What is the Programming Environment?

Your Applications & Jobs

- Resource & Workload Managers
- Scalable Debuggers & Analysis Utilities
- Performance Math & Parallel Libraries
- IO Service & Runtimes
- Programming Model Runtimes
- Compiler Toolchains
- Userland Tools & Utilities

 LLCA, Networks

 Common Linux Runtime Libraries

 Operating System

 Hardware
Programming Environment Overview

• At the highest level, the PE is your shell’s build- and run-time environment (see output of `env`).

• Software outside default (/usr/bin, /usr/lib, etc.) UNIX paths.

• Managed via session environment variables
  – Search paths
    • `PATH, LD_LIBRARY_PATH, LIBRARY_PATH, PKG_CONFIG_PATH`, etc...
  – Program environment options
    • `OMPI_*, CC, FC`, etc...

• Summit uses LMOD for this purpose
LMOD Environment Modules

- Much of the available software cannot coexist simultaneously in your environment.
- Build- and runtime-environment software managed with LMOD (https://lmod.readthedocs.io)

**Usage:**

```bash
$ module -t list    # list loaded modules
$ module avail     # Show modules that can be loaded given current env
$ module help <package> # Help info for package (if provided)
$ module show <package> # Show contents of module
$ module load <package> <package>... # Add package(s) to environment
$ module unload <package> <package>... # Remove package(s) from environment
$ module reset     # Restore system defaults
$ module restore <collection> # Load a saved collection
$ module spider <package>    # Deep search for modules
$ module purge       # Clear all modules from env.
```
Module Avail

- The `module avail` command shows only what can be loaded given currently loaded packages.
- Full or partial package names limit output to matches.

```bash
$ module avail
------------- /sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core
...
cuda/10.1.168 (D)
cuda/10.1.243 (D)
gcc/4.8.5     (L)
gcc/5.4.0     ...
py-nose/1.3.7 (D)
py-pip/9.0.1  python/3.5.2
readline/6.3

Where:
L: Module is loaded
D: Default Module

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".

Path in MODULEPATH
where a module exists. Printed in order of priority.

Future labels will have explanation in legend (shown on in non-terse output)
Modulefile Priority

• Loading some modules will alter the MODULEPATH.
  – Compilers, MPI: Only one module in families can be loaded at a time.

• First module among duplicate package/version names in MODULEPATH will be selected:

```
$ module -t avail hdf5/1.10.4
/sw/summit/modulefiles/site/.../spectrum-mpi/10.3.1.2-20200121-p6nrnt6/xl/16.1.1-5:
hdf5/1.10.4
/sw/summit/modulefiles/site/.../xl/16.1.1-5:
hdf5/1.10.4
```

Example: MPI-enabled builds replace serial builds when MPI implementation is loaded.
Modulefile Priority

To override behavior, alter the MODULEPATH yourself:

```
$ module use /path/to/module/file/tree
$ module unuse /path/to/remove/from/search/tree
```

- Path is prepended with higher priority
- Can also provide your own custom modulefiles.
  - Complete instructions for writing modulefiles:
Searching for Modules with Spider

- Use `module spider` (not avail) to search for modules
  - Finds packages that cannot be loaded given current environment
  - Shows requirements needed to make package available

```
$ module -t spider hdf5/1.10.4

hdf5: hdf5/1.10.4

You will need to load all module(s) on any one of the lines below before
the "hdf5/1.10.4" module is available to load.
...
gcc/4.8.5
gcc/4.8.5 spectrum-mpi/10.3.1.2-20200121
gcc/5.4.0
...
```
Spider (cont’d)

- Complete listing of possible modules is only reported when searching for a specific version:
  \texttt{module\ spider\ <package>/\<version>}

- Can search using limited regular expressions:
  - All modules with ‘m’ in their name: \texttt{module -t\ spider\ 'm'}
  - All modules starting with the letter ‘m’: \texttt{module\ -t\ -r\ spider\ '^m'}
Module Dependency Management

- Conflicting modules automatically reloaded or inactivated.
- Generally eliminates needs for $ module swap PKG1 PKG2

$ module load xl

Lmod is automatically replacing "gcc/4.8.5" with "xl/16.1.1-5".

Due to MODULEPATH changes, the following have been reloaded:
  1) spectrum-mpi/10.3.1.2-20200121
Module Dependency Management

- Check `stderr` for messages about deprecated modules.

- Modules generally only available when all dependencies are currently loaded.
  - Most provided packages use absolute RPATHs and RUNPATHs; obviates the need to explicitly load dependency modules.
  - Some exceptions, notably python extensions.

- Not all packages available in all compiler environments
  - Advanced approach to mix modules across compiler environments described in backup slides.
User Module Collections

• Save module collections for easy re-use

$ module save my_favorite_modules
Saved current collection of modules to: "my_favorite_modules", for system: "summit"

$ module reset
Resetting modules to system default

$ module restore my_favorite_modules
Restoring modules from user's my_favorite_modules, for system: "summit"

$ module savelist
# Show what collections you’ve saved

$ module describe <collection>
# Show modules in a collection

$ module disable <collection>
# Make a collection un-restorable (does not delete)
User Module Collections

- Modulefile updates *may break saved collections.*
  - **To fix:** manually load desired modules, save to same name to update.

- Collection named `default` automatically loaded on login
  - Use caution with personal `default` collections due to above

- To delete a collection: `rm ~/.lmod.d/<collection>.[system]`
Default Applications

• DefApps meta module
  - XL compiler
  - SMPI
  - HSI – HPSS interface utilities
  - XAlt – Library usage
  - LSF-Tools – Wrapper utility for LSF
  - darshan-runtime – An IO profiler; unload if using other profilers.
Compilers and Toolchains

- Compiler Environments
- Common Flags
## Compiler Environments

### IBM XL (default)
- xl/16.1.1-7
- xl/16.1.1-6
- xl/16.1.1-5 (default)
- xl/16.1.1-4
  - Older not recommended

### GCC
- gcc/9.1.0
  - Supports OMP-Offload and OpenACC
- gcc/8.1.1
- gcc/7.4.0
  - Latest w/ CUDA10 NVCC
- gcc/6.4.0 (default)
- gcc/5.4.0
- gcc/4.8.5
  - RHEL7 OS compiler in `/usr`
  - “Core” modulefiles

### PGI
- pg/20.1
- pg/19.10
- pg/19.9 (default)
- pg/18.10
- pg/18.7

---

New compiler releases added regularly.
IBM XL (Default toolchain)

- Base compilers `xlc`, `x1C`, `x1c++`, `xlf`
  - Many wrappers exist to apply preset flags for various language standards. Thread safe option wrappers suffixed `*_r`
  - See `${OLCF_XLC_ROOT}/etc/xlc.cfg.*` and `${OLCF_XLF_ROOT}/etc/xlf.cfg.*` for options enabled by wrappers.

- Single version in `/opt/ibm`, to reference module version, use `${OLCF_XL_ROOT}`, `${OLCF_XLC_ROOT}`, `${OLCF_XLF_ROOT}`
LLVM

- Base compilers `clang`, `gfortran` (OS)
- `llvm/1.0-20190225` based on Clang v8, despite module name
- Full software environment provided only for v9.0.0+
- Experimental; minimal support.
CUDA/NVCC

- Module `cuda/10.1.243` (default)
- Available under all compiler environments
- If you’re not using the GPUs, you’re not really using the machine
- Provides cuBLAS, cuDNN
  - cuBLAS located according to CUDA ≤ 9 scheme: `${OLCF_CUDA_ROOT}/{lib64,include}`
- Older modules available, but not recommended for use
  - Recompile against latest version available if possible
Software, Libraries, and Programming Models
Provided Software

• Vendor-supplied
  – IBM: ESSL (blas/lapack**/fftw), MASS, SMPI
  – NVIDIA: CUDA, cuBLAS, cuDNN
  – Debuggers: Allinea Forge, Perf. Reports; Score-P/Vampir

• Built by OLCF
  – Built in userspace without superuser privileges.
  – Often general-purpose builds
  – Optimized as possible while still being generally applicable
    • May not always be as optimized as you want;
      notable example: BLAS/LAPACK for CPU is mostly a reference implementation.
    • Encourage users to build own packages for special needs
MPI Implementation – IBM Spectrum-MPI

• Based on OpenMPI, similar compiler wrappers and flags
  mpicc, mpic++, mpiCC, mpifort, mpif77, mpif90, mpixl*

• Modules spectrum-mpi/10.3.1.2-20200121 (default)
  – Avoid hidden older releases

• Updates usually require recompilation.

• Uses jsrun MPI launcher (See separate talk in this series)

• Avoid Alt. implementations (OpenMPI)
  – openmpi/4.0.3 is experimental and not recommended for general use.
MPI environment

• When using XL, default `$OMPI_FC` is `xlf2008_r`
  – Works with standard MPI wrappers despite XL-specific wrappers `mpixlc`, `mpix1C`, `mpixlf`
  – F77 codes must use alternate xlf wrapper `export OMPI_FC=xlf_r`
    or set additional xlf options via `FFLAGS`, build-system, etc.

• Adaptive routing enabled by default
  
  `PAMI_IBV_ENABLE_000_AR=1`
  `PAMI_IBV_QP_SERVICE_LEVEL=8`
Building your own software

• Where to build?
  – Recommend /tmp/$USER
    • faster performance than NFS
    • doesn’t leave detritus in quota’d $HOME, /ccs/proj dirs.
  – GPFS also acceptable

• Where to install?
  – NFS filesystem /ccs/proj/<PROJECTID> preferred: not purged, RO.
  – Avoid $HOME, especially ~/.local/{bin,lib,share}
    • Shared across architectures; likely to cause ABI or ISA runtime errors
Thanks for listening

• Questions or comments regarding the Summit programming environment?

  Contact `help@olcf.ornl.gov`  
  
We’re happy to help with any issues and questions you have.
Appendix

Environment Modules
help("GCC Compiler")
whatis("Description: ", "GCC compiler 8.1.1")

local package = "gcc"
local version = "8.1.1"
local moduleroot = myFileName():sub(1,myFileName():find(myModule FullName(),1,true)-7)
local gccdir = "/sw/ascent/gcc/8.1.1"

-- Setup Modulepath for packages built by this compiler
prepend_path("MODULEPATH", pathJoin(moduleroot, package, version ) )

-- Environment Globals
prepend_path("PATH", pathJoin(gccdir, "bin") )
prepend_path("MANPATH", pathJoin(gccdir, "share/man") )
prepend_path("LD_LIBRARY_PATH", pathJoin(gccdir, "lib64") )

-- OLCF specific Environment
setenv("OLCF_GCC_ROOT", gccdir)
Access to all the provided software

• Possible to use modules across compiler environments but not recommended.

• Use at your own risk
  – Modules may conflict with other software or otherwise not function
  – Read modulefile comments and build log for information about build
  – Check binaries and libraries with ldd for links against MPI version

• Modules named
  \{PKG\}-{VER}-{COMPILER}-{COMP_VER}-{[SUFFIXES]-}{HASH_STUB}

SPACK_MODULES=“/sw/summit/.swci/1-compute/share/spack/modules”
module use “${SPACK_MODULES}/20180914/linux-rhel7-ppc64le”
Appendix

Compilers and Toolchains
IBM XL Options and Flags

• Code standard using base compiler
  – xlc: `-std=gnu99`, `-std=gnu11`
  – xlc++: `-std=gnu++11`, `-std=gnu++1y` (partial support)
  – Wrappers available for many language standards

• Default signed char: `-qchar=signed`

• Define macro: `-WF, -D`

• IBM xlf does not mangle Fortran symbols by default, use `-qextname` to add trailing underscores.
GNU Compiler Suite (GCC)

- Base compilers: gcc, g++, gfortran
- OS compiler always in environment
  - Guaranteed ABI compatible with system libraries
- Code standards:
  - gcc: -std=c11, -std=c17, -std=c90, -std=c99 (and GNU variants)
  - g++: -std=c++11, -std=c++14, -std=c++17
  - gfortran: -std=f90, -std=f2003, -std=f2008
- Signed char: -fsigned-char
GNU Compiler Suite (GCC)

For gcc v8+, set explicit language standard (eg \texttt{-std=c++11}) if encountering the error:

\begin{quote}
\texttt{error: identifier \texttt{__ieee128} is undefined}
\end{quote}
PGI (Portland Group)

- Base compilers **pgcc**, **pg++**, **pgfortran**

- Code standards:
  - **pgcc**: `-c99`, `-c11`
  - **pg++**: `-std=c++11 -std=c++14 --gnu_extensions`, `-std=c++14 --gnu_extensions`
  - Fortran code standard detected by suffix: `.F90`, `.F03`, `.F08`

- Default signed char: `-Mschar`
CUDA/NVCC Options and Flags

- **C++11 support**: `-std=c++11`

- **host/device `lambdas` (experimental)**: `--expt-extended-lambda`

- **host/device `constexpr`s (experimental)**: `--expt-relaxed-constexpr`

- **Supports XL, GCC, and PGI C++ host compilers via** `--ccbin <PATH>`
  - Some version restrictions for latest PGI, GCC toolchains
OpenACC (Version 2.5)

<table>
<thead>
<tr>
<th>Supported Compiler Environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGI (All Versions)</td>
</tr>
</tbody>
</table>

-acc -ta=nvidia:cc70  -fopenacc
## OpenMP

<table>
<thead>
<tr>
<th>Compiler</th>
<th>3.1 Support</th>
<th>4.x Support</th>
<th>Enable OpenMP</th>
<th>Enable OpenMP 4.X Offload</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>FULL</td>
<td>PARTIAL</td>
<td>-qsmp=omp</td>
<td>-qsmp=omp -qoffload</td>
</tr>
<tr>
<td>GCC</td>
<td>FULL</td>
<td>PARTIAL</td>
<td>-fopenmp</td>
<td>-fopenmp</td>
</tr>
<tr>
<td>PGI</td>
<td>FULL</td>
<td></td>
<td>-fopenmp</td>
<td></td>
</tr>
<tr>
<td>LLVM</td>
<td>FULL</td>
<td>PARTIAL</td>
<td>-fopenmp</td>
<td>-fopenmp -fopenmp-targets=nvptx64-nvidia-cuda --cuda-path=${OLCF_CUDA_ROOT}</td>
</tr>
</tbody>
</table>
Appendix

Software, Libraries, and Programming Models
Detailed Information about provided software

- `$OLCF_{PKG}_ROOT/./spack/build.out`

```
$ head $OLCF_HDF5_ROOT/.spack/build.out

==> Executing phase: 'autoreconf'
==> Executing phase: 'configure'
==> '/autofs/nccsopen-svm1_sw/ascent/.swci/1-compute/var/spack/stage/hdf5-1.10.3-2llvf5hpxbqzgl5agzkstjqs2xv4v4uk/hdf5-1.10.3/configure' '--prefix=/autofs/nccsopen-svm1_sw/ascent/.swci/1-compute/opt/spack/20180914/linux-rhel7-ppc64le/xl-16.1.1-beta5/hdf5-1.10.3-2llvf5hpxbqzgl5agzkstjqs2xv4v4uk' '--enable-unsupported' '--disable-threadsafe' '--enable-cxx' '--enable-hl' '--enable-fortran' '--without-szlib' '--enable-build-mode=production' '--enable-shared' 'CFLAGS=-qpic' 'CXXFLAGS=-qpic' 'FCFLAGS=-qpic' 'CFLAGS=-qpic' 'CXXFLAGS=-qpic' 'FCFLAGS=-qpic' 'CFLAGS=-qpic' 'CXXFLAGS=-qpic' 'FCFLAGS=-qpic' 'CFLAGS=-qpic' 'CXXFLAGS=-qpic' 'FCFLAGS=-qpic' 'CFLAGS=-qpic' 'CXXFLAGS=-qpic' 'FCFLAGS=-qpic'
...
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
```
Building your own software

- Recommended to rebuild with new MPI, CUDA releases
- Recommended to use common build systems and utils
  - CMake, autotools, pkgconfig, etc.
  - Many provided packages automatically alter
    \$CMAKE_PREFIX_PATH, \$PKG_CONFIG_PATH
- All center-built modules set \$OFCF_{PKG}_ROOT vars for use in build/configure scripts
Using Spack for missing dependencies

  – Used to deliver most of the packages we provide
  – Not all Spack packages written to support ppc64le... Yet
  – OLCF uses come customized packages not available upstream

• Must configure to use external SMPI, CUDA, compilers.
  – .*/spack/etc/spack/packages.yaml

• Happy to share our Spack configs and settings on request.