Photo: Carlos Jones/ORNL
[CC BY 2.0] via Wikimedia Commons

High Performance Computing (HPC)

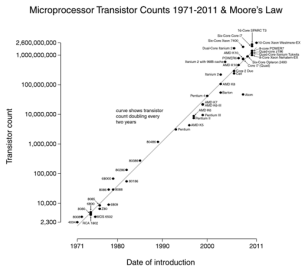
- Introduction to HPC and why it's useful
- How do you parallelize your code?
- The practice of running software on a cluster

Introduction to HPC and why it's useful

Gordon Moore

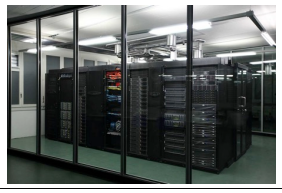
Born	January 3, 1929 (age 84) San Francisco, California, USA
Nationality	American
Alma mater	University of California, Berkeley; California Institute of Technology
Occupation	Chairman Emeritus, Intel Corporation
Net worth	▲$4 billion USD (2011)[1]

Jan 2015 update: $6.7 billion



Introduction to HPC and why it's useful

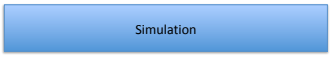
- Embarrassingly parallel problems
  - Graphics
  - Simulations with multiple parameters
- Non embarrassingly parallel problems
  - Fluid dynamics
  - A lot of the tasks run by a single program



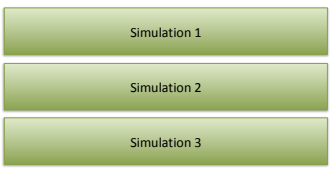
How do you parallelize your code? !

- `as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))`
  - Should be used in your code to give a simulation number

Before



Now



Using your PC

```
for ( i in 1 : 10 )
{
   do_simulation( i )
}
```

!

Shell script on the cluster

Using HPC

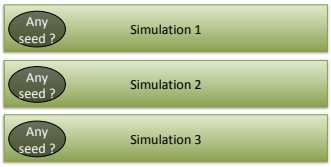
```
i <- as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))
do_simulation( i )
```

How do you parallelize your code?

- `as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))`
  - Should be used in your code to give a simulation number

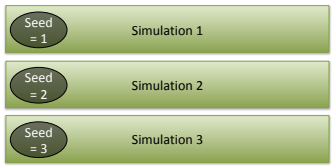
Before

Now

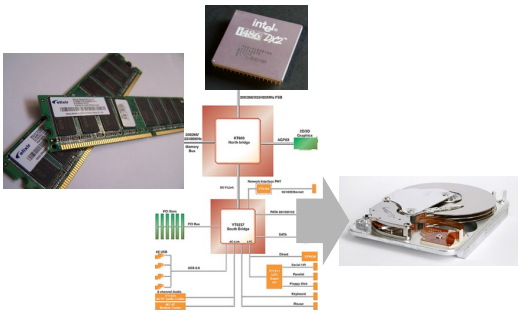


How do you parallelize your code?

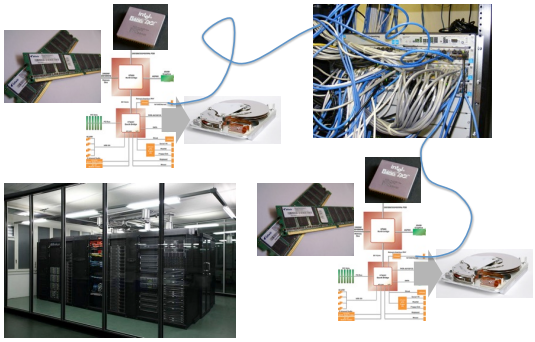
- `as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))`
  - Should be used in your code to give a simulation number
- Pseudo random numbers
  - Given a certain random number seed, you get the same sequence of random numbers every time
  - Each simulation should have a different seed



Handling memory

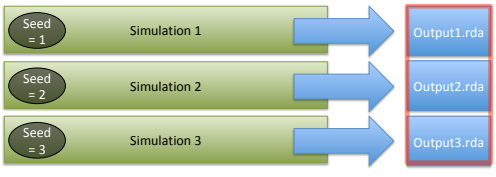


Handling memory



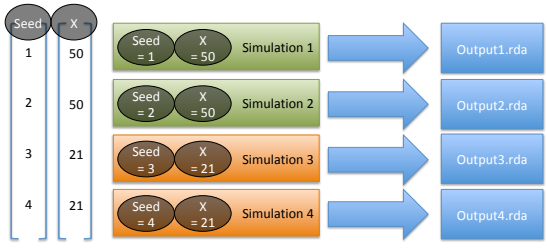
Handling memory

- Save your results in memory and then write to disk at the end
- Output your code to a series of files
- Write local code to read in your series of files automatically



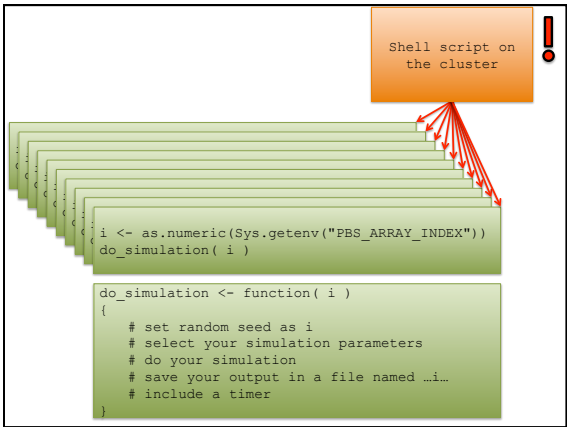
Handling memory

- Save your results in memory and then write to disk at the end
- Output your code to a series of files
- Write local code to read in your series of files automatically

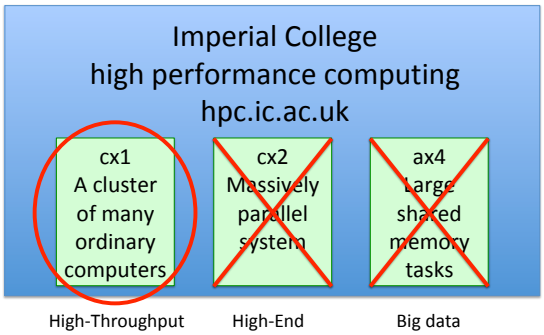


Handling memory

- Save your results in memory and then write to disk at the end
- Output your code to a series of files
- Write local code to read in your series of files automatically
- Build a timer into your code
- Test your code locally to know your memory and time requirements



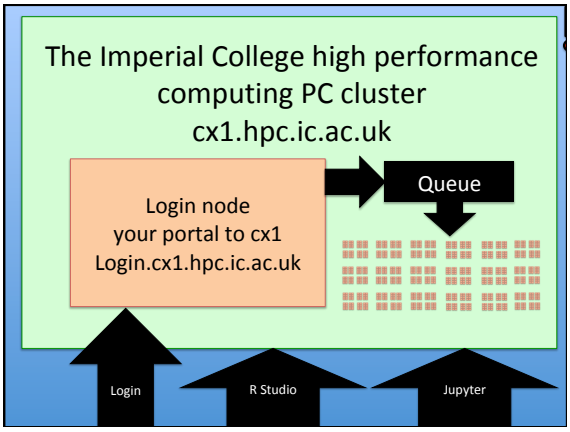
The practice of running code on a cluster



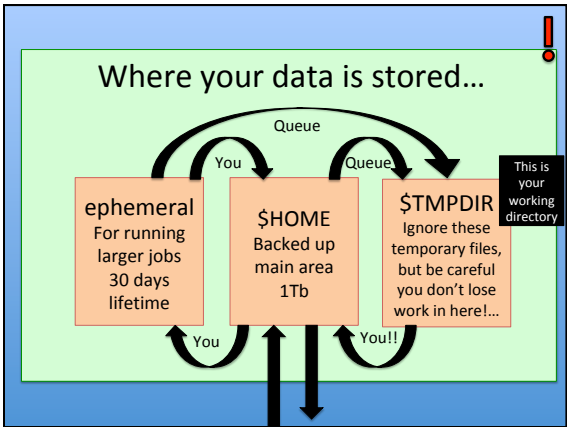
http://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/computing/

The Imperial College high performance computing PC cluster cx1.hpc.ic.ac.uk

The Imperial College high performance computing PC cluster cx1.hpc.ic.ac.uk



Where your data is stored…



Step 1: get your code onto the cluster

- Go to filezilla-project.org
- Download FileZilla Client and install it.
- Open FileZilla and put in the following settings
  - Host: sftp://login.cx1.hpc.ic.ac.uk
  - Username: your username for all Imperial services
  - Password: your usual Imperial password
  - Port: 22
- Press the Quickconnect button
- Copy your files and folders across onto HPC.

Step 1: get your code onto the cluster

Alternative method if you prefer command line…

- Use sftp: from the directory of your code in a shell window type ….
  - `sftp username@login.cx1.hpc.ic.ac.uk`
  - You will be asked for your standard cluster password
  - `put filename.R`
  - `exit`
- Or use scp:
  - `scp path/to/file.txt username@login.cx1.hpc.ic.ac.uk:/home/username/`
- Also see separate notes to be shared

Step 2: log into the cluster

- Use ssh: from a shell window type ….
  - `Ssh -l username login.cx1.hpc.ic.ac.uk`
  - You will be asked for your standard cluster password
  - Now it's as though you were sitting with a shell open at the login node.
  - `ls` (will list the files in $HOME)
  - `mkdir foldername` (make a new folder)
  - `mv filename $HOME/foldername` (move)
  - `cd foldername` (change directory)
  - `cat filename` (see your file to check it's contents)
  - `anaconda-setup` (set up anaconda – one time only)
  - `conda install r` (install R – one time only)

Step 3: make a file for your shell script

- This is the list of instructions to be executed when you get to the front of the queue is written in shell script. It should be a .sh file
- ***Do not*** run code on the login node – always write a shell script and wait in the queue.
- If you type `cat > filename.sh`
- You will then get the chance to type text (pressing enter for new lines) and the cat command will make the file containing the text that you typed.
- When you are finished typing the contents of your new file press control and D to complete the process.
- Type `cat filename.sh` to check that your file is correct before submitting it to the queue.

Step 3 continued: your shell script file

```
#!/bin/bash
#PBS -l walltime=12:00:00
#PBS -l select=1:ncpus=1:mem=1gb
module load anaconda3/personal
echo "R is about to run"
R --vanilla < $HOME/Rtest/ForwardsNTC_V5.R
mv datafilename* $HOME
echo "R has finished running"
# this is a comment at the end of the file
```

You can run Python code too –
just use different commands here

Step 3 continued: job sizing

Job class	Number of nodes N	ncpus/node	Max mem/node	Max walltime/hr	Max number of running jobs per user
throughput	1	1-8	96GB	up to 72hr	unlimited for jobs <=24hr in length
general	1 - 16	32	62GB or 124GB	up to 72hr	unlimited for jobs <=24hr in length
singlenode	1	48	124GB	up to 24hr	10
multinode	3 - 16	12	46GB	up to 48hr	unlimited
debug	1	1-8	96GB	up to 30 mins	1
large memory	1	12 or 24	190 or 380GB	48 hr	unlimited
GPU	1-2	1-8	1 - 32GB	48hr 72hr (P1000 only)	8
long	1	1-8	96GB	72 - 1000 hr	1

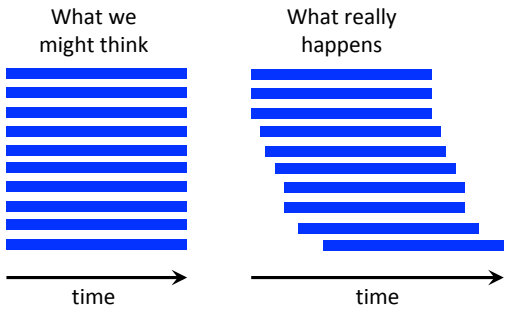
Step 4: submitting your job to the cluster

- You are now in the $HOME directory with your code and shell script both written.
- To submit your job type
  `qsub -J 1-32 filename.sh`
  `qstat` (S changes from Q to B when running)
- If you want to delete a job
  `qstat`
  `qdel job-id[]` (the [] is for array jobs only)
- qstat will give you a list of jobs and you would get the job-id from there.

Step 5: check that all is well

- Wait 5-10 minutes then check that nothing has gone wrong.
- `qstat` (is your job running still)
- `ls` (are output files as expected)
- `cat filename.sh.ejob-id.index` (are error files empty?)
- `cat filename.sh.ojob-id.index` (are standard output files as expected)
- `qstat` (is your job running still)
- `exit` (you're done for now come back later)

Step 5 continued: how jobs are run



Step 6: Getting your results back

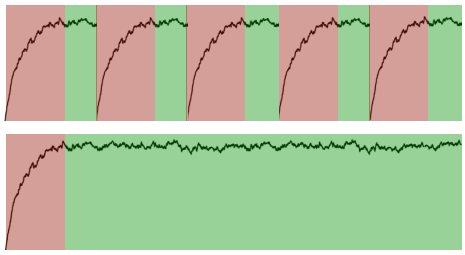
- `qstat` (is your job running still)
- `cd $HOME`
- `ls` (output files as expected?)
- `cat output filename` (contents as expected?)
- `cat filename.sh.ejob-id.index` (error files empty?)
- `cat filename.sh.ojob-id.index` (standard output files as expected?)
- `tar czvf filename.tgz *`
- `mv filename.tgz $HOME`

Step 6 continued: sftp to get results

- `exit`
- Use sftp: from a new directory on your own computer of where you want the results to be. Open a shell and type …
  - `sftp username@login.cx1.hpc.ic.ac.uk`
  - You will be asked for your standard cluster password
  - `get filename.tgz`
  - `exit`
- Your results are now all on your own computer
  - `tar xzvf filename.tgz`
- Your results are now complete uncompressed and ready for use. Now you need to write some R code to read in and analyze all those file.

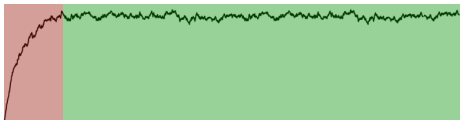
For your excercises

- You'll be asked to adapt your code from yesterday to run on the cluster for a much bigger ecological community size
- You'll need to collect species abundance data as before and average over a large number of parallel simulations.



For your excercises

- You'll be asked to adapt your code from yesterday to run on the cluster for a much bigger ecological community size
- You'll need to collect species abundance data as before and average over a large number of parallel simulations.
- Use a "burn in" period and check the species abundance distribution periodically. You should plot species richness against time and make a conservative judgment, but for neutral theory 8 * metacommunity size complete turnovers of the community is a good rule of thumb.



DO NOT …

Use the cluster without knowing memory and time requirements
Run jobs on the login node
Try to use cx2 or ax4 parts of the cluster
Output data to the hard disk regularly
Use the same random seed for your simulations
Copy and paste your shell script
Leave your results in $TMPDIR
Waste too much of your own time optimizing your code
Run code on the cluster that hasn't been tested locally first

DO …

Use the cluster for jobs that take a long time locally.
Optimize your code if there's going to be a huge benefit
Run repeat readings and different parameters as separate jobs.
Run jobs that take between 30 mins and 3 days to execute.
Write your shell script on the cluster itself.
Make your code output each result in a differently named file.
Check periodically that all is well on the cluster
Be ambitious – you can do loads of great stuff with a cluster.
imperial.ac.uk/admin-services/ict/self-service/research-support/rcs
wiki.imperial.ac.uk/display/HPC/High+Performance+Computing