Programing Environment and Compilation

New User Training
June 16, 2020

Zhengji Zhao
NERSC User Engagement Group
Outline

● Cori compilation overview
● Compile/link lines:
  - Compiler +
  - Compiler Flags +
  - `-I/path/to/headers` +
  - `-L/path/to/library -l<library>`
● Available libraries, and linking examples
● Spack - a package manager
● Summary
Cori Compilation Overview
Compilations on Cori

- Three programming environments are supported on Cori
  - Intel, GNU and Cray compilers are available; Intel is the default
- The programming environment modules, “PrgEnv-intel”, “PrgEnv-gnu”, and “PrgEnv-cray”, which include the compilers and matching libraries, provide user friendly programming environments
- Use “module swap PrgEnv-Intel PrgEnv-cray” to switch compilers
- Use the compiler wrappers provided by Cray, ftn, cc and CC for Fortran, C and C++ respectively, so that the header and library paths and libraries can be added on the compile/link lines automatically.
Cori System Configurations

- Cori KNL and Haswell – a Cray XC40
  - Cori has 9688 single-socket Intel® Xeon Phi™ Processor 7250 ("Knights Landing") nodes @1.4 GHz with 68 cores (272 threads) per node, two 512 bit vector units per core, and 16 GB high bandwidth on-package memory (MCDRAM) with 5X the bandwidth of DDR4 DRAM memory (>400 GB/sec) and 96 GB DDR4 2400 MHz memory per node
  - In addition, Cori has 2388 dual-socket 16-core Intel® Xeon™ Processor E5-2698 v3 ("Haswell") nodes @2.3GHz with 32 cores (64 threads) per node, two 256 bit vector units per core, 128 GB 2133 MHz DDR4 memory
  - Cori nodes are interconnected with Cray’s Aries network with Dragonfly topology

- Binary compatibility: Haswell binaries run on KNL, but not vise versa, because KNL supports the extended instruction sets

- Separate builds for Haswell and KNL are recommended for optimal performance
Compilations on Cori (Cont.)

- Cross compilation: compiling for compute nodes from login nodes (Haswell)
- Default environment loads the craype-haswell module on Cori, which sets “CRAY_CPU_TARGET=haswell” for Cori. So, compilers build binaries that are optimized for Haswell processors by default when compiling with the compiler wrappers

Default programming environment on Cori:

```
zz217@cori06:~$ module list
Currently Loaded Modulefiles:
  1) modules/3.2.11.4
  2) nsg/1.2.0
  3) altld/2.0
  4) darshan/3.1.7
  5) intel/19.0.3.199
  6) craype-network-aries
  7) craype/2.6.2
  8) cray-libscl/19.06.1
zz217@cori06:~$```

```bash
zz217@cori06:~$ module list
Currently Loaded Modulefiles:
  1) modules/3.2.11.4
  2) nsg/1.2.0
  3) altld/2.0
  4) darshan/3.1.7
  5) intel/19.0.3.199
  6) craype-network-aries
  7) craype/2.6.2
  8) cray-libscl/19.06.1
  9) udrreg/2.3.2-7.0.1.1_3.29__g8175d3d.ari
 10) ugni/6.0.14-0.7.0.1.1_7.32__ge78e5b0.ari
 11) pml/5.0.14
 12) dmapp/7.1.1-7.0.1.1_4.43__g38cf134.ari
 13) gnl-headers/5.0.12.0-7.0.1.1_6.27__g3b1768f.ari
 14) xpmem/2.2.20-7.0.1.1_4.8__g0475745.ari
 15) job/2.2.4-7.0.1.1_3.34__g36b56f4.ari
 16) dvs/2.12_2.2.156-7.0.1.1_8.6__g5aab709e
 17) alps/6.6.57-7.0.1.1_5.10__g1b735148.ari
 18) rca/2.2.20-7.0.1.1_4.42__g8e3fb5b.ari
 19) atp/2.1.3
 20) PrgEnv-intel/6.0.5
 21) craype-haswell
 22) cray-mpich/7.7.10
 23) craype-hugepages2M
```
To Compile for Cori Haswell

- Intel programming environment is the default

To use Intel compilers

```
ftn -O3 mycode.f90  # Fortran
cc -O3 mycode.c     # for C
CC -O3 myC++code.C  # for C++
```

To use GNU compilers

```
module swap PrgEnv-intel PrgEnv-gnu
ftn -O3 mycode.f90  # Fortran
cc -O3 mycode.c     # for C
CC -O3 myC++code.C  # for C++
```

To use Cray compilers

```
module swap PrgEnv-intel PrgEnv-cray
ftn -O3 mycode.f90  # Fortran
cc -O3 mycode.c     # for C
CC -O3 myC++code.C  # for C++
```

Note: the compiler wrappers, ftn, cc, and CC, are not Cray compilers; They invoke the Intel, GNU, or Cray compilers under the hood, depending on the loaded PE module (PrgEnv-<compiler>)}
To Compile for Cori KNL

- Applications are cross compiled for KNL nodes from the login nodes (Haswell)
- Do “module swap craype-haswell craype-mic-knl” before compiling for KNL to build binaries that are optimized for the KNL architecture

```bash
module swap craype-haswell craype-mic-knl
ftn -O3 mycode.f90 # Fortran
cc -O3 mycode.c # for C
CC -O3 myC++code.C # for C++
```
Compile/Link Lines
Compiler Recommendations

● Will not recommend any specific compiler
  ○ Intel - better chance of getting processor specific optimizations, especially for KNL
  ○ Cray compiler – many new features and optimizations, especially with Fortran; useful tools like reveal work with Cray compiler only
  ○ GNU - widely used by open software

● Start with the compilers that vendor/code developers used so to minimize the chance to hit compiler and code bugs, then explore different compilers for optimal performance
## Compiler Flags

<table>
<thead>
<tr>
<th>Intel</th>
<th>GNU</th>
<th>Cray*</th>
<th>Description/ Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>-O2</td>
<td>-O0</td>
<td>-O2</td>
<td>Default Optimization Level</td>
</tr>
<tr>
<td>default, or -O3</td>
<td>-O2 or -O3,-Ofast</td>
<td>default</td>
<td>Recommended Compiler Flags</td>
</tr>
<tr>
<td>-qopenmp</td>
<td>-fopenmp</td>
<td>-h omp for Fortran; -fopenmp for C/C++</td>
<td>OpenMP Flag</td>
</tr>
<tr>
<td>-g</td>
<td>-g</td>
<td>-g</td>
<td>Debug</td>
</tr>
<tr>
<td>-v</td>
<td>-v</td>
<td>-v</td>
<td>Verbose</td>
</tr>
</tbody>
</table>

*) Starting from CCE 9.0 version, Cray C/C++ compilers use LLVM as backend

- **Validity check after compilation**
- **Compilers’ default behavior could vary between compilers**
  - Default number of OpenMP threads used is all CPU slots available for Intel and GNU compilers; 1 for Cray compiler
  - Use compiler man page for available compiler optimization flags, e.g., man ifort
Header and Library Paths and Libraries

- Manually:
  - Find out the paths to the headers, and libraries, then add
    “-I <header path> -L<library path> -l<libraries>” to your compile/link lines

- Automatically:
  - Using the compiler wrappers, which can do this for you
  - Compiler wrappers are strongly recommended
Compiler Wrappers, ftn, cc and CC

● Use ftn, cc, and CC to compile Fortran, C and C++ codes, respectively, instead of invoking the native compilers directly, such as ifort, icc, icpc, gfortran, gcc, g++, etc.
  ○ The compiler wrappers wraps the underlying compilers with additional compiler and linker flags depending on the modules loaded in the environment
  ○ The same compiler wrapper command (e.g. ftn) is used to invoke any compilers supported on the system (Intel, GNU, Cray)

● Compiler wrappers do cross compilations
  ○ Compiling applications on login nodes to run on compute nodes
  ○ For some applications, may need to set the –host=x86_64 configure option (if available) when compiling for KNL from a login node
  ○ If compiling on a KNL node is needed, do “salloc –N 1 –q interactive –C knl –t 4:00:00” to get on to a compute node
Compiler Wrappers, ftn, cc and CC (Cont.)

- Compiler wrappers link dynamically by default on Cori
  - May need to load the same set of modules at run time or set the LD_LIBRARY_PATH env so that shared libraries can be found. Alternatively, consider using the “-Wl,--rpath=<library path>” option when compiling
  - A dynamically linked executable may take some time to load shared libraries when running with a large number of processes

- Use the -static option of the compiler wrappers or set the environment variable “CRAYPE_LINK_TYPE=static” to link statically
  - Preferred for performance at scale
Why Compiler Wrappers?

- They include the architecture specific compiler flags into the compilation/link lines automatically

<table>
<thead>
<tr>
<th></th>
<th>Intel*</th>
<th>GNU</th>
<th>Cray</th>
<th>Required Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cori KNL</td>
<td>-xMIC-AVX512</td>
<td>-march=knl</td>
<td>-h cpu=mic-knl</td>
<td>craype-mic-knl</td>
</tr>
<tr>
<td>Cori Haswell</td>
<td>-xCORE-AVX2</td>
<td>-march=core-avx2</td>
<td>-h cpu=haswell</td>
<td>craype-haswell</td>
</tr>
</tbody>
</table>

* for the latest Intel compilers, -march=knl,haswell can be used instead of -xcode.

- Automatically add header and library paths and libraries on the compilation/link lines
  - Compiler wrappers use the pkg-config tools to dynamically detect paths and libs from the environment (working with cray modules and some NERSC modules)
  - The architecture specific builds of libraries will be linked into
- Allow user provided options to take precedence
Verbose Output from Compiler Wrappers

- Depending on the modules loaded, compiler wrappers link to the MPI, LAPACK/BLAS/ScaLAPACK libraries, and more automatically.
- Library names on Cori could be different from what you used before.
Verbose Output from Compiler Wrappers (Cont.)
Available Libraries, and Linking Examples
Available Libraries

- **Access via modules, type** “module avail” or “module avail -S <your string>” **to see the available modules**

- **Cray supports many software packages – Cray Developer Toolkits (CDT)**
  - Modules from /opt/cray/pe/modulefiles, etc.
  - There are different builds for different compilers
  - Programming environment modules allow the libraries built with the matching compilers to be linked to

- **NERSC staff also supports many libraries**
  - Modules from /usr/common/software/modulefiles, etc.
  - Some of them interact with the Cray compiler wrappers while many of them do not
Available Libraries (Cont.)

● Where are the libraries and header files?
  ○ Use “module show <module name>” to see the installation paths
  ○ Run “ls -l <installation_path>/include” and “ls -l <installation_path>/lib” to see the library files
  ○ e.g., Cray MPICH library:

```
zz217@cori06:~> module show cray-mpich
.../opt/cray/pe/modulefiles/cray-mpich/7.7.10:
  ...
  setenv CRAY_MPICH_DIR /opt/cray/pe/mpt/7.7.10/gni/mpich-intel/16.0
  setenv MPICH_DIR /opt/cray/pe/mpt/7.7.10/gni/mpich-intel/16.0
  ...
zz217@cori06:~> ls -l $CRAY_MPICH_DIR
total 0
drwxr-xr-x 2 root root  628 Nov 14  2019 include
drwxr-xr-x 3 root root 1239 Nov 14  2019 lib
```
Example: Linking to Cray Provided Libraries

- Linking to the Cray MPI and Cray Scientific libraries are automatic by default if compiler wrappers are used
  
  ```
  CC parallel_hello.cpp   #or  ftn dgemmx1.f90
  ```

- Linking to HDF5 and NETCDF libraries are automatic, user just need to load the cray-hdf5 or cray-netcdf modules
  
  ```
  module load cray-hdf5
  cc h5write.c
  ```

- Note the library name could be different. Using the `-v` option to see the library names and other detail about the linking
Example: Linking to Cray Provided Libraries

- Linking to PETSc libraries are automatic, but users need to choose a proper module (e.g., real/complex, 32 or 64 bit integer builds)
  - E.g., “module load cray-petsc-complex-64”
  - Use “cc –v test1.c” to see the linking detail (test1.c can be any skeleton C code)

- Linking to fftw libraries
  - module load cray-fftw
  - Loading the cray-fftw module links to the pthread version of the library, “-lfftw3f_mpi -lfftw3f_threads -lfftw3f -lfftw3_mpi -lfftw3_threads -lfftw3”
  - Use -qopenmp to link with the OpenMP version of FFTW
Some of the NERSC provided modulefiles are written to interact with the Cray compiler wrappers, e.g., elpa module on Cori

- Type "module show <module name>" to check if the envs "<libname>_PKGCONFIG_LIBS", "PE_PKGCONFIG_PRODUCTS", and "PKG_CONFIG_PATH" are defined in the modulefiles, which compiler wrappers look for

Most of the NERSC provided modulefiles do not interact with the compiler wrappers, user need to provide the include and library paths and libraries manually, e.g., GSL

- Examples of linking to the NERSC provided library modules

```
module load elpa
# automatically link to elpa and MKL ScaLAPACK libraries
ftn -gopenmp -v test2.f90
```

```
module load gsl
ftn test3.f90 $GSL
# where GSL=-I/global/common/sw/cray/cnl7/haswell/gsl/2.5/intel/19.0.3.199/7twqxxq/include -L/global/common/sw/cray/cnl7/haswell/gsl/2.5/intel/19.0.3.199/7twqxxq/lib -l gsl -l gslcblas
```
Linking to Intel MKL Library

● Resource:
  ○ Learn from Intel compiler verbose output using the “-mkl={parallel,sequential,cluster}” flag

● For intel compiler, use \(-\text{mkl}\) flag
  ○ ftn test1.f90 \(-\text{mkl}\)  # default to parallel, the multi-threaded MKL
  # the cray-libsci will be ignored if -mkl is used.
Linking to Intel MKL Library (Cont.)

- For GNU compiler (e.g., to link to 32-bit integer build statically):
  - Save the $MKLROOT from the Intel compiler module, and then
  - Threaded: 
    ```
    -L$MKLROOT/lib/intel64 -Wl,--start-group -lmlk1_gf_lp64
    -lmlk1_gnu_thread -lmlk1_core -lgomp -Wl,--end-group
    -lpthread -lm -ldl
    ```
  - ScaLAPACK: 
    ```
    -L$MKLROOT/lib/intel64 -Wl,--start-group
    -lmlk1_gf_lp64 -lmlk1_gnu_thread -lmlk1_scalapack_lp64
    -lmlk1_blacs_intelmpi_lp64 -lmlk1_core –Wl,--end-group -lgomp
    -lpthread -lm -ldl
    ```

Note, 
```
-Wl,--start-group” … “-Wl,--end-group ” for static linking
```
Linking to Intel MPI library

- Cray MPICH libraries are recommended for performance especially at scale
- Compiler wrappers link to Cray MPICH libraries by default
- However, if you need to link to Intel MPI library, do

```bash
module load impi
mpifort test1.f90       #or mpiicpc test1.C
```

- Note that the binaries linked to the Intel MPI need to run with srun instead of mpirun to get a proper process/thread affinity, [https://docs.nersc.gov/jobs/examples/#intel-mpi](https://docs.nersc.gov/jobs/examples/#intel-mpi)
Spack - A Package Manager
Spack - A Package Manager

● Simple package installations
  ○ Spack installs the latest version of a package and its dependencies by default.

● Custom versions and configurations
  ○ Spack allows installation to be customized using a simple spec syntax to specify the version, build compiler, compile-time options, cross-compile platform, and dependencies, etc.

● Non-destructive installs
  ○ Uses hash from package/dependency configuration in the installation prefix

● Peaceful coexistence of packages
  ○ Uses _RPATH to link dependencies; no need to manipulate _LD_LIBRARY_PATH at runtime

● Easy package creation
  ○ package files are written in pure Python; one file for different builds
Spack Modules on Cori

- NERSC provides Spack modules for users
  - Allows access to NERSC’s recommended, up-to-date configuration files
  - Configured to make use of the NERSC staff installed software, so no need to build each dependent package by themselves redundantly, saving compilation time and storage space

- To access, module load spack  #currently spack 0.14.2 is made available
  - Create a directory (at the first time invocation only), $HOME/sw, and a few subdirectories under it on your account to build and install software on. E.g, software will be installed on your $HOME/sw/opt/spack directory, and the Spack generated modulefiles will be available at the $HOME/sw/share/spack/modules directory. You can create your own package files under the directory $HOME/sw/var/spack/repos/$USER/packages

- Users can overwrite these global setting with config files at ~/.spack
module load spack
spack list       # to see Spack supported packages
spack find -p    # to see the software installed by Spack

cd ~/sw/build
spack install <package name>   # to install a package

e.g.,
spack install wannier90@2.1.0 %intel@19.0.3.199 ^intel-mkl@19.0.3.199
spack install autoconf@2.69%intel@19.0.3.199 arch=cray-sles15-x86_64
spack install cp2k %intel blas=mkl +mpi ^intel-mkl
arch=cray-cn17-haswell
train467@cori09:/sw/build> spack install xerces-c
[+] /global/common/sw/cray/cnl7/haswell/libiconv/1.16/intel/19.0.3.199/vr7fhfz
[+] /global/common/sw/cray/cnl7/haswell/libiconv/1.16/intel/19.0.3.199/vr7fhfz
=> 41416: Installing xerces-c
=> Fetching https://archive.apache.org/dist/xerces/c/3/sources/xerces-c-3.2.2.tar.bz2
###########################################################################
###########################################################################
#### 100.0%
=> Staging archive:
  /global/cscratch1/sd/train467/sw/spackbuild/spack-stage-xerces-c-3.2.2-ggdbuk52jm72rebq67s3ro2vxzaezog4/xerces-c-3.2.2.tar.bz2
=> Created stage in
  /global/cscratch1/sd/train467/sw/spackbuild/spack-stage-xerces-c-3.2.2-ggdbuk52jm72rebq67s3ro2vxzaezog4
=> No patches needed for xerces-c
=> 41416: xerces-c: Building xerces-c [AutotoolsPackage]
=> 41416: xerces-c: Executing phase: 'autoreconf'
=> 41416: xerces-c: Executing phase: 'configure'
=> 41416: xerces-c: Executing phase: 'build'
=> 41416: xerces-c: Executing phase: 'install'
=> 41416: xerces-c: Successfully installed xerces-c
 Fetch: 2.30s. Build: 2m 34.05s. Total: 2m 36.35s.
[+] /global/homes/t/train467/sw/opt/spack/cray-cnl7-haswell/intel-19.0.3.199/xerces-c-3.2.2-ggdbuk5
If “spack install” fails

- Open an issue at https://github.com/spack/spack/issues

- Copy the failed package.py to your local repo to modify:
  
  ```
  cp -pr $SPACK_ROOT/var/spack/repos/builtin/packages/<package name>
  $HOME/sw/var/spack/repos/$USER/packages
  ```

  ```
  spack edit <package name>
  ```

  ```
  cd $HOME/sw/build
  ```

  ```
  spack install <package name> ...
  ```

  ______________

  train467@cori02:~/sw/build> spack config get repos
  repos:
  - $HOME/sw/var/spack/repos/$USER
  - /global/common/sw/spack/0.14.2/var/spack/repos/nersc
  - /global/common/sw/spack/0.14.2d/var/spack/repos/builtin
Summary
Summary

- Three supported programming environments: Intel, GNU, and Cray

- Use compiler wrappers where possible,
  - Add architecture specific optimization flags
  - Automatically add the header and library paths in to the compile/link lines, and link to the Cray MPI, LibSci and other Cray provided libraries if the modules are loaded

- To compile for Cori KNL, do
  - module swap craype-haswell craype-mic-knl
Summary (Cont.)

● There are many libraries available, use them where possible
  ○ Use “module avail” command to check available libraries
  ○ Use “module show <module name>” to see the installation paths if needed

● Most NERSC staff supported modules do not interact with the compiler wrappers
  ○ Users need to provide the header and library paths and libraries manually

● On Cori, applications are linked dynamically by default
  ○ Use of rpath is recommended
  ○ Use /global/common/software/<project name> directory to store your shared libraries and python modules as well as the dynamically linked applications
Summary (Cont.)

- To link applications statically, use the “-static” compiler wrapper option or set the env “CRAYPE_LINK_TYPE=static” before compiling
  - Static linking is recommended for large scale applications
- Learn from the compiler verbose output (-v)
- Spack is recommended to build your software
Recommended Readings

● NERSC website, especially,
  ○ Programming page at [http://docs.nersc.gov](http://docs.nersc.gov),
  ○ [https://docs.nersc.gov/programming/performance-debugging-tools/build-tools/#spack](https://docs.nersc.gov/programming/performance-debugging-tools/build-tools/#spack)

● Compiler and linker man pages:
  ○ ifort, icc, icpc, crayftn, etc.
  ○ man ld (“-Wl,-zmuldefs”, “-Wl,-y<symbol>”)

● Spack
  ○ [https://docs.nersc.gov/development/build-tools/spack/](https://docs.nersc.gov/development/build-tools/spack/)
Thank You and Welcome to NERSC!