Compiling and running code on CPU nodes

Xin Li

Cray programming environment (CPE)

Reference page: Compilers and libraries

The Cray Programming Environment (CPE) provides consistent interface to multiple compilers and libraries.

- In practice, we recommend
  - `ml cpeCray/23.03`
  - `ml cpeGNU/23.03`
  - `ml cpeAOCC/23.03`
- The `cpeCray`, `cpeGNU` and `cpeAOCC` modules are available after `ml PDC/23.03`
- No need to `module swap` or `module unload`

Compiler wrappers

- Compiler wrappers for different programming languages
  - `cc` : C compiler wrapper
  - `CC` : C++ compiler wrapper
  - `ftn` : Fortran compiler wrapper
- The wrappers choose the required compiler version and target architecture optinons.
- Automatically link to MPI library and math libraries
  - MPI library: `cray-mpich`
  - Math libraries: `cray-libsci` and `cray-fftw`

Compile a simple MPI code

- `hello_world_mpi.f90`

```
program hello_world_mpi
include "mpif.h"
integer myrank,mysize,ierr
call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD,myrank,ierr)
call MPI_Comm_size(MPI_COMM_WORLD,mysize,ierr)
write(*,*) "Processor ",myrank," of ",mysize,": Hello World!"
call MPI_Finalize(ierr)
end program
```

```
ftn hello_world_mpi.f90 -o hello_world_mpi.x
```

What flags does the `ftn` wrapper activate?

- Use the flag `-craype-verbose`

```
ftn -craype-verbose hello_world_mpi.f90 -o hello_world_mpi.x
```

Compile a simple MPI code

```
user@uan01:> srun -n 8 ./hello_world_mpi.x
 Processor             4  of             8 : Hello World!
 Processor             6  of             8 : Hello World!
 Processor             7  of             8 : Hello World!
 Processor             0  of             8 : Hello World!
 Processor             1  of             8 : Hello World!
 Processor             2  of             8 : Hello World!
 Processor             3  of             8 : Hello World!
 Processor             5  of             8 : Hello World!
```

Compile a simple linear algebra code

Link to the code

Use cray-libsci

```
ml PDC/23.03 cpeGNU/23.03
```

```
cc dgemm_test.c -o dgemm_test_craylibsci.x
```

Compile a simple linear algebra code

Use openblas

```
ml openblas/0.3.24-gcc-s34
```

```
export ROOTOPENBLAS=/pdc/software/23.03/spack/openblas-0.3.24-s34z4q2
cc dgemm_test.c -o dgemm_test_openblas.x -I$ROOTOPENBLAS/include -L$ROOTOPENBLAS/lib -lopenblas
```

Check the linked libraries

```
ldd dgemm_test_craylibsci.x
```

```
ldd dgemm_test_openblas.x
```

Check the linked libraries

```
ldd dgemm_test_craylibsci.x

...
libsci_gnu_82.so.5 => /opt/cray/pe/lib64/libsci_gnu_82.so.5
...
```

```
ldd dgemm_test_openblas.x

...
libopenblas.so.0 => /.../openblas-0.3.24-s34z4q2/lib/libopenblas.so.0
...
```

Exercise: Compile and run the dgemm_test code

- Run on a single core in the `shared` partition

```
salloc -n 1 -t 10 -p shared -A <name-of-allocation>
srun -n 1 ./dgemm_test_craylibsci.x
srun -n 1 ./dgemm_test_openblas.x
exit
```

- Expected output:

```
    2.700     4.500     6.300     8.100     9.900    11.700    13.500
    4.500     8.100    11.700    15.300    18.900    22.500    26.100
    6.300    11.700    17.100    22.500    27.900    33.300    38.700
```

Exercise: Compile and run `fftw_test` code

```
ml cray-fftw/3.3.10.3

wget https://people.math.sc.edu/Burkardt/c_src/fftw/fftw_test.c

cc --version
cc fftw_test.c -o fftw_test.x

ldd fftw_test.x

salloc -n 1 -t 10 -p shared -A <name-of-allocation>
srun -n 1 ./fftw_test.x
```

Compilation of large program

- Examples at https://www.pdc.kth.se/software

Environment variables for manual installation of software

- Environment variables for compilers

```
export CC=cc
export CXX=CC
export FC=ftn
export F77=ftn
```

- Environment variables for compiler flags
  - add `-I`, `-L`, `-l`, etc. to Makefile
- Environment variables at runtime
  - prepend to `PATH`, `LD_LIBRARY_PATH`, etc.

What happens when loading a module

```
ml show openblas/0.3.24-gcc-s34
```

```
whatis("OpenBLAS: An optimized BLAS library")
prepend_path("PATH","/pdc/software/23.03/spack/openblas-0.3.24-s34z4q2/bin")
prepend_path("LD_LIBRARY_PATH","/pdc/software/23.03/spack/openblas-0.3.24-s34z4q2/lib")
prepend_path("PKG_CONFIG_PATH","/pdc/software/23.03/spack/openblas-0.3.24-s34z4q2/lib/pkgconfig")
prepend_path("CMAKE_PREFIX_PATH","/pdc/software/23.03/spack/openblas-0.3.24-s34z4q2/.")
...
```

When running your own code

- Load correct programming environment (e.g. `cpeGNU`)
- Load correct dependencies (e.g. `openblas` if your code depends on it)
- Properly prepend to environment variables (e.g. `PATH`, `LD_LIBRARY_PATH`)
- Choose correct SLURM settings

SLURM settings for hybrid MPI/OpenMP code

- `--nodes` number of nodes
- `--ntasks-per-node` number of MPI processes
- `--cpus-per-task` 2 x number of OpenMP threads (because of SMT)
- `OMP_NUM_THREADS` number of OpenMP threads
- `OMP_PLACES` cores

Example job script

- 64 MPI x 2 OMP per node (main parition)

```
#!/bin/bash

#SBATCH -A ...
#SBATCH -J my_job
#SBATCH -t 01:00:00
#SBATCH -p main

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=64
#SBATCH --cpus-per-task=4

module load ...

export OMP_NUM_THREADS=2
export OMP_PLACES=cores

srun ...
```

Example job script

- 2 MPI x 2 OMP per node (shared partition)

```
#!/bin/bash

#SBATCH -A ...
#SBATCH -J my_job
#SBATCH -t 01:00:00
#SBATCH -p shared

#SBATCH --ntasks=2
#SBATCH --cpus-per-task=4

module load ...

export OMP_NUM_THREADS=2
export OMP_PLACES=cores

srun ...
```

Exercise: Hybrid MPI/OpenMP code for matrix-matrix multiplication

- Preparation

```
mkdir -p matmul_test && cd matmul_test
```

- Copy python code matmul_mpi_omp_test.py to the same folder

Exercise: Hybrid MPI/OpenMP code for matrix-matrix multiplication

- Copy job script job-n1.sh
  - for running on 1 MPI processes with different number of OpenMP threads
- Copy job script job-n2.sh
  - for running on 2 MPI processes with different number of OpenMP threads
- Submit two jobs

Exercise: Hybrid MPI/OpenMP code for matrix-matrix multiplication

- Result

Setting	Timing
1 MPI x 16 OMP	Time spent in matmul: 2.307 sec
1 MPI x 8 OMP	Time spent in matmul: 3.924 sec
1 MPI x 4 OMP	Time spent in matmul: 6.626 sec
2 MPI x 8 OMP	Time spent in matmul: 2.034 sec
2 MPI x 4 OMP	Time spent in matmul: 3.287 sec
2 MPI x 2 OMP	Time spent in matmul: 6.188 sec