Hewlett Packard Enterprise

Programming Environment and Modules

Comprehensive General LUMI Course

April 23–66, 2024

Agenda

- The Cray Programming Environment
- Controlling the Environment with modules

Vision

- HPE Cray EX systems are designed to be High Productivity as well as High Performance Computers

- The Cray Programming Environment (CPE) provides a simple consistent interface to users and developers.
  - Focus on improving scalability and reducing complexity

- The default Programming Environment provides:
  - the highest levels of application performance
  - a rich variety of commonly used tools and libraries
  - a consistent interface to multiple compilers and libraries
  - an increased automation of routine tasks

- We continue to develop and refine the PE
  - Frequent communication and feedback to/from users
  - Strong collaborations with third-party developers

Cray Developer Environment for EX

Development

Programming Models

- HPE Cray MPI
- SHMEM
- OpenMP | OpenACC
- AMD ROCm HIP | NVIDIA CUDA
- UPC | Fortran co-arrays
- Chapel

Programming Environments

- Compiling Environment
  - GNU
  - Intel Programming Environment
  - AMD Programming Environment
  - NVIDIA HPC SDK

Programming Languages

- C
- C++
- Fortran
- Python
- Chapel
- R

Optimized Libraries

- LibSci (BLAS)
- LAPACK & ScaLAPACK
- LibSci_ACC
- IRT
- FFTW

I/O Libraries

- NetCDF
- HDF5

DL / AI Tools

- Deep Learning Plug-in

Debugging

- Comparative Debugger
  Compare two versions of an application
- GDB for HPC
  Parallelized gdb for HPC
- Valgrind for HPC
  Memory debugging at scale
- Stack Trace Analysis Tool
  Stack tracing at scale
- Tool for Abnormal Termination Processing
  Manage core files at scale
- TotalView
- DDT

Performance Analysis & Optimization

- Performance Analysis Tool (PAT)
  Whole program performance analysis, exposing wide set of indicators, identifying bottlenecks and automatically generating suggestions to improve performance.
- Visualization Tool
  Complements text reports with summary of performance data in graphs and charts, allowing users to drill down and resolve issues
- Code Parallelization Assistant
  Reveal hidden potential of an application via code restructuring

Setup & Runtime

Environment Setup

- **Tool Enablement (for Spack, CMake, EasyBuild, etc)**
- **Modules / Lmod**

Supported systems:

- HPE Cray EX supercomputers
- HPE Apollo 2000
- HPE Apollo 6500
- HPE ProLiant DL systems
- Previous Cray XC systems

HPE –authored | HPE Added-value to 3rd party | 3rd party

Components of the Programming Environment

- Cray Compiling Environment (CCE)
  Optimizing compilers for Fortran, C, C++ and UPC
- Cray Scientific and Math Libraries (CSML)
  High Performance libraries for scientific applications (BLAS, LAPACK, ScaLAPCK, FFTW, NetCDF)
- Cray Message Passing Toolkit (CMPT)
  Provides the Message Passing Interface (MPI) and OpenSHMEM
- Cray Environment Setup and Compiling Support (CENV)
  Infrastructure to support the development environment.
  Includes compiler drivers, hugepages support and the PE packaging API
- Cray Performance Measurement and Analysis Tools (CPMAT)
  Provides tools to analyse performance and behaviour of programs and the PAPI performance API
- Cray Debugging Support Tools (CDST)
  Provides debugging tools including gdb4hpc and valgrind4hpc

Cray Compiling Environment (CCE)

- The default compiler on EX systems
  - Specifically designed for HPC applications
  - Takes advantage of Cray's experience with automatic vectorization and shared memory parallelization
- Excellent standards support for multiple languages and programming models
  - Fortran, C, C++
    – Supports most of Fortran 2018 (ISO/IEC 1539:2018) with some exceptions
    – C/C++ compiler is based on Clang/LLVM
  - OpenMP
    – Full OpenMP 4.5 and partial OpenMP 5.0 and 5.1, see `man intro_openmp`
  - OpenACC
    – Full OpenACC 2.0 and partial OpenACC 2.x/3.x, see `man intro_openacc`
  - HIP compilation
  - MPI Interfaces
- Full integrated and optimised support for PGAS languages
  - UPC 1.2 and Fortran 2008 coarray support
  - No preprocessor involved
  - Full debugger support (With ARM Forge)
- OpenMP and automatic multithreading fully integrated
  - Share the same runtime and resource pool
  - Aggressive loop restructuring and scalar optimization done in the presence of OpenMP
  - Consistent interface for managing OpenMP and automatic multithreading

Other Compilers and Programming Language Support

- Compilers
  - GNU
  - AOCC (AMD CPU Compiler)
  - AMD LLVM (AMD GPU support)

- Python: 3.10.10
- R: Includes R 4.2.1

Status of Clang into CCE (introduced in CCE9)

Cray Enhancements to Clang:

- Loopmark (-fsave-loopmark)
- Decompile (-fsave-decompile)
- UPC support (-hupc or automatic with the .hupc file extension)
- Enhanced assembly listings (line numbers, comments)
- Support for Cray Performance Measurement and Analysis Tools
- Cray-specific performance improvements
- Cray OpenMP runtime supports CCE Classic/Clang interoperability for codes that mix Fortran and C/C++
- Support for OpenMP 4.5 offload to NVIDIA and AMD GPUs

AMD Optimizing C/C++ Compiler (AOCC)

- The CPE enables this but does not bundle it
- Supported as a programming environment via PrgEnv-aocc
  - Based on LLVM 13
  - Provides C/C++ and Fortran (based “classical” FLANG)
- Loading the module will provide the compiler via the standard (ftn, cc , CC) wrappers and will link appropriate libraries

- More details from:
  https://developer.amd.com/wp-content/resources/AOCC-3.0-Install-Guide.pdf
  https://developer.amd.com/wp-content/resources/AOCC-3.0-User-Guide.pdf

Cray MPI and SHMEM

Cray MPI

- Implementation based on MPICH3 source from ANL
- Includes many improved algorithms and tweaks for Cray hardware
  - Improved algorithms for many collectives
  - Asynchronous progress engine allows overlap of computation and comms
  - Customizable collective buffering when using MPI-IO
  - Optimized Remote Memory Access (one-sided) fully supported including passive RMA
- Full MPI-3.1 support with the exception of
  - MPI_LONG_DOUBLE and MPI_C_LONG_DOUBLE_COMPLEX for CCE
- Includes support for Fortran 2008 bindings (since CCE 8.3.3)

Cray SHMEM

- Fully optimized Cray SHMEM library supported
- Fully compliant with OpenSHMEM v1.5
- Cray XC implementation close to the T3E model

Cray Scientific and Maths Libraries (CSML)

The programming environment provides integrated tuned versions of popular scientific libraries

- LibSci
  - BLAS (Basic Linear Algebra Subroutines)
  - BLACS (Basic Linear Algebra Communication Subprograms)
  - CBLAS (wrappers providing a C interface to the FORTRAN BLAS library)
  - IRT (Iterative Refinement Toolkit)
  - LAPACK (Linear Algebra Routines)
  - LAPACKE (C interfaces to LAPACK Routines)
  - ScaLAPACK (Scalable LAPACK)
- LibSci_ACC
  - Subset of GPU-optimized GPU routines from LibSci
- FFTW3
  - Fastest Fourier Transforms in the West, release 3
- Data libraries
  - NetCDF and HDF5

Packages where only Build Instructions Provided

- TPSL - Cray's collection of third-party scientific libraries
  - HYPRE, METIS, ParMETIS, MUMPS, SUNDIALS, SuperLU, SuperLU_DIST, Scotch, GLM, Matio
- Boost
- PETSc
- SLEPc
- Trilinos
- ADIOS
- CMake

- Build instructions can be found at https://github.com/Cray/pe-scripts
- Customer sites often have good documentation on building applications/packages

Cray DSMML

- Distributed Symmetric Memory Management Library (DSMML)
- Standalone proprietary library for managing distributed shared symmetric memory heaps
- Useful for implementing PGAS languages
- See
  intro_dsmml(3)
  Content will appear here in due course https://pe-cray.github.io/cray-dsmml/

Python and R Modules

- R, cray-R (version 4.2.1)
- Python 3
- cray-python/3.10.10 (from CPE 23.09) contains:

  python 3.10.10
  numpy 1.23.5
  pandas 1.4.2
  SciPy 1.10.0
  mpi4py 3.1.4
  dask   2022.7.0

Cray PE Deep Learning plugin

- Provides a highly-optimized communication layer for Deep Learning training
- This provides mechanisms for the following which can be added to a single-process application:
  - Process identification and job size
  - Weight and bias broadcasting
  - Gradient averaging
- Accessible via Python and C APIs
- Python API supports:
  TensorFlow, PyTorch, Keras and NumPy
- Modules
  `craype-dl-plugin-py3` (standard module)
  `craype-dl-plugin-ftr` (with fault tolerant support)
- See `man intro_dl_plugin`

- (note that you need to have gcc loaded to see above modules with module avail)

Debugging Support Tools (CDST)

- gdb4hpc
  Command-line parallel debugger
- valgrid4hpc
  Parallel-debugging tool for detection of memory leaks parallel application errors.
  (partially implemented on Shasta)
- Stack Trace Analysis Tool (STAT)
  Merged-stack backtrace analysis tool
- Abnormal Termination Processing (ATP)
  Scalable core file generator and analysis tool
- Cray Comparative Debugging (CCDB)
  Side by side debugging tool for two ‘versions’ of an application.

Cray Performance Measurement and Analysis Tools

- Perftools lite modules for one-shot build/run/analysis
  - perftools-lite, perftools-lite-events, perftools-lite-loops, perftools-lite-hbm
- Full featured perftools module for multi-step collection and analysis
  - pat_build (program instrumentation)
  - pat_report (report generation)
  - Craypat runtime library
- pat_run Launches a dynamically-linked program for analysis
- Also
  - PAPI, Apprentice2, stat_view and reveal

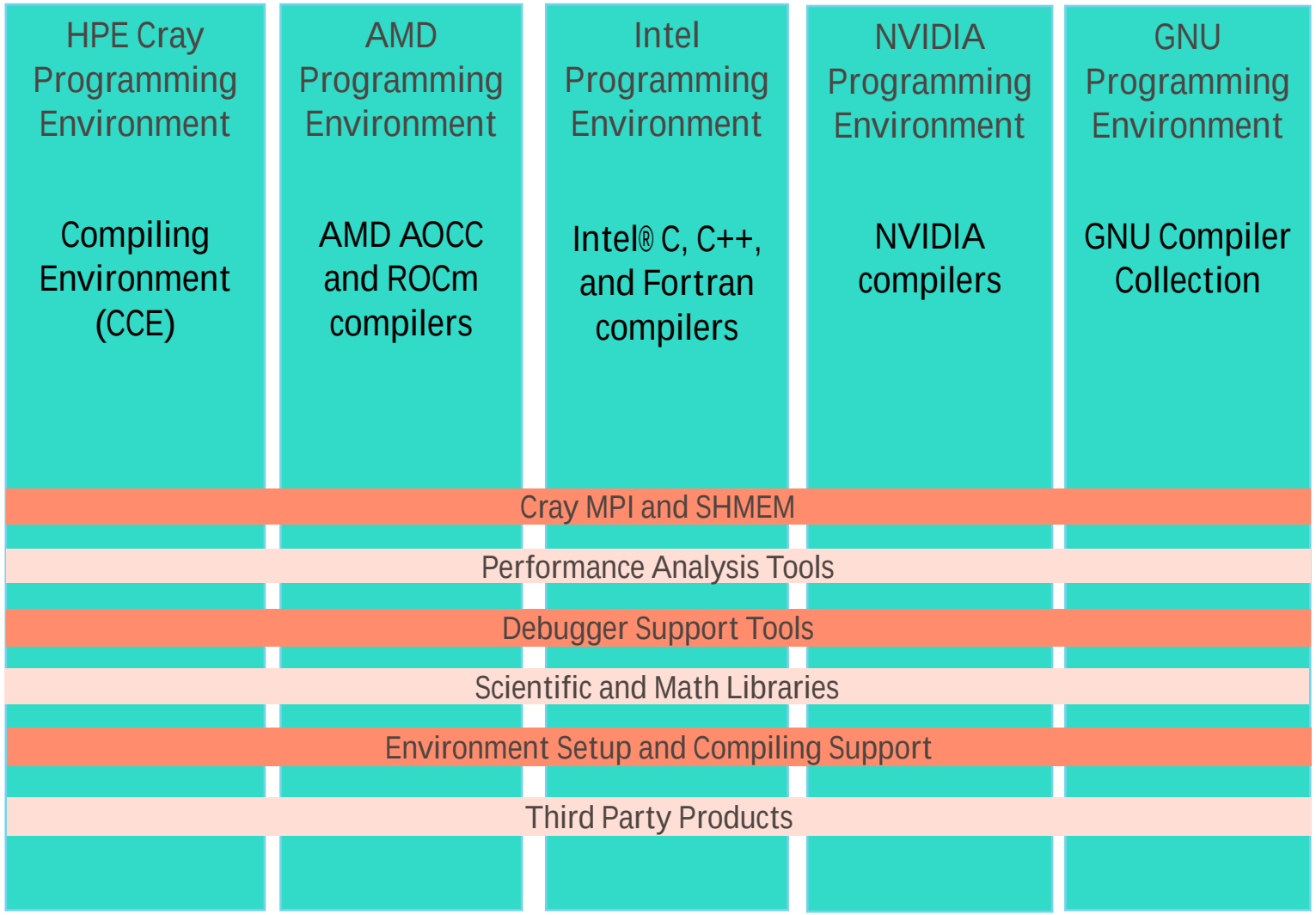
Controlling the Environment with Modules

Modules

- The Cray Programming Environment uses a “modules” framework to support multiple software versions and to create integrated software packages
- Either Environment Modules or Lmod will be set as the default system wide
- As new versions of the supported software and associated man pages become available, they are installed and added to the Programming Environment as a new module version, while earlier versions are retained to support legacy applications
- System administrators will set the default software versions, or you can choose another version by using modules system commands
- Users can create their own modules, or administrators can install site specific modules available to many users
- Modules are used both to set high-level context (for example choose a compiler toolchain) and to select individual tool and library components and versions

Compiler choice when using the PE

- Use modules to select compiling environment
  - Automatically uses our math, scientific, and communication libraries with chosen compiler
  - Can use debug and profiling tools with chosen compiler



Viewing the Current Module State

- Each login session has its own module state which can be modified by loading, swapping or unloading the available *modules*
- This state affects the functioning of the compiler wrappers and in some cases runtime of applications
- A standard, default set of modules is always loaded at login for all users
- Current state can be viewed by running:

```
$> module list
```

Default Modules Example: PE modules based on version 23.09 (*year.month*)

```
% module list
Currently Loaded Modules:
  1) craype-x86-rome                          8) cray-dsmml/0.2.2
  2) libfabric/1.15.2.0                       9) cray-mpich/8.1.27
  3) craype-network-ofi                      10) cray-libsci/23.09.1.1
  4) perftools-base/23.09.0                  11) PrgEnv-cray/8.4.0
  5) xpmem/2.5.2-2.4_3.20__gd0f7936.shasta   12) ModuleLabel/label     (S)
  6) cce/16.0.1                              13) lumi-tools/23.11      (S)
  7) craype/2.7.23                           14) init-lumi/0.2         (S)

  Where:
   S:  Module is Sticky, requires --force to unload or purge
```

Additional CPE versions

```
% module avail cpe

------------------------------ HPE-Cray PE modules -------------------------------
   cpe/22.08    cpe/22.12    cpe/23.03    cpe/23.09 (D)

  Where:
   D:  Default Module


% module load cpe/23.03

The following have been reloaded with a version change:
  1) PrgEnv-cray/8.4.0 => PrgEnv-cray/8.3.3
  2) cce/16.0.1 => cce/15.0.1
  3) cray-libsci/23.09.1.1 => cray-libsci/23.02.1.1
  4) craype/2.7.23 => craype/2.7.20
  5) perftools-base/23.09.0 => perftools-base/23.03.0
```

Viewing Available Modules

- There may be many hundreds of possible modules available to users
  - Beyond the pre-loaded defaults there are many additional packages provided by Cray
  - Sites may choose to install their own versions
- Users can see all the modules that can be loaded using the command:
  - **`module avail`**
- Cray PE modules will have a location in /opt, customer specific modules may also be installed.
- Searches can be narrowed by passing the first few characters of the desired module, e.g.

```
% module avail cce/

---------------------------- HPE-Cray PE modules -----------------------------
   cce/14.0.2    cce/15.0.0    cce/15.0.1    cce/16.0.1 (L,D)

  Where:
   L:  Module is loaded
   D:  Default Module
```

Further Refining Available Modules List

- `avail [switches] [sub-command [sub-command-args]]`
  - List all available modulefiles in the current `MODULEPATH`

- Useful options for filtering

  - `-d, --default`
    - List only default versions of modulefiles with multiple available versions

  - `--terse, t`
    - List in terse output format

  - `% module avail <string>`
    - List modules that include <string>

Module spider

- Lmod includes the *spider* modules command which is very useful in searching and providing information on how to use modules
- If you use it with a specific module name and version, it will show exactly which hierarchy of modules need to be loaded first

```
% module spider cray-netcdf/4.9.0.7

----------------------------------------------------------------------------
  cray-netcdf: cray-netcdf/4.9.0.7
----------------------------------------------------------------------------

    You will need to load all module(s) on any one of the lines below before the "cray-netcdf/4.9.0.7"
module is available to load.

      LUMI/22.08  partition/C  amd/5.2.3  cray-hdf5/1.12.2.7
      LUMI/22.08  partition/C  aocc/3.2.0  cray-hdf5/1.12.2.7
      LUMI/22.08  partition/C  gcc/12.2.0  cray-hdf5/1.12.2.7
      cray-hdf5/1.12.2.7
...
    Help:
      Release info:  /opt/cray/pe/netcdf/4.8.1.5/release_info
```

Module whatis info

- To find out what a module does (show whatis info)
  module whatis
- To search for a string in module name and the whatis info:
  module key

```
% module whatis cray-mpich
cray-mpich/8.1.27   : cray-mpich - Cray MPICH Message Passing Interface

% module key processor
-------------------------------------------------------------------

The following modules match your search criteria: "processor"
-------------------------------------------------------------------
...
  craype-x86-rome: craype-x86-rome
...
  craype-x86-trento: craype-x86-trento
...
-------------------------------------------------------------------
```

Modules Command Output

- Be aware that module commands output to standard error
- This makes it tricky to search the (voluminous) module avail output

- csh/tcsh

```
module avail >& mavail.txt ; grep netcdf mavail.txt
( module avail >/dev/null )  |& grep netcdf
module --redirect avail | grep netcdf
```

- bash/ksh

```
module avail 2> mavail.txt ; grep netcdf mavail.txt
module avail 2>&1 | grep netcdf
module --redirect avail | grep netcdf
```

- The modules commands paginate with less(1)

Modifying the Default Environment

Loading, swapping or unloading modules:

- The default version of any individual module can be loaded by name
  - e.g.: module load perftools-base
- A specific version can be specified after the forward slash
  - e.g.: module load perftools-base/23.09.0
- Modules can be swapped out in place
  - e.g.: module swap cce cce/15.0.1
- Or removed entirely
  - e.g.: module unload perftools

Modules will automatically change values of variables like PATH, MANPATH, LM_LICENSE_FILE… etc

- Modules also provide a simple mechanism for updating certain environment variables, such as PATH, MANPATH, and LD_LIBRARY_PATH
- In general, you should make use of the modules system rather than embedding specific directory paths into your startup files, makefiles, and scripts

Tips for Module Usage

- Put module list in job scripts
- If you want to test which programming environment is loaded (say in a Makefile)
  - Test the PE_ENV environment variable
    but be aware it is not supported
- If you really need to directly access something from the installed location of software (for example some build really wants to know where software is located)
  - Load (or show) the module
    module show cray-fftw
  - Look for envars that point to the location of interest

Summary of Useful Module Commands

Which modules are available?

- `module avail`, `module avail cce/`

Which modules are currently loaded?

- `module list`

Load software

- `module load perftools`

Change software version

- `module swap cce/16.0.1  cce/15.0.1`

Unload module

- `module unload cce`

Display module release notes

- `module help cce`

Show summary of module environment changes

- `module show cce` (or `module display cce`)

Compiler Driver Wrappers

- All applications *that will run in parallel* on the Cray EX should be compiled with the standard language wrappers
- The compiler drivers for each language are:
  - **`cc`** – wrapper for the C compiler
  - **`CC`** – wrapper for the C++ compiler
  - **`ftn`** – wrapper for the Fortran compiler
- These wrappers will choose the required compiler version, target architecture options, scientific libraries and required include files automatically from the module environment.
  - They enable MPI compilation by default
- Use them exactly like you would the original compiler, e.g. to compile `prog1.f90` run

```
ftn -c prog1.f90
```

- It can be necessary to set the env variables CC, CXX, FC for building tools, e.g.

```
CC=cc CXX=CC FC=ftn ./configure <flags>
```

About the –I, –L and –l Flags

- For libraries and include files covered by module files, you should not add anything to your Makefile
  - No additional MPI flags are needed (included by wrappers)
  - You do not need to add any `-I`, `-l` or `–L` flags for the Cray provided libraries
- If your Makefile needs an input for `–L` to work correctly, try using '.'
- If you really need a specific path, try checking 'module show X' for some environment variables
  - You will want to avoid a reference to a specific version as this fixes you build to that version or may create a conflict if the module environment changes

Choosing a Programming Environment

- The wrappers choose which compiler to use from the `PrgEnv` collection loaded

PrgEnv	Description	Real Compilers
`PrgEnv-cray`	Cray Compilation Environment	`crayftn, craycc, crayCC`
`PrgEnv-gnu`	GNU Compiler Collection	`gfortran, gcc, g++`
`PrgEnv-aocc`	AMD Optimizing Compilers (AOCC, CPU only support)	`flang, clang, clang++`
`PrgEnv-amd`	AMD LLVM Compilers (GPU support)	`amdflang, amdclang, amdclang++`
`PrgEnv-cray-amd`	AMD Clang C/C++ compiler and the Cray Compiling Environment (CCE) Fortran compiler	
`PrgEnv-gnu-amd`	AMD Clang C/C++ compiler and the GNU compiler suite Fortran compiler	

Choosing a Programming Environment

- `PrgEnv-cray` is loaded by default at login
  - use `module list` to check what is currently loaded
- List the PrgEnv- meta modules

```
> module avail PrgEnv
```

- Switch to a new programming environment

```
> module swap PrgEnv-cray PrgEnv-gnu
Lmod is automatically replacing "cce/16.0.1" with "gcc/12.2.0".


Due to MODULEPATH changes, the following have been reloaded:
  1) cray-mpich/8.1.27
```

  - The Cray MPI module is loaded by default (`cray-mpich`) and reloaded accordingly upon PrgEnv change
  - Compiler wrappers will link to the proper compiler version of the modules
  - Check the compiler version to know which backend you are using (e.g. `cc -v`)

Backend Compiler Version

- Check the compiler version to know which backend you are using in the compiler wrappers
  - `--version` flag
- E.g.
  - PrgEnv-cray

```
> cc --version
Cray clang version 16.0.1
> ftn --version
Cray Fortran : Version 16.0.1
```

  - PrgEnv-amd

```
> cc --version
AMD clang version 14.0.0 (https://github.com/RadeonOpenCompute/llvm-project roc-5.2.3
22324 d6c88e5a78066d5d7a1e8db6c5e3e9884c6ad10e)
> ftn --version
AMD flang-new version 14.0.0 (https://github.com/RadeonOpenCompute/llvm-project roc-
5.2.3 22324 d6c88e5a78066d5d7a1e8db6c5e3e9884c6ad10e)
```

PE modules: non-default modules

- The default GCC for PrgEnv-gnu is version 12.2.0
  - Wrong libs (GCC 11.2.0) are taken in `/opt/cray/pe/gcc-libs`, suggested to use GCC 11.2.0

```
module load gcc/11.2.0
```

- New PE rocm module based on version 5.2.3
  - Newer modules are available provided by LUST (check via `module spider rocm`)

- If you want to use non-default modules, set LD_LIBRARY_PATH before running the application

```
export LD_LIBRARY_PATH=${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}
```

  - `${CRAY_LD_LIBRARY_PATH}` is set by the PE modules
  - use the `ldd` command to check the linked libraries of you application executable

Compiler Versions

There are usually multiple versions of each compiler available to users.

- The most recent version is usually the default and will be loaded when swapping PrgEnvs.
- To change the version of the compiler in use, swap the Compiler Module.
  eg `module swap cce/16.0.1 cce/15.0.1`

PrgEnv	Compiler Module
PrgEnv-cray	cce
PrgEnv-gnu	gcc
PrgEnv-aocc	aocc
PrgEnv-amd	amd

- Note that you may not be able to swap vastly different versions without also swapping other modules
  - Example with PrgEnv-gnu loaded

```
> module swap gcc cce

Lmod is automatically replacing "PrgEnv-gnu/8.4.0" with "PrgEnv-cray/8.4.0".

Due to MODULEPATH changes, the following have been reloaded:
  1) cray-mpich/8.1.27
```

OpenMP

- OpenMP is supported by all of the `PrgEnvs`

PrgEnv	Enable OpenMP
PrgEnv-cray (Fortran)	-homp / -fopenmp
PrgEnv-cray (C/C++)	-fopenmp
PrgEnv-gnu	-fopenmp
PrgEnv-aocc	-fopenmp
PrgEnv-amd	-fopenmp

Compiler Man Pages and Information

- For more information on individual compilers

PrgEnv	C	C++	Fortran
PrgEnv-cray	man craycc	man crayCC	man crayftn
PrgEnv-gnu	man gcc	man g++	man gfortran
PrgEnv-aocc	clang --help	clang++ --help	flang --help
PrgEnv-amd	amdclang --help	amdclang++ --help	amdflang --help
Wrappers	man cc	man CC	man ftn

- All compilers provide --help output
  - E.g. for PrgEnv-cray: `crayftn --help`
- To verify that you are using the correct version of a compiler, use:
  - -V option on a cc, CC, or ftn for cray Cray
  - -dumpversion option on a cc, CC, or ftn command with GNU, AOCC, AMD

Recap

- The HPE Cray Programming Environment provides a consistent interface into a host of user and developer software
- Various toolchains are supported: Cray compilers, AMD, GNU, etc.
- Different architectures of CPU and GPU are supported

- The environment is used via a combination of
  - Modules
  - Compiler wrappers

Questions?