Compilers

Table of Contents

Available Compilers

- The following compilers are available on Sierra systems, and are discussed in detail below, along with other relevant compiler related information:
    - **XL:** IBM's XL C/C++ and Fortran compilers
    - **Clang:** IBM's C/C++ clang compiler
    - **GNU:** GNU compiler collection, C, C++, Fortran
    - **PGI:** Portland Group compilers
    - **NVCC:** NVIDIA's C/C++ compiler

Compiler Recommendations

- The recommended and supported compilers are those delivered from IBM (XL and Clang) and NVIDIA (NVCC):
    - Only XL and Clang compilers from IBM provide OpenMP 4.5 with GPU support.
    - NVCC offers direct CUDA support
    - The IBM **xlcuf** compiler also provides direct CUDA support
    - Please report all problems to the you may have with these to the LC Hotline so that fixes can be obtained from IBM and NVIDIA.
- The other available compilers (GNU and PGI) can be used for experimentation and for comparisons to the IBM compilers:
    - Versions installed at LC do not provide Open 4.5 with GPU support
    - If you experience problems with the PGI compilers, LC can forward those issues to PGI.
- Using OpenACC on LC's Sierra clusters is not recommended nor supported.
- XL Fortran is the recommended compiler for all Fortran; better performance and GPU support

Usage Notes, Recent News (8/22/18)

The Sierra systems environment is rapidly changing at this time. Regular status emails are sent to users with late breaking compiler (and other software) changes. Some important notes are included below.

- Volta versus Pascal GPUs: Recent XL and Clang compilers support both Volta and P100
    - LLNL-added options -qvolta and -qp100 can be used to manually pick GPU target
    - Automatically targets GPU for the platform you are compiling on otherwise
    - Cannot generate OpenMP 4.5 GPU code that works on both P100 and Volta. Performance hit makes this impractical, no JIT for OpenMP GPU code
    - Power8 executables works on Power9 but not necessarily other way around
    - CUDA-only GPU code (e.g. via RAJA) can work on both types of GPUs (due to JIT)
- Default MPI module is now **spectrum-mpi/rolling-release**
    - Tracks most recent "good" MPI release automatically
    - Do not have to recompile objects or libraries with new MPI releases
    - Relink existing builds or change rpath via chrpath to get benefit
    - Simplifies moving executables between CORAL EA and SIERRA clusters
- All bsub scripts run on a few shared launch nodes. Heavy duty compilations using make, spack, etc. can severely impact performance.
    - To get a dedicated backend/compute node for your work, use LC's **lrun -1** job launcher. For example:

        lrun -1 spack install zlib

- Don't use -O4, -O5, IPA, correctness checkers or rare options; may result in internal compiler issues at this time.
- The clang coral compilers claims its version 3.8 but really 6.0+ (makes nvcc happy). Most common clang question - it is really following

trunk

- Use static libraries for GPU code for a closer to normal library experience. NVIDIA's linker doesn't ignore unused object files, can get conflicts or unresolved symbols. Cleaning up 'libraries' generally resolves problems (duplicate 'main' routines common issue)
- Shared libraries for GPU code have severe constraints, XL bans outright
    - NVIDIA doesn't support dynamic linking of GPU code
    - If you construct your shared libraries so GPU code doesn't need dynamic linking, it can work
    - Generally shared library CPU code can launch self-contained GPU kernels, tricky in practice
- Complex build systems can get different compilers by mistake
    - Dangers of module system getting reset in sub-make command. Solution is to hard code full paths to compilers
- Need non-export controlled small reproducers to fix problems now
    - Almost all OpenMP 4.5 features work now in simple reproducers
    - Typically need your three layers of complexity to expose problems
    - Generating reproducers from descriptions no longer works, need to reduce from user code

Wrappers Scripts

- LC has created wrappers for most compiler commands, both serial and MPI versions.
- The wrappers perform LC customization and error checking. They also follow a string of links, which include other wrappers.
- To see the details on what a wrapper actually does:
    - use the **-v** flag for verbose output
    - use **-vvvv** to see the final full compiler line with MPI
- The wrappers located in /usr/tce/bin (in your PATH) will always point (symbolic link) to the default versions.
- Note: There may also be versions of the serial compiler commands in /usr/bin. Do not use these, as they are missing the LC customizations.
- If you load a different module version, your PATH will change, and the location may then be in either /usr/tce/bin or /usr/tcetmp/bin.
- To determine the actual location of the wrapper, simply use the command which *compilercommand* to view its path.
- Example: show location of default/current xlc wrapper, load a new version, and show new location:

```
% which xlc
/usr/tce/packages/xl/xl-beta-2017.11.28/bin/xlc

% module load xl/beta-2018.02.05

Due to MODULEPATH changes the following have been reloaded:
  1) spectrum-mpi/2017.11.10

The following have been reloaded with a version change:
  1) xl/beta-2017.11.28 => xl/beta-2018.02.05

% which xlc
/usr/tce/packages/xl/xl-beta-2018.02.05/bin/xlc
```

Versions

- There are several ways to determine compiler versions, discussed below.
- The default version of compiler wrappers is pointed to from /usr/tce/bin.
- To see available compiler module versions use the command module avail:
    - An **(L)** indicates which version is currently loaded.
    - A **(D)** indicates the default version.

For example:

```
% module avail
------------------- /usr/tce/modulefiles/Compiler/xl/beta-2017.11.28 --------------------
   spectrum-mpi/2017.11.10 (L)

----------------------------- /usr/tcetmp/modulefiles/Core -----------------------------
   StdEnv                 (L)      gsl/2.3                   python/2.7.14
   clang/coral-2017.11.09 (D)      gsl/2.4            (D)    python/3.6.4         (D)
   clang/coral-2017.12.06          ibmppt/alpha-2.4.0        scorep/3.0.0
   cmake/3.7.2                     ibmppt/beta-2.4.0         tau/2.26.2
   cmake/3.9.2            (D)      ibmppt/beta2-2.4.0 (D)    tau/2.26.3           (D)
   cuda/8.0                        ibmppt/2.3                totalview/2016.07.22
   cuda/9.0.176                    makedepend/1.0.5          totalview/2017X.3.1
   cuda/9.0.184                    petsc/3.7.6               totalview/2017.0.12
   cuda/9.1.76            (L,D)    petsc/3.8.3        (D)    totalview/2017.1.21  (D)
   cuda/9.1.85                     pgi/17.4                  totalview/2017.2.11
   flex/2.6.4                      pgi/17.7                  xl/beta-2017.11.28   (L,D)
   gcc/4.9.3                       pgi/17.9           (D)    xl/beta-2018.02.05
   git/2.9.3                       pgi/17.10
   gmake/4.2.1                     python/2.7.13

------------------------- /usr/share/lmod/lmod/modulefiles/Core -------------------------
   lmod/6.5.1    settarg/6.5.1

  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".
```

- You can also use any of the following commands to get version information:
  module display *compiler*
  module help *compiler*
  module key *compiler*
  module spider *compiler*.

  Examples below, using the IBM XL compiler (some output omitted):

```
% module display xl

-------------------------------------------------------------------------------------------
   /usr/tcetmp/modulefiles/Core/xl/beta-2017.11.28.lua:
-------------------------------------------------------------------------------------------
help([[LLVM/XL compiler beta 2017.11.28

IBM XL C/C++ for Linux, V13.1.6 (5725-C73, 5765-J08)
Version: 13.01.0006.0000
The license for the ESP version of IBM XL C/C++ for Linux, V13.1.6 (Beta) compiler
product will expire in 330 days on Wed Oct 31 22:00:00 2018.

IBM XL Fortran for Linux, V15.1.6 (5725-C75, 5765-J10)
Version: 15.01.0006.0000
The license for the ESP version of IBM XL Fortran for Linux, V15.1.6 (Beta) compiler
product will expire in 330 days on Wed Oct 31 22:00:00 2018.
]])
whatis("Name: XL compilers")
whatis("Version: beta-2017.11.28")
whatis("Category: Compilers")
whatis("URL: http://www.ibm.com/software/products/en/xlcpp-linux")
family("compiler")
prepend_path("MODULEPATH","/usr/tce/modulefiles/Compiler/xl/beta-2017.11.28")
prepend_path("PATH","/usr/tce/packages/xl/xl-beta-2017.11.28/bin")
prepend_path("MANPATH","/usr/tce/packages/xl/xl-beta-2017.11.28/xlC/13.1.6/man/en_US")
prepend_path("MANPATH","/usr/tce/packages/xl/xl-beta-2017.11.28/xlf/15.1.6/man/en_US")

% module help xl

---------------------- Module Specific Help for "xl/beta-2017.11.28" ----------------------
LLVM/XL compiler beta 2017.11.28
```

```
IBM XL C/C++ for Linux, V13.1.6 (5725-C73, 5765-J08)
Version: 13.01.0006.0000
The license for the ESP version of IBM XL C/C++ for Linux, V13.1.6 (Beta) compiler
 product will expire in 330 days on Wed Oct 31 22:00:00 2018.

IBM XL Fortran for Linux, V15.1.6 (5725-C75, 5765-J10)
Version: 15.01.0006.0000
The license for the ESP version of IBM XL Fortran for Linux, V15.1.6 (Beta) compiler
 product will expire in 330 days on Wed Oct 31 22:00:00 2018.

% module key xl

-----------------------------------------------------------------------------------
The following modules match your search criteria: "xl"
-----------------------------------------------------------------------------------
  spectrum-mpi: spectrum-mpi/2017.11.10
  xl: xl/beta-2017.11.28, xl/beta-2018.02.05
-----------------------------------------------------------------------------------
To learn more about a package enter:
   $ module spider Foo
where "Foo" is the name of a module
To find detailed information about a particular package you
must enter the version if there is more than one version:
   $ module spider Foo/11.1

% module spider xl

-----------------------------------------------------------------------------------
  xl:
-----------------------------------------------------------------------------------
     Versions:
        xl/beta-2017.11.28
        xl/beta-2018.02.05
-----------------------------------------------------------------------------------
  For detailed information about a specific "xl" module (including how to load the
 modules) use the module's full name.
  For example:
     $ module spider xl/beta-2018.02.05
-----------------------------------------------------------------------------------

% module spider xl/beta-2018.02.05

-----------------------------------------------------------------------------------
  xl: xl/beta-2018.02.05
-----------------------------------------------------------------------------------
    This module can be loaded directly: module load xl/beta-2018.02.05
    Help:
      LLVM/XL compiler beta beta-2018.02.05

      IBM XL C/C++ for Linux, V13.1.7 (Beta 1)
      Version: 13.01.0007.0000
      The license for the ESP version of IBM XL C/C++ for Linux, V13.1.7 (Beta)
 compiler product will expire in 155 days on Thu Jul 12 22:00:00 2018.

      IBM XL Fortran for Linux, V15.1.7 (Beta 1)
      Version: 15.01.0007.0000
```

```
      The license for the ESP version of IBM XL Fortran for Linux, V15.1.7 (Beta)
 compiler product will expire in 155 days on Thu Jul 12 22:00:00 2018.
```

- Finally, simply passing the --version option to the compiler invocation command will usually provide the version of the compiler. For example:

```
% xlc --version
IBM XL C/C++ for Linux, V13.1.6 (5725-C73, 5765-J08)
Version: 13.01.0006.0000
The license for the ESP version of IBM XL C/C++ for Linux, V13.1.6 (Beta) compiler
 product will expire in 251 days on Wed Oct 31 22:00:00 2018.

% gcc --version
gcc (GCC) 4.9.3
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

% clang --version
clang version 3.8.0 (ibmgithub:/CORAL-LLVM-Compilers/clang.git 0c6657a4a7f35c9c0c4f7
eab6e2dafac297918c1) (ibmgithub:/CORAL-LLVM-Compilers/llvm.git 2782ffd085eaa09d1c1ea
0670851fa403b532f0)
Target: powerpc64le-unknown-linux-gnu
Thread model: posix
InstalledDir: /usr/tce/packages/clang/clang-coral-2017.11.09/ibm/bin
```

Selecting Your Compiler and MPI Version

- Compiler and MPI software is installed as packages under /usr/tce/packages and/or /usr/tcetmp/packages.
- LC provides default packages for compilers and MPI. To see the current defaults, use the module avail command, as shown above in the Versions discussion. Note that a **(D)** next to a package shows that it is the default.
- The default versions will change as newer versions are released.
  - It's recommended that you use the most recent default compilers to stay abreast of new fixes and features.
  - You may need to recompile your entire application when the default compilers change.
- LMOD modules are used to select alternate compiler and MPI packages.
- To select an alternate version of a compiler and/or MPI, use the following procedure:
  1. Use module list to see what's currently loaded
  2. Use module key compiler to see what compilers and MPI packages are available.
  3. Use module load *package* to load the selected package.
  4. Use module list again to confirm your selection was loaded.

  Some examples (some output omitted):

```
% module list

Currently Loaded Modules:
  1) xl/beta-2017.11.28   2) spectrum-mpi/2017.11.10   3) cuda/9.1.76   4) StdEnv

% module key compiler

-------------------------------------------------------------------------------
The following modules match your search criteria: "compiler"
-------------------------------------------------------------------------------
  clang: clang/coral-2017.11.09, clang/coral-2017.12.06
  cuda: cuda/8.0, cuda/9.0.176, cuda/9.0.184, cuda/9.1.76, cuda/9.1.85
  gcc: gcc/4.9.3
  pgi: pgi/17.4, pgi/17.7, pgi/17.9, pgi/17.10
  spectrum-mpi: spectrum-mpi/2017.11.10
  xl: xl/beta-2017.11.28, xl/beta-2018.02.05
-------------------------------------------------------------------------------

% module load xl/beta-2018.02.05

Due to MODULEPATH changes the following have been reloaded:
  1) spectrum-mpi/2017.11.10

The following have been reloaded with a version change:
  1) xl/beta-2017.11.28 => xl/beta-2018.02.05

% module list

Currently Loaded Modules:
  1) cuda/9.1.76   2) StdEnv   3) xl/beta-2018.02.05   4) spectrum-mpi/2017.11.10
```

```
% module load pgi

Lmod is automatically replacing "xl/beta-2018.02.05" with "pgi/17.9"

Due to MODULEPATH changes the following have been reloaded:
  1) spectrum-mpi/2017.11.10

% module list

Currently Loaded Modules:
  1) cuda/9.1.76   2) StdEnv   3) pgi/17.9   4) spectrum-mpi/2017.11.10
```

- Notes:
  - When a new compiler package is loaded, the MPI package will be reloaded to use a version built with the selected compiler.
  - Only one compiler package is loaded at a time, with a version of the IBM XL compiler being the default. If a new compiler package is loaded, it will replace what is currently loaded. The default compiler commands for all compilers will remain in your PATH however.

IBM XL Compilers

- As discussed previously:
  - Wrapper scripts: Used by LC for most compiler commands.
  - Versions: There is a default version for each compiler, and usually several alternate versions also.
  - Selecting your compiler and MPI
- XL compiler commands are shown in the table below.

IBM XL Compiler Commands					
Language	**Serial**	**Serial + OpenMP 4.5**	**MPI**	**MPI + OpenMP 4.5**	**Comments**
C	xlc	xlc-gpu	mpixlc mpicc	mpixlc-gpu mpicc-gpu	The -gpu commands simply add the flags: -qsmp=omp -qoffload

C++	xlC xlc++	xlC-gpu xlc++-gpu	mpixlC mpiCC mpic++ mpicxx	mpixlC-gpu mpiCC-gpu mpic++-gpu mpicxx-gpu
Fortran	xlf xlf90 xlf95 xlf2003 xlf2008	xlf-gpu xlf90-gpu xlf95-gpu xlf2003-gpu xlf2008-gpu	mpixlf mpixlf90 mpixlf95 mpixlf2003 mpixlf2008	mpixlf-gpu mpixlf90-gpu mpixlf95-gpu mpixlf2003-gpu mpixlf2008-gpu

- Thread safety: LC always aliases the XL compiler commands to their _**r** (thread safe) versions. This is to prevent some known problems, particularly with Fortran. Note:the /usr/bin/xlf commands are not aliased as such, and they are not LC wrapper scripts - use is discouraged.
- OpenMP with NVIDIA GPU offloading is supported. For convenience, LC provides the -gpu commands, which set the option -qsmp=omp for OpenMP and -qoffload for GPU offloading. Users can do this themselves without using the -gpu commands.
- Optimizations:
  - The -O0 -O2 -O3 -Ofast options cause the compiler to run optimizing transformations to the user code, for both CPU and GPU code.
  - Options to target the Power8 architecture: -qarch=pwr8 -qtune=pwr8
  - Options to target the Power9 (Sierra) architecture: -qarch=pwr9 -qtune=pwr9
- Debugging - recommended options:
  - -g -O0 -qsmp=omp:noopt -qoffload -qfullpath
  - noopt - This sub-option will minimize the OpenMP optimization. Without this, XL compilers will still optimize the code for your OpenMP code despite -O0. It will also disable RT inlining thus enabling GPU debug information
  - -qfullpath - adds the absolute paths of your source files into DWARF helping TotalView locate the source even if your executable moves to a different directory.
- Documentation:
  - XLC/C++: Select the most recent version of Little Endian documents at http://www-01.ibm.com/support/docview.wss?uid=swg27036675
  - XLF: Select the most recent version of Little Endian documents at http://www-01.ibm.com/support/docview.wss?uid=swg27036672
  - IBM Redbook - Section 6.1.1 of "Implementing an IBM High-Performance Computing Solution on IBM Power System S822LC": http://www.redbooks.ibm.com/redbooks/pdfs/sg248280.pdf
  - IBM White Paper "Code Optimization with the IBM XL compilers on Power Architectures": http://www-01.ibm.com/support/docview.wss?uid=swg27005174&aid=1

IBM Clang Compiler

- The Sierra systems use the Clang compiler from IBM.
- As discussed previously:
  - Wrapper scripts: Used by LC for most compiler commands.
  - Versions: There is a default version for each compiler, and usually several alternate versions also.
  - Selecting your compiler and MPI
- Clang compiler commands are shown in the table below.

Clang Compiler Commands					
Language	**Serial**	**Serial + OpenMP 4.5**	**MPI**	**MPI + OpenMP 4.5**	**Comments**
C	clang	clang-gpu	mpiclang	mpiclang-gpu	Use the -gpu commands for OpenMP4.5 GPU support
C++	clang++	clang++-gpu	mpiclang++	mpiclang++-gpu	

- OpenMP with NVIDIA GPU offloading is supported. For convenience, LC provides the -gpu commands, which set the option -fopenmp for OpenMP and -fopenmp-targets=nvptx64-nvidia-cuda for GPU offloading. Users can do this themselves without using the -gpu commands. However, use of LC's -gpu commands is recommended at this time since the native Clang flags are verbose and subject to change.
- Documentation:
  - TO BE ADDED LATER

GNU Compilers

- As discussed previously:
  - Wrapper scripts: Used by LC for most compiler commands.
  - Versions: There is a default version for each compiler, and usually several alternate versions also.
  - Selecting your compiler and MPI
- GNU compiler commands are shown in the table below.

GNU Compiler Commands					
Language	**Serial**	**Serial + OpenMP 4.5**	**MPI**	**MPI + OpenMP 4.5**	**Comments**
C	gcc cc	n/a	mpigcc	n/a	For OpenMP use the flag: -fopenmp
C++	g++ c++	n/a	mpig++	n/a	
Fortran	gfortran	n/a	mpigfortran	n/a	

- OpenMP with NVIDIA GPU offloading is NOT currently provided. OpenMP 4.5 is supported starting with version 6.1, however it is not for NVIDIA GPU offload. Target regions are implemented on the multicore host instead.
- Optimization flags:
  - POWER8: -mcpu=power8 -mtune=power8
    Also see Section 6.1.2 of the IBM Redbook: Implementing an IBM High-Performance Computing Solution on IBM Power System S822LC
  - POWER9: -mcpu=powerpc64le -mtune=powerpc64le
- Documentation:
  - GNU online documentation at: https://gcc.gnu.org/onlinedocs/

PGI Compilers

- As discussed previously:
  - Wrapper scripts: Used by LC for most compiler commands.
  - Versions: There is a default version for each compiler, and usually several alternate versions also.
  - Selecting your compiler and MPI
- PGI compiler commands are shown in the table below.

PGI Compiler Commands					
Language	**Serial**	**Serial + OpenMP 4.5**	**MPI**	**MPI + OpenMP 4.5**	**Comments**
C	pgcc cc	n/a	mpipgcc	n/a	pgf90 and pgfortran are the same compiler, supporting the Fortran 2003 language specification For OpenMP use the flag: -mp
C++	pgc++	n/a	mpig++	n/a	
Fortran	pgf90 pgfortran	n/a	mpipgf90 mpipgfortran	n/a	

- OpenMP with NVIDIA GPU offloading is NOT currently provided. Most of OpenMP 4.5 is supported, however it is not for NVIDIA GPU offload. Target regions are implemented on the multicore host instead. See the product documentation (link below) "Installation Guide and Release Notes" for details.
- GPU support is via CUDA and OpenACC.
- Documentation:
  - PGI version 17.10 for OpenPOWER and NVIDIA Processors: http://www.pgroup.com/resources/docs/17.10/openpower/index.htm
  - Compiler options: http://www.pgroup.com/resources/docs/17.10/openpower/pgi-ref-guide/index.htm#cmdln-options-ref
  - PGI version 18.7 for OpenPOWER and NVIDIA Processors: http://www.pgroup.com/resources/docs/18.7/openpower/index.htm
  - Compiler options: http://www.pgroup.com/resources/docs/18.7/openpower/pgi-ref-guide/index.htm#cmdln-options-ref
  - Presentation from the ORNL Workshop Jan. 2017: Porting to OpenPower & Tesla with PGI

NVIDIA NVCC Compiler

- The NVIDIA nvcc compiler driver is used to compile C/C++ CUDA code:
  - nvcc compiles the CUDA code.
  - Non-CUDA compilation steps are forwarded to a C/C++ host (backend) compiler supported by nvcc.
  - nvcc also translates its options to appropriate host compiler command line options.
  - NVCC currently supports XL, GCC, and PGI C++ backends, with GCC being the default.
- Location:
  - The NVCC C/C++ compiler is located under usr/tce/packages/cuda/.
  - Other NVIDIA software and utilities (like nvprof, nvvp) are located here also.
  - The default CUDA build should be in your default PATH.
- As discussed previously:
  - Versions: There is a default version for each compiler, and usually several alternate versions also.

    - Selecting your compiler and MPI
- Architecture flag:
    - Tesla P100 (Pascal) for Early Access systems: -arch=sm_60
    - Tesla V100 (Volta) for Sierra systems: -arch=sm_70
- Selecting a host compiler:
    - The GNU C/C++ compiler is used as the backend compiler by default.
    - To select a different backend compiler, use the -ccbin=*compiler* flag. For example:

```
nvcc -arch=sm_70 -ccbin=xlC myprog.cu
nvcc -arch=sm_70 -ccbin=clang myprog.cu
```

    - The alternate backend compiler needs to be in your path. Otherwise you need to specify the full pathname.
- Source file suffixes:
    - Source files with CUDA code should have a **.cu** suffix.
    - If source files have a different suffix, use the -x cu flag. For example:

```
nvcc -arch=sm_70 -ccbin=xlc -x cu myprog.c
```

- Documentation:
    - NVDIA NVCC: http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/.