Compiling Codes at NERSC

Zhengji Zhao
NERSC New User Training,
February 23, 2017, Berkeley CA
This talk will focus on

- Cori (Haswell and KNL nodes) and Edison
- Compile/link lines
  - Compiler +
  - Compiler Flags +
    - -I/path/to/headers +
    - -L/path/to/library -l<library>
- Available compilers, libraries, how to compile - a couple of tips, compile for Cori KNL, summary
Separate builds for Cori and Edison is recommended

- Cori KNL and Haswell
  - Cori has over 9300 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 over 2000 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.

- Edison
  - Edison has 5586 dual-socket 12-core Intel(R) Xeon(R) CPU E5-2695 v2 ("Ivy Bridge") nodes @2.40GHz with 24 cores (48 threads) per node, two 256 bit vector units per core, 64 GB DDR3 1866 MHz memory. Edison nodes are interconnected with Cray’s Aries network with Dragonfly topology.

- Compilations on Edison and Cori are very similar, and binaries can be built to be compatible, however, however, we recommend to a separate build for each platform.
• **Intel** compiler is default on both Cori and Edison
• Access through modules
• Three programming environments
  – PrgEnv-intel, PrgEnv-gnu, and PrgEnv-cray
  – Module swap PrgEnv-Intel PrgEnv-cray to swap compilers and programming environment.

```
zz217@cori03:~> module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.5
  2) nsg/1.2.0
  3) intel/17.0.1.132
  4) craype-network-aries
  5) craype/2.5.7
  6) cray-libsci/16.09.1
  7) udreg/2.3.2-4.6
  8) ugni/6.0.12-2.1
  9) pmi/5.0.10-1.0000.11050.0.0.ari
 10) dmapp/7.1.0-12.37
 11) gni-headers/5.0.7-3.1
 12) xpmem/0.1-4.5
 13) job/1.5.5-3.58
 14) dvs/2.7.0.9.0-2.243
 15) alps/6.1.3-17.12
 16) rca/1.0.0-8.1
 17) atp/2.0.3
 18) PrgEnv-intel/6.0.3
 19) craype-haswell
 20) cray-shmem/7.4.4
 21) cray-mpich/7.4.4
 22) altd/2.0
 23) darshan/3.0.1.1
```

• Other available compilers, such as bupc, llvm, etc, will not be covered in this presentation.
Use ftn, cc, and CC to compile Fortran, C and C++ codes, respectively, instead of the underlying native compilers, such as ifort, icc, icpc, gfortran, gcc, g++, crayftn, craycc, crayc++

- The compiler wrappers are the same as the underlying compilers with the addition of flags included and libraries linked by default
- The same compiler wrapper command (.e.g. ftn) can refer to any underlying compiler available on the system (Intel, GNU, Cray)

**Compilers wrappers link statically**
- Preferred for performance at scale

**Use –dynamic to link dynamically**
- For larger scale runs, dynamically linked code may take a long time to load the shared libraries

**Cross compiling**
- Compile on login nodes to run on compute nodes
They include the architecture specific compiler flags into the compilation/link line automatically.

- They automatically add the header and library paths and libraries on the compilation/link lines, so you do not have to explicitly provide them.
  - Compiler wrappers use the pkg-config tools to dynamically detect paths and libs from the environment (loaded cray modules and some NERSC modules)
- Yet allow user provided options to take the precedence
  - You may need to remove --xhost option in your compilation/link line

<table>
<thead>
<tr>
<th></th>
<th>Intel</th>
<th>GNU</th>
<th>Cray</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cori KNL</td>
<td>-xAVX512</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>
<tr>
<td>Edison Ivy Bridge</td>
<td>-xCORE-AVX-I</td>
<td>-march=corei7-avx</td>
<td>-h cpu=ivybridge</td>
<td>craype-ivybridge</td>
</tr>
</tbody>
</table>
What do compiler wrappers link by default?

- Depending on the modules loaded, MPI, LAPACK/BLAS/ScalAPACK libraries, and more
- Library names could be different from what you used before
Compiler recommendations

• Will not recommend any specific compiler
  – Intel - better chance of getting processor specific optimizations
  – Cray compiler – many new features and optimizations, especially with Fortran
  – GNU - widely used by open software

• Start with the compilers that vendor/code developers used, to minimize the chance to hit the compiler and code bugs, then explore different compilers if you care the 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</td>
</tr>
<tr>
<td>default, or -O3</td>
<td>-O2 or -O3,-Ofast</td>
<td>default</td>
<td>recommended</td>
</tr>
<tr>
<td>-qopenmp</td>
<td>-fopenmp</td>
<td>default, or –h omp</td>
<td>OpenMP</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>

- **Validity check after compilation**
  - Run tests and check with the references if provided
  - If higher Debug version to check the validity

- **Default behaviors could varies between compilers**
  - Default number of OpenMP threads
  - * -g: Cray compiler disables all optimizations, try –G2
Verbose output from compiler wrappers contains many useful information
Available libraries

- **Cray supports many software packages**
  - Access via modules, type “module avail” to see the available modules
  - Different builds for different compilers
  - Programming environment can selectively pick the matching libraries to load

- **NERSC also supports many libraries**
  - many of them interact with the Cray compiler wrappers, while some of them do not.

- **Where are the libraries ?**
  - Module avail
  - Module show <module name>

```
zz217@cori06:~> echo $MODULEPATH
/opt/cray/pe/ari/modulefiles:/opt/cray/ari/modulefiles:/opt/cray/pe/craype/2.5.7/modulefiles:/opt/cray/pe/modulefiles:/opt/cray/modulefiles:/opt/modulefiles:/usr/common/software/modulefiles:/usr/syscom/nsg/modulefiles:/usr/syscom/nsg/opt/modulefiles:/usr/common/das/modulefiles:/usr/common/ftg/modulefiles:/opt/cray/craype/default/modulefiles
```
Demo for a few commands

- Module avail
- Module show <module name>
- Module load cray-petsc
- Module load fftw
- Module load cray-netcdf

- Intel® Math Kernel Library Link Line Advisor
  - Learn from compiler verbose output, 
    - -mkl={serial,thread,cluster}

- cdt modules could be useful to recover the previous programming environments
Resolving missing libraries and unresolved symbols

• Only shared libraries available
  – E.g., /usr/bin/ld: cannot find -ldl
  – Try -dynamic

• Static library linking order, try `-Wl,--start-group` ...
  `-Wl,--end-group`, e.g.,

  ```
  LLIBS = -Wl,--start-group $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a \
           $(MKLROOT)/lib/intel64/libmkl_scalapack_lp64.a \
           $(MKLROOT)/lib/intel64/libmkl_blacs_intelmpi_lp64.a \
           $(MKLROOT)/lib/intel64/libmkl_intel_thread.a $(MKLROOT)/lib/intel64/libmkl_core.a \
           -Wl,--end-group \
           /usr/common/software/wannier90/1.2/knl/intel/lib/libwannier.a \
           -Wl,-zmuldefs
  ```

• Use `grep -R and readelf --s |grep <symbol>` to search for the unresolved symbols.
How to compile for Cori KNL

Best: Use compiler options to build for KNL
module swap craype-haswell craype-mic-knl

- The loaded craype-* module sets the target that the compiler wrappers (cc, CC, ftn) build for
  - Eg -mknl (GNU compiler),
  - -hmic-knl (Cray compiler)
- **craype-haswell** is default on login nodes
- **craype-mic-knl** is for KNL nodes
Best: Compiler settings to target KNL

Alternate:

CC -axMIC-AVX512,CORE-AVX2 <more-options> mycode.c++

- Only valid when using Intel compilers (cc, CC or ftn)
- -ax<arch> adds an “alternate execution paths” optimized for different architectures
  - Makes 2 (or more) versions of code in same object file
- **NOT AS GOOD as the craype-mic-knl module**
  - (module causes versions of libraries built for that architecture to be used - eg MKL)
How to compile for Cori KNL

Recommendations:

• For best performance, use the craype-mic-knl module
  
  module swap craype-haswell craype-mic-knl
  CC -O3 -c myfile.c++

• If the same executable must run on KNL and Haswell nodes, use craype-haswell but add KNL-optimized execution path
  
  CC -axMIC-AVX512,CORE-AVX2 -O3 -c myfile.c++
Utility libraries

- **Not performance-critical (by definition)**
  - KNL can run Xeon binaries .. can use Haswell-targeted versions

- **I/O libraries (HDF5, NetCDF, etc) should fit in this category too**
  - (for Cray-provided libraries, compiler wrapper will use craype-* to select best build anyway)
Performance-critical libraries

• MKL: has KNL-targeted optimizations
• PETsc, SLEPc, Caffe, Metis, etc:
  – (soon) has KNL-targeted builds
• Modulefiles will use craype-{haswell,mic-knl} to find appropriate library
  – Currently FFTW, LibSci, Petsc, TPSL have separate builds for KNL

• Key points:
  – Someone else has already prepared libraries for KNL
  – No need to do-it-yourself
  – Load the right craype-module
Summary

• Use compiler wrappers where possible, which allows to include architecture specific optimization flag and link to the libraries
  – Check for the –xHost option
• Explore available compilers, Intel, Cray, GNU
  – Validity check after builds is important
• Use provided libraries where available
  – Use module show <librarymodule> to see the paths if needed
For Cori KNL, do
  - module swap craype-haswell craype-mic-knl

Build on login nodes whenever possible
Learn from the compiler verbose output
Read compiler man pages
Recommended readings

• NERSC website, especially,
  – http://www.nersc.gov/users/computational-systems/cori/programming/compiling-codes-on-cori/
  – http://www.nersc.gov/users/computational-systems/edispon/programming/

• man pages:
  – ifort, icc, icpc, crayftn, etc
  – man ld (-Wl,-zmuldefs, -Wl,-y<symbol>
Building your application separately for each platform could be important to get optimal performance.
Acknowledgement

• Steve Leak for providing some of the slides used in this talk.

Thank you!