



**Hewlett Packard
Enterprise**

Programming Environment and Modules

Introduction to GPU - PDC

Oct 12-13, 2023

Agenda

- The Cray Programming Environment
- Controlling the Environment with modules
- Compilers
- Compilers for GPU



HPE CRAY PROGRAMMING ENVIRONMENT

Essential toolset for HPC organizations developing HPC code in-house

Fully integrated software suite with compilers, tools, and libraries designed to increase programmer productivity, application scalability, and performance.

Complete toolchain

For the whole application development process.

Holistic solution

Unlike processor-specific tools, the suite enables software development for the full system (including CPUs, GPUs and interconnect) for the best performance.

Programmability

Offering users intuitive behavior, automation of tasks and best performance for their applications with little effort.

Scalability

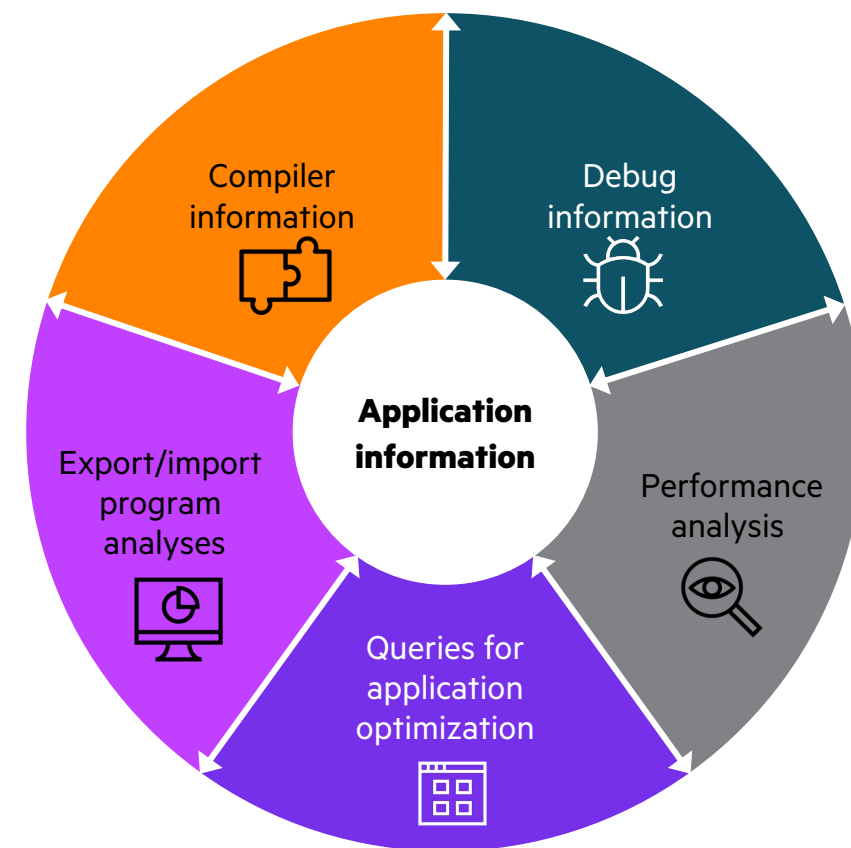
Improving performance of applications on systems of any size—up to Exascale deployments.

Complete Support

HPE Pointnext Services support the whole suite, not just the tools we developed.

From HPC experts for HPC experts

Developed for over 30 years in close interaction and contributions from our users.

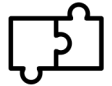


THE HPE CRAY PROGRAMMING ENVIRONMENT

Consists of:



Compilers



Scientific, Math and I/O Libraries



Debugging Tools



Performance Measurement, Analysis and Porting Tools



MPI



HPE CRAY PROGRAMMING ENVIRONMENT ADVANTAGES

Gold standard in HPC—Technology for real-life applications, not just benchmarks

Performance and programmability

- **Automatic optimizations** deliver performance for a **new target** through a simple recompile
- **Automatically exploits** the scalar, vector, and multithreading hardware capabilities of the systems
- Compiler optimization **feedback** for application tuning
- World-class MPI

Fully integrated heterogeneous optimization capability

- Providing **consistency** across all HPE Cray systems
- **Supporting x86-64** (both Intel and AMD) processors, Arm-based processors, and NVIDIA accelerators

Integration with other tools for productivity and performance

- Parallel **debuggers**: Rogue Wave TotalView and Arm DDT
- For advanced debugging, **performance analysis** and optimization tools additional insights into compiler needs

Focus on application portability and investment protection

Focus on compliance and language support:

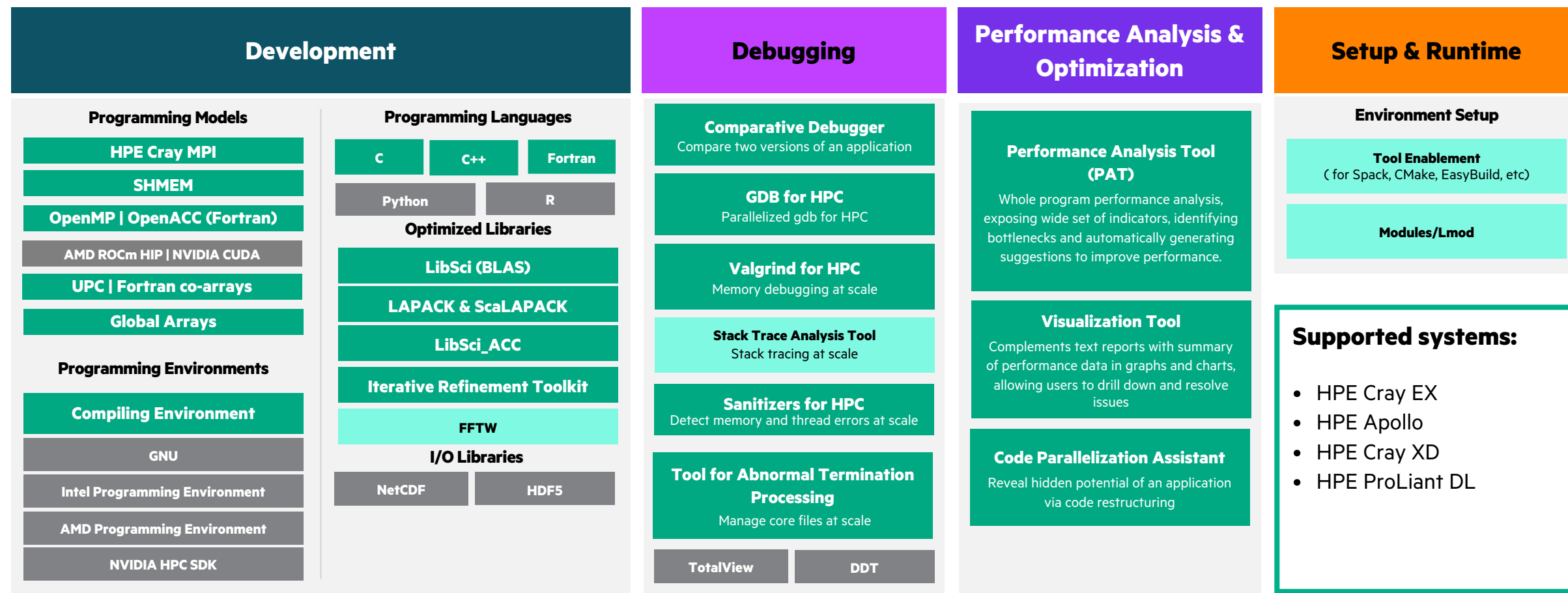
Accepts the **latest version** of:

- OpenMP for host and NVIDIA and AMD GPUs with Fortran, C and C++
- OpenACC for NVIDIA and AMD GPUs for Fortran
- Latest versions of C and C++ supported by Clang



HPE CRAY PROGRAMMING ENVIRONMENT

Comprehensive set of tools for developing, porting, debugging, and tuning of HPC applications on HPE HPC systems



HPE—authored

HPE Added-value to 3rd party

3rd party

Compiler Choices for Developers

Use modules to select compiling environment.

- Automatically uses HPE Cray Programming Environment’s math, scientific, and communication libraries with chosen compiler
- Can use debug and profiling tools with chosen compiler

HPE Cray Programming Environment	AMD Programming Environment	Intel Programming Environment	NVIDIA Programming Environment	GNU Programming Environment
Compiling Environment	AMD AOCC and ROCm compilers	Intel® C, C++, and Fortran compilers	NVIDIA compilers	GNU Compiler Collection
Cray MPI and SHMEM				
Performance Analysis Tools				
Debugger Support Tools				
Scientific and Math Libraries				
Environment Setup and Compiling Support				
Third Party Products				



CPU-GPU Workflow

Many HPC platforms today have one or more GPUs on the nodes

- GPUs provide significant computational power

Applications must be adapted to make use of GPUs

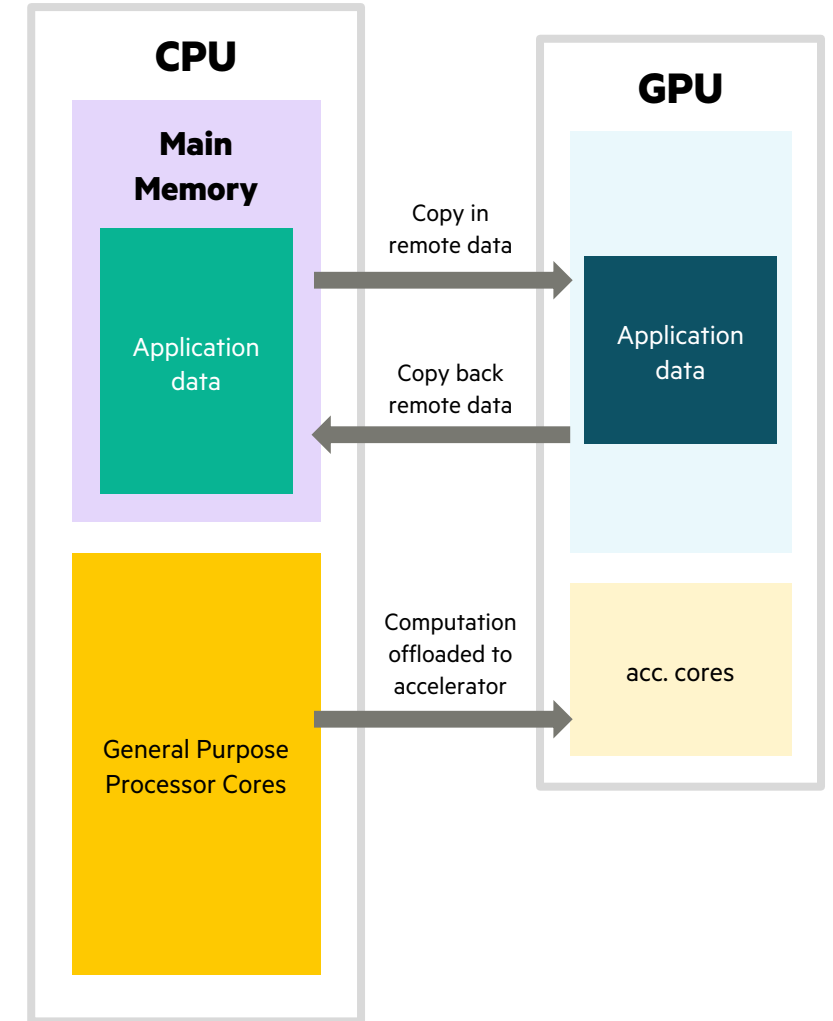
- Insertion of instructions to perform the GPU computation and transfer data to/from GPU
- Unified memory may automate data transfer
- Restructuring might be needed to ensure cost of data motion does not outweigh the benefits
 - Integrated performance tools can help

HPE Cray Programming Environment supports AMD and NVIDIA GPUs

- AMD HIP
- NVIDIA CUDA

HPE Cray Programming Environment's compilers support directives

- OpenMP
- OpenACC



LIBRARIES AND TOOLS

Set of high-performance libraries and tools customized for HPE/Cray HPC systems includes:

Customized libraries

Scientific libraries

LibSci library for HPE/Cray systems:

- Enhanced Basic Linear Algebra Subroutines (BLAS). Auto-tuned BLAS library (CrayBLAS) with custom optimizations for selected routines on HPE/Cray systems.
- Serial numerical linear algebra routines customized specifically for HPE/Cray hardware—LAPACK and scalable LAPACK (ScaLAPACK).

Iterative Refinement Toolkit

Tools that enable faster solutions to linear equations by working with scientific libraries and using mixed precision and progressively getting more accurate solutions as the solution converges. For well-conditioned problems, IRT can provide up to 70% performance improvement with little or no code changes.

Accelerated versions

LibSci_ACC library for HPE/Cray systems:

Accelerated versions of the scientific libraries are included for GPU accelerated systems to enhance user application performance by generating and executing auto-tuned kernels on GPUs. (Available from August 2020 for HPE Apollo systems with GPUs).

Bundled software

I/O libraries

NetCDF and HDF5 **I/O libraries** are built with the supported compiling environments and included in the suite for convenience.

Environment & tool enablement

Lmod, Lua and TCL are supported for use of environment setup. Additional support for Spack, Cmake, Easybuild, and autotools is being added for HPE and HPE Cray supercomputers.



DEBUGGING TOOLS

Traditional debuggers and new innovative techniques for 360° troubleshooting at scale for maximum productivity

The suite offers with comprehensive collection of debuggers addressing so users can spend less time debugging and more time creating, including:

Tools for deconstructing large-scale defects:

- **Stack Trace Analysis Tool (STAT)**
- **Tool for abnormal termination processing (ATP)**

Scalable debugging tools:

- **Comparative debugger** we developed featuring a GUI enabling users to easily compare data structures between two executing applications to detect issues
- **GDB for HPC**—based on popular GDB debugger with enhancements for scalability. Supports comparative debugger technology that enables programmers to compare data structures between two executing applications
- **Valgrind for HPC**—parallel memory analysis tool based on Valgrind debugger
- **Sanitizers for HPC** - help to detect and fix memory and thread errors

Support for traditional debuggers:

Integration with leading 3rd party tools: TotalView by Perforce and Arm DDT (Arm Forge Professional).

Controlling the Environment with Modules



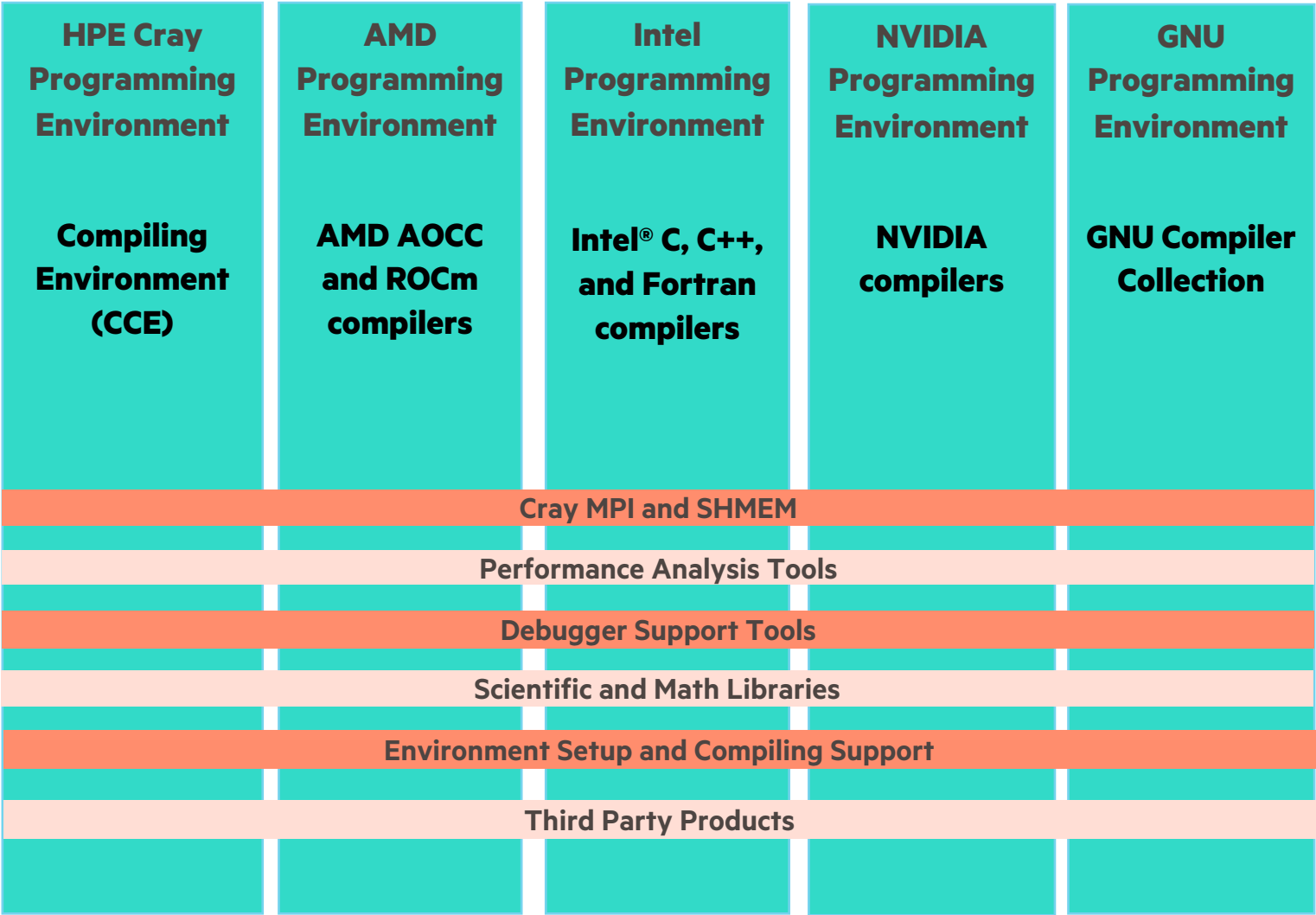
Modules

- 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 22.12 (year.month)

```
% $ module list
```

Currently Loaded Modules:

- | | | |
|---|---------------------------|------------------------------|
| 1) craype-x86-rome | 6) cce/14.0.1 | 11) <u>PrgEnv-cray/8.3.3</u> |
| 2) libfabric/1.15.0.0 | 7) craype/2.7.16 | 12) snic-env/1.0.0 |
| 3) craype-network-ofi | 8) cray-dsmml/0.2.2 | 13) systemdefault/1.0.0 (S) |
| 4) perftools-base/22.06.0 | 9) cray-mpich/8.1.17 | |
| 5) xpmem/2.3.2-2.2_9.4__g93dd7ee.shasta | 10) cray-libsci/21.08.1.2 | |

Where:

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



Newer PE version

```
% $ module av cpe
```

```
----- /opt/cray/pe/lmod/modulefiles/core -----  
cpe-cuda/22.06    cpe/22.06
```



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 (L,D)  cce/15.0.1
```

Where:

L: Module is loaded
D: Default Module

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/4.8.1.5
```

```
-----  
cray-netcdf: cray-netcdf/4.8.1.5  
-----
```

You will need to load all module(s) on any one of the lines below before the "cray-netcdf/4.8.1.5" module is available to load.

```
LUMI/21.12  partition/C  amd/5.2.3  cray-hdf5/1.12.1.5  
LUMI/21.12  partition/C  aocc/3.2.0  cray-hdf5/1.12.1.5  
LUMI/21.12  partition/C  cpeAOCC/21.12  cray-hdf5/1.12.1.5  
LUMI/21.12  partition/C  gcc/10.3.0  cray-hdf5/1.12.1.5
```

```
...
```

Help:

Release info: /opt/cray/pe/netcdf/4.8.1.5/release_info

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 envvars 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/15.0.0 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/15.0.0" with "gcc/12.2.0".

Due to `MODULEPATH` changes, the following have been reloaded:

1) `cray-mpich/8.1.23`

- 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 15.0.0
> ftn --version
Cray Fortran : Version 15.0.0
```
 - 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)
```



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/15.0.0 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/8.3.3" with "PrgEnv-cray/8.3.3".

Due to MODULEPATH changes, the following have been reloaded:

1) `cray-mpich/8.1.23`

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



GPU TARGET MODULE

- No GPU-specific modules are loaded by default
 - DARDEL GPU requires **craype-accel-amd-gfx90a** module
module load craype-accel-amd-gfx90a
 - By loading this module, you enable:
 - GPU support in the PrgEnv modules, e.g. MPI
 - Specific OpenMP and OpenACC offload flags, e.g. CCE
- ```
> module load craype-accel-amd-gfx90a
> ftn -craype-verbose --version
ftn_driver.exe -hcpu=x86-trento -haccel=amdgcN-gfx90a -hnetwork=ofi -hdynamic --version
> cc -craype-verbose --version -fopenmp
clang -march=znver3 -fopenmp-targets=amdgcN-amd-amdhsa -Xopenmp-target=amdgcN-amd-amdhsa -march=gfx90a -dynamic --version -fopenm
```
- 
- NOTE:
    - OpenACC supported only in PrgEnv-cray Fortran, see **man intro\_openacc**
    - OpenMP offload supported in PrgEnv-cray (see **man intro\_openmp**) and PrgEnv-amd



# ROCM MODULE

- ROCM module sets the environment to use ROCM

```
> $ module av rocm
```

```
----- /opt/cray/pe/lmod/modulefiles/core -----
rocm/5.0.2 (D) rocm/5.3.3
```

- The module add paths to LD\_LIBRARY\_PATH and set ROCM\_PATH env variable

```
> echo $ROCM_PATH
```

```
/opt/rocm
```

```
> echo $LD_LIBRARY_PATH
```

```
/opt/rocm/lib64:/opt/rocm/lib:/opt/rocm/rocprofiler/lib:/opt/rocm/rocprofiler/
tool:/opt/rocm/roctracer/lib:/opt/rocm/roctracer/tool:/opt/rocm/hip/lib:...
```

```
> echo $PATH
```

```
/opt/rocm/bin:/opt/rocm/rocprofiler/bin:/opt/rocm/hip/bin:...
```



# BUILD FOR GPUS: CRAY COMPILER

```
> module load PrgEnv-cray
> module load craype-accel-amd-gfx90a
> module load rocm
```

- Load the ROCM module as well as the accelerator module for the target GPU

```
Fortran OpenACC (enabled by default)
> ftn -hacc saxpy_acc_mpi.f90 -o saxpy_acc_mpi.x
Fortran OpenMP
> ftn -fopenmp gemv-omp-target-many-matrices.f90 -o gemv-omp-target-many-matrices.x
C/C++ OpenMP
> CC -fopenmp -DOMP main.cpp omp/OMPStream.cpp -I. -Iomp -DOMP_TARGET_GPU -o omp.x
C/C++ HIP
> CC -x hip -I. -Ihip -std=c++11 -DHIP main.cpp hip/HIPStream.cpp -o hip.x
```

- No OpenACC support for C/C++
- Check `CRAY_ACC_DEBUG` in `man intro_openacc` and `man intro_openmp`
- Generate compiler listings: `-hlist=a` for Fortran and `-fsave-loopmark` for C/C++

# BUILD FOR GPUS: AMD COMPILER

```
> module load PrgEnv-amd
> module load craype-accel-amd-gfx90a
> module load rocm
```

- Load the ROCM module as well as the accelerator module for the target GPU

```
> ftn -fopenmp gemv-omp-target-many-matrices.f90 -o gemv-omp-target-many-matrices.x
C/C++ OpenMP
> CC -fopenmp -DOMP main.cpp omp/OMPStream.cpp -I. -Iomp -DOMP_TARGET_GPU -o omp.x
C/C++ HIP
> CC -x hip -I. -Ihip -std=c++11 -DHIP main.cpp hip/HIPStream.cpp -o hip.x
```





## SUMMARY – HOW TO COMPILE FOR GPUS

---

- Load the **PrgEnv-xxx** module
- Load the CPU target module **craype-x86-trento**
- Load the GPU target module **craype-accel-amd-gfx90a**
- Load the ROCM module **rocm**
- Use the compiler wrappers



# Approaches to Accelerate Applications

## Accelerated Libraries

- The easiest solution, just link the library to your application without in-depth knowledge of GPU programming
- Many libraries are optimized by GPU vendors, eg. algebra libraries

## Directive based methods

- Add acceleration to your existing code (C, C++, Fortran)
- Can reach good performance with somehow minimal code changes
- OpenACC, OpenMP

## Programming Languages

- Maximum flexibility, require in-depth knowledge of GPU programming and code rewriting (especially for Fortran)
- Kokkos, RAJA, CUDA, HIP, OpenCL, SYCL



# Summary – Compilers

---

- Multiple compiler environments are available
  - All of them accessed through the wrappers ftn, cc and CC – just do module swap to change a compiler!
  - Load the proper modules
- There is no universally fastest compiler – but performance depends on the application, even input
  - We try however to excel with the Cray Compiler Environment
  - If you see a case where some other compiler yields better performance, let us know!
- Compiler flags do matter – be ready to spend some effort for finding the best ones for your application





# Questions?