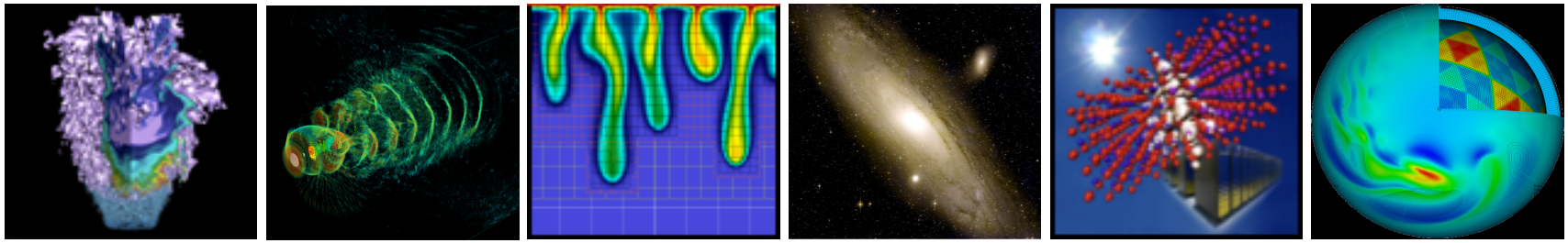

Tips to Compile Materials Science and Chemistry Codes at NERSC

Zhengji Zhao
NERSC user Services Group
April 5, 2011

U.S. DEPARTMENT OF ENERGY | Office of Science
NERSC National Energy Research Scientific Computing Center
BERKELEY LAB Lawrence Berkeley National Laboratory

Outline

- **Available compilers at NERSC**
- **Compiler flags**
- **Libraries**
  - Where to find them
- **A couple of common compilation errors**
- **A loading error and LD_LIBRARY_PATH**
- **Summary**

Available compilers at NERSC

- **Where we start**
  - We will address codes that work at least with one compiler and on one archietecture (third party software application packages)
  - The authors have provided with makefiles or configure scripts, we just need to work out the system dependent part of the makefiles

Available compilers

Codes	Hopper	Franklin	Carver	default
PGI	✔	✔	✔	yes
GNU	✔	✔	✔	
INTEL			✔	
Pathscale	✔	✔		
Cray	✔	✔		

- **Default on all major computing platforms at NERSC are pgi compilers**
- **Access through modules**
- **Programming environments**
  - **Module PrgEnv-pgi**
  - **Module pgi openmpi**

Comments from Cray

Compiler Choices – Relative Strengths

…from Cray's Perspective

- **PGI – Very good Fortran, okay C and C++**
  - Good vectorization
  - Good functional correctness with optimization enabled
  - Good manual and automatic prefetch capabilities
  - Very interested in the Linux HPC market, although that is not their only focus
  - Excellent working relationship with Cray, good bug responsiveness
- **Pathscale – Good Fortran, C, probably good C++**
  - Outstanding scalar optimization for loops that do not vectorize
  - Fortran front end uses an older version of the CCE Fortran front end
  - OpenMP uses a non-pthreads approach
  - Scalar benefits will not get as much mileage with longer vectors
- **(Not NERSC supported) Intel – Good Fortran, excellent C and C++ (if you ignore vectorization)**
  - Automatic vectorization capabilities are modest, compared to PGI and CCE
  - Use of inline assembly is encouraged
  - Focus is more on best speed for scalar, non-scaling apps
  - Tuned for Intel architectures, but actually works well for some applications on AMD

Comments from Cray

Compiler Choices – Relative Strengths

…from Cray's Perspective

- **GNU so-so Fortran, outstanding C and C++ (if you ignore vectorization)**
  - Obviously, the best for gcc compatability
  - Scalar optimizer was recently rewritten and is very good
  - Vectorization capabilities focus mostly on inline assembly
  - Note the last three releases have been incompatible with each other (4.3, 4.4, and 4.5) and required recompilation of Fortran modules
- **CCE – Outstanding Fortran, very good C, and okay C++**
  - Very good vectorization
  - Very good Fortran language support; only real choice for Coarrays
  - C support is quite good, with UPC support
  - Very good scalar optimization and automatic parallelization
  - Clean implementation of OpenMP 3.0, with tasks
  - Sole delivery focus is on Linux-based Cray hardware systems
  - Best bug turnaround time (if it isn't, let us know!)
  - Cleanest integration with other Cray tools (performance tools, debuggers, upcoming productivity tools)
  - No inline assembly support

Available compilers

- **From user perspective, compilation is no more than finding the paths to the needed header files and libraries, and provide them to the compile line and/or link line.**
- **Native compiler and compiler wrappers**
  - Use compiler wrappers to compile
  - Ftn,cc,CC on Hopper
  - Mpif90,mpicc, mpiCC on Carver
- **Dynamic and static linking**
  - Carver dynamic
  - Hopper static, Hopper support dynamic linking too

Compiler flags

PGI	Pathscale	Cray	GNU	Description
-fast	-Ofast	-O3	-O3	Produce high level of optimization
-mp=nonuma	-mp	-Oomp	-fopenmp	Activate OpenMP directives and pragmas in the code
-Mfixed	-fixedform	-f fixed	-ffixed-form	Process Fortran source using fixed form specifications.
-Mfree	-freeform	-f free	-ffree-form	Process Fortran source using free form specifications.
-V	-dumpversion	-V	--version	Show version number of the compiler.
-v				

Libraries

- **Modules**
  - module avail
- **How to find the paths to the header files and library files?**
  - Use Module show command
  - Compiler wrapper verbose outputs
    - mpif90 –v hello.f
    - ftn –v hello.f

Libraries

- **On hopper:**
  - Different builds for different compilers
  - Cray supports many software packages
  - Programming environment can selectively pick the matching libraries to load
- **On Carver**
  - You are on your own
  - It is your job to find the matching libraries among many available software and different builds

Libraries

- **LAPACK/ScaLAPCK libraries**
  - Libsci, acml, mkl
- **FFT libraries**
  - FFTW 2,3, acml, mkl
- **Quantum Espresso makefile**
  - Make.sys
- **VASP makefile**
- **Where do libraries and other software reside? (MODULEPATH)**
  - /opt –Cray directories
  - /usr/common/usg

A couple common errors

- **Syntax errors**
  - due different compiler behaviors
- **Library linking order**
  - Missing standard libraries, mixed fortran/C/C++ compilation
  - undefined symbols, try -Wl, --start-group, …, -Wl, --end-group
  - -Wl,-z muldefs –allow multiple defined symbols, use the first one.
- **Loading error (Carver)**
  - Provide the LD_LIBRARY_PATH
  - Set env OMP_NUM_THREADS to the number of threads for hybrid execution

Good practice

- **Use compiler wrappers**
- **Use the system provided libraries whenever applicable for a better performance**
- **Start with the compilers that vendor/ authors used, to minimize the chance to hit the compiler and code bugs, then try different compilers if you care the performance.**
- **Validity check after compilation**
  - Run tests and check with the references if provided
  - Debug version to check the validity

- **Recommended readings:**
  - NERSC website, especially
    - http://www.nersc.gov/nusers/systems/carver/programming/index.php
    - http://newweb.nersc.gov/users/computational-systems/hopper/programming/
    - man pages:
    - Pgf90,pgcc,pgCC
    - Other compilers

Dynamic Shared Objects and Libraries (DSL) on Hopper

- **Using system provided dynamic shared libraries**
  1. Link codes with -dynamic
  2. Set runtime env, CRAY_ROOTFS=DSL

```
hopper01> ftn -dynamic mpi_test.f90
hopper01> qsub -I -V -l mppwidth=2 -q debug
qsub: waiting for job 141142.sdb to start
qsub: job 141142.sdb ready

nid05430> cd $PBS_O_WORKDIR
nid05430> export CRAY_ROOTFS=DSL
nid05430> aprun -n 2 a.out
  Hello World, I am process              0
  Hello World, I am process              1
Application 536003 resources: utime ~0s,
stime ~0s
```

Dynamic Shared Objects and Libraries (DSL) on Hopper

- **Using user defined dynamic shared libraries**
  1. Build shared libraries:
     a) **Compile with –shared –fPIC**
     b) **Create dynamic shared libraries with cc –shared**
  2. Set runtime env, CRAY_ROOTFS=DSL , LD_LIBRARY_PATH

Continued…

```
nid05430> ftn -shared -fPIC -c callC.f
nid05430> cc -shared -o libflib.so callC.o
nid05430> cc -dynamic callF.c -L./ -lflib
nid05430> export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:./
nid05430> aprun -n 2 a.out
 reached Fortran
         …
the Long int is 12345678901
Application 536015 exit codes: 28
Application 536015 resources: utime ~0s, stime ~0s
```