Introduction to the module system and compiler toolchains

Birgitte Brydsö, Pedro Ojeda-May, Jerry Eriksson

Umeå University, 2016-11-08
The module system - Abisko

- Most programs are accessed by first loading them as a 'module'
- See which modules exist
  
  \texttt{module avail}

- Different versions of software
  
  \texttt{module avail} \texttt{<module name>}

- Example: loading the default intel compilers
  
  \texttt{module load intel}

- Unload the module
  
  \texttt{module unload intel}
The module system - Kebnekaise

- See which modules exist
  `ml spider`
- Modules available without loading a compiler and/or mpi toolchain
  `ml av` (or `module avail`)
- See which modules are currently loaded
  `ml` (or `module list`)
- Example: print more information about a module, here iccifort
  `ml show iccifort/2017.1.132-GCC-5.4.0-2.262017`
  (or `module show iccifort/2017.1.132-GCC-5.4.0-2.26`)
The module system - Kebnekaise

- Example: loading the intel compilers, iccifort
  
  \texttt{ml iccifort/2017.1.132-GCC-5.4.0-2.26}

  (or \texttt{module load iccifort/2017.1.132-GCC-5.4.0-2.26})

- Example: Unload the above module
  
  \texttt{ml -iccifort/2017.1.132-GCC-5.4.0-2.26}

  (or \texttt{module unload iccifort/2017.1.132-GCC-5.4.0-2.26})

- Example: loading a compiler toolchain, here for GCC
  
  \texttt{ml foss/2016.09} (or \texttt{module load foss/2016.09})
Compiler toolchains - Kebnekaise

Compiler toolchains load bundles of software making up a complete environment for compiling / using a specific prebuilt software. Usually includes a compiler suite, an MPI version, BLAS, LAPACK, ScaLapack, and FFTW versions.

Currently available toolchains:

- **GCC**: GCC only
- **foss**: GCC, OpenMPI, OpenBLAS/LAPACK, FFTW, ScaLAPACK
- **gompi**: GCC, OpenMPI
- **gimpi**: GCC, IntelMPI
- **gimkl**: GCC, IntelMPI, IntelMKL
- **gompic**: GCC, OpenMPI, CUDA
- **goolfc**: gompic, OpenBLAS/LAPACK, FFTW, ScaLAPACK
- **iccifort**: icc, ifort
- **iimpi**: icc, ifort, IntelMPI
- **intel**: icc, ifort, IntelMPI, IntelMKL
Currently Loaded Modules:
   1) snlcevironment (S)  2) systemdefault (S)

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

b-an01 [~]$ ml list

Currently Loaded Modules:
   1) snlcevironment (S)  2) systemdefault (S)

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

b-an01 [~]$
<table>
<thead>
<tr>
<th>Module</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allinea/6.1.1</td>
<td></td>
</tr>
<tr>
<td>Autoconf/2.69</td>
<td></td>
</tr>
<tr>
<td>Autotools/20150215</td>
<td></td>
</tr>
<tr>
<td>CMake/3.5.2</td>
<td></td>
</tr>
<tr>
<td>EasyBuild/2.9.0</td>
<td></td>
</tr>
<tr>
<td>GnuPLM/2.4.2</td>
<td></td>
</tr>
<tr>
<td>GCC/4.9.3-binnuts-2.25</td>
<td></td>
</tr>
<tr>
<td>GCC/4.0-2.26</td>
<td></td>
</tr>
<tr>
<td>GCC/6.2.0-2.27</td>
<td>(D)</td>
</tr>
<tr>
<td>GCCcore/4.9.3</td>
<td></td>
</tr>
<tr>
<td>GCCcore/5.4.0</td>
<td>(D)</td>
</tr>
<tr>
<td>GNU/4.9.3-2.25</td>
<td></td>
</tr>
<tr>
<td>Intel/1.4.17</td>
<td></td>
</tr>
<tr>
<td>PGI/16.5-GCC-5.4.0-2.26</td>
<td></td>
</tr>
<tr>
<td>PGI/16.7-GCC-5.4.0-2.26</td>
<td>(D)</td>
</tr>
<tr>
<td>Toss/2016b</td>
<td></td>
</tr>
<tr>
<td>Toss/2016.09</td>
<td>(D)</td>
</tr>
<tr>
<td>gcccuda/2016.16.0</td>
<td></td>
</tr>
<tr>
<td>gettext/8.17.9</td>
<td></td>
</tr>
<tr>
<td>gomp/2016b</td>
<td></td>
</tr>
<tr>
<td>gomp/2016.16.0</td>
<td>(D)</td>
</tr>
<tr>
<td>gomp/2016.16.0</td>
<td></td>
</tr>
<tr>
<td>gcc/2015.3.187-GNU-4.9.3-2.25</td>
<td></td>
</tr>
<tr>
<td>gcc/2016.1.150-GCC-4.9.3-2.25</td>
<td></td>
</tr>
<tr>
<td>gcc/2016.3.210-GCC-5.4.0-2.26</td>
<td></td>
</tr>
<tr>
<td>gcc/2017.0.898-GCC-5.4.0-2.26</td>
<td></td>
</tr>
<tr>
<td>tccfort/2016.3.210-GCC-5.4.0-2.26</td>
<td></td>
</tr>
<tr>
<td>tccfort/2017.0.898-GCC-5.4.0-2.26</td>
<td></td>
</tr>
<tr>
<td>tccfort/2018.4.0-2.26</td>
<td></td>
</tr>
<tr>
<td>lmod/6.5</td>
<td>seth3d/6.5</td>
</tr>
</tbody>
</table>

Where:
- **S**: Module is sticky, requires --force to unload or purge
- **L**: Module is loaded
- **D**: Default Module

Use "module spider" to find all possible modules.
Use "module keyword keys key2 ..." to search for all possible modules matching any of the keys.
The following is a list of the modules currently available:

**Autoconf:** Autoconf/2.69
Autoconf is an extensible package of M4 macros that produce shell scripts to
amatically configure software source code packages. These scripts can adapt the
packages to many kinds of UNIX-like systems without manual user intervention.
Autoconf creates a configuration script for a package from a template file that lists
the operating system features that the package can use, in the form of M4 macro
calls. - Homepage: http://www.gnu.org/software/autoconf/

**Automake:** Automake/1.15
Automake: GNU Standards-compliant Makefile generator - Homepage:
http://www.gnu.org/software/automake/automake.html

**AutoTools:** AutoTools/20150815
This bundle collect the standard GNU build tools: Autoconf, Automake and libtool
- Homepage: http://autoconf.io

**Boost:** Boost/1.61.0
Boost provides free peer-reviewed portable C++ source libraries. - Homepage:
http://www.boost.org/

**CMake:** CMake/3.5.2
CMake, the cross-platform, open-source build system. CMake is a family of tools
designed to build, test and package software. - Homepage: http://www.cmake.org

**CUDA:** CUDA/8.0.44
CUDA (formerly Compute Unified Device Architecture) is a parallel computing platform
and programming model created by NVIDIA and implemented by the graphics processing
units (GPUs) that they produce. CUDA gives developers access to the virtual
instruction set and memory of the parallel computational elements in CUDA GPUs.

**EasyBuild:** EasyBuild/2.9.0
EasyBuild is a software build and installation framework written in Python that
allows you to install software in a structured, repeatable and robust way.
- Homepage: http://hpccompute.github.com/easybuild/

**FFTW:** FFTW/3.3.4, FFTW/3.3.5
FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in
one or more dimensions, of arbitrary input size, and of both real and complex data.
- Homepage: http://www.fftw.org

**GCC:** GCC/4.9.3-binutils-2.25, GCC/5.4.0-2.26, GCC/6.2.0-2.27
GCC is a software build and installation framework written in Python that
allows you to install software in a structured, repeatable and robust way.
- Homepage: http://www.gcc.gnu.org/
Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".

```
ml splicer lcc
```

```
Description:
C and C++ compiler from Intel - Homepage:

Versions:
  tcc/2015.3.187-GNU-4.9.3-2.25
tcc/2016.1.158-GCC-4.9.3-2.25
tcc/2016.3.210-GCC-5.4.0-2.26
  tccfort/2016.1.158-GCC-4.9.3-2.25
tccfort/2016.3.210-GCC-5.4.0-2.26
  tccfort/2017.0.098-GCC-5.4.0-2.26 (D)
tccfortcuda/2016.10.0

Other possible modules matches:
tccfort tccfortcuda

To find other possible module matches do:
module -r spider "*tcc.*"
```

For detailed information about a specific "tcc" module (including how to load the modules) use the module's full name.

For example:
```
$ module spider tcc/2017.0.098-GCC-5.4.0-2.26
```
```
Example - getting started
Connecting from Linux or macOS

- Open a terminal window
- Type (change username to the right value)
  
  ```
  ssh -X username@kebnekaise.hpc2n.umu.se
  ```
- Enter your password.
- If you need to reset the password, go here:
  
  https://www.hpc2n.umu.se/forms/user/suprauth?action=pwreset
Connecting from a Windows System - PuTTY

- Go to http://www.chiark.greenend.org.uk/~sgtatham/putty/.
- Follow the link there to Download PuTTY.
- Get the Zip file with both PuTTY, PSCP, and PSFTP. Unzip, run putty.exe
Connecting from a Windows System - PuTTY
Connecting from a Windows System - PuTTY

- Logging on:
Getting a job to run - simple example

- cd /pfs/nobackup/$HOME
- nano hello-mpi.c

```
#include <stdio.h>
#include <mpi.h>

int main (int argc, char *argv[]) {

    int myrank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    printf("Processor %d of %d: Hello World!\n", myrank, size);

    MPI_Finalize();
}
```

- To save and exit: Ctrl-x
Getting a job to run - simple example

- Load the needed modules. Here just the GCC compilers and MPI libraries (using OpenMPI for this):
  - ml gompi
- Let us see that we have what we expect:
  - ml
- Compile the C/MPI program:
  - mpicc mpi_hello.c -o mpi_hello
Getting a job to run - simple example. Job script

#!/bin/bash
# Change to your own project id!
#SBATCH -A SNIC2016-1-450
#SBATCH -n 14
#SBATCH --time=00:10:00

module purge
module add gompi

mpirun ./mpi_hello

• Submitting: sbatch < jobscript >
Job status

- Check the job’s status:
  - `scontrol show job <job id>`
- Show a list of your jobs:
  - `squeue -u <username>`
- Delete job:
  - `scancel <job id>`
Job output

- Output and errors in:
  - slurm-<job id>.out
- Look at it with vi, nano, emacs, cat, less...
- To get output and error files split up, you can give these flags in the submit script:
  - #SBATCH --error=job.%J.err
  - #SBATCH --output=job.%J.out
- To run on the 'fat' nodes, add this flag to your script:
  - #SBATCH -p largemem (Kebnekaise)
  - #SBATCH -p bigmem (Abisko)
Job output - example

b-an01 [~/pfs/slurm]$ cat slurm-15952.out

The following modules were not unloaded:
(Use "module --force purge" to unload all):

  1) systemdefault   2) snicenvironment
Processor 12 of 14: Hello World!
Processor 5 of 14: Hello World!
Processor 9 of 14: Hello World!
Processor 4 of 14: Hello World!
Processor 11 of 14: Hello World!
Processor 13 of 14: Hello World!
Processor 0 of 14: Hello World!
Processor 1 of 14: Hello World!
Processor 2 of 14: Hello World!
Processor 3 of 14: Hello World!
Processor 6 of 14: Hello World!
Processor 7 of 14: Hello World!
Processor 8 of 14: Hello World!
Processor 10 of 14: Hello World!
Requesting GPU nodes

• Currently there is no separate queue for the GPU nodes
• You request them by adding the following to your batch script:
  ◆ #SBATCH --gres=gpu:k80:x where x=1, 2, 4
• x = the number of K80 cards, each with 2 GPU engines
• There are 32 nodes with dual K80 cards and 4 nodes with quad K80 cards
Compiling and linking with libraries
Compiling and linking with libraries

- MPI C program
  - Intel, Intel MPI:
    ```
    ml iimpi
    mpicc <program> -o <outfile.name>
    ```
  - GCC, OpenMPI:
    ```
    ml gompi
    mpicc <program> -o <outfile.name>
    ```
- OpenMP Fortran program
  - Intel:
    ```
    ml iccifort
    ifort -qopenmp <program> -o <outfile.name>
    ```
  - GCC:
    ```
    ml GCC
    gfortran -fopenmp <program> -o <outfile.name>
    ```
Compiling and linking with libraries - continued

- C program, BLAS, LAPACK
  - Intel, MKL:
    ml intel
    -L${MKLROOT}/lib/intel64 -lmkl_intel_ilp64 -lmkl_sequential -lmkl_core -lpthread -lm -ldl
  - GCC, OpenBLAS/LAPACK:
    ml foss
    gcc -o program program.o -lopenblas
Compiling and linking with libraries - continued

- Fortran program, ScaLAPACK, OpenMPI:
  - GCC, OpenBLAS/LAPACK, ScaLAPACK, OpenMPI:
    ```
    ml foss
    gcc -o program program.o -lscalapack -lopenblas
    ```
  - Intel, MKL, Intel MPI:
    ```
    ml intel
    -L${MKLROOT}/lib/intel64 -lscalapack_ilp64 -lmkl_intel_ilp64 -lscalapack -lmkl -lpthread -ldl
    ```

- C program, OpenMPI, CUDA
  - GCC:
    ```
    ml goolfc
    -lcuda -lcudart or nvcc program.cu -o program
    ```
Linking with Intel

- Very useful for linking with Intel and Intel MKL:
Questions?

- http://www.hpc2n.umu.se/
- support@hpc2n.umu.se