

The Summit Programming Environment

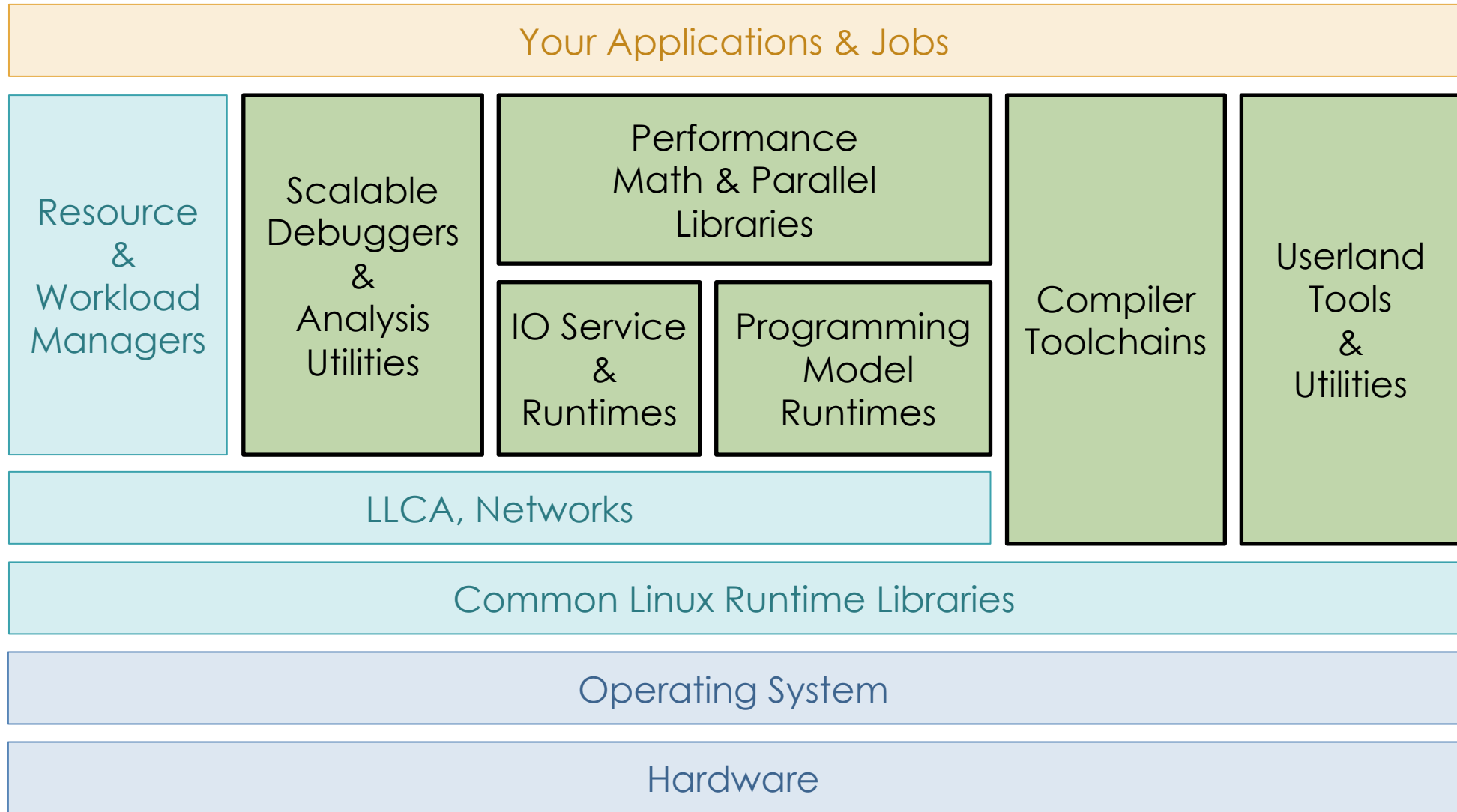
Matt Belhorn
Oak Ridge Leadership Computing Facility
Summit Training Workshop
11 February 2019

ORNL is managed by UT-Battelle LLC for the US Department of Energy



U.S. DEPARTMENT OF
ENERGY

What is the Programming Environment?



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 paths (`/usr/bin`, `/usr/lib`, etc.)
- 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 software available cannot coexist simultaneously in your environment.
- Build- and runtime-environment software managed with LMOD (<https://lmod.readthedocs.io>)
- Usage:

```
$ 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.

```
$ module avail
----- /sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core -----
...
cuda/9.1.85
cuda/9.2.64      (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 certain modules will alter the MODULEPATH.
 - Compilers, MPI
- First module among duplicate package/version names in MODULEPATH will be selected:

```
$ module -t avail hdf5/1.10.0-patch1
/sw/summit/modulefiles/site/.../spectrum-mpi/10.2.0.0-20180508-riohv7q/xl/20180502:
hdf5/1.10.0-patch1
/sw/summit/modulefiles/site/.../xl/20180502:
hdf5/1.10.0-patch1
```

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
```

- Path is prepended with higher priority
- Can also provide your own custom modulefiles.
 - Complete instructions for writing modulefiles:
https://lmod.readthedocs.io/en/latest/015_writing_modules.html

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 hdf5/1.10.0-patch1
```

```
-----  
hdf5: hdf5/1.10.0-patch1  
-----
```

```
You will need to load all module(s) on any one of the lines below before  
the "hdf5/1.10.0-patch1" module is available to load.
```

```
...
```

```
gcc/4.8.5
```

```
gcc/4.8.5  spectrum-mpi/10.1.0.4-20170915
```

```
gcc/4.8.5  spectrum-mpi/10.2.0.0-20171117
```

```
...
```


Spider (cont'd)

- Complete listing of possible modules is only reported when searching for a specific version:
`module spider <package>/<version>`
- Can search using limited regular expressions:
 - All modules with 'm' in their name: `module -t spider 'm'`
 - All modules starting with the letter 'm': `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/20180502".

Due to MODULEPATH changes, the following have been reloaded:

1) spectrum-mpi/10.2.0.0-20180508

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 actually 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
 - Recommend not using `default` due to above
- 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 Compiling

- Compiler Environments
- Common Flags



Compiler Environments

IBM XL (default)

- xl/16.1.1-1
 - Older modules not recommended

LLVM/Clang

- llvm/1.0-20180531
 - Older modules not recommended

GCC

- gcc/4.8.5
 - RHEL7 OS compiler
 - Compiler always available
- gcc/5.4.0
- gcc/6.4.0 (default)
 - Latest useable with NVCC
- gcc/8.1.1
 - OpenACC Capable

PGI

- pgi/19.1
- pgi/18.10 (default)
- pgi/18.7

New compiler releases added regularly.

IBM XL (Default toolchain)

- Base compilers `xlc`, `xlC`, `xlc++`, `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}`

IBM XL Options and Flags

- Code standard using base compiler
 - xlc: -std=gnu99, -std=gnu11
 - xlc++: -std=gnu++11, -std=gnu++1y (partial support)
 - xlf: -qlanglvl=90std, -qlanglvl=2003std, -qlanglvl=2008std
 - 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=gnu99`, `-std=gnu11`
 - `g++`: `-std=gnu++11`, `-std=gnu++1y`
 - `gfortran`: `-std=f90`, `-std=f2003`, `-std=f2008`
- Signed char: `-fsigned-char`

PGI (Portland Group)

- Base compilers `pgcc`, `pg++`, `pgfortran`
- Code standards:
 - `pgcc`: `-c99`, `-c11`
 - `pg++`: `-std=c++11 --gnu_extensions`, `-std=c++14 --gnu_extensions`
 - Fortran code standard detected by suffix: `.F90`, `.F03`, `.F08`
- Default signed char: `-Mschar`

LLVM

- Base compilers clang, gfortran (OS)
- Build actually based on Clang v3.8 despite modulefile name
- Full software environment **not provided**
- Experimental, minimal support.
 - May need to use older cuda (whichever latest release was built for) for OpenMP offload.
- Default signed char: `-fsigned-char`

CUDA/NVCC

- Module `cuda/9.2.148` (default)
- Available under all compiler environments
- If you're not using the GPU's, you're not really using the machine
- Provides cuBLAS, cuDNN
- Older modules available, but not recommended for use
 - Use latest version available when possible
 - Recompile on new releases

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

Software, Libraries, and Programming Models



Provided Software

- Vendor-supplied
 - IBM: ESSL (blas/lapack/fftw), MASS, SMPI
 - NVIDIA: CUDA, cuBLAS, cuDNN
 - Debuggers: Allinea Forge,
- Built by OLCF
 - Built in userspace without superuser privileges.
 - Often general purpose
 - 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

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=-qp' 'CXXFLAGS=-qp' 'FCFLAGS=-qp'
'--enable-parallel' ...
...
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
```

MPI Implementation – IBM Spectrum-MPI

- Based on OpenMPI, similar compiler wrappers and flags
mpicc, mpic++, mpiCC, mpifort, mpif77, mpif90, mpixl*
- Modules spectrum-mpi/10.2.0.10-20181214 (Default)
 - Avoid older releases.
- Updates require recompilation.
- Launcher jsrun (See separate talk in this series)
- Alt. implementations (OpenMPI, MPICH) **unavailable**

MPI environment

- When using XL, default `$OMPI_FC` is `xlf2008_r`
 - Works with standard MPI wrappers despite XL-specific wrappers `mpixlc`, `mpixlC`, `mpixlf`
 - Must change `$OMPI_FC` if using F77 code standards
- Adaptive routing enabled by default
 - `PAMI_IBV_ENABLE_000_AR=1`
 - `PAMI_IBV_QP_SERVICE_LEVEL=8`

OpenACC (Version 2.5)

Supported Compiler Environments

PGI (All Versions)

GCC 8.1.0+

`-acc -ta=nvidia:cc70`

`-fopenacc`

OpenMP

Compiler	3.1 Support	4.x Support	Enable OpenMP	Enable OpenMP 4.X Offload
IBM	FULL	PARTIAL	-qsmp=omp	-qsmp=omp -qoffload
GCC	FULL	PARTIAL	-fopenmp	-fopenmp
PGI	FULL		-fopenmp	
LLVM	FULL	PARTIAL	-fopenmp	-fopenmp -fopenmp-targets=nvptx64-nvidia-cuda --cuda-path=\${OLCF_CUDA_ROOT}

Building your own software

- Where to install?
 - NFS filesystem `/ccs/proj/<PROJECTID>` preferred: not purged, RO.
 - Avoid `$HOME`, especially `~/ .local/{bin,lib,share}`
 - Shared across architectures; may cause ABI or instruction set errors
- Where to build?
 - Recommend `/tmp/$USER`
 - faster performance than NFS
 - doesn't leave detritus in quota's `$HOME`, `/ccs/proj` dirs.

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 `$OLCF_{PKG}_ROOT` vars for use in build/configure scripts

Using Spack for missing dependencies

- Spack is a homebrew-like source-build package manager (<https://spack.readthedocs.io/en/latest/>)
 - Used to deliver most of the packages we provide
 - Not all Spack packages written to support ppc64le... Yet
 - OLCF uses some 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.

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.



Backup Slides



Sample Modulefile

```
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(myModuleFullName(),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

- 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"`