UCLan Supercomputer facility (HPC Wildcat)

Working on the cluster - some definitions

You may hear a supercomputer referred to as an HPC (High Powered Computing), a cluster or, in the case of the machine at UCLan, Wildcat. These all refer to the same thing. A cluster is a group of connected computers that can work together to perform tasks. Each computer on the cluster is called a *node*. Wildcat has 64 *compute nodes*, these are the machines that your jobs will run on. There is also a *head node* (some clusters have more than one). This is the node you will login to and submit your jobs. The head node on Wildcat is called **leopard**.

Just like a regular machine, each node has a number of cores. Each of the compute nodes on Wildcat has 8 cores and ~25Gb of memory.

I use the word 'job' to describe any code that you run on the computer, setting a simulation running is an example of a job.

Logging on to Wildcat

You will be using SSH to login to the supercomputer. To make the process a bit easier, you can add the following to your config file in ~./ssh, if that file is not present do not worry, you can just create it.

```
Host uclanhpc
    HostName 193.61.251.71
    User your_username
    Port 22

Host leopard
    HostName leopard
    User your_username
    Port 22
    ProxyCommand ssh -W %h:%p uclanhpc
```

This makes it easier to log in to the HPC. When you have made this change you will be able to type `ssh -XY your_username@leopard` and login to the leopard node of the HPC. It will ask you for your password twice, once to login to the 193.61.251.71 server (called **tiger**) and a second time to login to the head node called **leopard**. This is where you will submit your jobs.

When you login for the first time, your home area will be empty. Here you can create your own file structure to keep things organised. The file system on Wildcat has a maximum capacity of 44Tb and you can see your own disc usage with `du -sh`. Keep in mind that the space is not infinite, so if you can compress/remove any unwanted files or transfer them to starlink etc then feel free. It is unlikely that you will use huge amounts of storage but it is something to be aware of.

Modules

Working on the cluster is different from working on a regular laptop or PC. Instead of software being availible to you right away, it is controlled through *modules*. Let me show you an example:

```
[afenton@leopard ~]$ module list
No Modulefiles Currently Loaded.
[afenton@leopard ~]$ module avail

------------------------------------------------- /apps/modules/Modules/versions --------------------------------------------------
3.2.10 3.2.6  3.2.9
intel(5):PROB:155: Invalid modulename 'iFort/17.0.1' found
modulefiles/intel(5):PROB:155: Invalid modulename 'iFort/17.0.1' found

------------------------------------------------- /apps/modules/Modules/modulefiles -------------------------------------------------
amber/11(default)                icomp/2011.1.8.273-testing       openmpi/1.3.3/intel_11.1_64_openib
anaconda/2.7                     ifftw/2.1.5                      openmpi/1.3.3-intel_11.1
ansys/CFX/v121                   ifftw/3.2.2                      openmpi/1.4.1/intel_11.1_64_openib
ansys/v150                       imkl/10.2.2.025                  openmpi/1.4.2/gnu_4.1.2_64_openib
ansys/v180(default)              imkl/10.2.5.035(default)         openmpi/1.4.2/intel_10.1_64_openib
ansys/v192                       impi/2011.1.8.273-testing        openmpi/1.4.2/intel_11.1_64_openib
atk/2.12.0                       impi/3.2.2                       openmpi/1.4.2-intel_11.1
BLAS/19-04-2011                  impi/4.0.0                       openmpi/1.4.3-testing
clhep/gnu_4.1.2                  impi/4.0.3(default)              openmpi/1.4.4-intel_11.1
clisp/2.49                       intel/compiler/10.1.026          openmpi/1.6.4-gcc_4.4
cuda/10.1                        intel/compiler/11.1              openmpi/1.6.5/gcc
desmond/2018_4                   intel/compiler/11.1.059          paraview/3.10.0
dlpoly/4.04                      intel/compiler/11.1.073          paraview/5.0.1
fds/5.5.3(default)               intel/fftw/2.1.5                 phoebe/0.31a
ffmpeg/0.6.1(default)            intel/fftw/3.2.2                 phoebe/2.0a1
ffmpeg/4.2.2                     intel/iFort/16.0.4               py2cairo/latest
flex/2.5.39                      intel/iFort/17.0.1               pygobject/2.28.6
gdk-pixbuf/2.30.8                intel/impi/3.2.2                 pygobject/3.8.3
geant/4.10.00/gnu_4.1.2          intel/impi/4.0.0                 python/2.7.6
geant/4.9.5/gnu_4.1.2            intel/mkl/10.2.2.025             python/2.7.7
geant/4.9.6/gnu_4.1.2            intel/mkl/10.2.5.035             python/3.8.1
gfortran/9.2.0                   java/jdk1.6.0_21                 qt/4.7.1
```

In the screenshot above, you can see I use the command `module list` to show me which modules I currently have loaded (none at first). I then use `module avail` to show me which modules are availible on the cluster. I then use the `module load` command to load the `python/3.8.1` module. Using the `module list` command again shows me that python3 has been loaded.

```
[afenton@leopard ~]$ module load python/3.8.1
[afenton@leopard ~]$ module list
Currently Loaded Modulefiles:
  1) python/3.8.1
[afenton@leopard ~]$
```

I am just illustrating the process here, you would not be loading python3 on the head node because **we do not run code on the headnode**. The head node is only for compiling and submitting jobs, which I will get on to shortly.

Bashrc file

If you are familiar with bash then you may know about the bashrc file already. If not, the bashrc file contains a list of commands that are executed every time you login. This saves you having to type the same things in every time! (Actually it is .bashrc, the dot means it is hidden and will not be listed when you type `ls`.) For example, my .bashrc file contains the following

```
export HDF5_DIR=/home/afenton/hdf5-gfortran
export LD_LIBRARY_PATH=/home/afenton/hdf5-gfortran/lib:$LD_LIBRARY_PATH
```

What these commands do is not important at the moment but because they are in my .basrc file, the *enviroment variable* `HDF5_DIR` is always set to `/home/afenton/hdf5-gfortran` and I don't need to set it every time I open a new terminal. You can also define aliases in here, for example:

```
alias c="clear"
```

I have the above alias set so, when I want a clean terminal, I don't have to type the full word, I just press c and hit return.

Keep in mind that for any changes you make to the .bashrc file to work the first time, you need to reload it with `source .bashrc`. Note that you only need to do this immediately after making the changes, after that it will be done automatically when you login.

Submission of jobs

I mentioned before that we do not run code on the head node. I will use the SPH code PHANTOM as an example in the following. When logged on to the headnode using the process outlined above, you **cannot** just run the code as you normally would with `./phantom disc.in`. This will use up all the resources availible on the head node and stop all other users from submittion their own jobs. Instead you use a PBS script, see the example below, to request resources, give your job a name and load the appropriate modules. When you execute this script, your job will be submitted to the *scheduler* and, if the resources you requested are availible, your job will run.

```
 GNU nano 2.0.9                          File: submitPhantom.pbs                                      Modified

#PBS -S /bin/bash
#PBS -q testq               # This is the queue the job goes into, on wildcat it is not important, keep it as testq
#PBS -l nodes=1:ppn=8       # request 1 node, 8 cores per node. ppn must be 8 so this uses 8 cores
#PBS -l mem=20gb            # requesting 20gb of RAM per node at maximum
#PBS -l walltime=48:00:00   # default walltime is 48:00:00 - The code will stop after max walltime is reached
#PBS -N 0.65d_1.0s          # Give your run a useful name

module load impi/4.0.3      # I like to load my modules in the submission script at time of execution
module load gsl/1.16
module load gfortran/9.2.0
ulimit -s unlimited
export OMP_SCHEDULE="dynamic"
export OMP_NUM_THREADS=8
export OMP_STACKSIZE=512m
module load intel/compiler/11.1.073
module load openmpi/1.10.6/intel_11.1
module load hdf5/10.6
module load intel/fftw/3.2.2
cd $PBS_O_WORKDIR

./phantom disc.in
```

Again using PHANTOM as an example, this script will be in the same directory as your `disc.in`, `disc_00000.tmp.h5` files (i.e. any files that the code requires to run, such as initial conditions).

Compiling code on Wildcat

The process of compiling code and installing dependancies (if they are not already installed as a **module**) is a bit nuanced and can require a little bit of fiddling. If you have any specific problems, let me know and I will give you a hand.

Do not worry when you submit a job and you don't see anything on your screen. Jobs that you submit run 'in the background' and you do not see the usual output to terminal that you would see when running on a laptop or PC

Addendum - running jobs interactively

As mentioned above, when running jobs on wildcat or any HPC, the standard output (stdout) of the code does not appear in terminal as it would if you were running it on your own machine. This can be irritating when trying to debug something as you will not see the error messages. One way, and perhaps the best way, of circumventing this is to use *bash redirection* to pipe the stdout to a file using something like `./phantom disc.in >> output.txt`. There is an option in the submission script to do this for you but I have known it to not work correctly and consistently and have found it better to do it myself.

In addition to this, to further aid debugging, you can run a job in **interactive mode**. To do this you use the `-I` flag in your `qsub` command, for example `qsub -I submitPhantom.pbs`. This will submit the job in interactive mode and you will be transfered to a terminal on a specific node (meaning you only have access to 8 cores). Here you will be able to run your code using the commands you would use if you were running on your own machine (for example `./phantom disc.in`). **NOTE:that you will have to reload any modules and change directory to your working directory**. To exit an interactive job, you use `exit`. It is rare that you will have to do this, but it can be helpful for debugging because the stdout is printed to screen and any error messages are readable. If you do want to do this, give me a shout and we will make sure you aren't inadvertently running code on the headnode.

That is a very brief rundown on the cluster (and clusters in general). You may find it a bit tricky to get things going right away but any problems at all then come and find me.

If you are using the UCLan HPC then you have probably heard from Christian Kay. If you have any big problems then he is the guy to talk to!