HPE Cray Programming Environment

Kurt Lust
LUMI User Support Team (LUST)
University of Antwerp

LUMI

Why do I need to know this?

- **Q:** "I only want to run some programs, why do I need to know about programming environments?"
- **A:** They are an intrinsic part of an HPC system
  - Programs are preferably installed from sources to generate binaries optimised for the system. When running, the build environment needs to be at least partially recreated.
    - Less relevant on HPE Cray systems though as they do something that is unsual on most HPC systems
  - Even when installing from prebuild binaries, some modules might still be needed
    - Inject an optimised MPI library

LUMI

The Operating System

- The login nodes run SUSE Linux Enterprise Server 15 SP4
- The compute nodes run Cray OS, a restricted version of SUSE with some daemons inactive or configured differently and Cray's way of accessing file systems.
  - Goal is to limit OS jitter for better scalability of large applications
  - On the GPU nodes there was still the need to reserve 1 core for OS and drivers
  - This also implies that some software may not be able to run on the compute nodes
    - E.g., no /run/user/$UID
- Small system image, don't expect everything you find on a workstation

LUMI

Programming models

- C/C++ and Fortran provided by several compilers in the PE
- MPI and OpenSHMEM for distributed memory, RCCL
- OpenMP, also for GPU programming
- OpenACC only in Cray Fortran
- HIP, AMD alternative for CUDA
- Commitment to OpenCL unclear
- Not part of the regular PE, but we try to provide SYCL also
- Users reported success running Julia
- NO CUDA!!!!
  - And there will never be as this is NVIDIA proprietary technology

LUMI

Development environment on LUMI

- Compilers: Cray Compiling Environment, GNU compiler collection (AMD GPU support not enabled), AMD AOCC and ROCm compilers
- Cray Scientific and Math Libraries: FFTW, BLAS+Lapack, …
- Cray Message Passing Toolkit
- Additional tools to integrate everything and offer hugepages support
- Cray Performance Measurement and Analysis Tools
- Cray Debugging Support Tools
- Python and R
- A number of AMD tools (not always pre-installed)
- 3rd party: Linaro (ARM) Forge, Vampir, …

LUMI

Cray Compiling Environment

- Default compiler on most Cray systems
  - Designed for scientific software in an HPC environment
  - LLVM-based with extensions by HPE Cray for vectorization and shared memory parallelization
- Standards support:
  - C/C++ compiler essentially Clang/LLVM
  - Fortran is HPE Cray's own frontend and optimizer, supporting most of Fortran 2018
  - OpenMP support including offload: Full 4.5, partial 5.0/5.1 and working on more, see `man intro_openmp`
  - OpenACC support only in Fortran: 2.0, partial 2.x/3.x, see `man intro_openacc`
  - PGAS: UPC 1.2 and Fortran 2008 coarray support
  - MPI bindings

Scientific and math libraries

- LibSci
  - BLAS (Basic Linear Algebra Subroutines) and CBLAS (C interface wrappers)
  - LAPACK and LAPACKE (C interface to LAPACK)
  - IRT (Iterative Refinement Toolkit)
  - BLACS (Basic Linear Algebra Communication Subprograms) and ScaLAPACK
- LibSci_ACC: A subset of GPU-optimized routines from LibSci
- FFTW3: Fastest Fourier Transforms in the West
- Data libraries: netCDF and HDF5

Cray MPI

- Derived from ANL MPICH 3.4
- With tweaks for Cray
  - Improved algorithms for many collectives
  - Asynchronous progress engine for overlap of computation and communications
  - Customizable collective buffering when using MPI-IO
  - Optimized Remote Memory Access (one-sided) fully supported including passive RMA
- GPU-aware communications
- Support for Fortran 2008 bindings
- Full MPI 3.1 support except for dynamic process management and MPI_LONG_DOUBLE/MPI_C_LONG_DOUBLE_COMPLEX for CCE
- No mpirun/mpiexec, but Slurm srun as the process starter
- Layered on libfabric with the Cassini provider, and a GPU Transfer Library

Lmod

- The HPE Cray PE is configured through modules
  - On LUMI we use Lmod
- Basic module commands are similar in all 3 implementations of modules:
  - `module avail` : list available modules
  - `module list` : Show loaded modules
  - `module load` : To load a module
- Lmod supports a hierarchical module system: Distinguishes between available modules and installed modules
  - Some modules only become available after loading another module
  - Can be used to support multiple configurations with a single module name and version
  - Used extensively in the programming environment

LUMI

Compiler wrappers

- The HPE Cray PE compilers are usually used through compiler wrappers:
  - `cc` for C
  - `CC` for C++
  - `ftn` for Fortran
- Compiler selected based on the modules loaded
- Wrappers select the target CPU and GPU architectures based on target modules
- Some libraries are linked in automatically when the corresponding module is loaded
  - MPI: There is no `mpicc`, `mpiCC`, `mpif90`, etc.
  - LibSci and FFTW
  - netCDF and HDF5
- You can see what the wrapper does by adding `--craype-verbose`

LUMI

Selecting the version of the CPE

- Release numbers are of the type yy.dd, e.g., 22.08 for the release made in August 2022.
- There is always a default version assigned by the sysadmins
- The cpe module allows to change the default version
  - `module load cpe/22.08`
  - This module will try to switch loaded PE modules to the version corresponding to that PE version, but it sometimes fails due to bugs in that module. Loading it twice in separate `module load` commands fixes the issue.
  - Will produce a warning when unloading, but this is only a warning.
- Not needed when using the LUMI stacks, see later

The target modules

- CPU:
  - `craype-x86-rome`: Works everywhere, CPU in the login nodes and data and visualisation partition
  - `craype-x86-milan`: LUMI-C compute nodes
  - `crapye-x86-trento`: LUMI-G CPU
- GPU:
  - `craype-accel-host`: Will tell some compilers to compile for the host instead
  - `craype-accel-amd-gfx90a`: The MI200 series used in LUMI-G
- Network:
  - `craype-network-ofi`: Needed for the Slingshot 11 interconnect
  - `craype-network-none`: Omits network-specific libraries
- Compiler wrappers have corresponding options to overwrite these settings

LUMI

PrgEnv and compiler modules

- The PrgEnv-* modules load compiler, MPI and LibSci (and some other stuff)

PrgEnv	Description	Comp. mod.	Compilers
PrgEnv-cray	Cray Compiling Environment	cce	craycc, crayCC, crayftn
PrgEnv-gnu	GNU Compiler Collection	gcc	gcc, g++, gfortran
PrgEnv-aocc	AMD Optimizing Compilers (CPU only)	aocc	clang, clang++, flang
PrgEnv-amd	AMD ROCm LLVM compilers (GPU support)	amd	amdclang, amdclang++, amdflang

- `rocm` module needed when using PrgEnv-cray or PrgEnv-gnu on the GPUs

Getting help

PrgEnv	C	C++	Fortran
PrgEnv-cray	`man craycc`	`man crayCC`	`man crayftn`
PrgEnv-gnu	`man gcc`	`man g++`	`man gfortran`
PrgEnv-aocc/PrgEnv-amd	-	-	-
Wrappers	`man cc`	`man CC`	`man ftn`

- `--help` flag, e.g., for compilers
- `-dumpversion` (wrappers), `--version` (many compiler commands) for version information
- The `explain` command for Cray Fortran compiler error messages
- LUMI documentation for developers

LUMI

Google, ChatGPT and LUMI

- Google may not be your best friend to find information about the HPE Cray Programming Environment or error messages on LUMI
  - As it is currently mostly used on a few rather large systems
- ChatGPT won't do much better
  - Little data on the internet
  - And currently trained with data from 2 years ago
- HPE Cray has a command line alternative to Google: `man -K`
  - Example: Try
    `man -K FI_CXI_RX_MATCH_MODE`

Other modules

- cray-mpich for MPI
- cray-libsci for LibSci
- cray-fftw for the Cray FFTW library
- cray-netcdf, cray-netcdf-hdf5parallel, cray-parallel-netcdf, cray-hdf5, cray-hdf5-parallel
- cray-python with a selection of packages including mpi4py, NumPy, SciPy and pandas
- Cray-R

Warning 1: You do not always get what you expect…

LUMI

- The default behaviour of the Cray PE is to use default versions of libraries at runtime
  - The versions of these libraries do not correspond to the modules loaded
  - In fact, many applications will run without reconstructing the compile environment
  - BUT: If the default version on the system changes, the behaviour of your application might change…
- Solution: Prepend LD_LIBRARY_PATH with CRAY_LD_LIBRARY_PATH:
  export LD_LIBRARY_PATH=${CRAY_LD_LIBRARY_PATH}:$LD_LIBRARY_PATH
  - Experimental module `lumi-CrayPath` can be used to manage this
- Or use rpath linking when building:
  `export CRAY_ADD_RPATH=yes`

Warning 2: Order matters

- Some modules are only available when others are loaded first
- Some examples
  - cray-fftw only available when a processor target module is loaded
  - cray-mpich requires both craype-network-ofi and a compiler to be loaded
  - cray-hdf5 requires a compiler module to be loaded and cray-netcdf in turn requires cray-hdf5
  - And there are several other examples
- See next presentation to learn how to find modules