

Introduction to HPC2N

Birgitte Brydsø, Jerry Eriksson, and Pedro Ojeda-May

HPC2N, Umeå University

12 September 2019



- Our systems - Kebnekaise (and Abisko)
- Using our systems
- The File System
- The Module System
 - Overview
 - Compiler Tool Chains
 - Examples
- Compiling/linking with libraries
- The Batch System (SLURM)
 - Overview
 - Simple example

Kebnekaise and Abisko

Abisko



- 1 328 nodes / 15744 cores (10 fat, 318 thin)
- 2 Thin: 4 AMD Opteron 6238, 12 core 2.6 GHz proc.
- 3 Fat: 4 AMD Opteron 6344, 12 core 2.6 GHz proc.
- 4 10 with 512 GB RAM/node, 318 with 128 GB RAM/node
- 5 Interconnect: Mellanox 4X QSFP 40 Gb/s
- 6 Theoretical performance: 163.74 TF
- 7 HP Linpack: 131.9 TF
- 8 Date installed: Fall 2011. Upgraded Jan 2014. **Retired: ??**

Kebnekaise and Abisko

Kebnekaise



- 1 602 nodes / 19288 cores (of which 2448 are KNL)
 - 432 Intel Xeon E5-2690v4, 2x14 cores, 128 GB/node
 - 52 Intel Xeon Gold 6132, 2x14 cores, 192 GB/node
 - 20 Intel Xeon E7-8860v4, 4x18 cores, 3072 GB/node
 - 32 Intel Xeon E5-2690v4, 2x NVidia K80, 2x14, 2x4992, 128 GB/node
 - 4 Intel Xeon E5-2690v4, 4x NVidia K80, 2x14, 4x4992, 128 GB/node
 - 10 Intel Xeon Gold 6132, 2x NVidia V100, 2x14, 2x5120, 192 GB/node
 - 36 Intel Xeon Phi 7250, 68 cores, 192 GB/node, 16 GB MCDRAM/node
- 2 501760 CUDA “cores” (80*4992 cores/K80+20*5120 cores/V100)
- 3 More than 136 TB memory
- 4 Interconnect: Mellanox FDR / EDR Infiniband
- 5 Theoretical performance: 728 TF (+ expansion)
- 6 Date installed: Fall 2016 / Spring 2017 / Spring 2018

Using our systems

- 1 Get an account (<https://www.hpc2n.umu.se/documentation/access-and-accounts/users>)
- 2 Connect to:
`kebnekaise.hpc2n.umu.se`
or
~~`abisko.hpc2n.umu.se`~~
- 3 Transfer your files and data (optionally)
- 4 Compile own code, install software, or run pre-installed software
- 5 Create batch script, submit batch job
- 6 Download data/results

Using our systems

Connecting to HPC2N's systems

- **Linux, OS X:**

- `ssh username@kebnekaise.hpc2n.umu.se`
- Use `ssh -Y` if you want to open graphical displays.

- **Windows:**

- Get an SSH client (PuTTY, Cygwin, MobaXterm ...)
- Get an X11 server if you need graphical displays (Xming ...)
- Start the client and login to

`kebnekaise.hpc2n.umu.se`

- More information here:

<https://www.hpc2n.umu.se/documentation/guides/windows-connection>

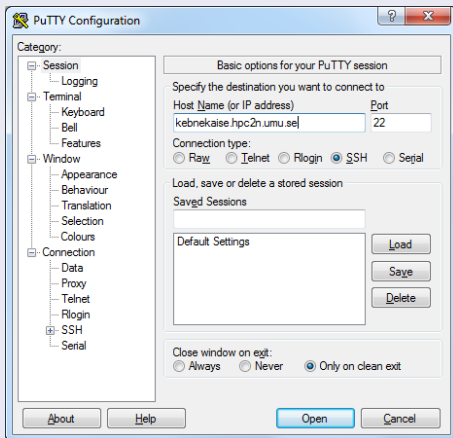
- **Mac/OSX:** Guide here:

<https://www.hpc2n.umu.se/documentation/guides/mac-connection>

Using our systems

Connecting from a Windows System with PuTTY

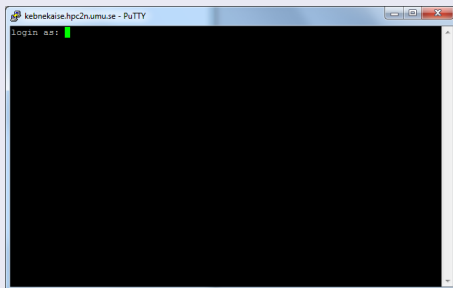
Get the Zip file (<http://www.putty.org/>) with both PuTTY, PSCP, and PSFTP. Unzip, run putty.exe



Using our systems

Connecting from a Windows System with PuTTY

Enter your username and then your password.



Using our systems

Transfer your files and data

- **Linux, OS X:**

- Use scp for file transfer:

```
local> scp username@kebnekaise.hpc2n.umu.se:file .  
local> scp file username@kebnekaise.hpc2n.umu.se:file
```

- **Windows:**

- Download client: WinSCP, FileZilla (sftp), PSCP/PSFTP, ...
- Transfer with sftp or scp

- <https://www.hpc2n.umu.se/documentation/filesystems/filetransfer>

- **Mac/OSX:**

- Transfer with sftp or scp (as for Linux) using Terminal
- Or download client: Cyberduck, Fetch, ...

- More info in guides (see previous slide) and here:

<https://www.hpc2n.umu.se/documentation/filesystems/filetransfer>

Editing your files

- Various editors: vi, vim, nano, emacs ...
- Example, nano:
 - `nano <filename>`
 - Save and exit nano: `Ctrl-x`
- Example, Emacs:
 - Start with: `emacs`
 - Open (or create) file: `Ctrl-x Ctrl-f`
 - Save: `Ctrl-x Ctrl-s`
 - Exit Emacs: `Ctrl-x Ctrl-c`
 - (If you want to run in an a separate emacs window, and with full functionality, you need to login with `ssh -Y` or similar, for X11 forwarding):

The File System

There are 2 file systems

More info here: <http://www.hpc2n.umu.se/filesystems/overview>

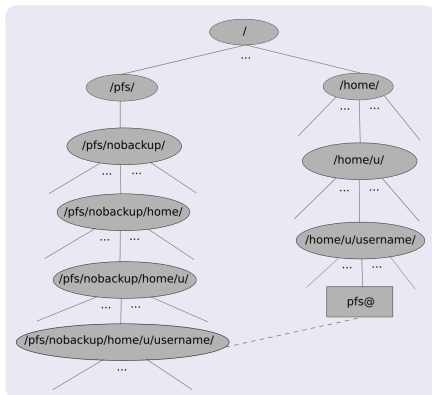
• AFS

- This is where your home directory is located (cd \$HOME)
- Regularly backed up
- NOT accessible by the batch system (except the folder

Public with the right settings)

• PFS

- Parallel File System
- NO BACKUP
- Accessible by the batch system



The File System

AFS

- Your home directory is located in `/home/u/username` and can also be accessed with the environment variable `$HOME`
- It is located on the AFS (Andrew File System) file system
- **Important!** The batch system cannot access AFS since ticket-forwarding to batch jobs do not work
- AFS does secure authentication using Kerberos tickets

The File System

PFS

- The 'parallel' file system, where your 'parallel' home directory is located in `/pfs/nobackup/home/u/username` (`/pfs/nobackup/$HOME`)
- Offers high performance when accessed from the nodes
- The correct place to run all your batch jobs
- NOT backed up, so you should not leave files there that cannot easily be recreated
- For easier access, create a symbolic link from your home on AFS to your home on PFS:

```
ln -s /pfs/nobackup/$HOME $HOME/pfs
```

You can now access your pfs with `cd pfs` from your home directory on AFS

The Module System (Lmod)

Most programs are accessed by first loading them as a 'module'

Modules are

- used to set up your environment (paths to executables, libraries, etc.) for using a particular (set of) software package(s)
- a tool to help users manage their Unix/Linux shell environment, allowing groups of related environment-variable settings to be made or removed dynamically
- allows having multiple versions of a program or package available by just loading the proper module
- are installed in a hierarchial layout. This means that some modules are only available after loading a specific compiler and/or MPI version.

The Module System (Lmod)

Useful commands (Lmod)

- See which modules exists:
`ml spider`
- Modules depending only on what is currently loaded:
`module avail` or `ml av`
- See which modules are currently loaded:
`module list` or `ml`
- Example: loading a compiler toolchain, here for GCC:
`module load foss/version` or `ml foss/version`
- Example: Unload the above module:
`module unload foss` or `ml -foss`
- More information about a module:
`ml show <module>`
- Unload all modules except the 'sticky' modules:
`ml purge`

The Module System

Compiler Toolchains

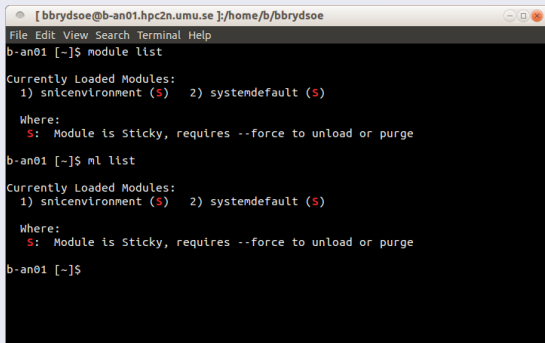
Compiler toolchains load bundles of software making up a complete environment for compiling/using a specific prebuilt software. Includes some/all of: compiler suite, MPI, BLAS, LAPACK, ScaLapack, FFTW, CUDA.

- Some currently available toolchains (check `m1 av` for versions and full, updated list):
 - **GCC**: GCC only
 - **gcccuda**: GCC and CUDA
 - **foss**: GCC, OpenMPI, OpenBLAS/LAPACK, FFTW, ScaLAPACK
 - **gimkl**: GCC, IntelMPI, IntelMKL
 - **gimpi**: GCC, IntelMPI
 - **gomp**: GCC, OpenMPI
 - **gompic**: GCC, OpenMPI, CUDA
 - **goolfc**: gompic, OpenBLAS/LAPACK, FFTW, ScaLAPACK
 - **icc**: Intel C and C++ only
 - **iccifort**: icc, ifort
 - **iccifortcuda**: icc, ifort, CUDA
 - **ifort**: Intel Fortran compiler only
 - **iimpi**: icc, ifort, IntelMPI
 - **intel**: icc, ifort, IntelMPI, IntelMKL
 - **intelcuda**: intel and CUDA
 - **iomkl**: icc, ifort, Intel MKL, OpenMPI
 - **pomkl**: PGI C, C++, and Fortran compilers, IntelMPI
 - **pompi**: PGI C, C++, and Fortran compilers, OpenMPI

The Module System

Examples, listing loaded modules

```
module list
ml list
ml
```



```
[bbrydsoe@b-an01.hpc2n.umu.se]:/home/b/bbrydsoe
File Edit View Search Terminal Help
b-an01 [-]$ module list

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

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

b-an01 [-]$ ml list

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

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

b-an01 [-]$
```


The Module System

Examples, listing all modules

```
module spider  
ml spider
```

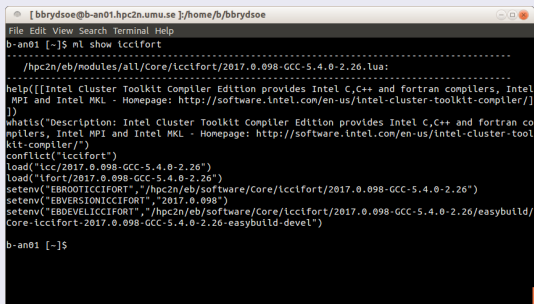
```
File Edit View Search Terminal Help  
.....  
The following is a list of the modules currently available:  
.....  
ABINIT: ABINIT/9.0-0-Python-2.7.13, ABINIT/9.4-2-Python-2.7.13, ABINIT/9.4-2  
ABINIT is a package whose main program allows one to find the total energy, charge density and electronic structure of systems made of electrons and nuclei (molecules and periodic solids) within Density Functional Theory (DFT), using pseudopotentials and a plane-wave or wavelet basis.  
  
ACTIC: ACTIC/1.1  
ACTIC converts Independent Triangles into Triangle Strips or Fans.  
  
ADDA: ADDA/1.3b4  
ADDA is an open-source parallel implementation of the discrete dipole approximation, capable to simulate light scattering by particles of arbitrary shape and composition in a wide range of particle sizes.  
  
ANTLR: ANTLR/2.7.7  
ANTLR, Another Tool For Language Recognition, (formerly PCCTS) is a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing Java, C#, C++, or Python actions.  
  
ASE: ASE/3.9.1-Python-2.7.12-rogul, ASE/3.9.1-Python-2.7.12, ASE/3.9.1-Python-2.7.13, ASE/3.15.0-Python-2.7.14, ASE/3.16.2-Python-3.6.4, ASE/3.17.0-Python-2.7.15, ASE/3.17.0-Python-3.7.2  
ASE is a python package providing an open source Atomic Simulation Environment in the Python scripting language.  
  
ATK: ATK/2.22.0, ATK/2.27.1, ATK/2.28.1, ATK/2.32.0  
ATK provides the set of accessibility interfaces that are implemented by other toolkits and applications. Using the ATK interfaces, accessibility tools have full access to view and control running applications.  
  
AUGUSTUS: AUGUSTUS/3.3.2-Python-2.7.15  
AUGUSTUS is a program that predicts genes in eukaryotic genomic sequences  
  
AdapterRemoval: AdapterRemoval/2.2.2  
AdapterRemoval searches for and removes remnant adapter sequences from High-Throughput Sequencing (HTS) data and (optionally) trims low quality bases from the 3' end of reads following adapter removal.  
  
Advisor: Advisor/2018_update1  
Vectorization Optimization and Thread Prototyping - Vectorize & thread code or performance "dies" - Easy workflow + data + tips = Faster code Faster - Prioritize, Prototype & Predict performance gain  
  
Amber: Amber/16-AmberTools-16-patchlevel-20-7-hpc2b, Amber/16-AmberTools-17-patchlevel-8-12-tilt_ras-100, Amber/16-AmberTools-17-patchlevel-8-12, Amber/16-AmberTools-17-patchlevel-10-15, ...  
Amber (Originally Assisted Model Building with Energy Refinement) is software for performing molecular dynamics and structure prediction.  
  
Anaconda2: Anaconda2/4.3.1  
Built to complement the rich, open source Python community, the Anaconda platform provides an enterprise-ready data analytics platform that empowers companies to adopt a modern open data science analytics architecture.  
  
Anaconda3: Anaconda3/4.3.1, Anaconda3/4.4.0  
Built to complement the rich, open source Python community, the Anaconda platform provides an enterprise-ready data analytics platform that empowers companies to adopt a modern open data science analytics architecture.  
  
Autocconf: Autocconf/2.09  
Autocconf is an extensible package of M4 macros that produce shell scripts to automatically configure software source code packages. These scripts can adapt the packages to many kinds of UNIX-like systems without manual user intervention. Autocconf 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.  
  
Autonake: Autonake/1.15, Autonake/1.15.1, Autonake/1.16.1  
Autonake: GNU Standards-compliant Makefile generator  
  
Autotools: Autotools/20150215, Autotools/20170616, Autotools/20180311  
This bundle collect the standard GNU build tools: Autocconf, Autonake and libtool  
lines 1-59
```

The Module System

Examples, show more info about a module

```
module show <module>
```

```
ml show <module>
```



```
[bbrydsoe@b-an01.hpc2n.umu.se ~]:/home/b/brydsoe
File Edit View Search Terminal Help
b-an01 [-]$ ml show iccifort
-----
/hpc2n/eb/modules/all/Core/iccifort/2017.0.098-GCC-5.4.0-2.26.lua:
-----
help([[Intel Cluster Toolkit compiler Edition provides Intel C,C++ and fortran compilers, Intel
MPI and Intel MKL - Homepage: http://software.intel.com/en-us/intel-cluster-toolkit-compiler/
]])
whats("Description: Intel Cluster Toolkit Compiler Edition provides Intel C,C++ and fortran co
mpilers, Intel MPI and Intel MKL - Homepage: http://software.intel.com/en-us/intel-cluster-toolkit-compiler/
")
conflict("iccifort")
load("icc/2017.0.098-GCC-5.4.0-2.26")
load("ifort/2017.0.098-GCC-5.4.0-2.26")
setenv("EBROOTICCIFORT", "/hpc2n/eb/software/Core/iccifort/2017.0.098-GCC-5.4.0-2.26")
setenv("EBVERSIONICCIFORT", "2017.0.098")
setenv("EBDEVELICCIFORT", "/hpc2n/eb/software/Core/iccifort/2017.0.098-GCC-5.4.0-2.26/easybuild/
Core-iccifort-2017.0.098-GCC-5.4.0-2.26-easybuild-devel")
b-an01 [-]$
```

The Module System

Examples, loading and unloading modules

```
module load <module>/<version> and module unload <module>/<version>  
ml <module>/<version> and ml -<module>/<version>
```

In general, you should always load the specific version of a module

```
File Edit View Search Terminal Help  
b-an01 [-]$ ml  
Currently Loaded Modules:  
  1) snicenvironment (S)  2) systemdefault (S)  
  
Where:  
  S: Module is Sticky, requires --force to unload or purge  
  
b-an01 [-]$ ml iccifortcuda/2019a  
b-an01 [-]$ ml  
Currently Loaded Modules:  
  1) snicenvironment (S)  3) GCCcore/8.2.0      5) icc/2019.1.144-GCC-8.2.0-2.31.1  7) CUDA/10.1.105  
  2) systemdefault (S)  4) binutils/2.31.1  6) ifort/2019.1.144-GCC-8.2.0-2.31.1  8) iccifortcuda/2019a  
  
Where:  
  S: Module is Sticky, requires --force to unload or purge  
  
b-an01 [-]$ ml unload iccifortcuda/2019a  
b-an01 [-]$ ml  
Currently Loaded Modules:  
  1) snicenvironment (S)  2) systemdefault (S)  
  
Where:  
  S: Module is Sticky, requires --force to unload or purge  
  
b-an01 [-]$
```

Compiling and Linking with Libraries

MPI and OpenMP - load the desired version found with `ml spider`. Some examples. There are other toolchains that will work.

- MPI C program:
 - Intel compilers, Intel MPI:
`ml iimpi/version`
`mpicc <program.c> -o <outfile>`
 - GCC compilers, OpenMPI:
`ml gOMPI/version`
`mpicc <program.c> -o <outfile>`
- OpenMP Fortran program:
 - Intel compilers:
`ml iccifort/version`
`ifort -qopenmp <program.f90> -o <outfile>`
 - GCC compilers:
`ml GCC/version`
`gfortran -fopenmp <program.f90> -o <outfile>`

Compiling and Linking with Libraries

Linking

Figuring out how to link

- Intel and Intel MKL linking:

<https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor>

- GCC, etc. **Use buildenv**

- After loading a compiler toolchain, load 'buildenv' and use 'ml show buildenv' to get useful linking info
- Example, foss (add relevant version):

```
ml foss/version
ml buildenv
ml show buildenv
```

- Using the environment variable (prefaced with \$) for linking is highly recommended!

Compiling and Linking with Libraries

Example: ml foss, ml buildenv, ml show buildenv

```
File Edit View Search Terminal Help
setenv("CXX","g++")
setenv("CXXFLAGS","-O2 -ftree-vectorize -march=native -fno-math-errno")
setenv("F77","gfortran")
setenv("F90","gfortran")
setenv("F90FLAGS","-O2 -ftree-vectorize -march=native -fno-math-errno")
setenv("FC","gfortran")
setenv("FCFLAGS","-O2 -ftree-vectorize -march=native -fno-math-errno")
setenv("FFLAGS","-O2 -ftree-vectorize -march=native -fno-math-errno")
setenv("FFTW_INC_DIR","/hpc2n/eb/software/MPI/GCC-CUDA/8.2.0-2.31.1-10.1.105/OpenMPI/3.1.3/FFTW/3.3.8/include")
setenv("FFTW_LIB_DIR","/hpc2n/eb/software/MPI/GCC-CUDA/8.2.0-2.31.1-10.1.105/OpenMPI/3.1.3/FFTW/3.3.8/lib")
setenv("FFTW_STATIC_LIBS","libfftw3.a")
setenv("FFTW_STATIC_LIBS_MT","libfftw3.a,libpthread.a")
setenv("FFT_INC_DIR","/hpc2n/eb/software/MPI/GCC-CUDA/8.2.0-2.31.1-10.1.105/OpenMPI/3.1.3/FFTW/3.3.8/include")
setenv("FFT_LIB_DIR","/hpc2n/eb/software/MPI/GCC-CUDA/8.2.0-2.31.1-10.1.105/OpenMPI/3.1.3/FFTW/3.3.8/lib")
setenv("FFT_STATIC_LIBS","libfftw3.a")
setenv("FFT_STATIC_LIBS_MT","libfftw3.a,libpthread.a")
setenv("FLIBS","-lgfortran")
setenv("LAPACK_INC_DIR","/hpc2n/eb/software/Compiler/GCC/8.2.0-2.31.1/OpenBLAS/0.3.5/include")
setenv("LAPACK_LIB_DIR","/hpc2n/eb/software/Compiler/GCC/8.2.0-2.31.1/OpenBLAS/0.3.5/lib")
setenv("LAPACK_MT_STATIC_LIBS","libopenblas.a,libgfortran.a")
setenv("LAPACK_STATIC_LIBS","libopenblas.a,libgfortran.a")
setenv("LDFLAGS","-L/hpc2n/eb/software/Compiler/GCC/8.2.0-2.31.1/CUDA/10.1.105/lib64 -L/hpc2n/eb/software/Core/GCCcore/8.2.0/lib64 -L/hpc2n/eb/software/Compiler/GCC/8.2.0-2.31.1/OpenBLAS/0.3.5/lib -L/hpc2n/eb/software/MPI/GCC-CUDA/8.2.0-2.31.1-10.1.105/OpenMPI/3.1.3/ScalAPACK/2.0.2-OpenBLAS-0.3.5/OpenMPI/3.1.3/FFTW/3.3.8/lib")
setenv("LIBBLAS","-lopenblas -lgfortran")
setenv("LIBBLAS_MT","-lopenblas -lgfortran")
setenv("LIBFFT","-lfftw3")
setenv("LIBFFT_MT","-lfftw3 -lpthread")
setenv("LIBLAPACK","-lopenblas -lgfortran")
setenv("LIBLAPACK_MT","-lopenblas -lgfortran")
setenv("LIBLAPACK_MT_ONLY","-lopenblas -lgfortran")
setenv("LIBLAPACK_ONLY","-lopenblas -lgfortran")
setenv("LIBS","-lm -lrt -lcudart -lpthread")
setenv("LIBSCALAPACK","-lscalapack -lopenblas -lgfortran")
setenv("LIBSCALAPACK_MT","-lscalapack -lopenblas -lpthread -lgfortran")
setenv("LIBSCALAPACK_MT_ONLY","-lscalapack -lgfortran")
setenv("LIBSCALAPACK_ONLY","-lscalapack -lgfortran")
setenv("MPICC","mpicc")
setenv("MPICXX","mpicxx")
setenv("MPIF77","mpifort")
setenv("MPIF90","mpifort")
setenv("MPIFC","mpifort")
```


The Batch System (SLURM)

- Large/long/parallel jobs **must** be run through the batch system
- SLURM is an Open Source job scheduler, which provides three key functions
 - Keeps track of available system resources
 - Enforces local system resource usage and job scheduling policies
 - Manages a job queue, distributing work across resources according to policies
- Same batch system on Abisko and Kebnekaise. The differences are that there are GPUs and KNLs which can be allocated on Kebnekaise
- Guides and documentation at:
<http://www.hpc2n.umu.se/support>

The Batch System

Accounting, Compute nodes, Abisko

- Physically, a socket is 12 cores, but for SLURM allocation purposes a socket is 6 cores (a NUMA node)
- Thus allocation is in groups of 6 cores (one NUMA island). This also means 6 cores is the smallest unit you can allocate.
- This is how your project is charged, depending on how many cores you ask for:

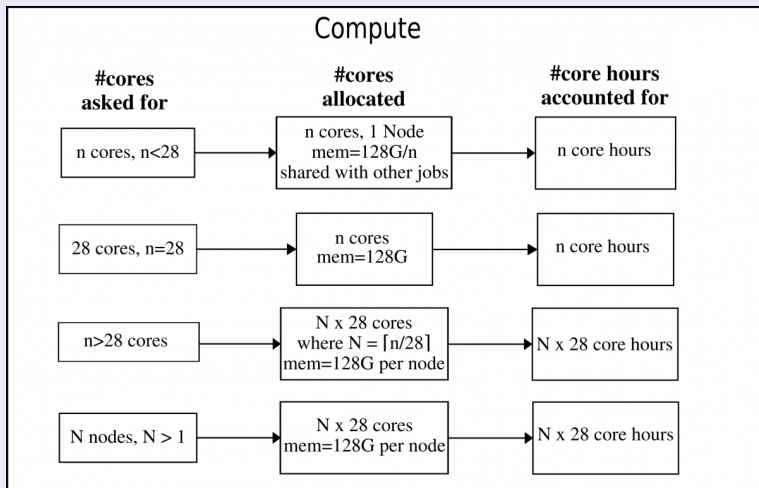
You ask for	Cores you get	Project is charged
1 core	6 cores	6 cores
6 cores	6 cores	6 cores
7 cores	12 cores	12 cores
c cores	$\text{ceil}(c/6)$ cores	$\text{ceil}(c/6)$ cores

If you request resources using **#SBATCH -c** you request c cores per task, and SLURM only allocates cores on a single node.

If you request resources using **#SBATCH -n** you request tasks which can be allocated on multiple nodes.

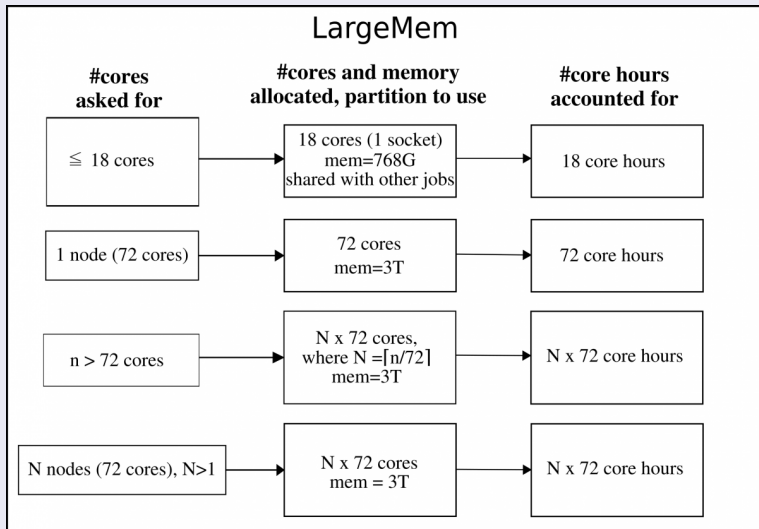
The Batch System

Accounting, Compute nodes, Kebnekaise



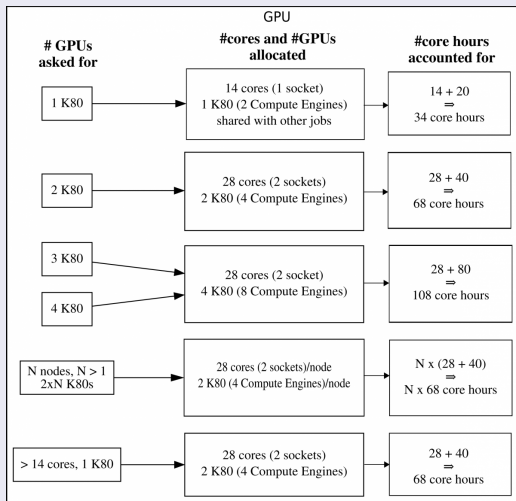
The Batch System

Accounting, largemem nodes, Kebnekaise



The Batch System

Accounting, GPU nodes, Kebnekaise. Same for the V100 as for the K80.



Note: V100s accounts like K80s and have **one** engine per card.

The Batch System (SLURM)

Useful Commands

- Submit job: `sbatch <jobscript>`
- Get list of your jobs: `squeue -u <username>`
- `srun <commands for your job/program>`
- `salloc <commands to the batch system>`
- Check on a specific job: `scontrol show job <job id>`
- Delete a specific job: `scancel <job id>`
- Info about jobs:

```
sacct -l -j <jobid> -o jobname,NTasks,nodelist,MaxRSS,MaxVMSize...
```

- More flags can be found with `man sacct`
- The output will be **very** wide. Use something like
`sacct -l -j | less -S`
to view (makes it sideways scrollable, using the left/right arrow key)

Use `man sbatch`, `man srun`, `man` for more information

The Batch System (SLURM)

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`

The Batch System (SLURM)

Using different parts of Abisko/Kebnekaise

- To run on the 'fat' nodes, add this flag to your script:
#SBATCH -p largemem (Kebnekaise - largemem does not have general access)
#SBATCH -p bigmem (Abisko)
- Specifying Intel Broadwell or Skylake CPUs only (Kebnekaise):
#SBATCH --constraint=broadwell
or
#SBATCH --constraint=skylake
- Using the GPU nodes (Kebnekaise)
#SBATCH --gres=gpu:<type-of-card>:x where
<type-of-card> is either k80 or v100 and x = 1, 2, or 4 (4 only for the K80 type).

More on

https://www.hpc2n.umu.se/documentation/guides/using_kebnekaise

The Batch System (SLURM)

Simple example, serial

Example: Serial job on Kebnekaise, compiler toolchain 'foss'

```
#!/bin/bash
# Project id - change to your own after the course!
#SBATCH -A SNIC2019-5-82
# Asking for 1 core
#SBATCH -n 1
# Asking for a walltime of 5 min
#SBATCH --time=00:05:00

# Purge modules before loading new ones in a script.
ml purge
ml foss/2019a

./my_serial_program
```

Submit with:

```
sbatch <jobscript>
```

The Batch System (SLURM)

Example, MPI C program

```
#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();
```

The Batch System (SLURM)

Simple example, parallel

Example: MPI job on Kebnekaise, compiler toolchain 'foss'

```
#!/bin/bash
#SBATCH -A SNIC2019-5-82
#SBATCH -n 14
#SBATCH --time=00:05:00
##SBATCH --exclusive
#SBATCH --reservation=intro-cpu

module purge
ml foss/2019a

srun ./my_parallel_program
```

The Batch System (SLURM)

Simple example, output

Example: Output from a MPI job on Kebnekaise, run on 14 cores (one NUMA island)

```
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!
```

The Batch System (SLURM)

Longer example

```
#!/bin/bash
#SBATCH -A SNIC2019-5-82
#SBATCH -n 14
#SBATCH --time=00:05:00

module purge
ml foss/2019a

echo "Running on hosts:  $SLURM_NODELIST"
echo "Running on $SLURM_NNODES nodes."
echo "Running on $SLURM_NPROCS processors."
echo "Current working directory is 'pwd'"

echo "Output of srun hostname:"
srun /bin/hostname

srun ./mpi_hello
```

The Batch System (SLURM)

Starting more than one serial job in the same submit file

```
#!/bin/bash
#SBATCH -A SNIC2019-5-82
#SBATCH -n 5
#SBATCH --time=00:15:00

module purge
ml foss/2018b

srun -n 1 ./job1.batch &
srun -n 1 ./job2.batch &
srun -n 1 ./job3.batch &
srun -n 1 ./job4.batch &
srun -n 1 ./job5.batch
```

The Batch System (SLURM)

Multiple Parallel Jobs Sequentially

```
#!/bin/bash
#SBATCH -A SNIC2019-5-82
#SBATCH -n 14
# Remember to ask for enough time for all jobs to complete
#SBATCH --time=02:00:00

module purge
ml foss/2018b

# Here 14 tasks with 2 cores per task. Output to file.
# Not needed if your job creates output in a file
# I also copy the output somewhere else and then run
# another executable...

srun -n 14 -c 2 ./a.out > myoutput1 2>&1
cp myoutput1 /pfs/nobackup/home/u/username/mydatadir
srun -n 14 -c 2 ./b.out > myoutput2 2>&1
cp myoutput2 /pfs/nobackup/home/u/username/mydatadir
srun -n 14 -c 2 ./c.out > myoutput3 2>&1
cp myoutput3 /pfs/nobackup/home/u/username/mydatadir
...
```

The Batch System (SLURM)

Multiple Parallel Jobs Simultaneously

Make sure you ask for enough cores that all jobs can run at the same time, and have enough memory. Of course, this will also work for serial jobs - just remove the `srun` from the command line.

```
#!/bin/bash
#SBATCH -A SNIC2019-5-82
# Total number of cores the jobs need
#SBATCH -n 56
# Remember to ask for enough time for all of the jobs to
# complete, even the longest
#SBATCH --time=02:00:00

module purge
ml foss/2018b

srun -n 14 --cpu_bind=cores ./a.out &
srun -n 28 --cpu_bind=cores ./b.out &
srun -n 14 --cpu_bind=cores ./c.out &
...
wait
```