HPC introduction at MPI CBG, Jan 24th, 2018

This etherpad is meant to help you take notes collaboratively. The course does not yield a cheat sheet of any sort.

Peter: world peace
Benedict: Reconstruct partially coherent imaging data from the TESCAN Q-PHASE from the microscopy facility to get back the 3D refractive index distribution. It belongs to a collaboration with the HYMAN Group and Patrick McCall. We need more memory!
Lena: To get some tips and tricks for best practise; to get some ideas how I can maybe use it in combination with R ;-)

Tobias (Deep Learning - Training and Application of CNNs)

Markus: Submitting NGS alignments etc to the cluster
Daniela: how to connect DNN to HPC?

Marcus: I want to combine data from multiple protein databases. I have some code that works on single entries, but now want to spread this to "all" entries.

Christopher: automated and parallel image processing

Oscar: just helping out :)

Anmol: To get an idea what best practices can I use while using HPC.

Notes:

Start connecting to the cluster-computer
=> ssh <user>@falcon #replace <user> with your CBG account
enter password
logging out:
=> logout/exit

cluster (called furiosa): three parts
- compute nodes
- login nodes (job submission) - called falcon
- export nodes (just data transfer) - called mack

(watch mad max for deeper understanding)

do not connect to cluster with Wifi hotspot

HPC: system always visible under the same IP address; has its own operating system
Cloud: where you can request a virtual machine; IP address changes
transferring files from the cluster: scp USER@falcon:FILE
transferring files to the cluster: scp FILE USER@falcon:FILE
transferring folders: scp -r USER@falcon:PATH_TO_FILE
ssh tunnel: if you want to get a file from a server via accessing another server

commands:

w   list all users log into the machine
ls -F   give more information on folders, executables, etc.
cd -   to get back to the path where you have been before
cp -v -r   the -v command tells you what it actually does

echo:
simly prints out a string on the command-line

This is for managing qued jobs - provided by the HPC
slurm:

- srun COMMAND   willl be execute on the cluster
- srun hostname   give the name of the node#
- sbatch < SCRIPT will execute the bash file on the cluster; returns number of batch job and creates a file (slurm-JOBID.out) in the current folder with information on the submission (i.e. data and node); to change the name of the output file specify the name my using the argument -o
- sbatch -o OUTPUT_FILE -e ERROR_OUTPUT < SCRIPT   with -e it's possible to store error messages in a separate file (i.e. myout.err)

when checking the uptime, the slum-JOBID.out: gives information on the load average (i.e. 21.68, 21.98, 22.01). The information refer to the acutal minute, 5 minutes ago and 15 minutes ago and can be used to check the node load

history|grep interactive   check the wiki for more information

squeue -u USERNAME   all jobs listed for the user (ST: R for running, PD for waiting list; last two columns: number of requested nodes, actual node)

scancel JOBID   to cancel a submitted job
scancel -u USERNAME   to cancel all jobs submitted by the user

wall time: time from submitting the job to the when it was finished; the expected wall time can be set via the parameter --time in sbatch, i.e. --time=00:01:00 for 1 minute

if you need GPUs you need to specify this with the submission. The same applies for the expected usage of memory (i.e. --mem). You can specify all parameters also in the bash file, i.e.:

```
#!/bin/bash
#SBATCH -o test.log
#SBATCH -t 00:10:00
#SBATCH -n 1 number of nodes
#SBATCH -c 12 number of cores
```

If you additionally specifiy these parameter as arguments of sbatch in the command line the arguments in the batch script will basically be overwritten

```
sbatch --mem=10000 --partition=gpu --gres=gpu:0 calc_pi.sh
```

queue_summary   to get information on the current 'resources' of the cluster

check the time: when running a script use the time command to check how much time it took to run the script, i.e.: time python3 SCRIPT

/proc/cpuinfo exist on each linux machine and gives information on the CPUs

The shared parallel filesystem (called lustre) of the cluster nodes means that each of the nodes has /tmp/ folder for instance, as it is local. This is something which we need to keep in mind when defining our output file. To store our output we need to address a project space on lustre, i.e. /projects/hpcsupport/course/

bash command to check the capacity of your computer/node: df -h

echo $? after running a command will tell you whether the previous command worked properly (0) or whether an error occurred (2)

check your code to find positions/hotspots which you can use to parallize:
    add @profile above the function you would like to test (good practice: start with the main function) and run the following commands:
    kernprof -l ./PYTHONSCRIPT   run the script (-l indicates line-wise); returns a file named PYTHONSCRIPT.lprof
    python3 -m line_profiler ./PYTHONSCRIPT.lprof   run this command to get the script profile, i.e. gives information on the main consumer, i.e. line

Amdahl's law to test which parallized implementation is the fastest

Copying stuff from the local machine to the HPC do the following:
    scp local.file user@falcon:local.file

list behavior of rm command

- rm -r folder1 folder2 folder3

list behavior of cp command

- cp file1 file2 file3 file4 <source>
- if <source> = . (dot) it means current directory

history

- command shows the bash history

c**d tricks**

- cd . : *current directory*
- cd .. : *one up*
- cd   : *go to home*
-

ls can also list recursively-- use -r

- ls -r
- ls -r -F (shows classifiers / for folders * for executable files)

- ls -r -F <file_path>

Now the discusion begins on cluster::

- **srun echo "Hello World"** this job will be submitted and run on cluster
- **echo "Hello World"** will run on falcon
- To see what's happening
- type **srun hostname**
- type **hostname**
-

difference between sbatch and and srun:

- sbatch takes .sh
-

- **Writing a batch script**
  - type **nano**
  - write **#!/bin/bash**
  - write your commands
  -

- sbatch -o myoutput.log < multiple_commands.sh
- means any output that this creates please put it into myoutput.log
-

- sbatch -o myoutput.log -e myoutput.err < multiple_commands.sh
- transfer all the error messages in myoutput.err
-

- output of bash file which has hostname and uptime as commands
  - c02n02
  - 11:39:53 up 124 days, 21:42, 0 users, load average: 20.39, 20.63, 21.07
- wall time: time to complete the job
-

- sbatch --mem=10000 calc_pi.sh
-

- finding the hotspots::
- start **profiling** with **main and then move on to next functions after seeing the results of lineprofiler**
  -

- amdal's law helps in getting the estimate for speedup what one can get after parallelizing the code
-

```
# Data Transfer
# scp - secure copy
# For documentation use
man scp
scp <filetotransfer> <address>:<filetotransferNewName>
```

```
# or
scp <filetotransfer> <address>:
# if you want to transfer in the other direction just reverse the command like so:
scp <address>:<filentotransferNewName> <filetotransfer>

# go to tmp
cd /tmp/
ls this_weeks_canteen_menus/
# Copy multiple files
scp -r falcon1:/tmp/this_weeks_canteen_menus .
-r # Recursively copy entire directories
. # copy to this location

# Absolut path starts with / as in the root directory
# if one uses falcon1:<filetotransfer> one refers to a relativ path in this case the home
directory

# Tips and Tricks:
# go from home directory
~/<fileordirectory>
# go from current directory
./<fileordirectory
```

General comment: be aware of the distinction between absolute paths and relative paths (abbreviations: "~" for home directory, "." for current directory)

```
# On login node falcon1
w # gives the active sessions
# HPC cluster is always a shared resource! So mind the other users.
```

# ls - list directory contents

```
ls --color # for color coded file/diretories
# has different flags for more information:
-l  # permissions; owner; group; size; last time of access; name
-h # gives you human readable file sizes
```

# pwd - path to working directory

```
# shows you where you currently are
```

# cd - change directory

```
cd .. # one directory above
cd - # last visited directory
```

# cp - copying

```
# everything but the last argument of cp will be taken as a source
cp -r -v <sourcefile> <sourcefile> <targetlocation>
cp -r -v /tmp/this_weeks_canteen_menus/ some_stuff
-r # Recursively copy entire directories
-v # verbose explain what is being done
```

# rm - remove

# Use carefully!! If it is gone it is gone!
-r # remove directories and their contents recursively

# mv - move
# move or rename file

# Job scheduler
# Since the cluster is a shared resource there must be a system to distribute the resources to the different users.
# This is done by the job scheduler which is in this case here SLURM (https://slurm.schedmd.com/)

# srun - Run parallel jobs
# srun echo "Hello World"
# To see it in action run:
hostname # gives the head node
srun hostname # gives the hostname of the node this command was executed

# Batch script execution on the cluster
# use editor nano

```
#!/bin/bash
hostname
date
```

save as multiple_commands.sh
# run as
sbatch < multiple_commands.sh
< # explicit multiple commands execution
sbatch multiple_commands.sh
# output is slurm-<jobid>.out
slurm-9776889.out

# open text file
less
cat

# if you want a nicer log file name do:
sbatch -o myoutput.log < multiple_commands.sh

Node stats:

c02n02
up 124 days, 21:39

c02n02: 11:38:22 up 124 days, 21:40

c01n14
11:37:41 up 166 days, 19:33

```
c01n14
 11:37:32 up 166 days, 19:33,  0 users,  load average: 21.69, 21.99, 22.01
```

c01n14 - 166 days

```
c02n02
 11:39:53 up 124 days, 21:42,  0 users,  load average: 20.39, 20.63, 21.07
```

```
c02n02
 11:41:12 up 124 days, 21:43,  0 users,  load average: 21,09, 20,78, 21,09
```

Questions

Why does the computer department run the canteen??
main: course entre-bad-code steak
side: french open source fries
Bad idea!

How to check the available computing resources on the nodes, like Python version, available packages etc.?
python --version
installed software is in /sw/apps/

What do those numbers mean in load average? Is there any useful information that one can get from it about the job?
the information of uptime is global, i.e. it refers to the health of the entire node.
the 3 numbers at the end refer to the 1-/5-/15-minute average of the node load

Is there any shortcut that if I write nano <filename>.sh then automatically it writes **#!/bin/bash** on the first line**?**

feedback of the morning session
(red)
- more slurm examples
- bash part was not so new to me
- everything fine so far
- maybe explain before more theoretical what a script does, then introduce the script
- what do load averages mean
- don't understand user privileges
- more explanation of node concepts

(green)
- well prepared
- connect to remote machine
- send job and get back data
- easy to follow
- well explained

- everything is fine, +1
- scp command,
- cd tricks,
- writing a batch script using nano
- learned etherpad !
- cp, +1
- srun, sbatch
- slurm simpler than expected

Monitoring jobs
after starting a batch i.e. with sbatch the ongoing job can be watched using:
  squeue -u USER
  Displays: JobID, Partition, Name, User, Time, #of Nodes, NodeID

# Killing a job
scancel <jobid>
scancel

# Walltime
# the actual time a job needs.
# If you send a job it gets a default amount of time the job can take before it gets killed by the system.
# IMPORTANT: since it is a shared resource there is no "infinite" runtime for you jobs (except you specify it).
# Depending on the resources the job needs the job scheduler distributes resources.
# This means the less resources and/or time a job needs the faster one gets resources.
# Thus, if you specify a walltime, the queuing system might put your processes ahead!
--time=00:00:00

# How to check the cluster load
queue_summary

# Exercise
download script with get for example:
https://psteinb.github.io/hpc-in-a-day/downloads/calc_pi.py

measure runtime with time utility.
Estimate runtime for your calculation > calc_pi.py scales linearly

```
#!/bin/bash
#SBATCH -o test.log
#SBATCH -t 00:10:00
#SBATCH -mem=10000
#SBATCH --partition=<gpu, long, bigmem, batch>
#SBATCH

python3 ./calc_pi.py 500000000
```

# Notes about the Filesystem
node_info.sh

```
#!/bin/bash

# specifies a filename
FILENAME=/tmp/${HOSTNAME}_info.log

# cat puts the contents of the file into the file with the specified filename
cat /proc/cpuinfo > ${FILENAME}

sbatch -o test.log -t 00:00:10 node_info.sh

# test.log has no content
# but also no .log file
# What happened?

# The problem is the shared filesystem
# HPC needs a shared and high performance filesystem
# here lustre is used
# the .log file is created and saved on a filesytem that is local to the node
# Thus one needs to save in a different diretory:

#!/bin/bash
FILENAME=/projects/hpcsupport/course/${HOSTNAME}_${USER}_info.log

echo /proc/cpuinfo > ${FILENAME}
```

Ratio remote/local space
2.1PB = 2100TB = 2100000 GB
2.150x

```
# Lustre is optimized for parallel I/O so it is fast for writing an saving files
# IMPORTANT: NO BACKUPS
# filesystem has a problem with directories with many many tiny files
# just the shear number of files can be a problem
# lustre is optimized for a smaller number of larger files
# BAD >10000 files
```

# Exercise serial_pi.py

```
# Calculate pi based on the Monte Carlo Method
import numpy as np
import argparse
import sys

# pseudo random number generator
np.random.seed(2018)

# function inside_circle
def inside_circle(total_count):

    # gets x and y coordinates of a random point
    x = np.float32(np.random.uniform(size=total_count))
    y = np.float32(np.random.uniform(size=total_count))

    # calculates radius of that point
    radii = np.sqrt(x * x + y * y)

    # filters for points with radius smaller than 1
    filtered = np.where(radii <= 1.0)

    # gets counts from filtered radii
    count = len(radii[filtered])

    # returns counts
    return count

# function estimate_pi
def estimate_pi(total_count):

    #
    count = inside_circle(total_count)
    return (4 * count / total_count)


def main():

    # uses argparse
    # required for parsing the commandline
    parser = argparse.ArgumentParser(description="Estimate Pi using a MC method")
    parser.add_argument('n_samples', metavar='N', type=int, nargs=1, default=10000,
                help='number of samples to draw')

    args = parser.parse_args()
    n_samples = args.n_samples[0]

    # carries out the calculation
    my_pi = estimate_pi(n_samples)
    sizeof = np.dtype(np.float32).itemsize

    # output for the user
    print("[serial version] memory consumption %.3f MB" % (n_samples*sizeof*3/
(1024*1024)) )
    print("[serial version] pi is %f from %i samples" % (my_pi, n_samples))

    # exits the program with exit status 0
    sys.exit(0)

if __name__ == '__main__':

    main()
```

# Exit status

```
# get the exit status type:
echo $?
```

ls / && echo "it worked"

# Parallelization - Finding the hotspot
# First we need to find out which operation of our code profits most from parallelization
# Thus some idea about the resource usage would be nice
# For this we use the tool kernprof
# wrappes around the code
# creates a profile for the code

# Install lineprofiler
# Don't do this normally!!
pip3 install --user line_profiler

# now it exists in ~/.local/bin/
# but how do we call it? >> Modify the environment
export PATH=$HOME/.local/bin:$PATH

# to enable the profile you need to annotate the code by
# adding @profile before main method for example
# execute within kernprof
kernprof -l ./serial_pi.py 50000

# access the output of line profiler
python3 -m line_profiler ./serial_pi.py.lprof

# inside_circle consumes most of the time
# getting x and y coordinates of point is again most of that

# Hotspot finding excerise
count_pylibs.py

# Strategy: call flow of function
# first go with the function that is called the first in the code
# here it means to profile the main method and identify the method in there that consumes the most time
# go to this method and see which operation in this method consumes the most there
# this is the hotspot

# for this code it is: result = re.split(word_pattern,text)
# this consumes exclusively 70% of the time of the program!
# its time to parallelize!

# Parallelisation - how to slice the pie
# inside_circle consumes most of the time 73%
# getting x and y coordinates of point is again most of that 80%
# how does one approach that?
x = np.float32(np.random.uniform(size=total_count))
y = np.float32(np.random.uniform(size=total_count))

# These two lines are independent... example solutions?
# Solution A: run these lines in parallel
# Solution B: chunking up total_counts

```
# How does decide what solution is best?
# Either code, test, measure and compare
# Or make a calculation

# Amdahl's law (1967)
# Theoretical possible speed up

speed-up S
parallel portion p
serial speed-ups s


# Solution A
# The parallel portion amounts to 73% of the time thus:
p = 0.73
# speed up of 2
s = 2

S = 1 / (1 - p) + (p / s)
S = 1.575

# Solution B
# The parallel portion amounts to 73% of the time thus:
p = 0.73
# speed up of 24 >> number of chunks
s = 24

S = 1 / (1 - p) + (p / s)
S = 3

# Important here is the more parallel operations the better but there is a point of diminishing
returns
# there is always parts of the code that cannot be parallelized
# maybe there is overhead... Gustafsons law

# Parallelization by map reduce

#!/bin/bash

#SBATCH -n 1
#SBATCH -c 12
#SBATCH -t 00:01:00
#SBATCH -o parallel12.log

time python3 ~


# get an IDE/editor for remote editing
# Development > separate the slurm code from the processing code

feedback of the afternoon
```

(red)
- put code on indico
- put wiki links which are useful
- any further reading material
- put exercise link
- module system only mentioned in passing
- would be nice to have some homework
- maybe more examples of parallelization
- maybe not use the pi example as it took long to understand

(green)
- enough for one day
- well documented and online resources
- good examples with good insight into super computer without high effort
- good structure
- nice exercises and Amdahl's law, +1
- getting up & running with HPC
- good to go step-by-step
- well timed
- loved approach for finding hot spots and get method of parallelisation
- maybe better to discuss big picture up front
- explain straight forward things quick
- keep the plot, don't digress
- from time to time, define something more precise