Linux Application Installation and HPC Libraries

Linux Application Installation and configuring software

- Multiple ways exist to install software on Linux systems
  Following are 3 major ways:
  - **RPM packages**
    [Typically on RHEL based systems]

Linux Application Installation and configuring software

- Multiple ways exist to install software on Linux systems
  Following are 3 major ways:
  - **RPM packages**
    [Typically on RHEL based systems]
  - **DEB packages**
    [Typically on Debian based systems]

Linux Application Installation and configuring software

- Multiple ways exist to install software on Linux systems
  Following are 3 major ways:
  - **RPM packages**
    [Typically on RHEL based systems]
  - **DEB packages**
    [Typically on Debian based systems]
  - **Source code**
    [Could be used for any platform]
    - *make* utility
    - *cmake* utility

Linux Application Installation and configuring software

- Multiple ways exist to install software on Linux systems
  Following are 3 major ways:
  - **RPM packages**
    [Typically on RHEL based systems]
  - **DEB packages**
    [Typically on Debian based systems]
  - **Source code**
    [Could be used for any platform]
    - *make* utility
    - *cmake* utility

Package manager software are used for easy installation and management of packages and their dependencies using software repositories

RPM Packages

- It is a common Linux free software package management tool developed by Red Hat. This method is popular because users don’t need to compile the code by themselves. The software is ready to be installed RPM packages

RPM Packages

- It is a common Linux free software package management tool developed by Red Hat.
  This method is popular because users don’t need to compile the code by themselves.
  The software is ready to be installed RPM packages

- As for RPM, user needs to perform the extraction of files by already defined options (such as destination, name etc.) which are hidden within the responsible utilities (rpm, yum).
  Installing RPM packages is fairly straightforward.
  To install such software package, you can run the following command:

```
# rpm –i RPMPackage.rpm
```

RPM Packages

- An alternative tool here is yum: the main difference is automatic upgrades and package management (including necessary dependencies).
YUM is analog for APT (DEB packages) and manage repositories. Example:

```
# yum update
# yum install Package
# yum remove Package
```

- For example:

```
# yum install openmpi
# yum remove gedit
```

Debian Packages

- Debian packages almost the same as RPM but for usage in Debian GNU/Linux systems. The extension of such packages are *.deb.x

Debian Packages

- Debian packages almost the same as RPM but for usage in Debian GNU/Linux systems. The extension of such packages are *.deb.x

- To install the packages (whether source or binary) use APT (Advance Packaging Tool) or dkpg. These are package management system for Debian and also includes a lot of different tools.

Debian Packages

- Debian packages almost the same as RPM but for usage in Debian GNU/Linux systems.
  The extension of such packages are *.deb.x

- To install the packages (whether source or binary) use APT (Advance Packaging Tool) or dkpg.
  These are package management system for Debian and also includes a lot of different tools.

- Installing new software is simple as well.
  You can run similar commands as compared to the ones used for RPMs.

Debian Packages

- Using dpkg:

```
# dpkg –i DebianPackage.deb
```

Debian Packages

- Using dpkg:

```
# dpkg –i DebianPackage.deb
```

- Using apt the common flow is as follows:

```
# apt-get update
# apt-cache search Package
# apt-get install Package
# apt-get remove Package
```

Source code installation

- TARBALLs are so-called archives distributed with the following extensions: “.tar.gz”, “.tar.bz2”, or “.zip” (there are even more based on type of compression and archivers). Originally tarballs are used for programs which are not compiled, i.e. they are presented as source code.
  There are significant differences on how to install software this way.

- The main idea here: - If you cannot find your program in the repositories, just download the source code from any open source program website and then install it according to the listed instructions.

Source code installation

- Typically the archives are required to be extracted to obtain the data:
  Unarchive tools could be used to perform the extraction.
  Following are few examples of the same:

```
$ tar -zxvf TarBall.tar.gz
$ tar -jxvf TarBall.tar.bz2
$ unzip TarBall.zip
```

Source code installation

- Typically the archives are required to be extracted to obtain the data:
  Unarchive tools could be used to perform the extraction.
  Following are few examples of the same:

```
$ tar -zxvf TarBall.tar.gz
$ tar -jxvf TarBall.tar.bz2
$ unzip TarBall.zip
```

- The program is not prepared for installation.
  One should perform the preparation procedure called pre-installation configuration.

Source code installation (using make)

- Just run ./configure and your system will be checked for any necessary libraries or configurations required to fulfill test of your system.

```
$ cd PackageDirectory $
./configure
```

Source code installation (using make)

- Just run ./configure and your system will be checked for any necessary libraries or configurations required to fulfill test of your system.
- ‘make’ command is used to compile the source code, which typically reads a rule file (known as Makefile) to build the binaries for the package / application.

```
$ cd PackageDirectory $
./configure


$ make
```

Source code installation (using make)

- Just run ./configure and your system will be checked for any necessary libraries or configurations required to fulfill test of your system.
- ‘make’ command is used to compile the source code, which typically reads a rule file (known as Makefile) to build the binaries for the package / application.
- ‘make check’ or ‘make test’ command is used to verify that the built software actually runs and behaves the expected way.

```
$ cd PackageDirectory $
./configure


$ make


$ make check
```

Source code installation (using make)

- Just run ./configure and your system will be checked for any necessary libraries or configurations required to fulfill test of your system.
- ‘make’ command is used to compile the source code, which typically reads a rule file (known as Makefile) to build the binaries for the package / application.
- ‘make check’ or ‘make test’ command is used to verify that the built software actually runs and behaves the expected way.
- Finally, for installation ‘make install’ command is used.

```
$ cd PackageDirectory $
./configure


$ make


$ make check

$ make install
```

Source code installation (using cmake)

- CMake is an open-source, cross-platform family of tools designed to build, test and package software.
  CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles.

Source code installation (using cmake)

- CMake is an open-source, cross-platform family of tools designed to build, test and package software.
  CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles.

- CMake is much more high-level.
  It's tailored to compile C++, for which you write much less build code, but can be also used for general purpose build.
  Following is an example of a typical application installation workflow using cmake:

  Note: CMakeLists.txt is file that contains basic rules and requirements for building the package / application.

```
#Extract the archive and create a build directory
$ tar -zxvf TarBall.tar.gz
$ cd PackageDirectory
$ mkdir build

# Generate configuration files using cmake
$ cmake /path/to/CMakeLists.txt

#Compilation, testing and installation
$ make   check   install
```

Source code installation (Important points)

- Installation from source code depends upon a number of factors.

  Following are a few important to list out:
  - Environment variables
  - Compilers and language support
  - Dependent libraries and their version support
  - Optimization libraries
  - System architecture

Source code installation (Important points)

- Installation from source code depends upon a number of factors.

  Following are a few important to list out:
  - **Environment variables**
  - Compilers and language support
  - Dependent libraries and their version support
  - Optimization libraries
  - System architecture

- **Crucial environment variables**:

  CC, CXX, FC, PATH, LD_LIBRARY_PATH, CPATH, MPICC, MPICXX, MPIFC, PREFIX, etc.

Source code installation (Important points)

- Installation from source code depends upon a number of factors.
  Following are a few important to list out:
  - Environment variables
  - **Compilers and language support**
  - Dependent libraries and their version support
  - Optimization libraries
  - System architecture

- Choice of compiler is important for considering **performance and accuracy** of the application.
  Few C/C++/Fortran application have restrictions on the language **standards** used.

Source code installation (Important points)

- Installation from source code depends upon a number of factors.
  Following are a few important to list out:
    - Environment variables
    - Compilers and language support
    - **Dependent libraries and their version support**
    - Optimization libraries
    - System architecture

- Most of the HPC application are built having common ground of libraries for I/O, communication, and computation. It is important that the **correct version of libraries** is used.

Source code installation (Important points)

- Installation from source code depends upon a number of factors.
  Following are a few important to list out:
  - Environment variables
  - Compilers and language support
  - Dependent libraries and their version support
  - **Optimization libraries**
  - System architecture

- Most often, HPC applications depend upon optimization libraries to perform critical operation. This allows the user to choose specific optimization libraries as per their requirement.

Source code installation (Important points)

- Installation from source code depends upon a number of factors.
  Following are a few important to list out:
    - Environment variables
    - Compilers and language support
    - Dependent libraries and their version support
    - Optimization libraries
    - **System architecture**

- Finally, the system architecture does play a vital role, since, few **application intrinsics** may use certain operation which are **supported on one platform**, but not on the other one.

Source code installation (Important points)

- Most of the times, the source application installation process is made available using the tools that we saw (make, cmake, package managers). However, complex applications do require different steps due to dependencies and user configurations.
  Thus, it is always recommended to read the **Documentation** or **ReadMe** file or **INSTALL** file.

Source code installation (Important points)

- Most of the times, the source application installation process is made available using the tools that we saw (make, cmake, package managers). However, complex applications do require different steps due to dependencies and user configurations.
  Thus, it is always recommended to read the **Documentation** or **ReadMe** file or **INSTALL** file.
- HPC applications often make the use of **make and cmake** utilities and installation process may have bottlenecks due to system architecture, compilers, libraries and many other factors.
  It is very important to **understand the process** designed for make (**Makefile / makefile**) and cmake (**CMakeLists.txt**)

Source code installation (Important points)

- Most of the times, the source application installation process is made available using the tools that we saw (make, cmake, package managers). However, complex applications do require different steps due to dependencies and user configurations.
  Thus, it is always recommended to read the **Documentation** or **ReadMe** file or **INSTALL** file.
- HPC applications often make the use of **make and cmake** utilities and installation process may have bottlenecks due to system architecture, compilers, libraries and many other factors.
  It is very important to **understand the process** designed for make (**Makefile / makefile**) and cmake (**CMakeLists.txt**)
- Understanding and usage of **shell scripts** is crucial to automate repetitive work.

Workout 1

A) The list of names displayed here are actual libraries that are required for most of the ocean and climate modelling applications.

B) While you are installing the library, observe and note the purpose of the library. This will give an idea on where the specific library will be used by your application.

C) List out / document the errors and issues that you face during the installation.

Ocean and climate modeling applications play a crucial role in understanding and predicting Earth's climate system.
A widely used applications in this domain is the Weather Research and Forecasting model (WRF).

Following are the list of dependencies for WRF.
The task is to install these libraries:
1) ZLIB
2) SZIP
3) HDF5
4) PNETCDF
5) NETCDF-C
6) NETCDF-F

Important Note:
Make sure you use the same compiler for compilation of all the dependent libraries.

Workout 2

A) The list of names displayed here are actual libraries that are required for physics, molecular dynamics, astrophysics, cosmosoly, medical applications and many more.

B) While you are installing the library, observe and note the purpose of the library. This will give an idea on where the specific library will be used by your application.

C) List out / document the errors and issues that you face during the installation.

Molecular dynamics (MD) is a computational simulation technique used to study the behavior and properties of atoms and molecules over time. MD simulations involve solving the equations of motion for individual particles in a system, allowing researchers to investigate the dynamics and interactions at the atomic scale.

A popular application in this domain is LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator)

Following are the list of dependencies for LAMMPS.
The task is to install these libraries:
1) BLAS
2) LAPACK
3) FFTW

Important Note:
Make sure you use the same compiler for compilation of all the dependent libraries.

“Installing Linux applications is like dating a penguin – it might seem a little cold and complicated at first, but once you get to know each other, it's a match made in open-source heaven!”

Questions?

Thanks!

Contact:

Vineet More
vineet.more.bfab@gmail.com
[Make sure to use HPCAP23 as the subject line for your queries]

LinkedIn Handle:
https://www.linkedin.com/in/vineet-more-c-programmer/