UNIVERSITY OF CAMBRIDGE

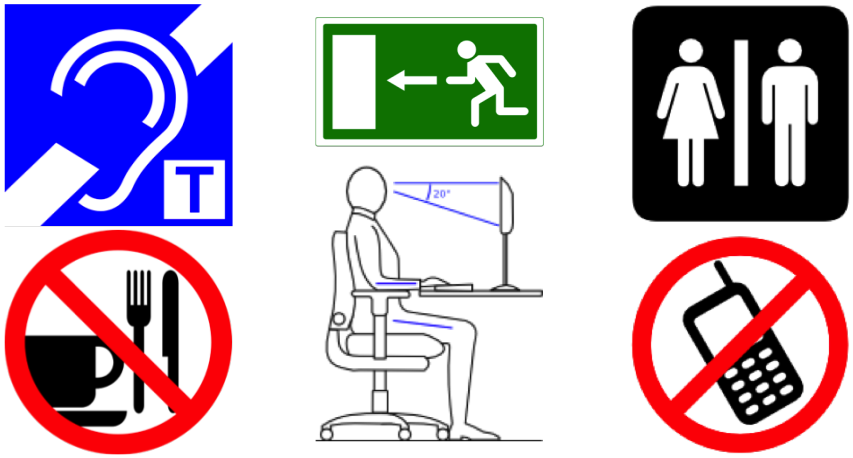
An Introduction to High Performance Computing

Stuart Rankin
support@hpc.cam.ac.uk

Research Computing Services (http://www.hpc.cam.ac.uk/ http://www.csd3.cam.ac.uk/)
University Information Services (http://www.uis.cam.ac.uk/)

25th January 2018 / UIS Training

Health and Safety



Welcome

- Please sign in on the attendance sheet.
- Please fill in the online feedback at the end of the course:
  `http://feedback.training.cam.ac.uk/ucs/form.php`
- Keep your belongings with you.
- Please ask questions and let us know if you need assistance.

Plan of the Course

Part 1: Introduction
Part 2: Cluster Basics
Part 3: Advanced HPC

09:30	WELCOME
10:00-11:00	Introduction and basics
11:00-11:30	Practical and break
12:00-12:30	Practical
12:30-13:30	LUNCH
14:00-14:30	Practical
14:45-15:15	Practical
15:30-CLOSE	Further discussion

Part I: **Introduction**

Basics: Outline

Who are we?

Training Accounts

CSD 3 and Login node for today

Security

Basics: Pre-requisites

Connecting

Basic practicals

Connecting from other clients

UIS: Research and Institutional Services Division

Your trainers for today will be:

- Paul Sumption: Research Computing Technical Liaison
- Arjen Tamerus: Research Software Engineer

Basics: Training accounts

- For our practical exercise's we will use HPC training accounts.
- You will find two pieces of paper on your desk.
- 1: A terms and conditions form for you to sign.
- 2: Your training account details.
- Your training account will only be valid for today.

Basics: Login node

- For our practical exercise's we will use the login node login-gfx2.hpc.cam.ac.uk.
- Normally you would use login.hpc.cam.ac.uk hence using this in the examples in our slides.
- We will be using nodes attached to Darwin
- Some of the details in theses slides will change as we now have a new cluster CSD3 and Darwin is in the process of being decommissioned.
- The main difference is the login node names and slightly different submit scripts as CSD 3 has more node types.

Basics: Security

- Boring but very, very important . . .
- Cambridge IT is under constant attack by would-be intruders.
- Your data and research career is threatened by intruders.
- Cambridge systems are high profile and popular targets.
- Don't let intruders in.

Basics: Security

1. Keep your password (or private key passphrase) safe.
2. Always choose strong passwords.
3. Your UIS password is used for multiple systems so keep it secure!
4. Keep the software on your laptops/tablets/PCs up to date this includes home computers especially if you are using the VPN to connect in.
5. Don't share accounts (this is against the rules anyway).

Pre-requisites

- A pre-requisite of this course:
- Basic Unix/Linux command line experience
- `https://www.training.cam.ac.uk/ucs/Course/ucs-unixintro1`
  Unix: Introduction to the Command Line Interface (Self-paced)
- Shell scripting experience is desirable
- `https://www.training.cam.ac.uk/ucs/Course/ucs-scriptsci`
  Unix: Simple Shell Scripting for Scientists

Basics: Connecting

- When connecting to the cluster will we use the SSH secure protocol only.
- We will use the Linux workstations during this course
- Please check your workstation is booted into Ubuntu Linux, ask if you need help with this.
- You are welcome to use your own laptop, however you may need to install some software in order to connect.
- Later in this section of the slides we will cover the software you may need to install on your own computer.

Basics: Connecting

- SSH secure protocol only.
  Supports login, file transfer, remote desktop...
- HPCS allows access from registered IP addresses only.
  Almost all Cambridge University addresses already registered.
  Connection from home possible via the VPN service
  http://www.ucs.cam.ac.uk/vpn
  or SSH tunnel through a departmental gateway.

Connecting: Linux Clients

- ssh, scp, sftp, rsync
  Installed (or installable), in Ubuntu we will use 'Terminal'.
- X Windows, for using graphical applications remotely This is already installed on your desktop.

Connecting: Login

- From Linux/MacOSX/UNIX (or Cygwin):
  ssh -Y **abc123**@login.hpc.cam.ac.uk
- From graphical clients:
  Host: login.hpc.cam.ac.uk
  Username: **abc123** (your UCAM account name)
- login.hpc will map to a random login node
  i.e. one of login-sand1, login-sand2, ... , login-sand8
  NB Not darwin.hpc (the head node).
- Non-registered addresses will fail with "Connection refused".
- Similarly for other systems (e.g. cardio-login.hpc, login-mrc-bsu.hpc,...).

Connecting: First time login

- The first connection to a particular hostname produces the following:

```
The authenticity of host 'login-sand2.hpc.cam.ac.uk (131.111.1.214)'
can't be established.
RSA key fingerprint is
0b:ef:59:90:fb:13:4a:c9:56:82:7b:cd:4b:2b:e1:3b.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'login-sand2.hpc.cam.ac.uk' (RSA) to the
list of known hosts.
```

- One should always check the fingerprint before typing "yes".
- Graphical SSH clients *should* ask a similar question.
- Designed to detect fraudulent servers.

Connecting: First time login

- You may be presented with any of the following fingerprints (depending on your client):

```
MD5:0b:ef:59:90:fb:13:4a:c9:56:82:7b:cd:4b:2b:e1:3b
SHA256:sSkVfzpwjwiFvxLcdPoDpN8IsN3kt0ZSywhDhPKZPAg

MD5:34:9b:f2:d2:c6:b3:5c:63:99:b7:27:da:5b:c8:16:fe
SHA256:HsiY1Oe0M8tS6JwR76PeQQA/VB7r8675BzG5OYQ4h34

MD5:64:7c:7c:ff:05:9d:0e:dc:06:fe:f1:c2:10:37:7a:85
SHA256:wq9ljBfPa7lXXpQq+rk5JTBXLJO/kXjOc5A7rp4ENzA
```


Basics: A practical example before some theory

- Some simple exercises will help us gauge your experience.
- Exercise 1: Login with SSH.
- Exercise 2: Navigating the command line.
- Exercise 3: SFTP file transfer.

Navigating your terminal

Useful commands for navigating your terminal.

- cd <dirname> - change into a directory
- ls <dirname> - list the contents of a directory
- cd or cd ~ - change into your home folder
- cd .. - change back one folder
- man ls - will bring up the manual page for the ls command
- pwd - print working directory

Exercise 1: Navigating your terminal

- Start a terminal by double clicking on the terminal icon
- Try the ls and cd commands in a terminal.
- Look at the man page for the ls command
- Close the terminal

Exercise 2: Login

Using a Linux terminal you will login to the cluster with your HPC training account.

- Start the terminal by double clicking on the terminal icon
- In your terminal enter:
- ssh -Y **abc123**@login.hpc.cam.ac.uk
  Replace abc123 with your training account username
- Enter your password as supplied on the sheet
- Leave this terminal open, you will need it for exercise 3!

Exercise 3: Transfer some files

You will need to transfer the exercise files to the cluster.

- Open a second Linux terminal on your training computer.
- Enter this command: cd `~\Course_material`
- Check the file 'exercises.tar.gz is in your directory listing
- Hint: ls

Exercise 3: Transfer some files

Transfer the exercises.tar.gz to your HPC home folder.

- sftp abc123@login.hpc.cam.ac.uk
  Change abc123 to your training account username
- The command: put exercises.tar.gz will transfer the file from your local computer to the remote one
- In terminal logged into the cluster type: ls
- Use: tar -xvf exercise.tar - to unzip the file

Connecting: File Transfer

In exercise 3 we used sftp. Another tool to be familiar with is rsync especially when you want to transfer lots of directories and files.

- rsync is a powerful tool and has many features.
  - You can rerun it to update or resume after interruption.
  - All transfers are checksummed.

Connecting: File Transfer

- From Linux/MacOSX/UNIX (or Cygwin):
  rsync -av **old_directory/** abc123@login.hpc.cam.ac.uk:scratch/new_directory
  copies contents of old_directory to ~/scratch/new_directory.
  rsync -av **old_directory** abc123@login.hpc.cam.ac.uk:scratch/new_directory
  copies old_directory (and contents) to ~/scratch/new_directory/old_directory.
  - For transfers in the opposite direction, place the remote machine as the first argument.

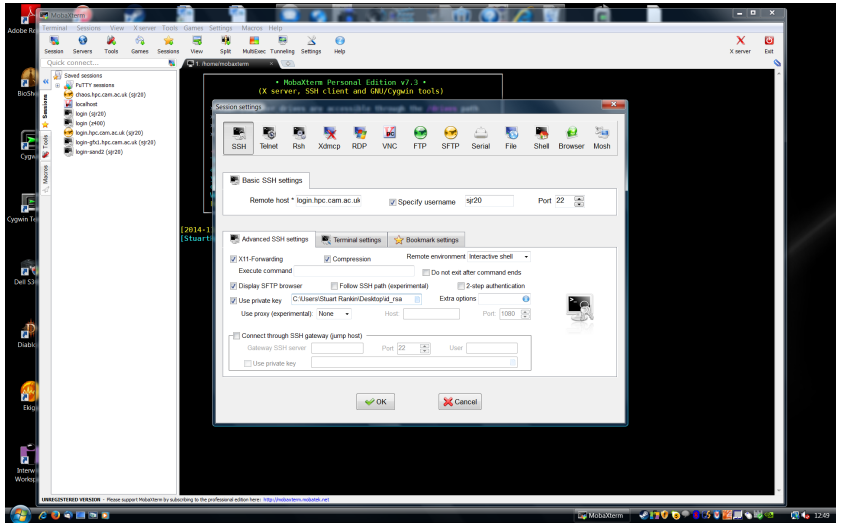
Basics: Connecting from other clients

- When using your own computer you you may need to install some software in order to connect.
- There are quite a few choices of software packages for this, we will just cover a few.
- Our website
  https://www.hpc.cam.ac.uk/using-clusters/connecting covers this in more detail.

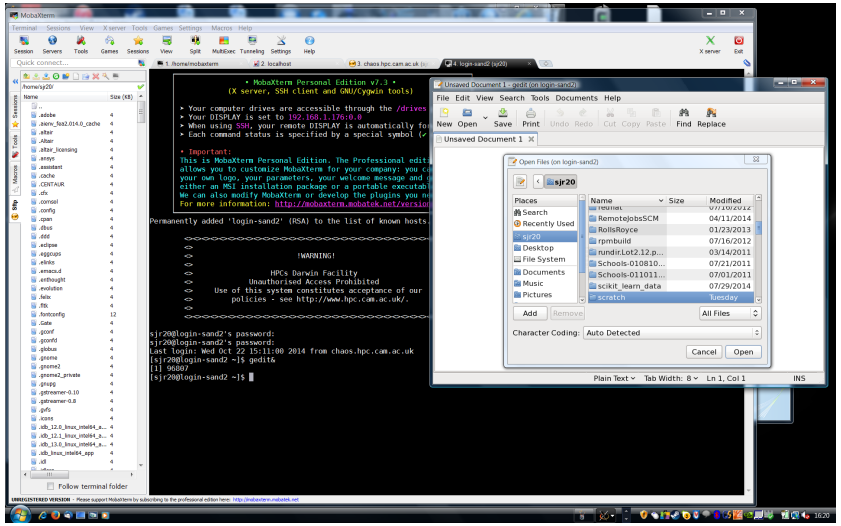
Connecting: MacOSX/UNIX Clients

- ssh, scp, sftp, rsync
  Installed (or installable) OS X has a native terminal package. This can be launched by clicking Apple >Go >Utilities and then clicking the Terminal icon.
- TurboVNC (for remote desktop, 3D optional)
  http://sourceforge.net/projects/turbovnc/files/
- On MacOSX, install XQuartz to display remote graphical applications.
  http://xquartz.macosforge.org/landing/

MobaXterm SSH (Windows)



MobaXterm SSH (Windows)



Part II: **HPC Basics**

HPC Basics: Our Clusters

- Our clusters are built using lots of commodity servers which then operate as a 'super computer'.
- We have CPU and GPU cluster nodes.
- A cluster has a scheduler which runs jobs from a queue.
- You submit jobs to the queue using a submission script.
- Jobs have service levels and QOS (quality of service) associated with them.
- There is a user environment this allows you to load or unload versions of software.
- We will look at each of these aspects in more detail during the course.

HPC Basics: Why use an HPC cluster?

There are many reasons why you may need to use a big computer or cluster.

Limited local resources: Your jobs can no longer run on your own computer or run very slowly.

Storage Intensive: Your data will exceed that of your local storage capacity.

Software: Many of the software packages you require have a complex installation process or require specialist support.

Grants and sharing: You want to make the best use of your grant when buying computing resources or need to share data or a workflow with your group.

HPC Basics: Types of problem part 1.

In this section we will cover high throughput and compute intensive jobs. We will introduce the user and module environment and submit some High Throughput jobs to some CPU nodes.

High Throughput: Many unrelated problems to be executed in bulk.

Compute Intensive: A single problem requiring a large amount of computation.

HPC Basics: High Throughput

- Distribute independent, multiple problems across multiple CPUs to reduce the overall execution time.
- Workload is trivially (or *embarrassingly*) parallel:
  - Workload breaks up naturally into *independent* pieces.
  - Each piece is performed by a separate process/thread on a separate CPU (concurrently).
  - Little or no inter-CPU communication.
- Emphasis is on throughput over a period, rather than on performance on a single problem.

HPC Basics: Data Intensive Problems

- Distribute the data for a single problem across multiple CPUs to reduce the overall execution time.
- The *same* work may be done on each data segment.
- Rapid movement of data to and from disk is more important than inter-CPU communication.
- Big Data problems of great current interest -
- Hadoop/MapReduce
- Life Sciences (genomics) and elsewhere.

HPC Basics: Summary

- Why have a supercomputer?
  - Big single problems, many problems, Big Data.
- Most current supercomputers are clusters of separate nodes.

HPC Basics: User Environment

- The user environment can be changed by loading modules.
- At login some default modules are loaded.
- When you start to compile software or write your own programs you will need to know more about your user environment and the underlying hardware.

HPC Basics: User Environment

- Scientific Linux 6.8 (Red Hat Enterprise Linux 6.8 rebuild)
  - bash
  - GNOME2 or XFCE4 desktop (if you want)
- Lustre (patched), Mellanox OFED, CUDA
- But you don't need to know that. (Probably...)
- Upgrade to Scientific Linux/Red Hat Enterprise Linux 7 underway (Wilkes and new CSD 3 nodes are already on SL7).

User Environment: Filesystems

When you apply for an HPC account a home directory is created for you.

- /home/abc123
  - 40GB quota.
  - Visible equally from all nodes.
  - Single storage server.
  - Hourly, daily, weekly snapshots copied to tape.
  - Not intended for job outputs or large/many input files.
- /scratch/abc123
  - Visible equally from all nodes.
  - Larger and faster (1TB initial quota).
  - Intended for job inputs and outputs.
  - Not backed up.

Our storage services

- We have several storage services for users that need to exceed 1TB.
- http://www.uis.cam.ac.uk/initiatives/storage-strategy/storage-services
- The most relevant services to HPC are RCS and RDS.
- RCS - Research Cold Store is for data that isn't changing, data goes to disk then two sets of tapes.
- RDS - Research Data Store, non backed up high performance storage for projects.

Filesystems: Quotas

- quota

```
==========================================================================================
Usage on /scratch (lfs quota -u abc123 /scratch):
==========================================================================================
Disk quotas for user abc123 (uid 456):
      Filesystem  kbytes   quota   limit   grace   files   quota   limit   grace
 /scratch/abc123 *1073742000  1073741824 1181116006   -   165588       0       0       -
...
```

- Aim to stay below the soft limit (*quota*).
- Once over the soft limit, you have 7 days grace to return below.
- When the grace period expires, or you reach the hard limit (*limit*), no more data can be written.
- It is important to rectify an out of quota condition ASAP.

Filesystems: Quotas

- quota

```
==========================================================================================
Usage on /scratch (lfs quota -u abc123 /scratch):
==========================================================================================
Disk quotas for user abc123 (uid 456):
      Filesystem  kbytes   quota   limit   grace   files   quota   limit   grace
 /scratch/abc123 9298708  1073741824 1181116006   -   165588       0       0       -
...
```

- Aim to stay below the soft limit (*quota*).
- Once over the soft limit, you have 7 days grace to return below.
- When the grace period expires, or you reach the hard limit (*limit*), no more data can be written.
- It is important to rectify an out of quota condition ASAP.

Filesystems: Backups

- Disk snapshots and tape (as of May 2017).
- They are not an undelete - take care when deleting.
- Successful restoration depends on:
  - The file having existed long enough to have been backed up at all.
  - The last good version existing in a current backup.
  - Request restoration as soon as possible with *location* and *exact time of loss*.
  - Scratch files are not backed up.

Filesystems: Automounter

- Directories under /scratch are automounted.
- They only appear under /scratch when explicitly referenced.
- Thus when browsing /scratch may appear too empty
  — use *ls* or *cd* to reference /scratch/abc123 explicitly.

Filesystems: Permissions

- Be careful and if unsure, please ask support@hpc.cam.ac.uk.
  - Can lead to accidental destruction of your data or account compromise.
- Avoid changing the permissions on your home directory.
  - Files under /home are particularly security sensitive.
  - Easy to break passwordless communication between nodes.

Using HPC: Software

- Free software accompanying Red Hat Enterprise is (or can be) provided.
- Other software (free and non-free) is available via modules.
- Some proprietary software may not be generally accessible.
- See http://www.hpc.cam.ac.uk/using-clusters/software.
- New software may be possible to provide on request.
- Self-installed software must be properly licensed.
- *sudo will not work.* (You should be worried if it did.)

User Environment: Environment Modules

- Modules load or unload additional software packages.
- Some are required and automatically loaded on login.
- Others are optional extras, or possible replacements for other modules.
- Beware unloading default modules in ~/.bashrc.
- Beware overwriting environment variables such as PATH and LD_LIBRARY_PATH in ~/.bashrc. If necessary append or prepend.

User Environment: Environment Modules

- Currently loaded:

```
module list
Currently Loaded Modulefiles:
  1) dot                      6) intel/impi/4.1.3.045   11) default-impi
  2) scheduler                7) global
  3) java/jdk1.7.0_60         8) intel/cce/12.1.10.319
  4) turbovnc/1.1             9) intel/fce/12.1.10.319
  5) vgl/2.3.1/64            10) intel/mkl/10.3.10.319
```

- Available:

```
module av
```


User Environment: Environment Modules

- Show:

```
module show castep/impi/7.0.3
-------------------------------------------------------------------
/usr/local/Cluster-Config/modulefiles/castep/impi/7.0.3:

module-whatis    adds CASTEP 7.0.3 (Intel MPI) to your environment

Note that this software is restricted to registered users.

prepend-path     PATH /usr/local/Cluster-Apps/castep/impi/7.0.3/bin:/usr/local/...
-------------------------------------------------------------------
```

- Load:

```
module load castep/impi/7.0.3
```

- Unload:

```
module unload castep/impi/7.0.3
```


User Environment: Environment Modules

- Matlab

```
module load matlab/r2015b
```

- Invoking matlab in batch mode:
  matlab -nodisplay -nojvm -nosplash command
  where the file command.m contains your matlab code.
- The University site license contains the Parallel Computing Toolbox.

User Environment: Environment Modules

- Purge:

```
module purge
```

- Defaults:

```
module show default-impi
module unload default-impi
module load default-impi-LATEST
```

- If you have compiled software yourself your run time environment must match compile time environment!

Excercise 4: Environment Modules

- Connect to the cluster using your training account: See excercise 2 if you have closed your terminal.
- Get a list of modules that are currently loaded

Hints: module list

- Get a list of available R modules

Hints: module av R

Excercise 5: Run an Rscript

- Connect to the cluster using your training account: See excercise 2 if you have closed your terminal.
- In the exercises folder there is a file called test.r
- Run this script using: Rscript hello.r
- Load the module for: R/3.3.2

Hints: module load R/3.3.2

- Run the script again: Rscript hello.r

HPCS: Storage

- Multi-petabytes split across multiple filesystems with tape.
- Lustre cluster filesystem:
  - Multiple RAID6 back-end disk volumes.
  - Multiple object storage servers.
  - Single metadata server.
  - Tape-backed HSM on newest filesystems.
  - 4 GB/sec overall read or write.
  - Prefers big read/writes over small.
- For active HPC work only.

HPCS: Service Levels

Service Level 1 Paying, intended for large projects with long-term, consistent requirement.
- Guaranteed fraction of resources per quarter.

Service Level 2 Paying, intended for medium-sized projects with irregular requirement.
- High priority, but no guarantees; *ad hoc*.

Service Level 3 Non-paying, intended for interim or pump priming, small-scale use.
- Low priority, limited usage (200,000 Darwin core hours) per quarter.

HPCS: Service Levels

Service Level 4 Non-paying, for when nothing else is available.

Very low priority, very restricted, very limited. Best efforts continuation.

HPCS: How To Apply

- Submit the online application form:
  https://www.hpc.cam.ac.uk/services/applying-for-resources/hpc-application
- The PI should be someone senior enough to have funding.
  E.g. supervisor, head of research group.
- Funding is not necessary.
- Please email support@hpc.cam.ac.uk for all support issues.
- Further information can be found on the web site:
  http://www.hpc.cam.ac.uk
- Imminent upgrades may introduce changes.

Part III: **Using HPC**

Advanced HPC: Types of problem.

In this section we will cover Memory Intensive and Data Intensive jobs.

Memory Intensive: A single problem requiring a large amount of memory.

Data Intensive: A single problem operating on a large amount of data.

Advanced: Compute Intensive Problems

- Distribute the work for a single problem across multiple CPUs to reduce the execution time as far as possible.
- Program workload must be *parallelised*:

  Parallel programs split into copies (processes or threads).
  Each process/thread performs a part of the work on its own CPU, concurrently with the others.
  A well-parallelised program will fully exercise as many CPUs as there are processes/threads.
- The CPUs typically need to exchange information rapidly, requiring specialized communication hardware.
- Many use cases from Physics, Chemistry, Engineering, Astronomy, Biology...
- The traditional domain of HPC and the Supercomputer.

Advanced: Scaling & Amdahl's Law

- Using more CPUs is not necessarily faster.
- Typically parallel codes have a scaling limit.
- Partly due to the system overhead of managing more copies, but also to more basic constraints;
- Amdahl's Law (idealized):

$$S(N) = \frac{1}{(1 - p + \frac{p}{N})}$$

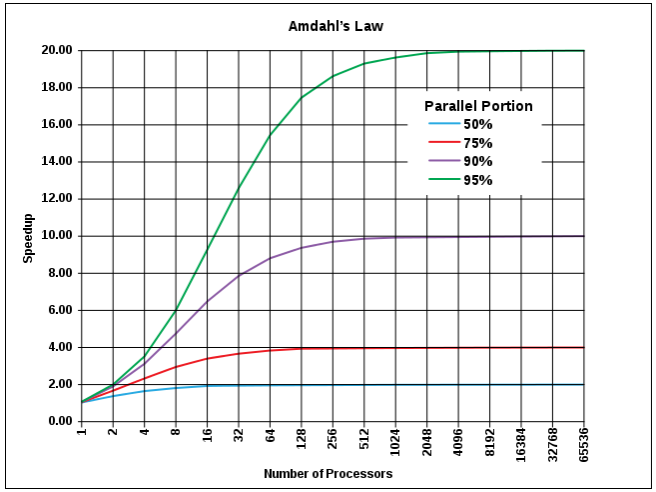
where

$S(N)$ is the fraction by which the program has sped up relative to $N = 1$

p is the fraction of the program which can be parallelized

N is the number of CPUs.

Advanced: Amdahl's Law



http://en.wikipedia.org/wiki/File:AmdahlsLaw.svg

Summary

- Parallelisation requires effort:
  - There are libraries to help (e.g. OpenMP, MPI).
  - First optimise performance on one CPU, then make p as large as possible.
- The scaling limit: eventually using more CPUs becomes detrimental instead of helpful.

Advanced HPC: Memory Intensive Problems

- Require aggregation of large memory rather than multiple CPUs.
  NB Memory (fast, volatile) vs disk (slow, non-volatile).
- Technically more challenging to build machines (interconnecting memory and CPUs).
- Coding/porting easier (memory appears seamless, allowing a single system image).
- Optimisation harder (nonuniform memory produces latency).
- Historically, the arena of large SGI systems.
- Nowadays, similar techniques are applicable to commodity servers.

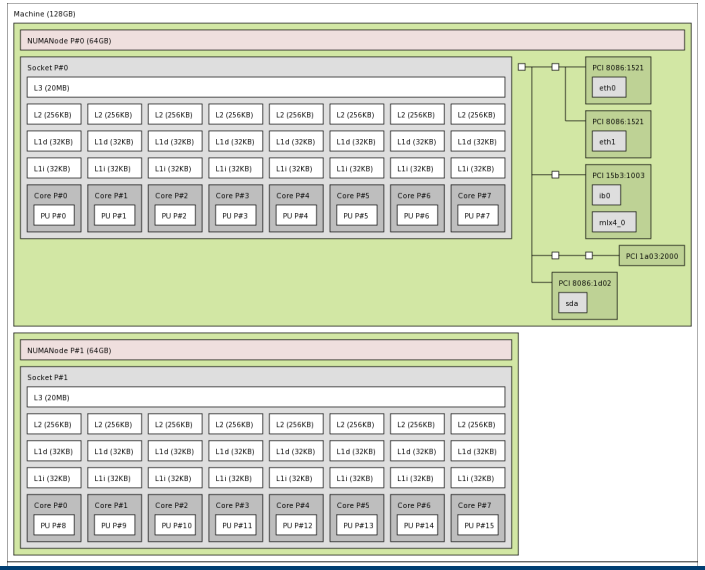
Advanced HPC: Architecture

- So far we have abstracted you from the architecture and the concept of compiling code.
- When does the underlying archictecture start to become important?
- When you are compiling code
- working with MPI/OpenMP
- large job optimisation

Advanced HPC: Inside a Modern Computer

- Today's commodity servers already aggregate both CPUs and memory.
- Even small computers now have multiple CPU cores per socket
  $\Longrightarrow$ each socket is a Symmetric Multi-Processor (SMP).
- Larger computers have multiple sockets (each with local memory):
  all CPU cores (unequally) share the node memory
  $\Longrightarrow$ the node is a shared memory multiprocessor
  with Non-Uniform Memory Architecture (NUMA)
  but users still see a single computer (single system image).

Advanced HPC: Inside a Modern Computer



Advanced: How to Build a Supercomputer

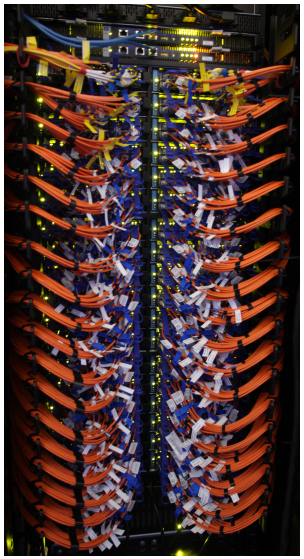
- A supercomputer aggregates contemporary CPUs and memory to obtain increased computing power.
- Usually today these are clusters.

1. Take some (multicore) CPUs plus some memory.
   - Could be an off-the-shelf server, or something more special.
   - A NUMA, shared memory, multiprocessor building block: a node.

Advanced: How to Build a Supercomputer

2. Connect the nodes with one or more networks. E.g.
   Gbit Ethernet: 100 MB/sec
   FDR Infiniband: 5 GB/sec

   Faster network is for inter-CPU communication across nodes.
   Slower network is for management and provisioning.
   Storage may use either.



Advanced: How to Build a Supercomputer

3. Logically bind the nodes
   - Clusters consist of distinct nodes (i.e. separate Linux computers) on common private network(s) and controlled centrally.
     * Private networks allow CPUs in different nodes to communicate.
     * Clusters are distributed memory machines:
       Each process/thread sees only its local node's CPUs and memory (without help).
     * Each process/thread must fit within a single node's memory.
   - More expensive machines logically bind nodes into a single system i.e. CPUs and memory.
     * E.g. SGI UV (COSMOS system in DAMTP).
     * Private networks allow CPUs to see CPUs and memory in other nodes.
     * These are shared memory machines.
     * Logically a single system - 1 big node - but very non-uniform.
     * A single process can span the entire system.

Advanced HPC: Programming a Multiprocessor Machine

- Non-parallel (serial) code
  * For a single node as for a workstation.
  * Typically run as many copies per node as CPUs, assuming node memory is sufficent.
  * Replicate across multiple nodes.
- Parallel code
  * Shared memory methods within a node.
    E.g. pthreads, OpenMP.
  * Distributed memory methods spanning multiple nodes.
    Message Passing Interface (MPI).

Advanced: Summary

- Why have a supercomputer?
  - Big single problems, many problems, Big Data.
- Most current supercomputers are clusters of separate nodes.
- Each node has multiple CPUs and non-uniform shared memory.
- Parallel code uses shared memory (pthreads/OpenMP) within a node, distributed memory (MPI) spanning multiple nodes.
- Non-parallel code uses the memory of one node, but may be copied across many.

Part V: **The HPC Service**

HPCS: A Brief History

Created:	1996 (as the HPCF).
Mission:	Delivery and support of a large HPC resource for use by the University of Cambridge research community.
Self-funding:	Paying and non-paying service levels.
User base:	Includes external STFC & EPSRC plus industrial users.
Plus:	Dedicated group nodes and research projects.

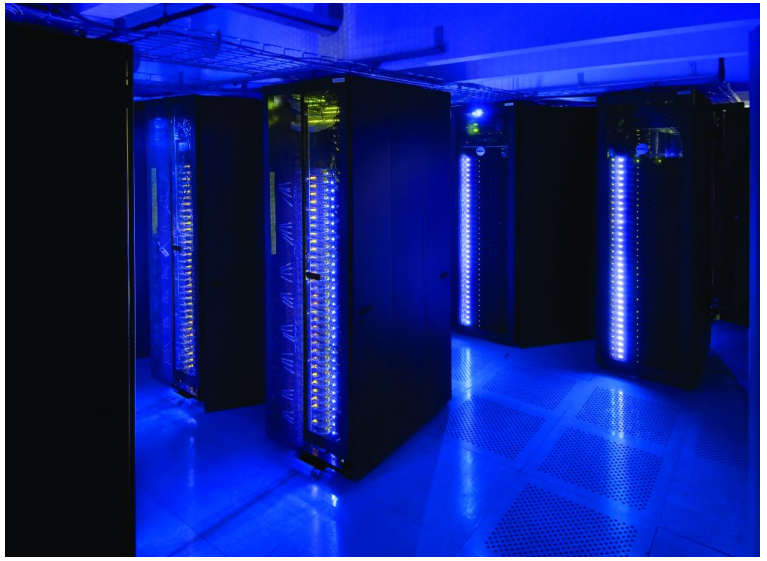
The West Cambridge Data Centre

HPCS: A Brief History of Darwin

1997 76.8 Gflop/s
2002 1.4 Tflop/s
2006 18.27 Tflop/s
2010 30 Tflop/s
2012 183.38 Tflop/s
2013 183.38 CPU + 239.90 GPU Tflop/s
2017 $\sim$ 1.0 CPU + 1.193 GPU + 0.508 KNL Pflop/s

http://www.top500.org

Darwin1 (2006–2012)



Darwin3 (2012)(b) & Wilkes (2013)(f)

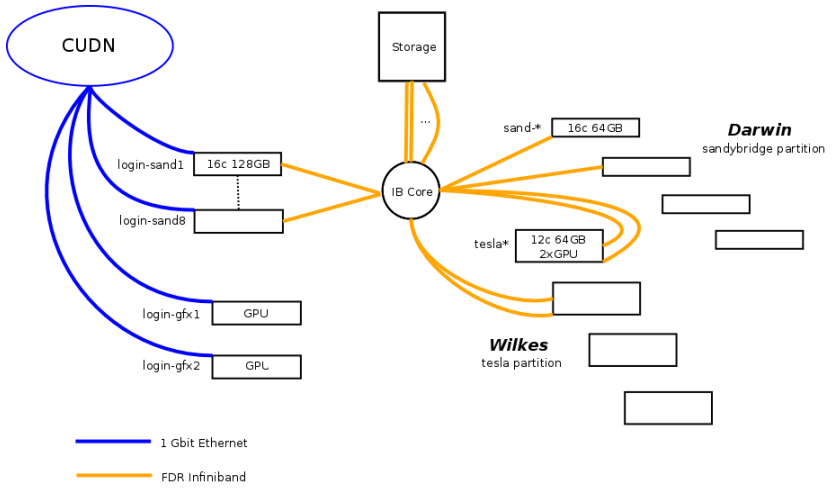
Darwin: an Infiniband CPU Cluster

- Each compute node:
  - 2x8 cores, Intel Sandy Bridge 2.6 GHz.16 cores
  - 64 GB RAM (63900 MB usable).63900 MB
  - 56 Gb/sec (4X FDR) Infiniband.5 GB/sec Infiniband (for MPI and storage)
- 600 compute nodes (300 belong to Cambridge).
- 8 login nodes (login.hpc.cam.ac.uk).
- 1 Petaflop upgrade availability October 2017

Wilkes: a Dual-Rail Infiniband GPU Cluster

- Each compute node:
  - 2 × NVIDIA Tesla K20c GPU.2 GPUs
  - 2x6 cores, Intel Ivy Bridge 2.6 GHz.12 cores
  - 64 GB RAM (63900 MB usable).63900 MB
  - 2 × 56 Gb/sec (4X FDR) Infiniband.2 × 5 GB/sec Infiniband (for MPI and storage)
- 128 compute nodes.
- 8 login nodes (login.hpc.cam.ac.uk).
- Environment shared with Darwin (same filesystems, user environment, scheduler).
- 1 Petaflop upgrade (Wilkes2) early access June 2017

HPCS Production Cluster Schematic



User Environment: Compilers

Intel: icc, icpc, ifort (recommended)

```
icc -O3 -xHOST -ip code.c -o prog
mpicc -O3 -xHOST -ip mpi_code.c -o mpi_prog
```

GCC: gcc, g++, gfortran

```
gcc -O3 -mtune=native code.c -o prog
mpicc -cc=gcc -O3 -mtune=native mpi_code.c -o mpi_prog
```

PGI: pgcc, pgCC, pgf90

```
pgcc -O3 -tp=sandybridge code.c -o prog
mpicc -cc=pgcc -O3 -tp=sandybridge mpi_code.c -o mpi_prog
```


Exercise 6: Modules and Compilers

- Go to the exercises directory of your HPCS training account.

  Hints: Firstly you may need to review Exercise 2 in order to reconnect to your HPCS training account. At the HPCS command prompt, change to the exercises directory (cd ~/exercises).
- Try to compile the hello.c program using the default icc compiler (it will fail because there is a deliberate bug).

  Hints: icc hello.c -o hello
- To fix the problem, open the hello.c file in the gedit editor.

  Hints: Launch gedit in the background by doing gedit&. A gedit window should appear. Remove the word BUG and save the file.

Exercise 6: Modules and Compilers (ctd)

- Try again to compile the hello.c program using the default icc compiler, and run it. You should see "*node* says: Hello, World!".

  Hints: icc hello.c -o hello then run: ./hello.
- Which version of icc did you use? Find this out by listing the current modules loaded.

  Hints: module list — the Intel compiler modules are named intel/cce/*version*. You can also work out the version directly by finding the location of the binary, e.g. doing which icc which should return the path: /usr/local/Cluster-Apps/intel/cce/*version*/bin/icc.
- Compile and run the hello.c program in the exercises directory using a non-default C compiler.

  Hints: E.g. load the latest PGI C compiler (pgcc) with module load pgi. module av will show all possible choices (not all of which are compilers).

Using HPC: Job Submission

Using HPC: Job Submission

- Compute resources are managed by a scheduler: SLURM/PBS/SGE/LSF/...
- Jobs are submitted to the scheduler
  - — analogous to submitting jobs to a print queue
  - — a file (*submission script*) is copied and queued for processing.

Using HPC: Job Submission

- Jobs are submitted from the login nodes
  - — not themselves managed by the scheduler.
- Jobs may be either non-interactive (batch) or interactive.
- Batch jobs run a shell script on the first of a list of allocated nodes.
- Interactive jobs provide a command line on the first of a list of allocated nodes.

Using HPC: Job Submission

- The HPCS has used SLURM since February 2014.
- Jobs may use part or all of one or more nodes
  — the owner can specify `--exclusive` to force exclusive node access (automatic on Wilkes but may change post-upgrade).
- Template submission scripts are available under /usr/local/Cluster-Docs/SLURM.

Job Submission: Using SLURM

- Prepare a shell script and submit it to SLURM:

```
[abc123@login]$ sbatch slurm_submission_script
Submitted batch job 790299
```


Job Submission: Show Queue

- Submitted job scripts are copied and stored in a queue:

```
[abc123@login]$ squeue -u abc123
             JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
            790299 sandybrid     Test3  abc123 PD       0:00      8 (PriorityResourcesAssocGrpCPUMinsLimit)
            790290 sandybrid     Test2  abc123  R   27:56:10      8 sand-6-[38-40],sand-7-[27-31]
```


Job Submission: Monitor Job

- Examine a particular job:

```
[abc123@login]$ scontrol show job=790299
```

Job Submission: Cancel Job

- Cancel a particular job:

```
[abc123@login]$ scancel 790299
```


Job Submission: Scripts

- SLURM
  See slurm_submit.darwin, slurm_submit.wilkes.

```
#!/bin/bash
#! Name of the job:
#SBATCH -J darwinjob
#! Which project should be charged:
#SBATCH -A CHANGEME
#! How many whole nodes should be allocated?
#SBATCH --nodes=1
#! How many tasks will there be in total? (<= nodes*16)
#SBATCH --ntasks=116
#! How much wallclock time will be required?
#SBATCH --time=02:00:00
#! Select partition:
#SBATCH -p sandybridge
...
```

- #SBATCH lines are *structured comments*
  — correspond to sbatch command line options.
- The above job will be given 1 cpu16 cpus on 1 node for 2 hours (by default there is 1 task per node, and 1 cpu per task).

Job Submission: Accounting Commands [HPCS]

- How many core hours available do I have?

```
mybalance

User            Usage |        Account     Usage | Account Limit Available (CPU hrs)
----------  --------- + -------------- --------- + ------------- ---------
abc123             18 |          STARS       171 |       100,000    99,829
abc123             18 |       STARS-SL2        35 |       101,000   100,965
abc123            925 |         BLACKH    10,634 |       166,667   156,033
```

- How many core hours does some other project or user have?

```
gbalance -p HALOS

User           Usage |   Account     Usage | Account Limit Available (CPU hrs)
---------- --------- + --------- --------- + ------------- ---------

pq345              0 |     HALOS   317,656 |       600,000   282,344
xyz10         11,880 |     HALOS   317,656 |       600,000   282,344

(Use -u for user.)
```

- List all jobs charged to a project/user between certain times:

```
gstatement -p HALOS  -u xyz10 -s "2014-01-01-00:00:00" -e "2014-01-20-00:00:00"
       JobID      User   Account  JobName  Partition               End      NCPUS CPUTimeRAW ExitCode      State
------------ --------- --------- -------- ---------- ----------------- ---------- ---------- -------- ----------
14505             xyz10    halos       help sandybrid+ 2014-01-07T12:59:40         16         32      0:9  COMPLETED
14506             xyz10    halos       help sandybrid+ 2014-01-07T13:00:11         16         48      2:0     FAILED
...
```


Job Submission: Single Node Jobs

- Serial jobs requiring large memory, or OpenMP codes.

```
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=

#SBATCH --mem=3994
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=  # For OpenMP across  cores
$application $options
...
```

Job Submission: Single Node Jobs

- Serial jobs requiring large memory, or OpenMP codes.

```
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=1
# Default is 1 cpu (core) per task
#SBATCH --mem=3994
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=  # For OpenMP across  cores
$application $options
...
```


Job Submission: Single Node Jobs

- Serial jobs requiring large memory, or OpenMP codes.

```
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=16 # Whole node

#SBATCH --mem=3994
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=16  # For OpenMP across 16 cores
$application $options
...
```


Job Submission: Single Node Jobs

- Serial jobs requiring large memory, or OpenMP codes.

```
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=8  # Half node

#SBATCH --mem=3994
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=8   # For OpenMP across 8 cores
$application $options
...
```


Job Submission: Single Node Jobs

- Serial jobs requiring large memory, or OpenMP codes.

```
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=16 # Whole node

#SBATCH --mem=3994
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=8   # For OpenMP across 8 cores (using all memory)
$application $options
...
```

Job Submission: MPI Jobs

- Parallel job across multiple nodes.

```
#!/bin/bash
...
#SBATCH --nodes=4
#SBATCH --ntasks=64     # i.e. 16x4 MPI tasks in total.
#SBATCH --cpus-per-task=2
...
mpirun -np 64 $application $options
...
```

- SLURM-aware MPI launches remote tasks via SLURM.
- The template script uses $SLURM_TASKS_PER_NODE to set PPN.

Job Submission: MPI Jobs

- Parallel job across multiple nodes.

```
#!/bin/bash
...
#SBATCH --nodes=4
#SBATCH --ntasks=32     # i.e.  8x4 MPI tasks in total.
#SBATCH --cpus-per-task=2
...
mpirun -ppn 8 -np 32 $application $options
...
```

- SLURM-aware MPI launches remote tasks via SLURM.
- The template script uses $SLURM_TASKS_PER_NODE to set PPN.

UNIVERSITY OF CAMBRIDGE 96 of 107

Job Submission: Hybrid Jobs

- Parallel jobs using both MPI and OpenMP.

```
#!/bin/bash
...
#SBATCH --nodes=4
#SBATCH --ntasks=32     # i.e.  8x4 MPI tasks in total.
#SBATCH --cpus-per-task=2
...
export OMP_NUM_THREADS=2   # i.e. 2 threads per MPI task.
mpirun -ppn 8 -np 32 $application $options
...
```

- This job uses 64 CPUs (each MPI task splits into 2 OpenMP threads).

Job Submission: High Throughput Jobs

- Multiple serial jobs across multiple nodes.
- Use srun to launch tasks (job steps) within a job.

```
#!/bin/bash
...
#SBATCH --nodes=4
...
cd directory_for_job1
srun --exclusive -N 1 -n 1 $application $options_for_job1 > output 2> err &
cd directory_for_job2
srun --exclusive -N 1 -n 1 $application $options_for_job2 > output 2> err &
...
cd directory_for_job64
srun --exclusive -N 1 -n 1 $application $options_for_job64 > output 2> err &
wait
```

Exercise 7: Submitting Jobs

- Submit a job which will run a copy of your hello program on all cores of 4 of the 24-core nodes which are available for training.

Hints:

1. Edit the script job_script in your exercises directory. Set:
   #SBATCH –nodes=4
   #SBATCH –ntasks=96
   application="./hello"
   In the module section, make sure that the module you used to compile hello is also loaded (last).
2. Submit the job with sbatch job_script. The jobid is then printed.
3. Watch the job in the queue with squeue.
4. After it has disappeared, open the output file slurm-jobid.out in your editor. There should be 24 "Hello, World!" messages from 4 different nodes.

Exercise 7: Submitting Jobs (ctd)

- Experiment with changing the number of nodes and tasks by changing and submitting job_script (you are limited to 8 nodes in total).

Job Submission: Interactive [HPCS]

- Compute nodes are accessible via SSH while you have a job running on them.
- Alternatively, submit an interactive job:

```
sintr -A MYPROJECT -N2 -n16 -t 2:0:0
```

- Within the window (screen session):
  - Launches a shell on the first node (when the job starts).
  - Graphical applications should display correctly.
  - Create new shells with ctrl-a c, navigate with ctrl-a n and ctrl-a p.
  - ssh or srun can be used to start processes on any nodes in the job.
  - SLURM-aware MPI will do this automatically.

Job Submission: Array Jobs

- This feature varies between versions.
- *http://slurm.schedmd.com/job_array.html*
- Used for submitting and managing large sets of similar jobs.
- Each job in the array has the same initial options.
- SLURM

```
[abc123@login]$ sbatch --array=1-7:21,3,5,7 -A STARS-SL2 submission_script
Submitted batch job 791609

[abc123@login-sand2]$ squeue -u abc123
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         791609_1 sandybrid      hpl    abc123  R       0:06      1 sand-6-32
         791609_3 sandybrid      hpl    abc123  R       0:06      1 sand-6-37
         791609_5 sandybrid      hpl    abc123  R       0:06      1 sand-6-59
         791609_7 sandybrid      hpl    abc123  R       0:06      1 sand-7-27
```

791609_1, 791609_3, 791609_5, 791609_7

i.e. ${SLURM_ARRAY_JOB_ID}_${SLURM_ARRAY_TASK_ID}

SLURM_ARRAY_JOB_ID = SLURM_JOBID for the first element.

Job Submission: Array Jobs (ctd)

- Updates can be applied to specific array elements using ${SLURM_ARRAY_JOB_ID}_${SLURM_ARRAY_TASK_ID}
- Alternatively operate on the entire array via ${SLURM_ARRAY_JOB_ID}.
- Some commands still require the SLURM_JOB_ID (sacct, sreport, sshare, sstat and a few others).

Scheduling

- SLURM scheduling is multifactor:
  - QoS — payer or non-payer?
  - Age — how long has the job waited?
    Don't cancel jobs that seem to wait too long.
  - Fair Share — how much recent usage?
    Payers with little recent usage receive boost.
  - sprio -j jobid
- Backfilling
  - Promote lower priority jobs into gaps left by higher priority jobs.
  - Demands that the higher priority jobs not be delayed.
  - Relies on reasonably accurate wall time requests for this to work.
  - Jobs of default length will not backfill readily.

Wait Times

- The cluster is currently very busy and we do not preempt.
- 36 or 12 hour job walltimes are permitted.
- This sets the timescale at busy times (*without* backfilling).
- Use backfilling when possible.
- Short (1 hour or less) jobs have higher throughput (dedicated nodes).
- Each user can submit one very high priority job with --qos=INTR.
- We are upgrading significantly in 2017!

Checkpointing

- Insurance against failures during long jobs.
- Restart from checkpoints to work around finite job length.
- Application native methods are best. Failing that ...
- Darwin & Wilkes nodes currently provide Berkeley Lab Checkpoint/Restart (BLCR)
  cr_run, cr_checkpoint, cr_restart
- sbatch --checkpoint=minutes
- scontrol checkpoint restart jobid

Job Submission: Scheduling Top Dos & Don'ts

- **Do ...**
  - Give reasonably accurate wall times (allows backfilling).
  - Check your balance occasionally (mybalance).
  - Test on a small scale first.
  - Implement checkpointing if possible (reduces resource wastage).
- **Don't ...**
  - Request more than you need
    — you will wait longer and use more credits.
  - Cancel jobs unnecessarily
    — priority increases over time.