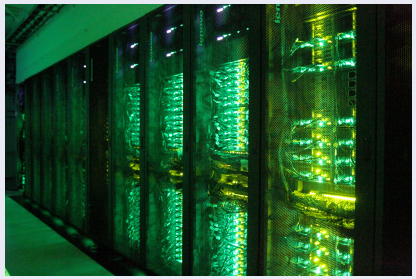
Heterogeneous computing with performance modelling - Introduction to HPC2N

Birgitte Brydsø, Mirko Myllykoski, Pedro Ojeda-May

HPC2N, Umeå University

4-5 November 2020

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
   - 36 Intel Xeon Phi 7250, 68 cores, 192GB/node, 16GB MCDRAM/node
2. 501760 CUDA "cores" (80*4992 cores/K80+20*5120 cores/V100)
3. More than 136 TB memory total
4. Interconnect: Mellanox FDR / EDR Infiniband
5. Theoretical performance: 984 TF
6. HP Linpack: 791 TF

Using our systems

Connecting to HPC2N's systems - ThinLinc

ThinLinc is a cross-platform remote desktop server developed by Cendio AB. It is especially useful when you need to use software with a graphical interface.

- Download the client from `https://www.cendio.com/thinlinc/download`. Install it.
- Start the client. Enter the name of the server: kebnekaise-tl.hpc2n.umu.se. Enter your username.
- Go to "Options" $->$ "Security". Check that authentication method is set to password.
- Go to "Options" $->$ "Screen". Uncheck "Full screen mode".
- Enter your HPC2N password. Click "Connect"
- Click "Continue" when you are being told that the server's host key is not in the registry. Wait for the ThinLinc desktop to open.

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

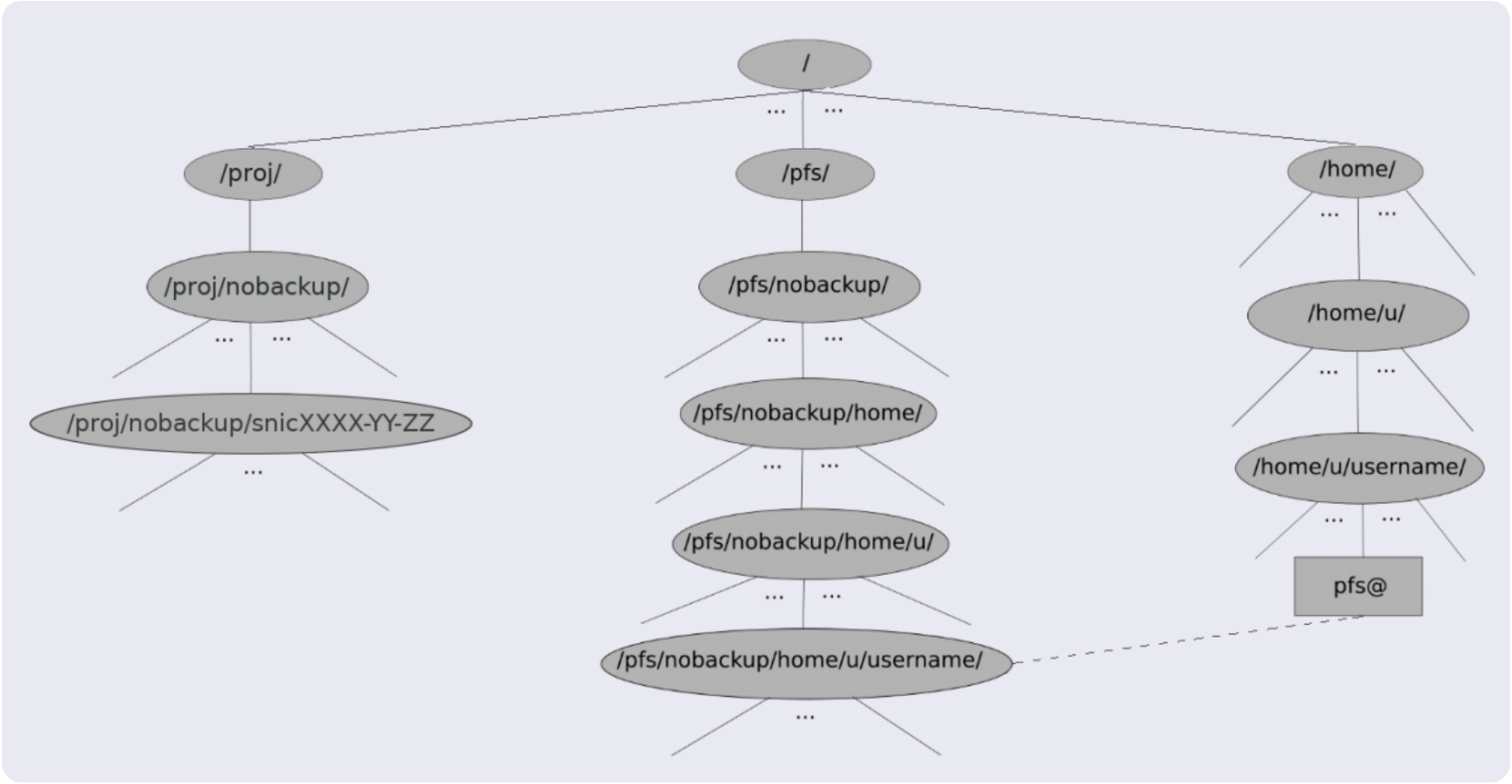
- Various editors: vi, vim, nano, emacs ...
- Example, vi/vim:
  - `vi <filename>`
  - Insert before: i
  - Save and exit vi/vim: `Esc :wq`
- Example, nano:
  - `nano <filename>`
  - Save and exit nano: `Ctrl-x`
- Example, Emacs:
  - Start with: emacs
  - Open (or create) file: Ctrl-x Ctrl-f
  - Save: Ctrl-x Ctrl-s
  - Exit Emacs: Ctrl-x Ctrl-c

The File System

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

	AFS (~/)	AFS (~/Public)	Project storage	PFS	/scratch
Good for batch jobs	No		Yes	Yes	
Backed up	Yes	Yes	No	No	No
Accessible by batch system	No	Yes, readable with right permissions	Yes	Yes	Yes (node only)
High performance	No	No	Yes	Yes	Medium
Default readability	Owner	World readable	Group only	Everyone on cluster	Owner
Permissions	chmod, chgrp, ACL	Cannot be changed	chmod, chgrp, ACL	chmod, chgrp, ACL	chmod, chgrp, ACL
Notes	Your home-directory is on AFS	Your home-directory is on AFS	This gets allocated through storage projects		Per node

The File 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 authentification using Kerberos tickets

The File System

PFS, project storage

- The 'parallel' file system, located in /pfs/nobackup/home/u/username (/pfs/nobackup/$HOME)
- As well, project storage is located on pfs, under /proj/nobackup/<your-project-storage>
- 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 during this course, 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
- 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/2019b` or `ml fosscuda/2019b`
- Example: Unload the above module:
  `module unload fosscuda/2019b` or `ml -fosscuda/2019b`
- 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 2019b:

    ```
    ml fosscuda/2019b
    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!

Compiling and Linking with Libraries

Compiling with nvcc, example using fosscuda/2019b

- Load any modules (once per session):
  `ml fosscuda/2019b buildenv`
- Compile with nvcc. The flag `-o` allowes you to name the output file
  `nvcc -o hello hello.cu`
- Remember to link with any libraries. If you loaded `buildenv` you can use the environment variables. Example with OpenBLAS:
  `nvcc -o hello hello.cu ${LIBBLAS}`

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/2019b'

```
#!/bin/bash
# Project id - change to your own after the course!
#SBATCH -A SNIC2020-9-16
# 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/2019b

./my_serial_program
```

Submit with:

`sbatch <jobscript>`

The Batch System (SLURM)

parallel example

```
#!/bin/bash
#SBATCH -A SNIC2020-9-16
#SBATCH -n 14
#SBATCH --time=00:05:00

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

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

NOTE: Instead of using a batch script, you can give the commands on the command line.

```
#!/bin/bash
#SBATCH -A SNIC2020-9-16
#SBATCH --time=00:05:00
# Asking for one V100 card
#SBATCH --gres=gpu:v100:1
#SBATCH --reservation=snic2020-9-161-day1

# Load any modules you need
ml fosscuda/2019b buildenv

./my_program
```

Various useful info

- A project has been set up for the workshop: `SNIC2020-9-16`
- You use it in your batch submit file by adding:

  ```
  #SBATCH -A SNIC2020-9-16
  ```

- There is a reservation for 3 V100 GPU nodes. This reservation is accessed by adding this to your batch submit file:

  WEDNESDAY
  ```
  #SBATCH --reservation=snic2020-9-161-day1
  ```

  THURSDAY
  ```
  #SBATCH --reservation=snic2020-9-161-day2
  ```

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