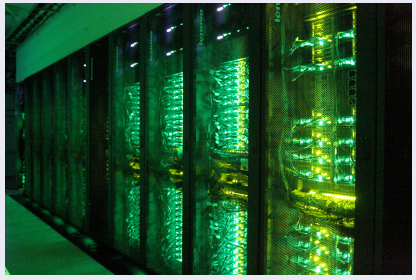
Introduction to HPC2N

Birgitte Brydsø

HPC2N, Umeå University

5 November 2019

Kebnekaise



1. 544 nodes / 17552 cores (of which 2448 are KNL)
   - 432 Intel Xeon E5-2690v4, 2x14 cores, 128G/node
   - 20 Intel Xeon E7-8860v4, 4x18 cores, 3072GB/node
   - 32 Intel Xeon E5-2690v4, 2x NVidia K80, 2x14, 2x4992, 128GB/node
   - 4 Intel Xeon E5-2690v4, 4x NVidia K80, 2x14, 4x4992, 128GB/node
   - 36 Intel Xeon Phi 7250, 68 cores, 192GB/node, 16GB MCDRAM/node
2. 399360 CUDA "cores" (80 * 4992 cores/K80)
3. More than 125 TB memory total
4. Interconnect: Mellanox 56 Gb/s FDR Infiniband
5. Theoretical performance: 728 TF
6. HP Linpack: 629 TF
7. Date installed: Fall 2016 / Spring 2017

Using Kebnekaise

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 SSH client (MobaXterm, PuTTY, Cygwin ...)
  - Get X11 server if you need graphical displays (Xming, ...)
  - Start the client and login with your HPC2N username 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 Kebnekaise

Transfer your files and data

- **Linux, OS X:**
  - Use scp (or sftp) for file transfer. Example, scp:

```
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
- **Mac/OSX:**
  - Transfer with sftp or scp (as for Linux) using Terminal
  - Or download client: Cyberduck, Fetch, ...
- More information in guides (see previous slide) and here:

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

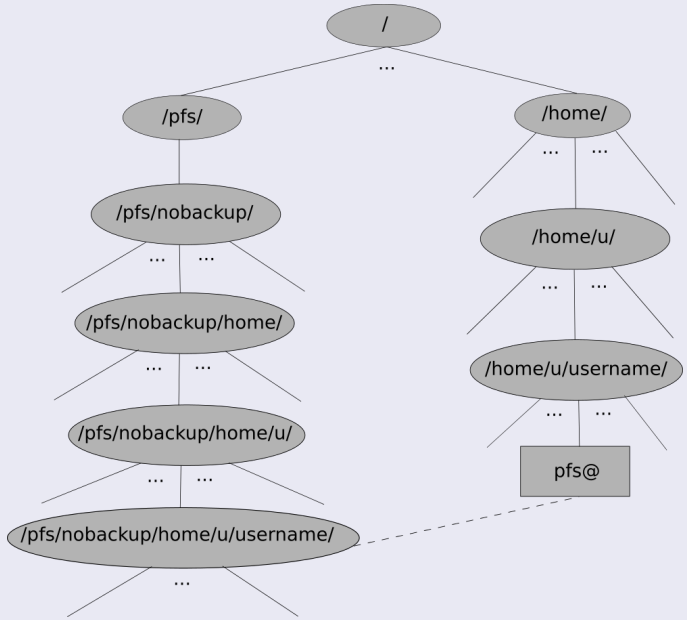
Editing your files

- Various editors: vi, vim, nano, emacs ...
- Example, vi/vim:
  - `vi <filename>`
  - Insert before: i
  - Save and exit vi/vim: `Esc :wq`
- 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

The File System

- **AFS**
  - Your home directory is here ($HOME)
  - Regularly backed up
  - NOT accessible by the batch system (ticket-forwarding doesn't work)
  - secure authentification with Kerberos tickets
- **PFS**
  - Parallel File System
  - NO BACKUP
  - High performance when accessed from the nodes
  - Accessible by the batch system
  - Create symbolic link from $HOME to pfs:

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



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
- 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)

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

- See which modules exists:
  `module spider` or `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 and version, here for GCC, OpenMPI, OpenBLAS/LAPACK, FFTW, ScaLAPACK and CUDA:
  `module load fosscuda/2019a` or `ml fosscuda/2019a`
- Example: Unload the above module:
  `module unload fosscuda/2019a` or `ml -fosscuda/2019a`
- More information about a module:
  `module show <module>` or `ml show <module>`
- Unload all modules except the 'sticky' modules:
  `module purge` or `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 of the currently available toolchains (check `ml av` for all/versions):
  - **GCC**: GCC only
  - **gcccuda**: GCC and CUDA
  - **foss**: GCC, OpenMPI, OpenBLAS/LAPACK, FFTW, ScaLAPACK
  - **fosscuda**: GCC, OpenMPI, OpenBLAS/LAPACK, FFTW, ScaLAPACK, and CUDA
  - **gimkl**: GCC, IntelMPI, IntelMKL
  - **gimpi**: GCC, IntelMPI
  - **gompi**: 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

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`

- Buildenv
  - After loading a compiler toolchain, load `'buildenv'` and use `'ml show buildenv'` to get useful linking info
  - Example, fosscuda, version 2019a:

    ```
    ml fosscuda/2019a
    ml buildenv
    ml show buildenv
    ```

  - Using the environment variable (prefaced with $) is highly recommended!
  - You have to load the buildenv module in order to be able to use the environment variables for linking!

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
- In order to run a batch job, you need to create and submit a SLURM submit file (also called a batch submit file, a batch script, or a job script).
- Guides and documentation at: http://www.hpc2n.umu.se/support

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>`
- Useful info about job: `sacct -l -j <jobid> | less -S`

The Batch System (SLURM)

Job Output

- Output and errors in:
  `slurm-<job-id>.out`
- 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 specify Broadwell or Skylake only:
  `#SBATCH --constraint=broadwell` or
  `#SBATCH --constraint=skylake`
- To run on the GPU nodes, add this to your script:
  `#SBATCH --gres=gpu:<card>:x`
  where <card> is k80 or v100, x = 1, 2, or 4 (4 only if K80).
- http://www.hpc2n.umu.se/resources/hardware/kebnekaise

The Batch System (SLURM)

Simple example, serial

Example: Serial job, compiler toolchain 'fosscuda/2019a'

```
#!/bin/bash
# Project id - change to your own after the course!
#SBATCH -A SNIC2019-5-142
# Asking for 1 core
#SBATCH -n 1
# Asking for a walltime of 5 min
#SBATCH --time=00:05:00

# Always purge modules before loading new in a script.
ml purge > /dev/null 2>&1
ml fosscuda/2019a

./my_serial_program
```

Submit with:

`sbatch <jobscript>`

The Batch System (SLURM)

parallel example

```
#!/bin/bash
#SBATCH -A SNIC2019-5-142
#SBATCH -n 14
#SBATCH --time=00:05:00

ml purge < /dev/null 2>&1
ml fosscuda/2019a

srun ./my_mpi_program
```

The Batch System (SLURM)

Requesting GPU nodes

Currently there is no separate queue for the GPU nodes

- Request GPU nodes by adding this to your batch script:

  ```
  #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)
- There are 32 nodes (broadwell) with dual K80 cards and 4 nodes with quad K80 cards
- There are 10 nodes (skylake) with dual V100 cards

The Batch System (SLURM)

Example: Asking for a GPU

```
#!/bin/bash
#SBATCH -A SNIC2019-5-142
#SBATCH --time=00:10:00
# Asking for one V100 card
#SBATCH --gres=gpu:v100:1

# Load any modules you need
...

./my_program
```

Various useful info

- A project has been set up for the workshop: `SNIC2019-5-142`
- You use it in your batch submit file by adding:

  ```
  #SBATCH -A SNIC2019-5-142
  ```

- There is a reservation for 2 V100 GPU nodes. This reservation is accessed by adding this to your batch submit file:

  ```
  #SBATCH --reservation=intro-gpu
  ```

- The reservation is ONLY valid for the duration of the course.