Introduction to HPC2N

Birgitte Brydsø

HPC2N, Umeå University

20 November 2017

Overview

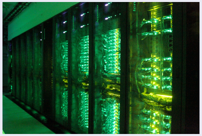
Kebnekaise and Abisko

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

Kebnekaise and Abisko

Kebnekaise



1. 544 nodes / 17552 cores (of which 2448 are KNL)
   - 432 Intel Xeon E5-2690v4, 2x14 cores, 128 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
   - 36 Intel Xeon Phi 7250, 68 cores, 192 GB/node, 16 GB MCDRAM/node
2. 399360 CUDA "cores" (80 * 4992 cores/K80)
3. More than 125 TB memory
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 and Abisko

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 Kebnekaise and Abisko

Connecting to HPC2N's systems

- **Linux, OS X:**
  - `ssh username@kebnekaise.hpc2n.umu.se`
    or
    `ssh username@abisko.hpc2n.umu.se`
  - Use `ssh -X ....` 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 to

    `kebnekaise.hpc2n.umu.se`
    or
    `abisko.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 and Abisko

Transfer your files and data

- **Linux, OS X:**
  - Use scp for file transfer:

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

or

```
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

Using Kebnekaise and Abisko

Editors

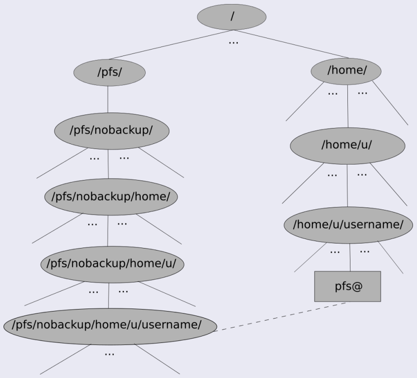
Editing your files

- 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

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 accesseable 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 authentification 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'

- 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 and version, here for GCC:
  `module load foss/2017b` or `ml foss/2017b`
- Example: Unload the above module:
  `module unload foss/2017b` or `ml -foss/2017b`
- 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.

- Currently available toolchains (check `ml av` for versions):
  - **GCC**: GCC only
  - **gcccuda**: GCC and CUDA
  - **foss**: GCC, OpenMPI, OpenBLAS/LAPACK, FFTW, ScaLAPACK
  - **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:

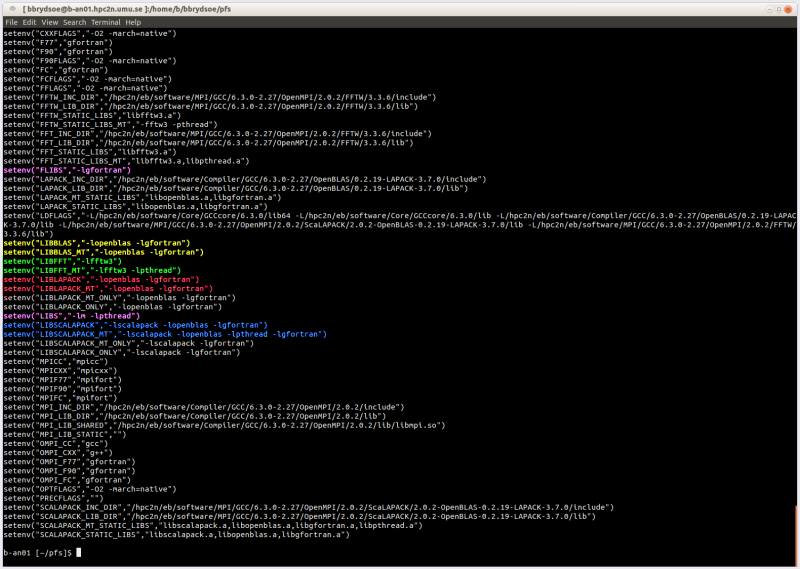
  `https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor`

- Buildenv
  - After loading a compiler toolchain, load `'buildenv'` and use `'ml show buildenv'` to get useful linking info
  - Example, foss, version 2017b:

    ```
    ml foss/2017b
    ml buildenv
    ml show buildenv
    ```
  - Using the environment variable (prefaced with $) is highly recommended!

Compiling and Linking with Libraries

Example: ml foss, ml buildenv, ml show buildenv


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

```
setenv("CXXFLAGS","-O2 -march=native")
setenv("F77","gfortran")
setenv("F90","gfortran")
setenv("F90FLAGS","-O2 -march=native")
setenv("FC","gfortran")
setenv("FCFLAGS","-O2 -march=native")
setenv("FFLAGS","-O2 -march=native")
setenv("FFTW_INC_DIR","/hpc2n/eb/software/MPI/GCC/6.3.0-2.27/OpenMPI/2.0.2/FFTW/3.3.6/include")
setenv("FFTW_LIB_DIR","/hpc2n/eb/software/MPI/GCC/6.3.0-2.27/OpenMPI/2.0.2/FFTW/3.3.6/lib")
setenv("FFTW_STATIC_LIBS","libfftw3.a")
setenv("FFTW_STATIC_LIBS_MT","-fftw3 -pthread")
setenv("FFT_INC_DIR","/hpc2n/eb/software/MPI/GCC/6.3.0-2.27/OpenMPI/2.0.2/FFTW/3.3.6/include")
setenv("FFT_LIB_DIR","/hpc2n/eb/software/MPI/GCC/6.3.0-2.27/OpenMPI/2.0.2/FFTW/3.3.6/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/6.3.0-2.27/OpenBLAS/0.2.19-LAPACK-3.7.0/include")
setenv("LAPACK_LIB_DIR","/hpc2n/eb/software/Compiler/GCC/6.3.0-2.27/OpenBLAS/0.2.19-LAPACK-3.7.0/lib")
setenv("LAPACK_MT_STATIC_LIBS","libopenblas.a,libgfortran.a")
setenv("LAPACK_STATIC_LIBS","libopenblas.a,libgfortran.a")
setenv("LDFLAGS","-L/hpc2n/eb/software/Core/GCCcore/6.3.0/lib64 -L/hpc2n/eb/software/Core/GCCcore/6.3.0/lib -L/hpc2n/eb/software/Compiler/GCC/6.3.0-2.27/OpenBLAS/0.2.19-LAPAC
K-3.7.0/lib -L/hpc2n/eb/software/MPI/GCC/6.3.0-2.27/OpenMPI/2.0.2/ScaLAPACK/2.0.2-OpenBLAS-0.2.19-LAPACK-3.7.0/lib -L/hpc2n/eb/software/MPI/GCC/6.3.0-2.27/OpenMPI/2.0.2/FFTW/
3.3.6/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 -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")
setenv("MPI_INC_DIR","/hpc2n/eb/software/Compiler/GCC/6.3.0-2.27/OpenMPI/2.0.2/include")
setenv("MPI_LIB_DIR","/hpc2n/eb/software/Compiler/GCC/6.3.0-2.27/OpenMPI/2.0.2/lib")
setenv("MPI_LIB_SHARED","/hpc2n/eb/software/Compiler/GCC/6.3.0-2.27/OpenMPI/2.0.2/lib/libmpi.so")
setenv("MPI_LIB_STATIC","")
setenv("OMPI_CC","gcc")
setenv("OMPI_CXX","g++")
setenv("OMPI_F77","gfortran")
setenv("OMPI_F90","gfortran")
setenv("OMPI_FC","gfortran")
setenv("OPTFLAGS","-O2 -march=native")
setenv("PRECFLAGS","")
setenv("SCALAPACK_INC_DIR","/hpc2n/eb/software/MPI/GCC/6.3.0-2.27/OpenMPI/2.0.2/ScaLAPACK/2.0.2-OpenBLAS-0.2.19-LAPACK-3.7.0/include")
setenv("SCALAPACK_LIB_DIR","/hpc2n/eb/software/MPI/GCC/6.3.0-2.27/OpenMPI/2.0.2/ScaLAPACK/2.0.2-OpenBLAS-0.2.19-LAPACK-3.7.0/lib")
setenv("SCALAPACK_MT_STATIC_LIBS","libscalapack.a,libopenblas.a,libgfortran.a,libpthread.a")
setenv("SCALAPACK_STATIC_LIBS","libscalapack.a,libopenblas.a,libgfortran.a")

b-an01 [~/pfs]$
```

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 (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>`

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
  ```
- 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)

The Batch System (SLURM)

Simple example, serial

Example: Serial job, compiler toolchain 'foss'

```
#!/bin/bash
# Project id - change to your own after the course!
#SBATCH -A SNIC2017-3-98
# 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
ml foss/2017b

./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, compiler toolchain 'foss'

```
#!/bin/bash
# Project to run in - change to own later
#SBATCH -A SNIC2017-3-98
# Asking for 14 cores
#SBATCH -n 14
# Asking for 5 minutes walltime
#SBATCH --time=00:05:00
##SBATCH --exclusive

ml purge
ml foss/2017b

mpirun ./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)

Requesting GPU nodes

Currently there is no separate queue for the GPU nodes

- You request GPU nodes 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

Note: This is only valid on Kebnekaise. Abisko has no GPUs.

The Batch System (SLURM)

Longer example

```
#!/bin/bash
#SBATCH -A SNIC2017-3-98
#SBATCH -n 14
#SBATCH --time=00:05:00

ml purge
ml foss/2017b

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 mpirun hostname:"
mpirun /bin/hostname

mpirun ./mpi_hello
```