

HPE Cray Programming Environment

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

November 2026

Why do I need to know this?

LUMI

- **Q:** "I only want to run some programs, why do I need to know about programming environments?"
- **A:** Programming environments are an intrinsic part of an HPC system
  - The middleware essential for parallel computing is often part of the PE
  - 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 prebuilt binaries, some modules from the programming environment might still be needed
    - Inject an optimised MPI library
    - But sometimes a prebuilt binary will not work due to middleware incompatibilities

The Operating System

LUMI

- The login nodes run SUSE Linux Enterprise Server 15 SP6 (soon SP7)
  - Subtle differences with Red Hat derivatives and Debian derivatives
- 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
- Communication library is libfabric
  - No UCX support!
- Small system image, don't expect everything you find on a workstation
  - Smaller than most tier-2 systems

Battling OS jitter: Low-noise mode

LUMI

- Can mean different things
- Cray OS already reduces noise by disabling some daemons on the compute nodes
  - Reduces compatibility with regular applications
  - So some Cray systems had 2 modes
    - "Emulation mode" or "Cluster Compatibility Mode" where those daemons were enabled
    - "Scalability mode" also often called "low-noise" mode with the regular Cray OS
- Disabling some daemons is not always enough for extreme scalability
  - LUMI-GPUs have a more extreme form of "low-noise" mode: Core specialization: OS processes restricted to a reserved core that is not available for applications (core 0)
  - Assymetry in the node: 1 CCD with 7 available cores and 7 with 8 available cores
  - Additional headache for proper setup of efficient GPU applications
    - So LUMI now actually reserves 1 core per CCD

Programming models on LUMI

LUMI

- 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
- Looking at Chapel too
- Users reported success running Julia
- NO CUDA!!!!
  - And there will never be as this is NVIDIA proprietary technology

Development environment on LUMI

LUMI

- Compilers: Cray Compiling Environment (CCE), 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 other AMD tools (not always pre-installed)
- 3rd party: Linaro (ARM) Forge, Vampir, …
- But no Intel oneAPI HPC Toolkit…

AMD tools and technologies on LUMI

LUMI

- GPU: ROCm stack is the alternative to the NVIDIA CUDA stack
  - Fully open-source
  - HIP: C++ runtime API and kernel language similar to CUDA
  - Many libraries are an equivalent of CUDA libraries
    - Often a rocX and hipX: rocX is the ROCm library, hipX a layer for CUDA compatibility
  - hipify-clang and hipify-perl tools to assist in porting
  - C/C++ and Fortran compilers with OpenMP offload support (Fortran requires ROCm 7 new generation flang compiler)
  - Debugger: rocgdb
  - Performance measurement and analysis: rocprof, Omnitrace/ROCm Systems Profiler, Omniperf/ROCm Compute Profiler
- CPU: AOCC compilers are integrated in the HPE Cray PE, the AOCL libraries are not
  - We provide some compiled from sources, but beware for compatibility problems with the HPE Cray PE!

Cray Compiling Environment

LUMI

- 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 with Cray extensions for optimization
  - Fortran is HPE Cray's own frontend and optimizer, supporting Fortran 2018 and more and more Fortran 2023 features (and strict about it)
  - OpenMP support including offload: Full 5.0, partial 5.1/5.2/6.0, 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

LUMI

- LibSci
  - BLAS (Basic Linear Algebra Subroutines) and CBLAS (C interface wrappers)
  - LAPACK and LAPACKE (C interface to LAPACK)
  - BLACS (Basic Linear Algebra Communication Subprograms) and ScaLAPACK
  - IRT (Iterative Refinement Toolkit)
- LibSci_ACC: A subset of GPU-optimized routines from LibSci
- FFTW3: Fastest Fourier Transforms in the West
- Data libraries: netCDF and HDF5
- Cray TPSL does no longer exist as a product

Cray MPI

LUMI

- Derived from ANL MPICH 3.4a2 (8.1.x) or MPICH 4.1.2 (9.0.1 in 25.09, 4.3a1 for 9.1.0)
- 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 *(kind of)*
- Full MPI 3.1 support except for dynamic process management and `MPI_LONG_DOUBLE`/`MPI_C_LONG_DOUBLE_COMPLEX` for CCE
  MPI 4.0 support in versions 9.0.x
- No mpirun/mpiexec, but Slurm `srun` as the process starter
- Layered on libfabric with the Cassini provider, and a GPU Transfer Library

GPU-aware MPI

LUMI

- Cray MPICH supports
  - GPU-attached communication buffers, can use device pointers
  - GPU-NIC RDMA for inter-node MPI transfers
  - GPU Peer2Peer IPC for intra-node transfers (but with restrictions, see later)
- Enable:
  - At build time: Ensure the GTL (Graphics Transfer Library) is linked in
  - At runtime: `export MPICH_GPU_SUPPORT_ENABLED=1`
- If the GPU code uses MPI operations that access GPU-attached memory regions:
  `export MPICH_OFI_NIC_POLICY=GPU`
  If only CPU buffers are used, is better `export MPICH_OFI_NIC_POLICY=NUMA`.
- See also later: Depending on how Slurm is used, Peer2Peer IPC might not work.
  Turn off: `export MPICH_GPU_IPC_ENABLED=0`.
  - Or alternatively: `export MPICH_SMP_SINGLE_COPY_MODE=NONE`
  - Both have severe consequences for performance

Lmod

LUMI

Later today…

- 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
  - `module unload` : To unload 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

Compiler wrappers

LUMI

- 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: no `mpicc`, `mpiCC`, `mpif90`, etc. needed
  - LibSci and FFTW
  - netCDF and HDF5
- You can see what the wrapper does by adding `-craype-verbose`
- Wrappers are provided by the `craype` module

Selecting the version of the CPE

LUMI

- Release numbers are of the type yy.dd, e.g., 25.03 for the release made in March 2025.
- There is always a system default version assigned by the sysadmins
  - Currently 25.03 even though we have a newer version on the system also
- The `cpe` module allows to change the default version
  - `module load cpe/25.09`
  - 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 often 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

LUMI

- CPU:
  - `craype-x86-rome`: Works everywhere, CPU in the login nodes and data and visualisation partition
  - `craype-x86-milan`: LUMI-C compute nodes
  - `craype-x86-trento`: LUMI-G CPU
- GPU:
  - `craype-accel-amd-gfx90a`: The MI200 series used in LUMI-G
  - `craype-accel-host`: Will tell some compilers to compile for the host instead
- 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

PrgEnv and compiler modules

LUMI

- 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-native gcc	(gcc-14, g++-14, gfortran-14) 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

LUMI

PrgEnv	C	C++	Fortran
PrgEnv-cray	`man craycc`	`man crayCC`	`man crayftn`
PrgEnv-gnu	`man gcc-14`	`man g++-14`	`man gfortran-14`
PrgEnv-aocc/PrgEnv-amd	-	-	-

- Web-based documentation from HPE Cray: cpe.ext.hpe.com/docs
- `--craype-help` for the compiler wrapper-specific help
- `--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: docs.lumi-supercomputer.eu/development

Google, ChatGPT and LUMI

LUMI

- Google used to be pretty bad for the CPE but things have improved
  - There used to be very little information on the web, so also little for Google to search in
- ChatGPT, gemini, copilot or similar AI assistants are becoming useful
  - Be critical. The answer is not always right, but it oftens put you in the right direction. Go into a dialog and ask for sources of information
  - AI is particularly good at inventing non-existing environment variables
- Try the search box on cpe.ext.hpe.com/docs
- HPE Cray has a command line alternative to Google for the included man pages: `man -K`
  - Example: Try
    `man -K FI_CXI_RX_MATCH_MODE`

Other modules

LUMI

- 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 may not get what you expect…

LUMI

- The default behaviour of the Cray PE is to use system 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
  - Automated via module `lumi-CrayPath`
- Or use rpath linking of the CPE components when building:
  `export CRAY_ADD_RPATH=yes`
- Or with no PE modules loaded, from 25.03 onwards:
  `export PE_LD_LIBRARY_PATH=system`

Warning 2: Order matters

LUMI

- 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 presentation on modules to learn how to find modules (though it doesn't always work well for the PE modules)

Note: Working without CPE wrappers

LUMI

- It is now possible to work without the HPE Cray PE compiler wrappers
  - Use the compilers from the "PrgEnv and compilers" slide
  - With cray-mpich loaded you can then use the traditional MPI compiler wrappers `mpicc`, `mpiCC`, `mpif90`, etc.
    - Issues with `gcc-native` are now resolved
  - You'll have to manually link all other libraries (BLAS, FFTW, netCDF, …) even if you use those provided in the HPE Cray PE environment
- Tip: Most modules define environment variables that point to the bin and lib subdirectories
  - `CRAY_MPICH`
  - `CRAY_LIBSCI_PREFIX_DIR` / `CRAY_PE_LIBSCI_PREFIX_DIR`
  - `FFTW_ROOT`, `FFTW_INC` and `FFTW_DIR`


LUMI
Questions?