DevOps School for HPC

Lmod: A Modern Environment Modules System for HPC

André Ramos Carneiro
Laboratório Nacional de Computação Científica (LNCC) - Brasil

Agenda

- Present myself
- Introduction
- History
- What is Lmod?
- Purpose
- Lmod vs. Environment Modules
- Installation & Configuration
- Examples
- Summary

André Ramos Carneiro

I'm a Technologist at the National Laboratory for Scientific Computing (LNCC), leading the HPC Support. I'm also System Manager of the SDumont Supercomputer***s***. I have 20 years of experience with HPC and Scientific Computing.

I have a Master's degree from the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), with focus on Parallel File System.

LNCC is located at Petrópolis, a city in the mountains region of Rio de Janeiro, Brazil. Our main goals are:

- Research and develop scientific computing
- Provide HPC service for the Brazilian community (SINAPAD and SDumont)
- Postgraduate program, with master's and doctoral degrees

Introduction

- In High-Performance Computing (HPC), managing multiple software versions is challenging. Different versions / needs / requirements
  - "*I want GCC v6.7*"
  - "*My software only works with OpenMPI 2.1 and GCC 4*"
  - "*I need the Quantum Espresso X.Y, with this patch and that plugin*"
- Each software **needs to be** installed in a specific directory
  - Cannot put everything under system path (`/usr, /bin, /lib` ...)
  - But we need to configure the environment (`PATH, LD_LIBRARY_PATH` ...)
- In the *very old days*, we used the "`source bash_file`" to set the variables
  - `export PATH=/dir/my/app/bin:$PATH`
  - `export LD_LIBRARY_PATH=/dir/my/app/lib:$LD_LIBRARY_PATH`
  - Can be *messy* and does not allow for cleaning up the environment.
- Environment Modules (*Old*, TCL based) simplify software setup.

History of Lmod

- Created by Robert McLay at TACC.
- Designed as a replacement for **TCL-based** Environment Modules.
- Written in Lua for flexibility and performance.
- Timeline
  - 1990s → Tcl-based Environment Modules
  - 2008 → Start of the development of Lmod at TACC
  - 2011 → First Stable Version
  - 2025 → Widely adopted by research centers and HPC facilities.

What is Lmod?

- A Lua-based environment modules system.
- **Lmod** is the **modern** and **advanced** alternative to traditional Environment Modules.
- Allows loading, unloading, and managing **multiple software versions**.
- Compatible with classic 'Environment Modules' commands.

Purpose of Lmod

- Simplify management of complex software environments.
- Allow users to switch between software versions easily.
- Integrate seamlessly with schedulers like Slurm.
- Reduce dependency conflicts in HPC clusters.
- Hierarchical module naming.
- Dynamic module loading/unloading.
- Version management and search via '`module spider`'.
- Sticky modules prevent accidental unloading.

Lmod vs Environment Modules

- Lmod is faster and more scalable.
  - Implement cache features
- Supports hierarchical modules for complex environments.
- Provides `'spider'` command for searching modules.
- Backward-compatible with older commands.
- Handles conflicts and dependencies automatically.
- Environment Modules needs the "`#!Module`" magic cookie signature
- Lmod files need to have the "`.lua`" extension

Installing Lmod

Installing Lmod on Ubuntu

```
# Install dependencies
sudo apt update && sudo apt install lmod

# Enable Lmod in shell
source /etc/profile.d/lmod.sh
module avail
```

Installing Lmod on Rocky Linux

```
# Enable EPEL repository
https://docs.fedoraproject.org/en-US/epel/getting-started/

# Install Lmod package
sudo dnf install Lmod

# Verify installation
module avail
```

Basic Lmod Commands & Hierarchy

Modulefile: Hierarchy

- The “main” directory is defined by the `MODULEPATH` environment variable
  - “Defaults” to `/usr/share/lmod/lmod/modulefiles`
- Below this dir, you can have the following structure

```
Compiler
   ├── gcc/12.1.0
   └── gcc/11.2.0
MPI
   ├── openmpi/4.1.5
   ├── mpich/3.4.3
IO-Libs
   ├── hdf5/1.14.0
   └── netcdf/4.9.2
```

Basic Lmod Commands

- `module avail [name]` → List available modules.
  - Can specify an "application" name to filter.
- `module overview [name]` → Resume of available modules
  - Number of versions. Can specify an "application" name to filter.
- `module spider <name>` → Search for a module
- `module load <name>` → Load a module
- `module list` → Show loaded modules
- `module unload <name>` → Unload a module
- `module purge` → Remove all loaded modules
- `module swap <name1> <name2>` → Exchange between modules

Modulefile: Listing available

```
$ module avail

---- /usr/share/lmod/lmod/modulefiles ----
   Compiler/gcc/11.2.0    Compiler/gcc/12.1.0 (D)    Core/lmod    Core/settarg (D)
IO-Libs/hdf5/1.14.0    IO-Libs/netcdf/4.9.2    MPI/mpich/3.4.3    MPI/openmpi/4.1.5

  Where:
   D:  Default Module
```

Modulefile: Listing available

```
$ module overview

---- /usr/share/lmod/lmod/modulefiles ----
Compiler/gcc (2)   Core (2)   IO-Libs/hdf5 (1)   IO-Libs/netcdf (1)   MPI/mpich (1)   MPI/openmpi
(1)

$ module overview gcc

---- /usr/share/lmod/lmod/modulefiles ----
Compiler/gcc (2)
```

Modulefile: Loading & Listing

```
$ module list
No modules loaded

$ module load Compiler/gcc
$ module list

Currently Loaded Modules:
  1) Compiler/gcc/12.1.0

$ module load MPI/openmpi/4.1.5
$ module list

Currently Loaded Modules:
  1) Compiler/gcc/12.1.0   2) MPI/openmpi/4.1.5
```

- When the “version” is not specified, it loads the default (**D**) one
  - Usually, the newer one!

Modulefile: UnLoading

```
$ module list

Currently Loaded Modules:
  1) Compiler/gcc/12.1.0   2) MPI/openmpi/4.1.5

$ module unload MPI/openmpi/4.1.5
$ module list

Currently Loaded Modules:
  1) Compiler/gcc/12.1.0

$ module purge
$ module list
No modules loaded
```

- "Good" practice to recommend the use of `module purge` at the beginning of the job submission script.
  - Make sure the *env* is clean, with no "*trash*" carried on from the current login session

Modulefile: Hierarchy (contd.)

- It's possible to have multiples `MODULEPATH` directories
  - Add to the list in the file /etc/lmod/modulespath (Ubuntu).

```
/etc/lmod/modules
/usr/share/lmod/lmod/modulefiles
/MY/SHARED-STORAGE/MODULEFILES
```

  - Modify the file `/etc/profile.d/lmod.sh`
    - add at the end of the file

```
export MODULEPATH=/MY/SHARED-STORAGE/MODULEFILES:${MODULEPATH}
```

  - Create a custom file `/etc/profile.d/modules_extra_path.sh`

```
if [ -z "$MODULEPATH" ]; then
   export MODULEPATH=/MODULEFILES-DIR1:/MODULEFILES-DIR2
else
   export MODULEPATH=$MODULEPATH:/MODULEFILES-DIR1:/MODULEFILES-DIR2
fi
```

- **Attention** to the "position" (order) of the custom dir in the `MODULEPATH` env

Creating a Modulefile

Creating a Modulefile

- Lua (language behind Lmod) have a simple syntax
  - More about lua: http://www.lua.org
- Lua modulefiles are written in the positive.
  - The actions necessary to configure the *env*
- A modulefile contains commands to add to the PATH or set environment variables.
  - When loading a modulefile, the commands are *executed* (run).
  - When unloading a modulefile, the actions are *reversed*.
- What is added/modified during the "`module load`" command is undone during the "module unload"
- The environment variables set during loading are unset during unloading.

Creating a Simple Modulefile

- Old, TCL, example: File - `$MODULEPATH/gcc/12.1.0`

```
#%Module
proc ModulesHelp { } {
    puts stderr "GCC Compiler"
}
prepend-path    PATH   /opt/gcc/12.1.0/bin
setenv          CC     gcc
setenv          CXX    g++
```

Creating a Simple Modulefile

- New, Lua, example: File - `$MODULEPATH/gcc/12.1.0.lua`

```
# Example: gcc/12.1.0.lua
help([[GCC Compiler]])
prepend_path('PATH', '/opt/gcc/12.1.0/bin')
setenv('CC', 'gcc')
setenv('CXX', 'g++')
```

Lmod Functions

Basic *Functions*

- `prepend_path("VAR", "VALUE", "delim")`: **pre**pend a **value** to a **var**iable. It is possible (optional) to add a **delim**iter. Default is **":"** (can be any **single** character, " " or ";")
  - Ex: PATH=/bin.
  - `prepend_path("PATH", "/home/me/bin") ->`
    - `PATH = /home/me/bin:/bin`
- `append_path("VAR", "VALUE", "delim")`: **ap**pend a **value** to a **var**iable. It is possible (optional) to add a **delim**iter. Default is **":"** (can be any **single** character, " " or ";")
  - Ex: PATH=/bin.
  - `append_path("PATH", "/home/me/bin") ->`
    - `PATH = /bin:/home/me/bin`
- `remove_path("VAR", "VALUE", "delim")`: **remove value** from a **var**iable, for both **load** and **unload** *modes*. It is possible (optional) to add a **delim**iter. Default is **":"** (can be any **single** character, " " or ";")
  - Ex: If it's necessary to remove '`/usr/bin`' from the `PATH` variable
  - **Not** reversible

Basic *Functions*

- `setenv("VAR", "VALUE")`: assigns a **value** to a **var**iable. Do not use this function to assign the initial value to a variable. Use append_path or prepend_path instead.
- `local VAR = "VALUE"`: assigns a **value** to a local **var**iable.
  - Like "`set`" does for the current shell or environment file
- `pushenv("NAME", "VALUE")`: sets **NAME** variable to **value**. In addition, it **saves** the **previous** value in a hidden environment variable. This way, the previous state can be returned when a module is unloaded.
- `whatis("STRING")`: The whatis command can be called repeatedly with different strings. See the Administrator Guide for more details.
- `help( [[ help string ]])`: What is printed out when the help command is called. Note that the help string can be multi-lined.

Not-So Basic *Functions*

- `purge()`: This command will unload all (non-sticky) modules except the one currently being loaded.
- `source_sh("shellName","shell_script arg1 ...")`: source a shell script as part of a module. Supported `shellName` are: `sh, dash, bash, zsh, csh, tcsh, ksh`.
  - This feature was introduced in Lmod 8.6+.
- `family("name")`: A user can only have one family "name" loaded at a time.
  - `family("compiler")` would mean that a user could only have one compiler loaded at a time.
    - Could be anything *user-defined*: MPI, HDFLIB, PYTHON
  - The "new" module loaded replaces the "current" module loaded, which has the same "family"
- `conflict("name1", "name2")`: The current modulefile will only load if all listed modules are **NOT** loaded.

Not-So Basic *Functions*

- `extensions("app/2.1, other_app/3.2, lib/1.3")`: Informative of which "extensions" this module provides.
  - Can be used for additional plugins used.
  - Place the list of extensions as a single string.
  - Shows during a "`module spider <module>`" execution. Ex:

```
$ module spider openmpi/4.1.4
…
This module provides the following extensions:
       gcc/12
```

Dependency *Functions*

- Usually a software "X" depends on module "A" (be it a compiler, a library). There are several ways to handle module dependency:
- Use `depends_on("A","B")` - automatic handling
  - Loads module "A" (and "B") when "X" loads. Unloads all during unload of "X"
  - `module load X; module unload X => unload A`
  - `module load A; module load X; module unload X => keep A`
- Use `depends_on_any("C","D", ...)` - automatic handling
  - Similar to `depends_on()`, except that Lmod picks the **first** available module listed.
- Use `prereq("A")` - explicit (manual) handling
  - Issues a warning if module "A" isn't **previously** and **manually** loaded.
  - When unloading "X", module "A" will remain loaded.

Dependency *Functions*

- *Usually a software "X" depends on module "A" (be it a compiler, a library). There are several ways to handle module dependency:*
- Use `load("A")`
  - **Always** load module "A", even if "A" is already loaded. When module "X" is unloaded, module "A" will **always** be unloaded as well.
  - Difference from `depends_on` is that, if "A" is already loaded, the unload of "X" will unload "A" as well
- Use `always_load("A")`
  - **Always** load module "A", even if "A" is already loaded. When module "X" is unloaded, module "A" **will** remain **loaded**.
  - … a bit confusing, but hey, it's an option 🤷

Dependency *Functions*

- On the old TCL Module Environment

```
if { [ module-info mode load ] } {
   module load module/A
} else {
   module unload module/A
}
```

Good Practices

… in LNCC, at least

Good Practices

… in LNCC, at least

- Use the "`module purge`" at the beginning of the job script
- Advise the users to NOT configure the environment at the `~/.profile` or `~/.bashrc` (and similar) files
  - This usually leads to unforeseen/unknown errors
  - They forgot what they have done
- Use a shared directory among nodes to store the module files
  - Only need to write them once, easy to maintain
- Define some useful standard variables (only inside the module file)
  - `APPNAME` = Name of the application/software/library
  - `APPVERSION` = Version of the application/software/library
  - `APPHOME` = Directory of the installation. EX: `/shared_dir/softwares/$APPNAME/$APPVERSION`

Good Practices

… in LNCC, at least

- Important variables to configure
  - Default: `PATH, LD_LIBRARY_PATH, MANPATH, PKG_CONFIG_PATH`
  - Good for compiling: `LDFLAGS` (with the libs directory, starting with `-L`)
  - Good for compiling: `CPPFLAGS` (with the include directory, starting with `-I`)
- Be informative on the `help`/`whatis` functions
  - What this module already loads as a dependency
  - Additional plugins
  - Any special feature: GPU support, parallel support (MPI / Threads)

Modulefile

improved with "Good Practices"

```
#Local variables
local APPNAME = "gcc"
local APPVERSION = "12.1.0"
local APPHOME = "/opt/" .. APPNAME .. "/" .. APPVERSION

help("GCC Compiler version " .. APPVERSION)
whatis("GCC Compiler version " .. APPVERSION)
prepend_path('PATH', APPHOME .. '/bin')
prepend_path('LD_LIBRARY_PATH', APPHOME .. '/lib')
prepend_path('MANPATH', APPHOME .. '/share/man')
prepend_path('PKG_CONFIG_PATH', APPHOME .. '/lib/pkgconfig')
prepend_path('LDFLAGS', '-L' .. APPHOME .. '/lib', " ")
prepend_path('CPPFLAGS', '-I' .. APPHOME .. '/include', " ")
family("COMPILER")
setenv('CC', 'gcc')
setenv('CXX', 'g++')
```

Modulefile

improved with "Good Practices"

```
#Local variables
local APPNAME = "openmpi"
local APPVERSION = "4.1.4"
local APPHOME = "/opt/softwares/" .. APPNAME .. "/" .. APPVERSION

help("OpenMPI Library v" .. APPVERSION .. " , compiled with GCC 12")
whatis("OpenMPI Library v" .. APPVERSION .. " , compiled with GCC 12")
prepend_path('PATH', APPHOME .. '/bin')
prepend_path('LD_LIBRARY_PATH', APPHOME .. '/lib')
prepend_path('MANPATH', APPHOME .. '/share/man')
prepend_path('PKG_CONFIG_PATH', APPHOME .. '/lib/pkgconfig')
prepend_path('LDFLAGS', '-L' .. APPHOME .. '/lib', " ")
prepend_path('CPPFLAGS', '-I' .. APPHOME .. '/include', " ")
setenv('MPICC', 'mpicc')
setenv('MPICXX', 'mpic++')
family("MPI")
extensions("gcc/12")
depends_on("gcc/12")
```

Modulefile

improved with "Good Practices"

```
$ module list
No modules loaded

$ module load openmpi/4.1.4
$ module list

Currently Loaded Modules:
  1) gcc/12   2) openmpi/4.1.4

$ echo $LDFLAGS
-L/shared_dir/softwares/gcc/12.1.0/lib -L/shared_dir/softwares/openmpi/4.1.4/lib

$ echo $PATH
/opt/softwares/gcc/12.1.0/bin:/opt/softwares/softwares/openmpi/4.1.4/bin:/usr/
bin:/home/user/bin:.....

$ module unload openmpi/4.1.4
$ module list
No modules loaded
```

Good Practices

… in LNCC, at least

- Organize the modulefiles directory structure
  - **Do**:
    - MODULEPATH/Software_X/VERSION_1
    - MODULEPATH/Software_X/VERSION_2
    - MODULEPATH/Software_Y/VERSION_1
    - MODULEPATH/Lib_Z/VERSION_1
  - **Don't**:
    - MODULEPATH/Software_X_VERSION_1
    - MODULEPATH/Software_X_VERSION_2
    - MODULEPATH/Software_Y_VERSION_1
    - MODULEPATH/Lib_Z_VERSION_1

Good Practices

… in LNCC, at least

- In multi-architecture environments, separate the modulefiles
- On SDumont (1st version of Santos Dumont Supercomputer), every modulefile are inside a “main” directory, mixing versions of a same software for CPU-Only with GPU-Support
  - Gets **very** *messy* with the 1000+ modulefiles we created

SDdumont

```
PROJ/5.2.0_sequana
R/3.6.3_gnu_sequana
R/4.0.3_gnu_sequana
R/4.1.0_gnu_sequana
R/4.1.1_gnu_sequana                                 (D)
abcluster/3.2_sequana
anaconda2/2019.10_sequana
anaconda3/5.2.0_sequana
anaconda3/2020.02_sequana
anaconda3/2020.07_sequana
anaconda3/2020.11
anaconda3/2024.02_sequana                           (D)
antismash/5.1_sequana
apbs/1.4.2_openmpi_gnu_sequana
arpack/3.9.1_gnu_gcc-9.3
autodock-gpu/4.2.6_cuda_sequana
autodock-gpu/4.2.6_opencl_sequana                   (D)
autogrid/4.2.6_sequana
automake/1.15
automake/1.16                                       (D)
bamtools/2.5.1_gnu_sequana
barrnap/0.9_sequana
bbmap/38.81_sequana
bcftools/1.10.2_gnu_sequana
beast/1.8.4_sequana
beast/1.10_sequana
beast/2.5_sequana
beast/2.6_sequana                                   (D)
bedtools/2.29_gnu_sequana
berkeleygw/3.0.1_intel_sequana
blast/2.6.0_gnu_sequana
blast/2.10.0_gnu_sequana                            (D)
boost/1.72.0_gnu_leo_sequana
boost/1.72.0_gnu_sequana.suporte
boost/1.72.0_gnu_sequana
boost/1.72.0_intel_sequana
boost/1.86.0_gnu+openmpi-4.1.4_sequana
boost/1.86.0_intel-2020_sequana
boost/1.87.0_gnu+openmpi-4.1.4_sequana              (D)
bowtie2/2.4_gnu_sequana
bwa/0.7_gnu_sequana
card-rgi/5.1.0_sequana
cdo/2.4.0_openmpi-4.1.6_sequana
cdo/2.4.2_openmpi-4.1.6_sequana                     (D)
cellranger/6.1.2
cgal/5.6_gcc-9.3
checkm/1.1.2_sequana
cmake/3.9.4_sequana
cmake/3.12_sequana
cmake/3.17.1_gnu_sequana
cmake/3.17.3_sequana
cmake/3.18_gcc-7.4_sequana
hdf5/1.8.22_openmpi-2.1.6_gcc-6.5
hdf5/1.10.7_openmpi-4.1.4_gnu_sequana
hdf5/1.12_intel_2018_sequana
hdf5/1.12_intel_2020_sequana
hdf5/1.12.2_threadsafe-HL_openmpi-4.1.6_gnu_sequana (D)
hdf5/1.14_intel_2020_sequana
hdf5/1.14.3_openmpi-4.1.6_gnu_sequana
hdf5/1.14.5_oneapi_2022_intel_sequana
hdf5/1.14.5_openmpi-4.1.4_gcc-12.4_sequana
hdf5/1.14.5_openmpi-4.1.4_gnu_sequana
hdf5/1.14.6_intel_2025_sequana
hdf5/1.14.6_oneapi_2022_intel_sequana               (D)
help2man/1.49.3_gcc-12.4_sequana
hisat/0.1.6-beta_sequana
hmmer/2.3.2_gnu_sequana
hmmer/3.3_gnu_sequana
hmmer/3.3_openmpi+gnu_sequana                       (D)
hmmer/3.4_openmpi
hs-blastn/0.0.5+_gnu_sequana
htop/3.3.0                                          (D)
htslib/1.10.2_gnu_sequana
hwloc/2.11.2_gcc-9.3_sequana
hwloc/2.11.2_gcc-12.4+cuda-12.6_sequana
hwloc/2.11.2_gcc-12.4_sequana
hwloc/2.11.2_sequana
hypre/2.14_intel_2020_sequana
hypre/2.15_intel_sequana
hypre/2.15_openmpi-2.0_gnu_sequana                  (D)
hypre/2.15_openmpi-2.0_intel_sequana
idba/1.1_gnu_sequana
igv/2.8.2_sequana
intel-oneapi/2021_sequana                           (D)
intel-oneapi/2022_sequana
intel-oneapi/2025.0_sequana
intel-oneapi/DISABLED_2025.0_sequana_testing_suporte_v2
intel-opencl/2020_sequana
intel_psxe/2016_sequana
intel_psxe/2017_sequana
intel_psxe/2018_sequana                             (D)
intel_psxe/2019_sequana
intel_psxe/2020_sequana
interproscan/5.42_sequana
iqtree/2.1.1_sequana
jasper/4.2.4_sequana                                (D)
java/jdk-8u201_sequana
java/jdk-11_sequana
java/jdk-12_sequana
jellyfish/2.3_gnu_sequana
jemalloc/gnu_sequana
julia/1.10.6_gcc-9.3_sequana
julia/1.10.7
julia/1.11.3
openmpi/gnu/2.1.1_sequana
openmpi/gnu/2.1.6_gcc-6.5
openmpi/gnu/4.1.1_sequana
openmpi/gnu/4.1.2+cuda-11.2_sequana
openmpi/gnu/4.1.4+cuda-11.2_sequana
openmpi/gnu/4.1.4+gcc-9.3+cuda-11.1_sequana
openmpi/gnu/4.1.4+gcc-12.4+cuda-10.2_sequana
openmpi/gnu/4.1.4+gcc-12.4+cuda-11.6_sequana
openmpi/gnu/4.1.4+gcc-12.4_sequana
openmpi/gnu/4.1.4_sequana
openmpi/gnu/4.1.6_sequana
openmpi/gnu/4.1.8_sequana                           (D)
openmpi/gnu/5.0.5_sequana_TESTING
openmpi/icc/4.0.3_sequana_testing
openmpi/icc/DISABLED-2.0.4.2
openmpi/icc/DISABLED-4.0.3_sequana
openmpi/icc/debug/DISABLED-2.0.2.10
openmpi/icc/ilp64/DISABLED-2.0.4.2                  (D)
openmpi/icc/mt/DISABLED-2.0.4.2
openmpi/icc/mt/debug/DISABLED-2.0.2.10
openmpi/icc/mt/ilp64/DISABLED-2.0.4.2
openpmix/3.1.5_sequana
openpmix/4.2.8_sequana
openpmix/5.0.4_sequana
openssl/3.0.9_sequana                               (D)
orca/4.2.1_openmpi-3.14_gnu
orca/5.0.4
orca/6.0.0_avx2
orca/6.0.1_avx2                                     (D)
p11-kit/0.23.21_gnu_sequana
pa-star/pa-star-v1.1-48_sequana
parabricks/2.5.0_singularity_sequana
parabricks/3.0_singularity_sequana
paraview/5.13.2_sequana
parmetis/4.0.3_gnu_sequana.suporte
parmetis/4.0.3_gnu_sequana
parmetis/4.0.3_intel_sequana
parmetis/4.0.3_openmpi-2.0_gnu-7.4_sequana
parmetis/4.0.3_openmpi-4.1.2_gnu-9.3_sequana
pcre2/10.37_sequana
pcre2/10.44_gcc-9.3_gnu_sequana                     (D)
perl/5.30_sequana
perl/5.38.2_sequana
petsc/3.10_intel_2020_sequana
petsc/3.10_openmpi-2.0_gnu_sequana
petsc/3.17_openmpi-4.1.4_gnu_sequana
pgi/compilers-18.10_sequana                         (D)
pgi/compilers-19.4_sequana
pgi/compilers-19.10_sequana
pgi/cuda-9.1-18.10_sequana
pgi/cuda-9.2-18.10_sequana
pgi/cuda-9.2-19.4_sequana
```

Good Practices

… in LNCC, at least

- *In multi-architecture environments, separate the modulefiles*
- On SDumont2nd (the new machine), we are dividing the modulefiles by architecture, with special modulefiles
  - The main directory (`MODULEPATH = /shared_dir/modulefiles`) have modulefiles for general availability, and the special modulefiles:
  - The `arch_apu_amd` module file configure the environment for the AMD MI300A APU
  - The `arch_arm` module file configure the environment for the GRACE-GRACE ARM CPU
  - The `arch_cpu_amd` module file configure the environment for the AMD GENOA CPU
  - The `arch_gpu` module file configure the environment for the NVIDIA H100 GPU
  - The `arch_gpu_sc` module file configure the environment for the NVIDIA GRACE-HOPER GH200+ARM
- `prepend-path('MODULEPATH', /shared_dir/modulefiles_extra/architecture)`

SDumont2nd

```
$ module load arch_apu_amd/current
Loading AMD APU (MI300A) architecture Software environment

$ module av
---- /shared_dir/modulefiles_extra/apu_amd ----
   gromacs/2025.2_openmpi-4.1_amd    lammps/27Jun2024_openmpi-4.1_amd
...

$ module unload arch_apu_amd/current
Unloading AMD APU (MI300A) architecture Software environment

$ module load arch_gpu_sc/current
Loading GPU and ARM64 CPU architecture (NVIDIA SuperChip Grace Hopper GH200) Software environment

$ module av

---- /shared_dir/modulefiles_extra/gpu_sc ----
anaconda3/2024.10 (D)      gromacs/2021.b_openmpi-4.1_gnu  lammps/15Jun2023_nv_container
cp2k/9.1_nv_container      gromacs/2022.3_nv_container          namd/3.0_nv_container
cp2k/2023.1_nv_container   gromacs/2023.2_nv_container          nvhpc/25.5
cp2k/2023.2_nv_container   gromacs/2024.3_plumed_openmpi-4.1    openmm/8.3
cuda/11.8                  lammps/3Nov2022_nv_container         plumed/2.9_aarch64_mpi
```

Good Practices- Module hierarchy

… not done in LNCC, yet

- One advantage of Lmod is the "Software Module hierarchy"
- Compiler and MPI-based Dependencies
  - Ex: When "gcc/12" is loaded, all the software compiled with it "shows up"
  - Ex: When "openmpi/4.1.1" (which was compiled with gcc/12) is loaded, all the software compiled with it "shows up"
- Only packages compatible with the currently loaded compiler/MPI/what-ever become visible and loadable
- Modulefile Organization: Modulefiles are typically organized in a hierarchical directory structure. A common structure might include:
  - `/opt/apps/modulefiles/Core`: For modules with no compiler or MPI dependencies.
  - `/opt/apps/modulefiles/Compiler`: For modules dependent only on a specific compiler.
  - `/opt/apps/modulefiles/MPI`: For modules dependent on a specific compiler-MPI pairing.

Good Practices - Module hierarchy

… not done in LNCC, yet

- Reduced Conflicts: Prevents users from loading incompatible software combinations.
- Simplified Module Availability: Users only see modules relevant to their current environment.
- Automated Dependency Management: Lmod handles the unloading and reloading of dependent modules during swaps.
- Improved User Experience: Makes it easier for users to find and load the correct software versions.
- At LNCC, we are still transitioning to this way of handling hierarchy, but it's difficult to change the mentality of the staff and the users
    - A lot of the OLD TCL-based modules working…

Summary

Summary

- Lmod is a modern, flexible alternative to classic Environment Modules.
- Simplifies managing multiple software versions.
- Offers advanced search, hierarchy, and conflict resolution features.
- Ideal for HPC environments.
- Information about transitioning to Lmod
  - https://lmod.readthedocs.io/en/latest/045_transition.html
  - https://lmod.readthedocs.io/en/latest/073_tmod_to_lmod.html

Q&A

- Questions?
- andrerc@lncc.br