Professional Skills for M.Sc in Climate Change and AI
Data and Models

Lecture 1B: Computing Hardware and Environments: where to compute?

Bryan Lawrence

May 5, 2023

Lecture 1B: Where to compute?

Contributes to objective

An understanding of the hardware component of the digital infrastructure available to UK academia (and you personally) for working with climate data, models, and machine learning, with some experience of

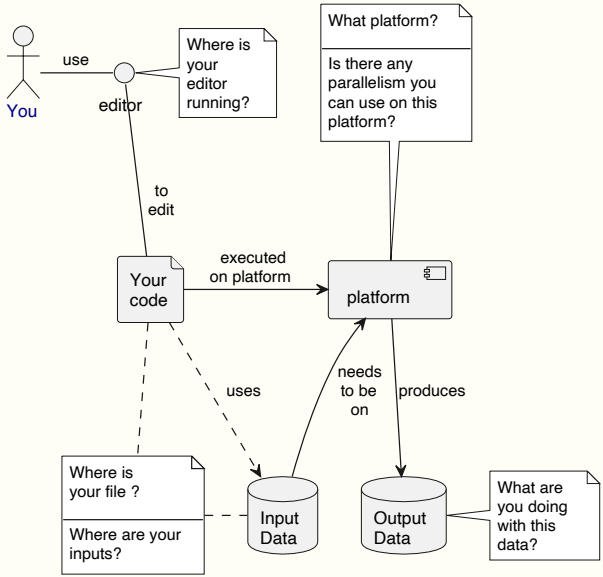
1. notions of parallel and batch computing.
2. working via SSH on a remote machine,
3. using containers (hopefully on your laptop),

Specific Goals

- You will understand the difference between on-node and off-node parallelism, and understand the notion of “embarassingly” parallel tasks.
- You will know about the university computing provision, and what is available from UKRI for bigger projects.
- You’ll understand why we going to provide you with an introduction to the SLURM scheduler.

Modes of Computing

- Local
  - Apps
  - Command Line
- Local but kind-of-remote
  - Containers (Docker)
  - Virtual Machines
- Cloud
  - Software-as-a-Service
  - Platform-as-a-Service
  - Infrastructure-as-Service
- Remote
  - Interactive Terminal
  - Jupyter notebook
  - Batch Computing

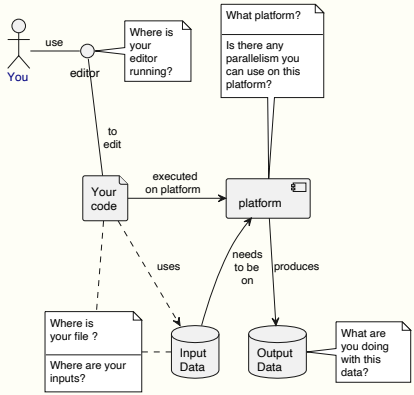


Key Questions

It's always about productivity (how can you work with your code efficiently?), performance (can you actually get an answer in a reasonable time?), and portability (can I develop in a different place than I execute? Can I execute in different places?).

Key questions:

1. Where do I want to develop?
2. Where is my main dataset? Can i use a subset for developing?
3. Can I get my main data onto the target platform?
4. What will I do with my data products?
5. Where and how can I use parallelism?



Basic Ideas: Serial, Parallel, Checkpointing

Consider a simple algorithm

$$A(\text{stuff}) = B(\text{stuff}) + C(\text{stuff}) + D(\text{stuff})$$

Serial Computation

- The algorithm is broken into tasks, with each calculated one at a time, one after another.
- In this example, we can calculate $b = B(\text{stuff})$ and then do $c = C(\text{stuff})$ and so on.
- (We might build up an answer as we go, or store the results as we go, so there are two possible sub- algorithms: $A = ((b) + c) + d$ or $A = b + c + d$).
- Note that if it was taking a long time, we could stop the computation and checkpoint it at $A^{\text{partial}} = b + c$ and then later do $A = A^{\text{partial}} + d$.

Basic Ideas: Serial, Parallel, Checkpointing

Consider a simple algorithm

$$A(\text{stuff}) = B(\text{stuff}) + C(\text{stuff}) + D(\text{stuff})$$

Parallel Computation

- The algorithm is broken into tasks, executed at the same time, before the task results are all gathered into a result.
- Now we are defintely calculating $b = B(\text{stuff})$, $c = C(\text{stuff})$, and $d = D(\text{stuff})$ at the same time so that we can simply calculate $A = a + b + c + d$ trivially at the end.
- Note that unless we can break each of these functions A, B, C, D into more pieces we can't checkpoint this!

Types of parallelism

Consider some things I might want to do with a collection of surface pressure maps (arrays of $x, y = \text{longitude}, \text{latitude}$):

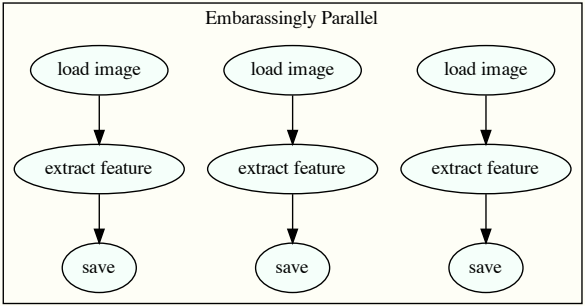
1. Process each map to extract a feature of interest (storms?), and save the features.
2. Take each of those features (storms) and count the number or each type (e.g cyclones, polar low) that were present.
3. Take each of those types, find the one with the lowest pressure.
4. We want to find the lowest pressure in our collection of storm types and label each of the storms in our images with the central low as a percentage of the deepest low we have seen before.

Embarassingly Parallel

Consider some things I might want to do with a collection of surface pressure maps (arrays of $x, y = \text{longitude}, \text{latitude}$):

1. Process each map to extract a feature of interest (storms?), and save the features.

The first task is embarassingly parallel: we can process each task completely independently. If we had 1000 maps, and 10 processors, we could simply parcel them up into groups of 100 and send them off.



Map-Reduce

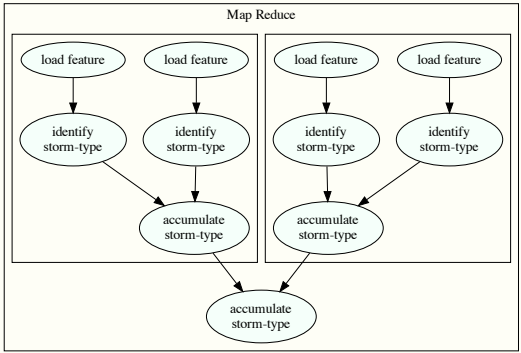
Consider some things I might want to do with a collection of surface pressure maps (arrays of $x, y = \text{longitude}, \text{latitude}$):

2. Process each map to extract a feature of interest (storms?), and save the features.
3. Take each of those types, find the one with the lowest pressure.

The second and third tasks are not so simple, although they could be addressed with map-reduce:

E.g. Break the collection into groups, do the count on each group, gather the partial counts and sum the partial counts into the result.

We have mapped the data onto the processors, got partial results and reduced the set of partial results to one result.

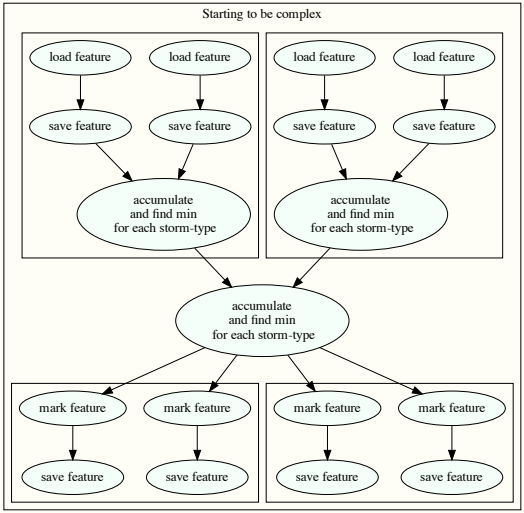


More complexity in the graph

We like problems like those, but consider one more example:

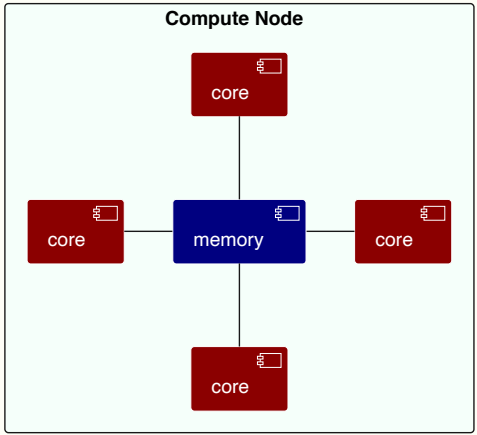
4. We want to find the lowest pressure in our collection of storm types and label each of the storms in our images with the central low as a percentage of the deepest low we have seen before

This involves a more complex pattern because of the shared dependency before we do something to each image.



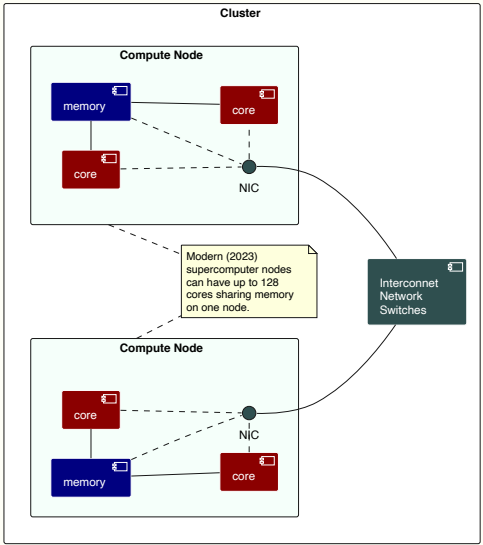
Shared Memory

- Processors (cores) on a node share main memory (but there is other memory).
- A programme running on a single computer (node) can generally rely on all the (CPU) cores having access to the same memory.
- Relatively easy to programme because the programmer doesn't have manage datacore locality. The compiler can do a lot of the heavy lifting.



Distributed Memory

- Processors have their own local memory.
- Changes made to memory on one processor are not known to another unless they are explicitly moved from one to another.
- Recall our simple example A, B, C, D? If we do B, C, D on different nodes, we have to move stuff to each node at the beginning, and then move b, c, d back to one node to calculate A.
- Requires libraries (e.g the Message Passing Interface, MPI) for managing the data movement, and the programmer needs to explicitly use them.



Why we care about the difference

Because for data analysis, we care about:

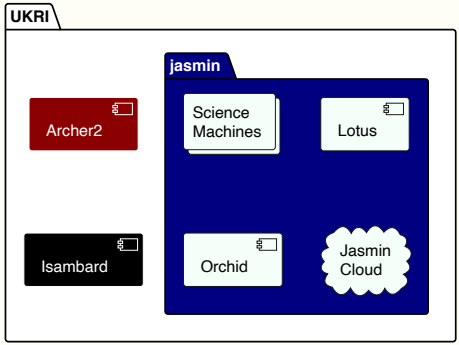
- How much memory we have and need,
- How many cores we have and need, and
- How we get access to the cores and memory if they are on multiple nodes.

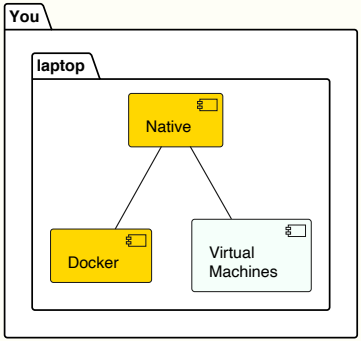
For most purposes you are only going to care about threads and processes.

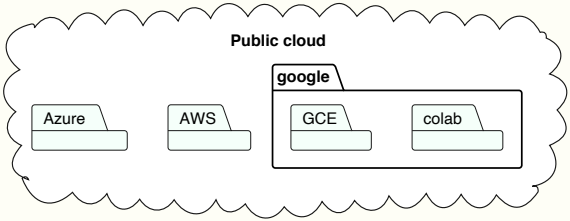
- A programme using threads can break a task manipulating one array up into threads, each of which is manipulating a piece of the array.
- A programme using processes will copy parts of the array between processes and then operate on the parts.
- It's the copy that hurts, but on Python, threads behave badly too.

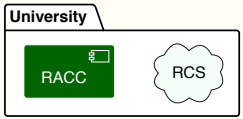
So for this course, we are introducing you to the use of a scheduler (slurm) so that you run programmes in parallel or if you need parallelism within a Python programme, we suggest Dask. We won't introduce you to Dask in detail, but we will show how libraries can support it, and how you can configure such libraries to use parallelism.

The landscape





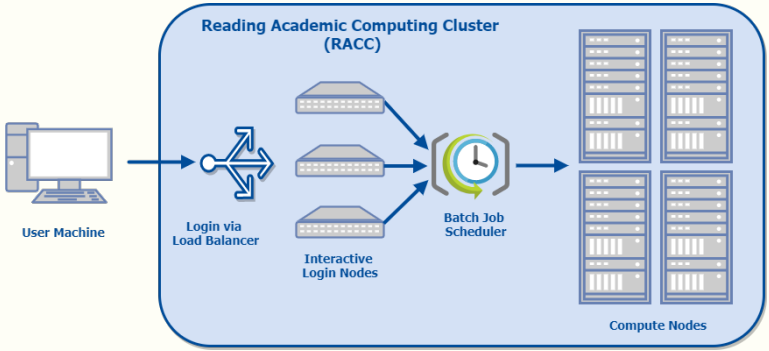




Reading University Academic Computing Cluster - RACC

University resource, provides some compute (CPU and GPU), and storage.
The size of the cluster varies as university users can buy in hardware as part of projects.

- heterogeneous CPU and GPU nodes
- storage split into home, scratch, and (chargeable) silver and gold.
- slurm scheduler.

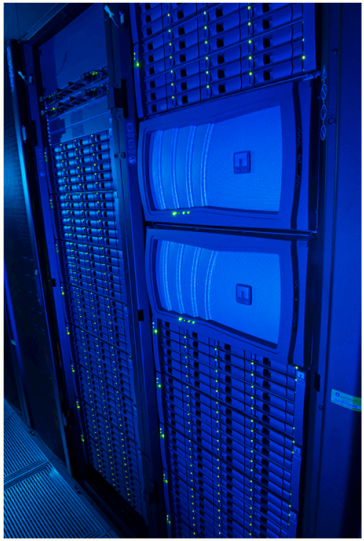


https://research.reading.ac.uk/act/knowledgebase/racc-introduction/

JASMIN

NERC data analysis system deployed at STFC's Rutherford Appleton Laboratory:

- Login and data transfer nodes.
- Interactive compute nodes (" the science machines").
- High performance computing with fast and/or big storage:
  - Batch computing on Lotus (CPU) and Orchid (GPU) — using the slurm scheduler.
- Private Cloud.
- Multiple tiers of storage for different kinds of requirements.
- Hosts the CEDA data archive (petabtyes of data online and accessible).



https://jasmin.ac.uk

JASMIN Services, March 2021

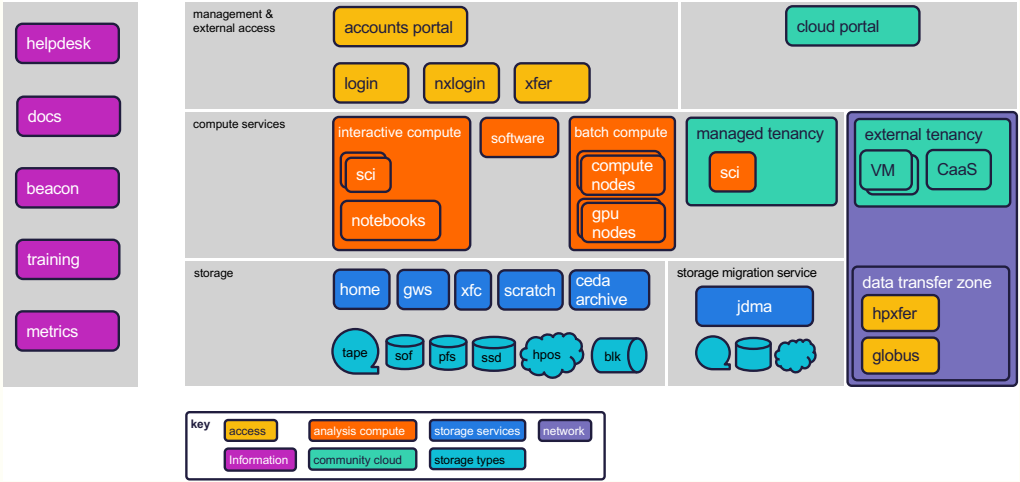


Figure: Matt Pritchard

ARCHER2

(EPCC): Provides CPU based batch supercomputing:

- Compute Nodes: 5,860 nodes (5,276 standard memory, 256GiB; 584 high memory, 512GiB)
- Login Nodes: (same as high memory compute)
- Processors: 2 x AMD EPYCTM 7742, 2.25 GHz, 64-core
- Cores per node: 128 (2 x 64-core processors)

https://www.archer2.ac.uk/

Uses slurm scheduler. Note that the home storage (1 PB) is not available on the compute nodes. The work storage (14.5 PB) is.

ISAMBARD

ISAMBARD is the collective noun for several systems which provide both computing and test environments for both CPU and GPU (uses PBSpro scheduler):

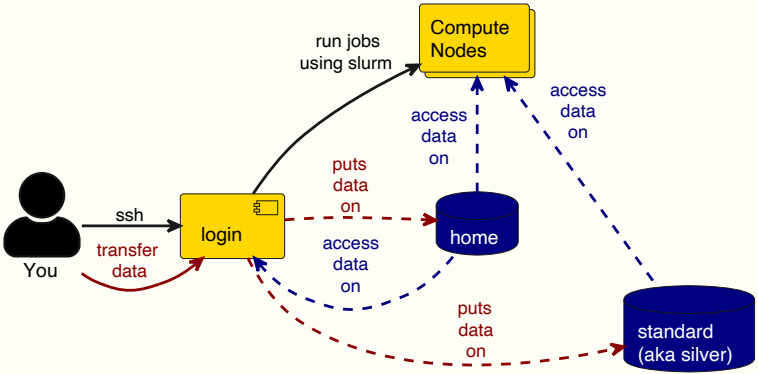
- XCI: ARM based supercomputer
  - 329 Marvel ThunderX2 ARM nodes (160 @ 256GB, 169 @ 512GB).
  - Each node has two 32-core processors.
- A64FX Fujitsu
  - 72 ARMv8.2 nodes each with 48 cores and 32 GB.
- MCS: Multi-architecture Comparison system
  - 6 different node architecures: Intel (Broadwell, Cascade Lake, Knights Landing); AMD Rome, IBM Power 9, Nvidia P100 and V100.

https://gw4-isambard.github.io/

Accessing the RACC

Read the docs!

https://research.reading.ac.uk/act/knowledgebase/racc-introduction/



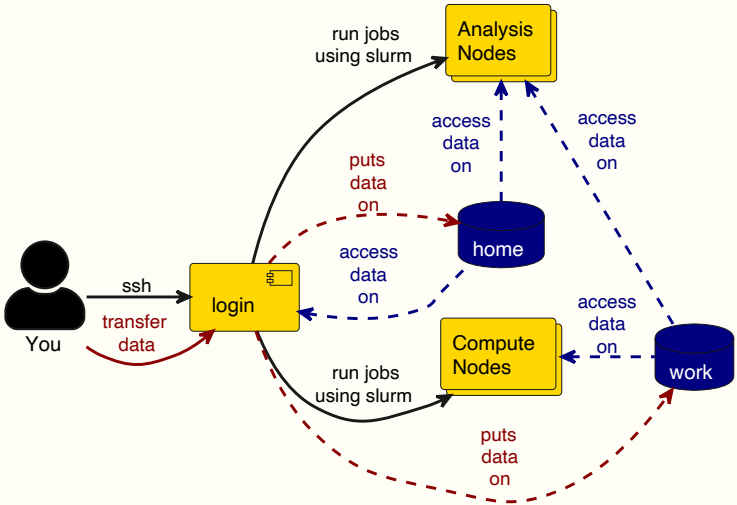
- Use ssh to get in.
- Transfer data in via or push or pull using ssh (more anon)
- (I think) the home directory is backed up.

Accessing ARCHER2

Read the docs!

https://docs.archer2.ac.uk/

- Use ssh to get in.
- Transfer data in via or push or pull using ssh (more anon)
- Only home is backed up.

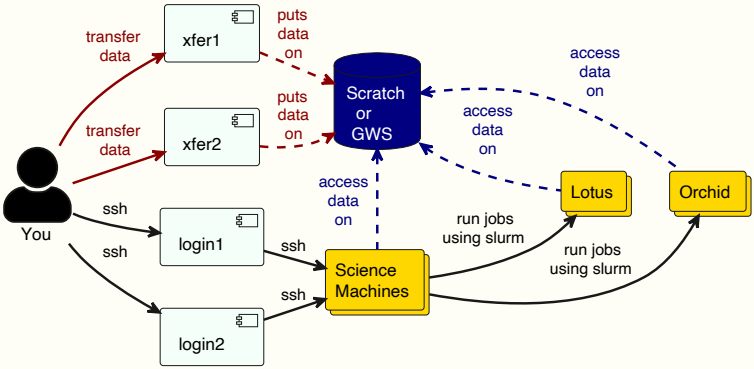


Accessing JASMIN

Read the docs!

https://help.jasmin.ac.uk/article/4856-jasmin-users-guide

- Use ssh to get in.
- Transfer data in via or push or pull using ssh (more anon)
- Only home is backed up (there are four different types of storage).



Cloud Computing or Batch Computing?

If our interest is in running our own code on our own data, then the difference comes down to:

- HPC: platform-as-a-service where you get access to all of a core or a node for a period of time, but someone else is responsible for the computing environment.
  - You get all that core or node until your job finishes or your run out of allocated time.
- Cloud: infrastructure-as-a-service, where you can get access to all or part of the memory of a node, and you are responsible for the computing environment (although you might get a menu of choices of environment).
  - If you have all the memory, you have all the node resources, if not, you are sharing the node resources, until you are finished (or pre-empted).

Interesting adjunct questions are:

- Am I sharing the network access (to data and/or other nodes)?
- Where is the data you want to access? Is it network local to the compute you want, and if not, can you get it there?
- Who is paying and what does it cost?

Batch Computing

Basic principles:

- Users submit “jobs” (programmes) that utilise some fraction of the computer to “queues” (also known as “partitions”) and the job executes when there is space and time on the system. While the job is running the user generally does not interact with the job.
- The compute resources are managed by a scheduler that tries to maximise the usage of the computer (all the nodes are busy) and optimise the throughtput (user jobs run as soon as possible).
- There is usually some mechanism to prioritise either particular users or types of jobs. For example: Archer2 tries to prioritise big jobs as the system was designed for capability. JASMIN tries to deliver capacity.
- There are generally login nodes where people can do small interactive tasks, including compiling and limited data analysis.

All the systems of interest to us (RACC, JASMIN, ARCHER2) use the slurm scheduler.

Summary

- A quick tour of the way we might want to be computing,
- Some questions to ask about where we might plan to compute,
- A very quick introduction to concepts of parallelism, and
- An intro to the UK computing landscape.

Next time:

- Introduction to tools and methods of accessing remote computing.