-c/4.3.2
export CFLAGS=" -I/uufs/chpc.utah.edu/sys/installdir/netcdf-c/4.3.2/include \
               -I/uufs/chpc.utah.edu/sys/installdir/hdf5/1.8.14/include "
export LDFLAGS=" -Wl,-rpath=/uufs/chpc.utah.edu/sys/installdir/netcdf-c/4.3.2/lib \
               -L/uufs/chpc.utah.edu/sys/installdir/netcdf-c/4.3.2/lib -lnetcdf \
               -Wl,-rpath=/uufs/chpc.utah.edu/sys/installdir/hdf5/1.8.14/lib \
               -L/uufs/chpc.utah.edu/sys/installdir/hdf5/1.8.14/lib -lhdf5 "
python setup.py build
python setup.py install
cd .. ; rm -R
```



- Following our documentation
<https://www.chpc.utah.edu/documentation/software/r-language.php#rpkg>
- R defines several environment variables for external packages
 - R_LIBS_USER for user specific packages
- Define user-specific LMOD R module which overrides the system default
 - Here add the user-specific R_LIBS_USER

- Create your own module directory (e.g. ~/MyModules):
`mkdir -p ~/MyModules`
- Create an R subdirectory in ~/MyModules:
`mkdir ~/MyModules/R`
- Copy the R/3.3.2 module from the CHPC modules directory into your own R module space:
`cp /uufs/chpc.utah.edu/sys/modulefiles/CHPC-c7/Core/R/3.3.2.lua
~/MyModules/R`
- Make the relative name of the new module unique:
`mv ~/MyModules/R/3.3.2.lua ~/MyModules/R/3.3.2.$USER.lua`
- Make own module directory visible to Lmod:
`module use ~/MyModules`
- Create a new directory where we will install our new R packages:
`mkdir -p ~/software/pkg/RLibs/3.3.2i`
- Edit the newly created module e.g. ~/MyModules/R/3.3.2.\$USER.lua, to add the following line:
`setenv("R_LIBS_USER", "/home/u0xxyyzz/software/pkg/RLibs/3.3.2i/")`
The string u0xxyyzz must be replaced by your user name.

- Function **install.packages()** :

R

```
>library(maRketSim)      # try to find a library
```

```
Error in library(maRketSim) : there is no package called  
'maRketSim'
```

```
>install.packages("maRketSim",lib="/home/$USER/software/pkg/R  
Libs/3.3.2i",repos="http://cran.us.r-  
project.org",verbose=TRUE)
```

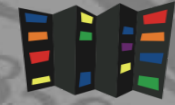
```
>library(maRketSim)
```

- **lib** – destination path for the library
- **repos** – location of the repository
- **configure.args, configure.vars** – arguments and variables for source installs

<https://www.rdocumentation.org/packages/utils/versions/3.4.1/topics/install.packages>

- Command **R CMD INSTALL**
 - For R packages that depend on external libraries
- E.g. RNetCDF
 - Needs netcdf-c and udunits2

```
export PATH=/uufs/chpc.utah.edu/sys/installdir/netcdf-c/4.3.2i/bin:$PATH
export PATH=/uufs/chpc.utah.edu/sys/installdir/udunits/2.2.20/bin:$PATH
wget https://cran.r-project.org/src/contrib/RNetCDF_1.8-2.tar.gz
R CMD INSTALL --library=/uufs/chpc.utah.edu/common/home/$USER/RLibs/3.3.2i \
--configure-args= \
"CPPFLAGS='-I/uufs/chpc.utah.edu/sys/installdir/udunits/2.2.20/include' \
LDFLAGS='-Wl,-rpath=/uufs/chpc.utah.edu/sys/installdir/netcdf-c/4.3.2i/lib \
-L/uufs/chpc.utah.edu/sys/installdir/netcdf-c/4.3.2i/lib -lnetcdf \
-Wl,-rpath=/uufs/chpc.utah.edu/sys/installdir/udunits/2.2.20/lib\
-L/uufs/chpc.utah.edu/sys/installdir/udunits/2.2.20/lib -ludunits2 ' \
--with-nc-config=/uufs/chpc.utah.edu/sys/installdir/netcdf-c/4.3.2i/bin/nc-config" \
RNetCDF_1.8-2.tar.gz
```

BACKUP DEMO

- MIT Photonic Bands (MPB)
 - Program to study photonic crystals
 - http://ab-initio.mit.edu/wiki/index.php/MIT_Photonic_Bands
 - Has a nice set of dependencies (BLAS, LAPACK, MPI, FFTW)
- Download the source
 - `wget http://ab-initio.mit.edu/mpb/mpb-1.5.tar.gz`
- Decide how to build
 - We want to optimize for highest performance – use Intel compilers and libraries (`module load intel impi`)

- Build in `/uufs/chpc.utah.edu/sys/builddir/mpb/1.5i`
- Run `configure -help` to see the options
- Set up configure script – `vi config.line`
 - I prefer to create a script with all the environment variables and configure options
 - `cp /uufs/chpc.utah.edu/sys/builddir/mpb/1.5i/config.line .`
 - Modify `--prefix`
- Run configure script - `./config.line`
- Run `make`
- Run `make install`
- There is no `make test`, so run own
 - `cd test2; ../mpb/mpb-mpi diamondctl`