Compiling code

Cheyenne and Casper users have access to Intel, NVIDIA/PGI, and GNU compilers. The **Intel compiler module** is loaded by default, which makes some Intel tools available by default as well.

2/2/2021: This documentation has been updated because of changes to the PGI compiler. It has become the NVIDIA HPC (nvhpc) compiler and all future versions will be released as such. PGI users should migrate to NVIDIA when possible. The nvhpc compiler has no license limitations.

Page contents

Compiler commands

After loading the compiler module that you want to use, identify and run the appropriate compilation wrapper command from the table below. (If your script already includes one of the following generic MPI commands, there is no need to change it: mpif90, mpif77, ftn; mpicc, cc; mpiCC and CC.)

Also consider using the compiler's diagnostic flags to identify potential problems.

Any libraries you build to support an application should be built with the same compiler, compiler version, and compatible flags that were used to compile the other parts of the application, including the main executable(s). Also, when you run the applications, be sure you have loaded the same module/version environment in which you created the applications. This will avoid job failures that can result from missing mpi launchers and library routines.

Compiler man pages

To get the **man** page for a compiler, log in to the system where you intend to use it, load the module, then execute **man** for the compiler.

```
module load nvhpc
man nvfortran
```

Compiler	Language	Commands for serial programs	Commands for programs using MPI	Flags to enable OpenMP (for serial and MPI)
Intel (default)	Fortran	ifort foo.f90	mpif90 foo.f90	-qopenmp
	C	icc foo.c	mpicc foo.c	-qopenmp
	C++	icpc foo.C	mpicxx foo.C	-qopenmp
Include these flags for best performance when you use the Intel compiler: -march=corei7 -axCORE-AVX2				
NVIDIA HPC	Fortran	nvfortran foo.f90	mpif90 foo.f90	-mp
	C	nvc foo.c	mpicc foo.c	-mp
	C++	nvc++ foo.C	mpicxx foo.C	-mp
GNU (GCC)	Fortran	gfortran foo.f90	mpif90 foo.f90	-fopenmp
	C	gcc foo.c	mpicc foo.c	-fopenmp
	C++	g++ foo.C	mpicxx foo.C	-fopenmp
PGI	Fortran	pgfortran (or pgf90 or pgf95) foo.f90	mpif90 foo.f90	-mp
	C	pgcc foo.c	mpicc foo.c	-mp
	C++	pgcpp (or pgCC) foo.C	mpicxx foo.C	-mp
The PGI compiler has become the NVIDIA HPC (nvhpc) compiler and all future versions will be released as such. PGI users should migrate to NVIDIA when possible.				

GPU compilers

To compile CUDA code to run on the Casper data analysis and visualization nodes, use the appropriate NVIDIA compiler command:

- **nvc** – NVIDIA C compiler
- **nvcc** – NVIDIA CUDA compiler (Using nvcc requires a C compiler to be present in the background; nvc, icc, or gcc, for example.)
- **nvfortran** – CUDA Fortran

For examples, see:

- Compiling GPU code on Casper nodes
- Compiling multi-GPU MPI-CUDA code on Casper

Changing compilers

Should you prefer to use a compiler other than the Intel default compiler, use **module swap** to make the change.

In this example, you are switching from Intel to NVIDIA:

```
module swap intel nvhpc
```

When you load a compiler module, the system makes other compatible modules available. This helps you establish a working environment and avoid conflicts. If you need to link your program with a library*, use **module load** to load the library as in this example:

```
module load netcdf
```

Then, you can just invoke the desired compilation command without adding link options such as **-l netcdf**. Here's an example:

```
mpif90 foo.f90
```

You can learn more about how using modules helps you manage your environment on our Environment modules page.

Where to compile

The use of different types of processors and operating systems in the Cheyenne environment makes your choice of where to compile your code especially important.

System	Processor
Cheyenne	Intel Broadwell
Casper DAV nodes	Intel Skylake (default) Intel Cascade Lake

Cheyenne uses SUSE Enterprise Linux, while the Casper data analysis and visualization (DAV) nodes run CentOS. The different operating systems provide different versions of some standard libraries, which may be incompatible with each other. Specify Skylake nodes to ensure your code will run on both Skylake and Cascade Lake nodes.

Even if you take care to build your programs for portability, you will achieve superior performance if you compile your code on the cluster where you intend for it to run.

Compile on Cheyenne if…

- You want to aggressively optimize CPU performance.
- You will run your code only on Cheyenne.

You can compile your code in a batch job in the "economy" or "regular" queue using your login environment (not a login node) if needed.

Compile on Casper if...

- You want to ensure that your code will run on Casper nodes.
- You want to use the latest CPU optimizations in Intel's Skylake or Cascade Lake architecture.
- Your programs use GPU tools like OpenGL, CUDA, and OpenACC.

Native commands

We recommend using the module wrapper commands described above. However, if you prefer to invoke the compilers directly, unload the NCAR default compiler wrapper environment by entering this on your command line:

```
module unload ncarcompilers
```

You can still use the environment variables that are set by the modules that remain loaded, as shown in the following examples of invoking compilers directly to compile a Fortran program.

Intel compiler

```
ifort -o a.out $NCAR_INC_<PROGRAM> program_name.f $NCAR_LDFLAGS_<PROGRAM> $NCAR_LIBS_<PROGRAM>
```

NVIDIA HPC compiler

```
nvfortran -o a.out $NCAR_INC_<PROGRAM> program_name.f $NCAR_LDFLAGS_<PROGRAM> $NCAR_LIBS_<PROGRAM>
```

GNU compiler collection (GCC)

```
gfortran -o a.out $NCAR_INC_<PROGRAM> program_name.f $NCAR_LDFLAGS_<PROGRAM> $NCAR_LIBS_<PROGRAM>
```

PGI compiler

```
pgfortran -o a.out $NCAR_INC_<PROGRAM> program_name.f $NCAR_LDFLAGS_<PROGRAM> $NCAR_LIBS_<PROGRAM>
```

To determine the correct name to substitute for <PROGRAM> in those commands, you can grep each of them as shown here.

```
env | grep NCAR_INC
env | grep NCAR_LDFLAGS
env | grep NCAR_LIBS
```

***** In addition to multiple compilers, CISL keeps available multiple versions of libraries to accommodate a wide range of users' needs. Rather than rely on the environment variable LD_LIBRARY_PATH to find the correct libraries dynamically, we encode library paths within the binaries when you build Executable and Linkable Format (ELF) executables. To do this, we use RPATH rather than LD_LIBRARY_PATH to set the necessary paths to shared libraries.

This enables your executable to work regardless of updates to new default versions of the various libraries; it doesn't have to search dynamically at run time to load them. It also means you don't need to worry about setting the variable or loading another module, greatly reducing the likelihood of runtime errors.