Introduction to High Performance Computing (HPC) – Session 2

using the "**C**omputational **S**hared **F**acility" (CSF)

Research Platforms, Research IT, IT Services

Course materials / Slides available from:
https://ri.itservices.manchester.ac.uk/course/rcsf/

CSF online documentation
https://ri.itservices.manchester.ac.uk/csf3/

Contact Research Platforms via the Connect Portal
https://ri.itservices.manchester.ac.uk/csf3/help/

Recap slide skipped in training room

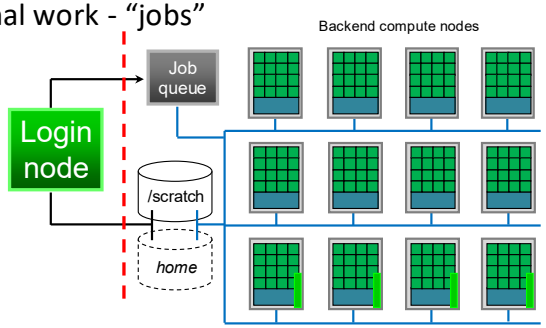
Housekeeping

- Please let me know if you're leaving
  - Morning: Session one: 10am – 12:30pm (practicals 1, 2, & 3)
  - Afternoon: Session two: 1:30pm - 4pm (practicals 4 & 5)
- 1-to-1 help is available if needed during exercises. We'll describe how this works before the first one.
- Please give feedback on this course
  - Quick form at https://goo.gl/forms/zfZyTLw4DDaySnCF3 (choose "*Introduction to HPC (Using CSF)*")
  - Feedback is important to help us improve our courses
  - Records your attendance on the course

Recap slide skipped in training room

Jobs, Jobscripts and the Batch System

- We want to do computational work - "jobs"



- You decide:
  - Which program(s) to run
  - Which resources it needs (#cores, CPU type, memory, GPU?)
  - How much time the job will need to complete its work
  - Which of your folders ("directory") to run the job in
- You'll put these requirements into a *jobscript* file
- Then submit your *jobscript* to the batch system ("Slurm")
- Slurm decides *when* the job runs and on which compute node(s). It ensures you get all of your requested resources.

Recap slide skipped in training room

A simple Jobscript – *Serial* (1 core)



`#!` on first line only (a special line)

First line indicates we use the *bash* scripting language to write our jobscript.

`myjob.txt`

`#SBATCH` indicates a **batch system parameter** to specify our job requirements. We'll use various combinations of these.

```
#!/bin/bash --login

#SBATCH -p serial
#SBATCH -n 1
#SBATCH -t 5

# Let's do some work
date
hostname
sleep 120
date
```

`-p` (`--partition=`) think of this as the queue, for serial (1-core) jobs in this case.

`-n` (`--ntasks=`) number of cores, which is 1 by default for serial jobs (**optional**).

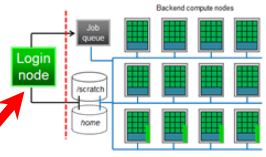
`-t` (`--time=`) maximum "wallclock" time the job is allowed to run for. Various formats. 5 is 5 minutes. 4-0 would be 4 days (0 hours).

`#` lines are just comments - anything on the line after it will be ignored.

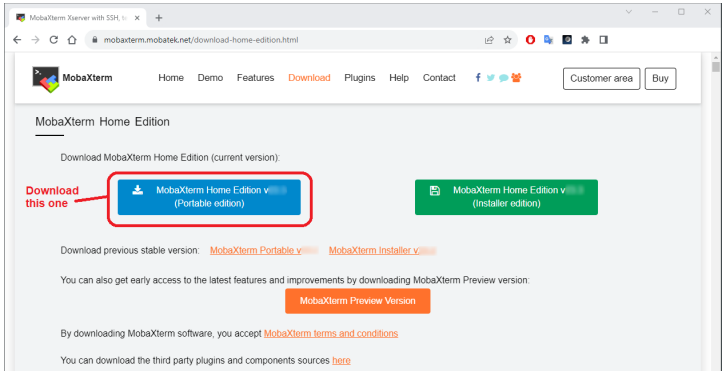
Actual Linux commands we run in our job. They will execute on a compute node.

Recap slide skipped in training room

Connect to CSF from Windows



- Access the CSF from a PC / laptop using an SSH (**S**ecure **Sh**ell) app
  - Sometimes called a "terminal".
  - There's no web-site or other fancy GUI on the CSF – use the "command-line".
- **Windows users** need to install a free *terminal* app called MobaXterm
- https://mobaxterm.mobatek.net/download-home-edition.html
  the Home edition (portable edition) does *not require* Administrator rights - just *extract* the small .zip file in your P-Drive or USB stick for example.



1. Download using the blue box.
2. Once downloaded, *right-click* on the .zip file and select:

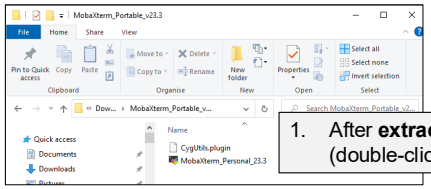
   "Extract all …"

   This will *unpack* the .zip file to a folder.

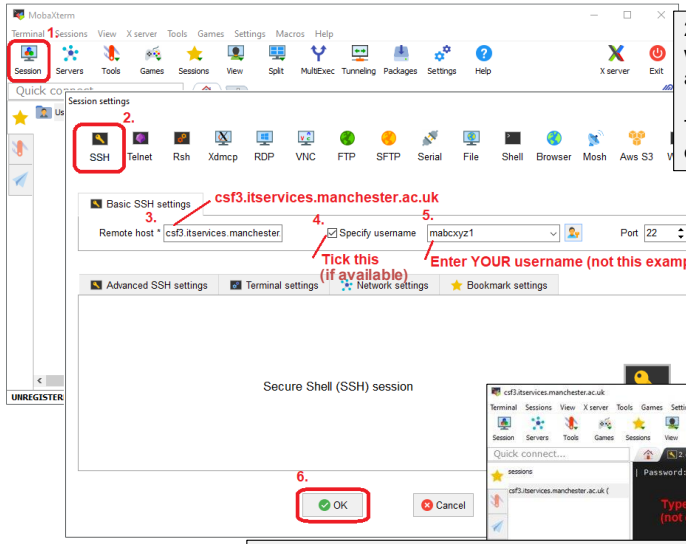
Recap slide skipped in training room

MobaXterm "Session"

(username saved in the session setup)



1. After **extracting** the .zip file, start MobaXterm_Personal_*xy.z* (double-click on the icon)



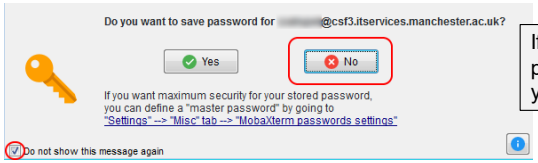
2 (1-6). Create a "Session" which saves the CSF's details along with *your* **username**.

This is needed to make file drag-n-drop work (see later.)

3. This will then start to log you into the CSF – it will ask for your password. Type carefully!



4. See slide about 2FA – you may be asked for DUO after your password



If asked to save your password, we recommend you say "No", for security.

Recap slide skipped in training room

Recap slide skipped in training room

Connecting from Linux / Mac

- From MacOS using a *Terminal* window (after installing Xquartz)

```
ssh -Y username@csf3.itservices.manchester.ac.uk
```

UPPERcase Y

Central IT Services username.
Answer 'Yes' to continue *if* asked.
Enter central IT password when asked (same as for email)

- From Linux using a Terminal window

```
ssh -X username@csf3.itservices.manchester.ac.uk
```

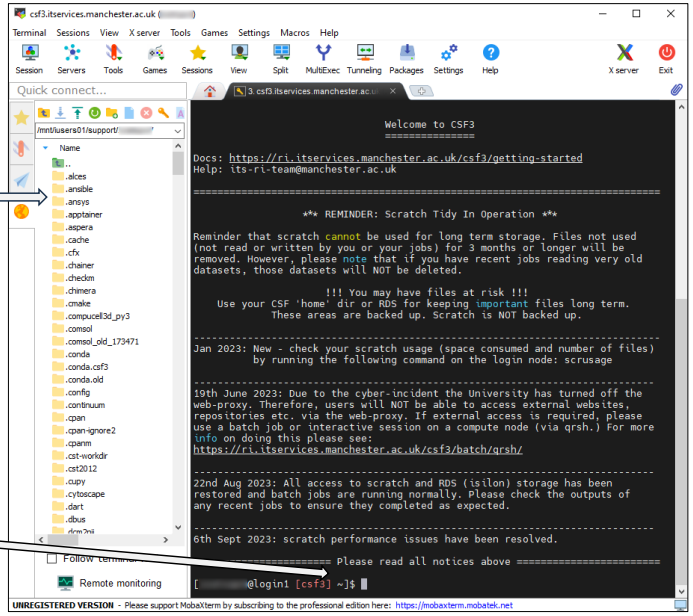
UPPERcase X

Central IT Services username.
Answer 'Yes' to continue *if* asked.
Enter central IT password when asked (same as for email)

- Finished using CSF? Log out with: `logout` or `exit`

Drag-n-drop file browser for upload / download

(new users won't have as many items in the list!)



We're on (one of) the CSF login nodes. Any commands you use will be typed "at the prompt", which shows your username and current directory (folder.)

https://ri.itservices.manchester.ac.uk/course/rcsf/

https://ri.itservices.manchester.ac.uk/csf3

ACCESSING APPLICATION S/W

Modules

Access to Application Software

- Lots of different pieces of software installed
  - Many different applications
  - Different *versions* of an application
  - Need to ensure job knows where an app is installed
    - Try `echo $PATH` to see all directories the CSF will look in
- Use "*modules*" to set up *environment* for software
  - In your jobscript, add some `module` commands
  - Sets up all necessary *environment variables*
  - Apps use these *env vars* to get various settings
  - Can also run `module` commands on the login node (e.g., to check what apps are available)

Module Commands

- `module avail` – lists all available modules
- `module search` *`keyword`* – lists all modules with *keyword* in their name
- `module list` – lists currently loaded modules
- `module load` *`modulename`* – loads module
- `module unload` *`modulename`* – unloads module
- `module purge` – **unload all modules**
- `man module` – man pages for the module command
- Examples:

```
module load apps/binapps/matlab/R2024b
module load apps/intel-19.1/amber/20-bf12-at21-bf12
module load apps/gcc/R/4.4.1
module unload apps/binapps/starccm/18.02-double
module help compilers/intel/19.1.2
module load tools/gcc/cmake/3.28.6
```

- See documentation for more info
  https://ri.itservices.manchester.ac.uk/csf3/software/modules/

Modulefile settings

- What "settings" do modulefiles actually make?
  - Depends on the application (eg the installation instructions)
- Try the following commands on the login node:

```
which matlab
/usr/bin/which: no matlab in(/usr/local/bin:.....

module load apps/binapps/matlab/R2024b

which matlab
/opt/apps/apps/binapps/matlab/R2024b/bin/matlab
```

- This shows that the modulefile made the matlab 2024b installation available.
- A job can do this to run that version of matlab.
- If interested, to see all of the settings that a modulefile will make:

```
module show apps/binapps/matlab/R2024b
```

But the idea is you don't need to know the settings - modulefiles take care of the details so you can concentrate on what your jobs actually *do* with the application.

- See documentation for more info
  https://ri.itservices.manchester.ac.uk/csf3/software/modules/

Loading modulefiles: On login nodes OR in the jobscript

Inherit from the login node (**not recommended**)

Jobs in Slurm will inherit any modulefile settings (i.e. loaded modules) from the login node at the point when you *submit* (sbatch) the job.

```
# On the login node:
module load apps/R/4.4.1
sbatch myjob.txt
```

myjob.txt

```
#!/bin/bash --login
#SBATCH -p serial
#SBATCH -t 2-0

# We'll use whichever version
# of R was loaded on the login
# node when job was submitted.

# Which version of R did I use
# 6 months ago???
R CMD BATCH myscr.R
```

Only in the jobscript (**recommended!**)

```
# On the login node:
sbatch myjob.txt
```

myjob.txt

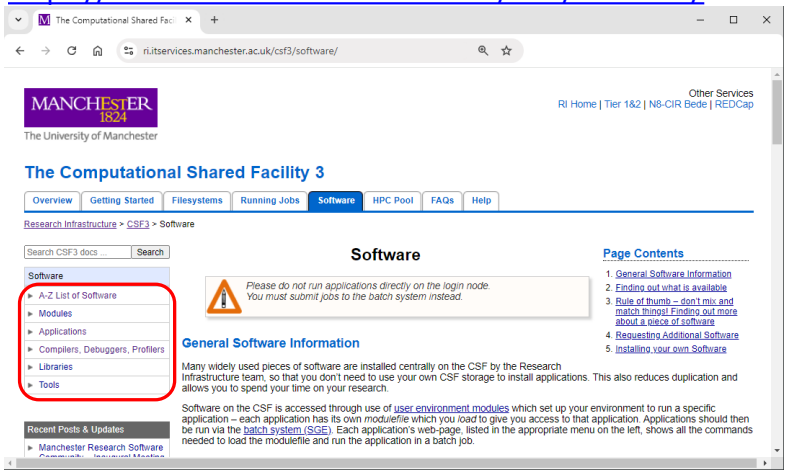
```
#!/bin/bash --login
#SBATCH -p serial
#SBATCH -t 2-0

# Start with a clean env then
# load module inside jobscript
module purge
module load apps/R/4.4.1

# We know the version of R!
R CMD BATCH myscr.R
```


Which Modulefiles to Load

- How do I know which modulefile to load for a particular app?
  - https://ri.itservices.manchester.ac.uk/csf3/software/



A note about our documentation

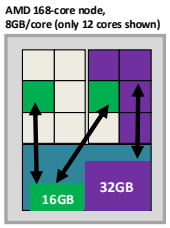
- Over the summer we change the batch system from SGE to Slurm
  - SGE uses #$ as the jobscript *sentinel*
  - Slurm uses #SBATCH (as we've seen earlier)
  - Our applications documentation is being updated to convert example jobscripts from SGE to Slurm
  - If you see #$ in our documentation, you'll need to write the equivalent Slurm jobscript (using #SBATCH).
  - See our SGE-to-Slurm guide, which shows how #$ flags map to #SBATCH flags https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/sge-to-slurm/

PARALLEL COMPUTING

Background

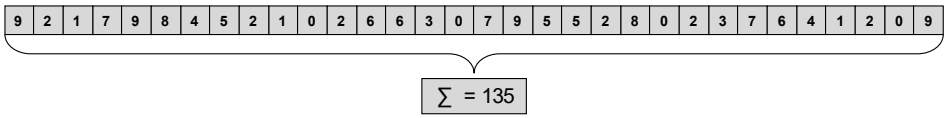
Motivations for Parallel Computing

- CSF compute nodes have multiple CPU cores (32, 40, 168)
- Many applications can use multiple cores to speed up the computation
  - Split the "work" over multiple CPU cores
    - Each core does a small(er) part of the computation, all in parallel
    - "Data parallelism" (same instructions run on each portion of "data")
  - May need to *combine* partial results together at end
  - Should get the final result quicker
    - Ideally *N* cores giving results *N* times quicker
- Also provides access to more memory
  - Each core has access to 8GB RAM (AMD nodes)
    - Ideally *M* cores for *M* times larger problem
- Both of the above!

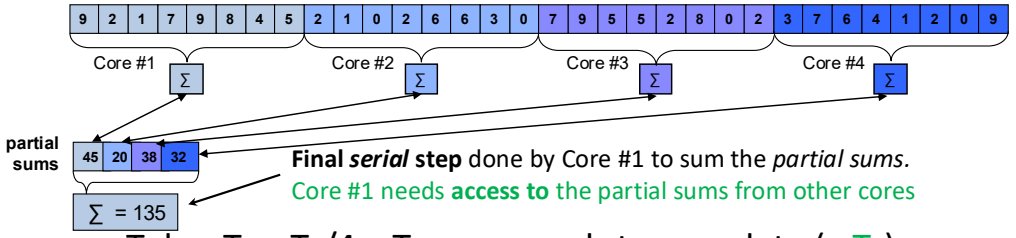


Simple example: sum a list of numbers

- Could do this example manually with 4 volunteers
- 1-core: sum = sum + number$_i$ (for i = 1 to N)
  - Let's say it takes T_1 seconds to complete



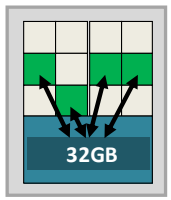
- 4-cores: Each core sums a smaller list of numbers



  - Takes $T_4 \approx T_1/4 + T_{serial}$ seconds to complete ($< T_1$)

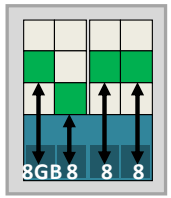
Parallel Job Type #1 - single node

- A program runs on *multiple CPU cores* of **one** compute node
- Two common techniques used by apps:
  - Typically, one copy of the program runs
    - *"Shared memory"* (all cores see same memory)
    - Cores synchronize access to shared memory (data)
    - Look for "OpenMP" / "multi-threaded" / "Java threads" … in an application's docs



Shared Memory

  - Or coordinated copies of the program run, each communicating with each other
    - *"Distributed memory"* (each core has its own mem)
    - They communicate to share data, results
    - Look for "MPI" or "message passing" in the application's docs

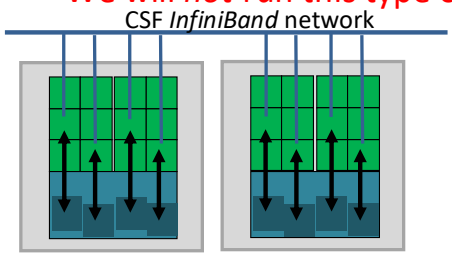


Distributed Memory

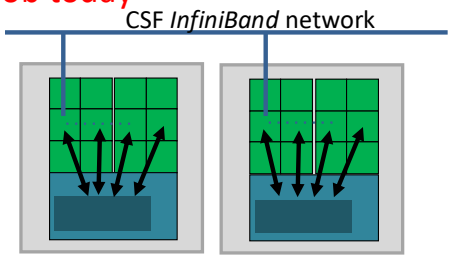
- **Your app must have been written to use one (or both) of the above parallel techniques!**
- We'll run this "single compute-node" type of job today

Parallel Job Type #2 - multi-node

- Running a program over *several* compute nodes (and the 40 cores on each of those nodes)
  - Must be the "MPI" / "message passing" style of app (see above)
  - Uses *more* cores than available in a *single* compute node
    - On CSF we require you to use *all* of the cores in *each* compute node
  - They communicate to share data, results etc (as before)
    - Over the fast internal *InfiniBand* network
    - Possibly via shared memory as before, if on same compute node
- **Your app must have been written to support this!**
- We will *not* run this type of job today



Distributed Memory



Hybrid Memory (often MPI+OpenMP)

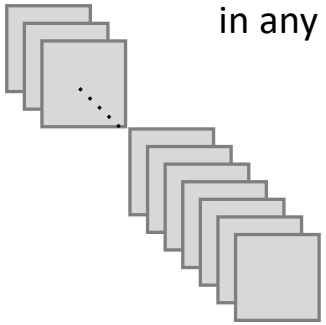
Note: the diagrams only show a few cores in use for simplicity. On the CSF you must use *all* 40 cores in *each* node.

Parallel Job Type #3 - High Throughput Computing (HTC) [Covered Later]

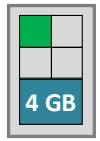
- Lots of *independent* computations. EG:
  - Processing lots of data files (e.g., image files)
  - Running the same simulation many times over with different parameters ("parameter sweeps")
- Run many copies of your program
  - Programs may be serial (single core) but running lots of them at once. They don't communicate.
- Easy to do on CSF. See also the UoM Condor Service (formerly the EPS Condor Pool)
  - Free resource, uses UoM idle desktops over night

Example: Image Analysis

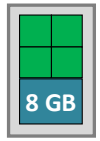
- High Throughput Computing
  - Not all s/w is "HPC" / parallel
  - But you might have *lots* of data
  - Each image takes 1hr to process (and are *independent* - process in any order)



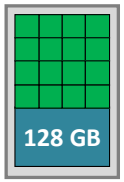
Example: 10,000 image scans to be analysed by an image processing application. Each image takes *1 hour* to process.



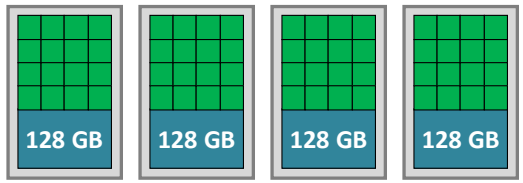
Laptop: 1 copy of software running.
Over 1 year to complete!!



Desktop: 4 cores, 4 copies of software running.
~100 days to complete!



Single HPC compute node: 16 copies of software running.
~26 days to complete



Multiple HPC compute nodes: 64 copies of software running.
~6 days to complete

Which style of parallel job to use

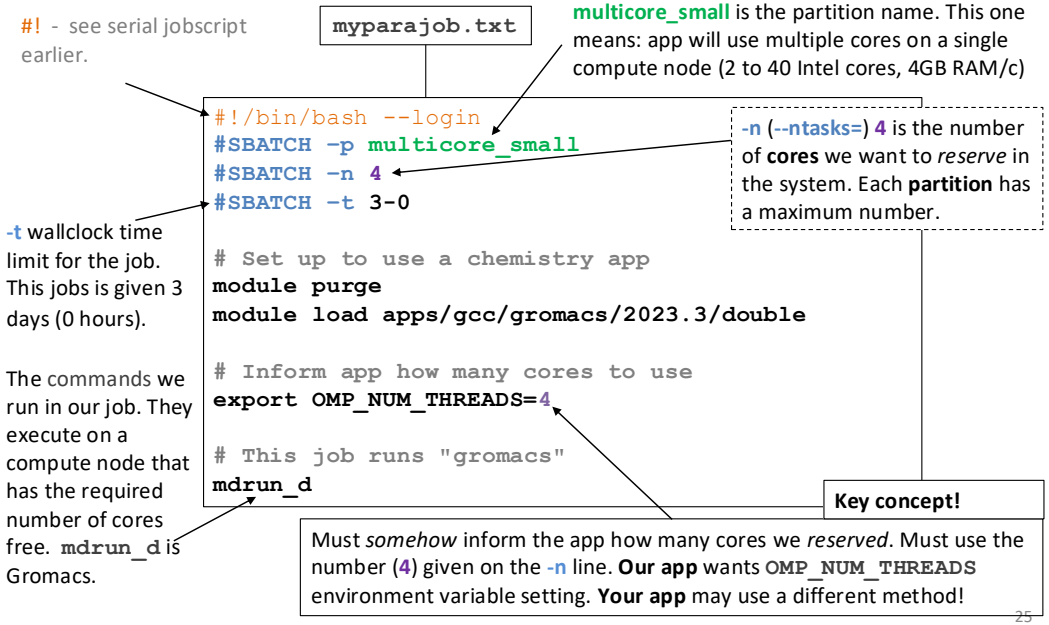
- Mostly determined by the capability of your app
  - Is it serial (1-core) only? Is it multi-core (single-node) only? Is it multi-node capable?
- A serial app will only ever use 1 core
  - But run as an HTC job, you can still process lots of data in parallel
    - Use many cores, running multiple *independent* jobs (see later)
- Parallel app using only *shared memory*
  - "OpenMP", "multithreaded", "Java threads", "shared memory"
  - Can only use 1 compute node (2--40 Intel or 2--168 AMD cores)
- Parallel app using *distributed memory*
  - "MPI" (message passing interface), "distributed memory"
  - Can use many cores across multiple compute nodes
  - But consider: the network
    - Communication faster *within* the same compute node
    - Communication slower on network *between* nodes
    - Apps may not speed up, the more cores (and nodes) you use (see later)

Parallel Jobscript on CSF

- Use a jobscript to ask the batch system to find *N* free cores
  - While matching other requirements (memory, architecture, fast networking, GPU etc).

1. Add extra lines in jobscript to request:
   - *a parallel partition* (for multi-core or multi-node jobs)
   - *and number of cores* to reserve
2. Inform your app how many cores to use
   - Remember, the jobscript says how many cores your *job* requires (the batch system will allocate those cores to your job.)
   - **But** *you* must still ensure your app uses no more!!
     - This is not always automatic and how you do it varies from app to app

Parallel Jobscript – Multicore Small (*single compute-node)*



Avoid a common mistake

- Can use **$SLURM_NTASKS** to get the number of cores reserved by the job

myparajob.txt

```
#!/bin/bash --login
#SBATCH -p multicore_small  # Use Intel nodes
#SBATCH -n 4          # Can be 2-40 in multicore_small
#SBATCH -t 3-0        # 3-day wallclock (max is 7-0)

# Set up to use a chemistry app
module purge
module load apps/gcc/gromacs/2023.3/double

# Inform app how many cores to use
export OMP_NUM_THREADS=$SLURM_NTASKS

# This job runs "gromacs"
mdrun_d
```

Our app wants `OMP_NUM_THREADS` environment variable to be set.
Your app might use a different method!

$SLURM_NTASKS is automatically set to the number, **4** in this case, given on **-n** line. Will be **1** in "serial" partition.

Parallel jobscript - Multi-core (cont…)

- That was a multicore_small (*single* compute node, no larger than 40 cores) example
- Using an app named Gromacs as an example
  https://ri.itservices.manchester.ac.uk/csf3/software/applications/gromacs/
- Requested a partition (-p) and number of cores (-n)
  - Job will run the app on a single Intel node, allocating multiple cores on that node to the job (4-5GB/core RAM)
- Then informed the app to use 4 cores via **OMP_NUM_THREADS** environment variable (very common).
  - Special **$SLURM_NTASKS** variable is always set to the number of cores requested on the -n (--ntasks=) line.

Parallel jobscript – Submitting the job

- As with the serial job, submit it to the system with
  `sbatch jobscript`
- Monitor with `squeue`
- It may take longer for *more* cores to become free in the system)
- You'll get the usual output file
  `slurm-JOBID.out`

Parallel Jobscript – Multi-core (*single compute-node*)

- Another *partition* provides AMD CPUs ("Genoa" architecture), and supports larger jobs - the multcore partition
  - Jobs run on a single compute node
  - Between 2 and 168 cores, 8Gb/core RAM

```
#!/bin/bash --login
#SBATCH -p multicore  # Use the AMD "Genoa" nodes
#SBATCH -n 84         # Can be 2-168 in multicore
#SBATCH -t 3-0        # 3-day wallclock limit (max 7-0)

# Set up to use "gromacs"
module purge
module load apps/gcc/gromacs/2023.3/double

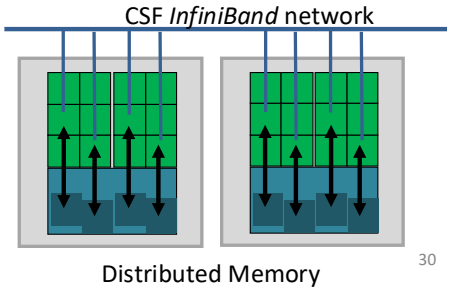
# Inform app how many cores to use
export OMP_NUM_THREADS=$SLURM_NTASKS

# This job runs "gromacs"
mdrun_d
```


Parallel Jobscript – Multi-node (*multiple compute-nodes*)

- For MPI-capable applications requiring large core-counts - the "multinode" partition
  - Jobs run on 2 or more (up to 32) compute nodes
  - Each Intel "Cascadelake" compute node has 40 cores
  - Job sizes are 80 to 1280 cores in total, 4Gb/core RAM
  - You need to specify number of compute nodes AND total number of cores
    - This may seem redundant but Slurm wants both

- Reminder – this is the style of job where the app sends data amongst its processes both within a node and over the network to other nodes.
- (Diagram only shows 12 cores per node for ease of drawing, but nodes in "multinode" have 40 cores.)



Parallel Jobscript – multi-*node*

#! from serial jobscript earlier.

-N (--nodes) where *numnodes* is between 2 and 32. **Required!** 8 in this example.

-t wallclock limit (*days-hours*). **Required!** Max permitted is: 7 days (7-0) for jobs of 80-560 cores. 4 days (4-0) for jobs of 600-1280 cores.

```
#!/bin/bash --login
#SBATCH -p multinode
#SBATCH -N 8          # 3 compute nodes
#SBATCH -n 320        # 320 in total (8x40)
#SBATCH -t 5-0        # 5-day wallclock

# Example: Set up to use MrBayes
module purge
module load apps/gcc/mrbayes/3.2.6

# App uses MPI to run across nodes. It
# knows to use $SLURM_NTASKS cores.
mpirun pmb in.nex

# You can specify num cores if you like
# mpirun -n $SLURM_NTASKS pmb in.nex
```

-p (--partition) **Required**. multinode has **Intel Cascadelake** 40-core nodes.

-n (--ntasks) where *numcores* is between 80 and 1280 (**total number of cores**). **Required!** 320 in this example

mpirun will run **$SLURM_NTASKS** (-n above) MPI processes, i.e. 320 in this example, using 40 core on each compute node, 8 nodes in total.

Parallel jobscript - Multi-node (cont…)

- NOT available to free-at-point-of-use (fatpou01) users
- A multi-node example – uses all cores on each node (a requirement).
- Requested the **`multinode`** partition: `#SBATCH -p multinode`
- Requested job to run using 8 nodes (40-cores per node).
  - Can request up to 32 nodes: `#SBATCH -N 8 (or --nodes=)`
- Requested job to start 320 tasks (8 x 40-core).
  - Can request between 80 – 1280 cores: `#SBATCH -n 320 (or --ntasks=)`
- Informed the app to use 320 cores via **`mpirun mpi-app.exe`**
  - **`mpirun`** starts multiple copies of an MPI app on allocated nodes
  - It reads the special **`$SLURM_NTASKS`** variable, which is always set to number of cores on the `-n (--ntasks=)` line.
  - **Your app must be capable of running as a multinode app. Using mpirun will not magically make your app multinode capable. Check the app's manual.**
- Further examples, including mixed mode (MPI+OpenMP), are available at: https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/parallel-jobs-slurm/#multinode

Parallel Partitions

Partition Name	Description
multicore_small (**recommended**)	2-40 cores, single compute node. ~4-5GB RAM per core. Jobs will be placed on Intel Haswell (max 24 cores/job), Skylake (max 32 cores/job), or Cascade Lake (max 40 cores/job)
-C architecture	Not recommended! (haswell, skylake or cascadelake)

Partition Name	Description
multicore	2-168 cores, single compute node. 8GB RAM per core. Jobs will be placed on AMD EPYC "Genoa" (max 168 cores/job)
No optional flags	

Partition Name	Description
multinode	80-1280 cores (in multiples of 40), 40-core Intel Cascade Lake nodes with 4GB RAM per core. The nodes are connected via Mellanox 100 Gb/s (4× EDR) InfiniBand.
No optional flags	

- **Max permitted runtime limit** *(must be specified by you in the jobscript)* :
  - **multicore_small & multicore**: 7days
  - **multinode:** 80–560 cores: 7 days, 600–1,280 cores: 4 days
- Our simple jobscript did *not* use any of the above optional (-C) flags. Not needed in most cases.
- If you limit a job by *architecture* it may wait longer in the queue.
- **https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/parallel-jobs-slurm**

Choosing the partition

- Choosing the partition is fairly simple, but:
  - Check the app's webpage for advice and examples https://ri.itservices.manchester.ac.uk/csf3/software
  - Check the partition page for limits on number of cores https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/partitions/
  - Avoid using **`#SBATCH -C architecture`**
- Use Intel (**multicore_small**) or AMD (**multicore**) nodes?
  - Most (all) apps will run on both, but AMD nodes are newer
  - The high memory nodes are all Intel CPUs (e.g., **`-p himem`**)
  - Large multicore jobs (>40 cores) must use **multicore**
  - Submitting to **multicore_small** may have shorter wait times
  - **multicore** nodes have 8GB/core (**multicore_small** nodes have ~4 or 5-6GB/core)
- Use **multinode** for larger MPI-capable apps
  - But do some tests to ensure your app "scales"

Parallel Software Performance

- You'll probably be running an app many times
- Worth small investigation to find optimal performance parameters (#cores & #nodes)
  - How many cores should I use?
- Do a few runs, change the number of cores
  - Plot time vs number of cores
  - Easy to do on CSF: remove "**`-n numcores`**" from jobscript, add it to **`sbatch`** command instead:

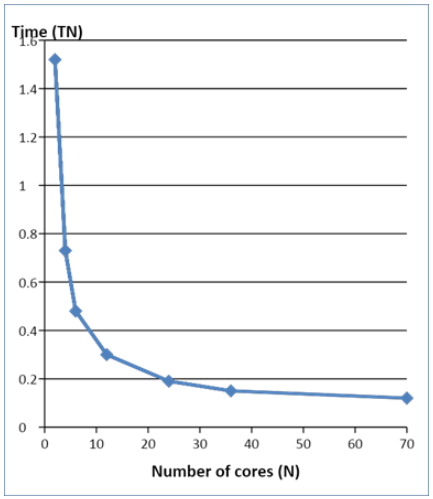
```
sbatch -n 2 myjobscript.txt
sbatch -n 4 myjobscript.txt
sbatch -n 8 myjobscript.txt
```

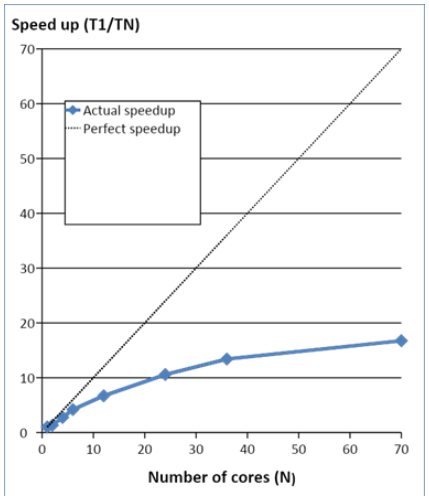

To Assess Parallelism

- Plot the following against "Number of Cores":
  - "Speed-up" or "Parallel Efficiency"
  - Total memory usage?
- Look for the sweet-spot
- Calculate: Speed-up = T_1 / T_N
  - Compare results against "ideal" scaling (where *N*-cores makes it go *N*-times faster)
- Calculate: Parallel Efficiency = $T_1 / (N \times T_N)$
  - N = number of cores, T_N = time take on N cores
- Pick a typical problem size for your work

Examples of Speed-up

- Data for popular Finite Element app on CSF
  - The 'Time' graphs shows it getting faster. But…





Examples of Speed-up & Efficiency

- Example showing Speed-up and Efficiency values
  - App multiplies two square matrices
    - Measured a single multiplication of two 2000x2000 matrices

No. cores	Time (Seconds)	Speed-up	Efficiency
1	45.0	1x	1.00
2	22.8	1.97x	0.99
4	11.7	3.84x	0.96
8	7.1	6.33x	0.80

- The speed-up is reasonably close to “perfect” & efficiency is reasonably close to 100% but…
  - How will this scale as we go multi-node?
  - How will this scale as the problem size increases?
  - How will this scale on other hardware?

PRACTICAL SESSION 4

Parallel job and scaling

Practical Session 4 - Parallel jobs

- Follow the handout ‘Practical Session 4’
  - Use `sbatch` to submit a simple parallel (2-core) job on the CSF
  - Modify the jobscript to use different numbers of cores
  - Determine whether the application "scales" with the number of cores.
- Exercise sheet (pdf) available at: https://ri.itservices.manchester.ac.uk/course/rcsf/

MULTIPLE SIMILAR JOBS

High Throughput Computing and "Job arrays"

Multiple Runs of an Application

- We want to run an application many times to process *many different* input files
  - For example, on a desktop PC you might run

```
myapp.exe -in mydata.1.tif -out myresult.1.tif
(wait for it to finish)
myapp.exe -in mydata.2.tif -out myresult.2.tif
(wait for it to finish)
myapp.exe -in mydata.3.tif -out myresult.3.tif
…
myapp.exe -in mydata.1000.tif -out myresult.1000.tif
```

  - If it takes 5 minutes to process one file, it will take 1000 x 5 minutes to process them all (~3.5 days)

How **Not** To Do It on the CSF (1)

- Inefficient method 1: one after another in *one* job? `sbatch jobscript-all.txt`

jobscript-all.txt

```
#!/bin/bash --login
#SBATCH -p serial
#SBATCH -t 4-0      # Wallclock is 4 days for all images

myapp.exe -in mydata.1.tif -out myresult.1.tif
(will wait for it to finish)
myapp.exe -in mydata.2.tif -out myresult.2.tif
(will wait for it to finish)
myapp.exe -in mydata.3.tif -out myresult.3.tif
…
myapp.exe -in mydata.1000.tif -out myresult.1000.tif
```

- This is *no better* than the desktop PC method

How **Not** To Do It on the CSF (2)

- Inefficient method 2: lots of individual jobscripts?

jobscript1.txt

```
#!/bin/bash --login
#SBATCH -p serial
#SBATCH -t 10    # Wallclock is 10 mins for one image

myapp.exe -in mydata.1.tif -out myresult.1.tif
```

```
sbatch jobscript1.txt
sbatch jobscript2.txt
sbatch jobscript3.txt
…
sbatch jobscript1000.txt
```

Make 1000 copies of this jobscript, edit each one to process a different file (mydata.2.tif, …)

Then **submit each job**

- Strains the batch system queue manager
- But, you will get many jobs running in parallel
  - EG: approx 100-200 jobs running at same time

How To Do It - a "Job Array" Jobscript

1-1000 (start-end) says how many tasks to run and how they should be numbered. Note: **Can** start at **0**. Can use start-end:increment to increase the ID by more than 1.

-t wallclock. Each individual array task has a wallclock of 10 minutes. There is no overall time limit for the job.

-a an array job. Runs multiple copied of the job a specified number of times. These are called **array tasks**. Each is numbered uniquely 1, 2, 3 …, 1000.

arrayjob.txt

```
#!/bin/bash --login
#SBATCH -p serial     # Each task gets 1 core
#SBATCH -t 10         # 10 minutes per task
#SBATCH -a 1-1000     # 1000 tasks

echo "I am task ${SLURM_ARRAY_TASK_ID}"
myapp.exe \
   -in mydata.${SLURM_ARRAY_TASK_ID}.tif \
   -out myresult.${SLURM_ARRAY_TASK_ID}.tif
```

The commands we run in our job. They will execute on backend nodes (different cores and nodes for different tasks).

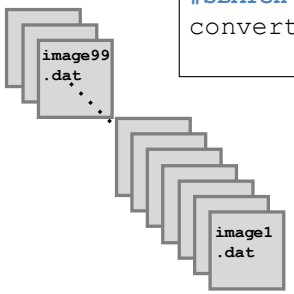
${SLURM_ARRAY_TASK_ID} is automatically set by the batch system and tells us which array task we are (1,2,…). We can use this to do something different in array task.

"Job Array" Jobscript

- Our app is a serial (1-core) app
  - But you *could* use the mutlicore partition if your app is multi-core capable.
- The total number of tasks can be 100s, 1000s, 10000s (max 25,000 on CSF)
- The system will run *many* of the tasks concurrently
  - Usually 100s - "High-throughput Computing"
  - You get lots of work done sooner
  - It will eventually churn through all of them
  - They are started in numerical order but no guarantee they'll finish in that order!
- The extra jobscript **#SBATCH -a** line is easy. Using the **$SLURM_ARRAY_TASK_ID** number *creatively* **is the key to job arrays**.
- **Note**: Do not confuse Slurm's **-n/--ntasks** (number of cores) flag with **-a/--array** (start-end for array tasks) flag. The names can be confusing :-(

The $SLURM_ARRAY_TASK_ID variable (1)

- Want to do something different in each task. EG:
  - Read a different data file to process
  - Pass a different parameter to an application
- You can get this different "thing" in many ways:
  - EG: Use the $SLURM_ARRAY_TASK_ID in filenames:




```
#SBATCH -a 1-1000
convert -i image_${SLURM_ARRAY_TASK_ID}.jpg \
        -o image_${SLURM_ARRAY_TASK_ID}.png
```

Task 1 reads image_1.jpg writes image_1.png
Task 2 reads image_2.jpg writes image_2.png
…
Task 1000 reads image_1000.jpg writes image_1000.png

The $SLURM_ARRAY_TASK_ID variable (2)

- Or have a "master" list (a text file) of names etc
- The Nth task reads the Nth line from that text file:

```
#SBATCH -a 1-4000
# Read the Nth line of filenamelist.txt and save in variable MYFILENAME
MYFILENAME=$(awk "NR==${SLURM_ARRAY_TASK_ID}" filenamelist.txt)

# Now use whatever the value of variable is in the next command
myapp.exe -input ${MYFILENAME} -output ${MYFILENAME}.out
```

filenamelist.txt

```
ptn1511.dat
ptn7235.dat
ptn7AFF.dat
ptn6E14.dat
ptn330D.dat
…
```

Task 1 reads ptn1511.dat writes ptn1511.dat.out
Task 2 reads ptn7235.dat writes ptn7235.dat.out
…

- Number of lines in master file **must** match number of tasks
- To get number of lines in master file use:
  wc -l filenamelist.txt
- bash: **VAR=$(*command arg1 arg2*)** captures output from *command* and assigns to variable VAR for use later.

https://ri.itservices.manchester.ac.uk/csf3/batch/job-arrays/

The $SLURM_ARRAY_TASK_ID variable (3)

- Another way to use the "master" list method
- The N[th] task reads the N[th] line from that text file:

```
#SBATCH -a 1-50
# Read the Nth line of foldernamelist.txt and save as $FOLDER
FOLDER=$(awk "NR==${SLURM_ARRAY_TASK_ID}" foldernamelist.txt)

# Now use whatever the value of variable is in the next command
cd ~/scratch/experiments/${FOLDER}
mdrun_d
```

foldernamelist.txt

```
znc24/100p/a1
znc24/200p/b2
ag80/100p/b1
ag81/100q/c1
ptn2/50a/a1
ptn3/50b/c1
…
```

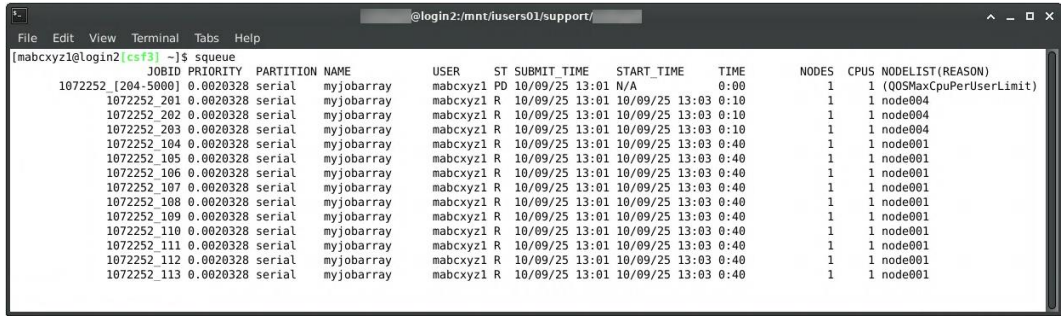
Task 1 reads `znc24/100p/a1` as folder name
Task 2 reads `znc24/200p/b2` as folder name
…

- Number of lines in master file **must** match number of tasks
- To get number of lines in master file use:
  `wc -l foldernamelist.txt`
- bash: **`VAR=$(`*`command arg1 arg2`*`)`** captures output from *command* and assigns to variable VAR for use later.

https://ri.itservices.manchester.ac.uk/csf3/batch/job-arrays/

Jobarrays – squeue and scancel

- `squeue` shows pending and running tasks
  - `QOSMaxCpuPerUserLimit` means the limit of number of CPUs in use at any one time (for this user) has been reached.
  - Hence some array tasks must wait until earlier ones have finished.



- scancel can remove all tasks or just some

`scancel 1072252`	Remove all running and waiting tasks
`scancel 1072252_300`	Remove task 300 (a bit strange)
`scancel 1072252_[4000-5000]`	Remove last 1000 tasks

Jobarray Output Files

- You'll get the usual Slurm output file but
  - One per task
  - Potentially a lot of files!
- Look for
  `slurm-`*`JOBID_TASKID`*`.out`

  EG: `slurm-`*`1046732_1`*`.out`
  `slurm-`*`1046732_2`*`.out`
  `...`
  `slurm-`*`1046732_1000`*`.out`
- You should delete empty / unwanted files soon and often

PRACTICAL SESSION 5

Job array examples

Practical Session 5 – Job arrays

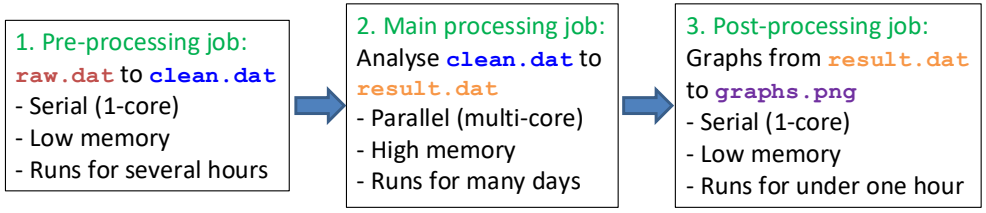
- Follow the handout 'Practical Session 5'
  - Modify, and run, a simple job-array jobscript to run an app with different input values.
  - Write a new job-array jobscript to do some python image processing on a set of Hubble Ultra Deep Field images.
- Exercise sheet (pdf) available at: https://ri.itservices.manchester.ac.uk/course/rcsf/

JOB PIPELINES

Ordering jobs

A Job Pipeline (aka workflow)

- Suppose you have several apps that:
  - Need to run in a specific order - a "pipeline"
    - There is a *dependency* between apps
  - Each might have different CPU-core and memory requirements
  - Each might take different amounts of time to run



How **not** to do it on the CSF (1)

- Put all steps in one job?
  - Wastes resources (some cores and memory)
  - May go over 7-day runtime limit

Only one command uses all of the cores

mypipeline_bad.txt

```
#!/bin/bash --login
#SBATCH -p himem      # Uses a high-memory node
#SBATCH --mem 2000G   # … with 2TB of RAM
#SBATCH -n 16         # … and 16 cores
#SBATCH -t 7-0        # … for 7 days
module purge
module load apps/……

# First app (1-core, low memory usage)
preproc -in raw.dat -out clean.dat
# Second app (multiple cores, high memory usage)
mapper -p $SLURM_NTASKS -in clean.dat -out result.dat
# Third app (1-core, low memory usage)
drawGraphs -in result.dat -out graphs.png
```

Better but still not perfect

- Split into multiple jobs, notice when jobs finish, submit next…?
  - Log in to CSF, check if previous job has finished…. wastes time!

A serial job (no wasted cores)

firstjob.txt

```
#!/bin/bash --login
#SBATCH -p serial    # 1-core job
#SBATCH -t 0-2       # 2 hour wallclock
module load apps/……
# First 'job' (1-core, low memory usage)
preproc -i raw.dat -o clean.dat
```

A parallel, high-mem job

secondjob.txt

```
#!/bin/bash --login
#SBATCH -p himem     # Uses a high-memory node
#SBATCH --mem 2000G # … with 2TB RAM
#SBATCH -n 16        # … and 16 cores
#SBATCH -t 7-0       # … 7 day wallclock
module load apps/……
# Second 'job' (multiple cores, high memory usage)
mapper -p $SLURM_NTASKS -i clean.dat -o result.dat
```

A serial job (no wasted cores)

thirdjob.txt

```
#!/bin/bash --login
#SBATCH -p serial    # 1-core job
#SBATCH -t 30        # 30 minute wallclock
module load apps/……
# Third 'job' (1-core, low memory usage)
drawGraphs -i result.dat -o graphs.png
```

sbatch firstjob.txt
(now you need to wait until this job has finished before submitting the next one!)
sbatch secondjob.txt
(now you need to wait until this job has finished before submitting the next one!)
sbatch thirdjob.txt

How **to** do it - Job Dependencies

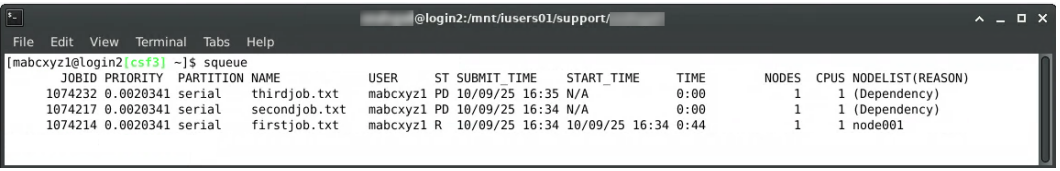
- Using the previous individual jobscripts
  - Add a `-d` / `--dependency=` flag when submitting them
  - Use the JOBID of the previous job to make the current job wait for it

```
sbatch firstjob.txt
Submitted batch job 1074214
                ↓
sbatch -d afterok:1074214 secondjob.txt
Submitted batch job 1074217
                ↓
sbatch -d afterok:1074217 thirdjob.txt
Submitted batch job 1074232
```

  - The **`afterok`** flag means do not run this job until the earlier dependency job has finished successfully.
  - Note, you cannot you use jobscript names. You must use the JOBID of an earlier job.
  - There are lots of dependency parameters (e.g, afterany, afternotok, and multiple dependencies can be setup.)

Job Dependencies

- You *must* submit the jobs in the correct order
  - EG: If `secondjob.txt` is submitted first, it runs immediately (no dependency job exists to wait for)
- `squeue` shows `(Dependency)` for jobs on hold



- Later jobs may still *wait* to be scheduled
  - They don't always run *immediately* after earlier jobs finish

Job Dependencies

- Can submit the jobs in a more programmatic manner:
  - Use `--parsable` flag to get *just* the JOBID of the submitted job (instead of 'long' message):
    - **`sbatch myjobscript`**
      `Subimtted batch job 1074233`
    - **`sbatch --parsable myjobscript`**
      `1074233`
  - Capture output of command into shell variable

```
JOBID=$(sbatch --parsable firstjob.txt)
JOBID=$(sbatch --parsable -d afterok:$JOBID secondjob.txt)
JOBID=$(sbatch --parsable -d afterok:$JOBID thirdjob.txt)
```

Job-Array Dependencies (1)

- An ordinary job can wait for a *job array* to finish
  - All tasks in the job array must have finished

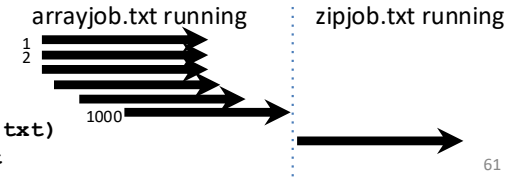
arrayjob.txt

```
#!/bin/bash --login
#SBATCH -p serial      # 1-core job
#SBATCH -t 30          # 30 minute wallclock
#SBATCH -a 1-1000      # Job array with 1000 tasks
convert img.${SLURM_ARRAY_TASK_ID}.tif img.${SLURM_ARRAY_TASK_ID}.pdf
```

zipjob.txt

```
#!/bin/bash --login
#SBATCH -p serial      # 1-core job
#SBATCH -t 5           # 5 minute wallclock
zip conference.zip img.*.pdf
```

Add a job dependency so that second job waits until entire job-array has finished.




```
JID=$(sbatch --parsable arrayjob.txt)
sbatch -d afterok:$JID zipjob.txt
```


Job-Array Dependencies (2)

- A *job array* can wait for a *job array* to finish
  - *All* tasks in the *first* job array must have finished

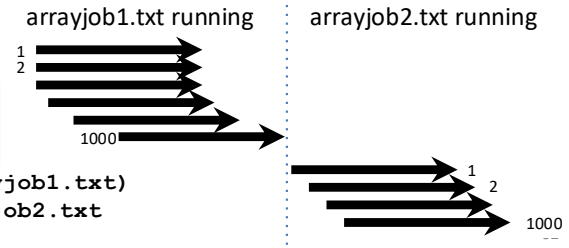
arrayjob1.txt

```
#!/bin/bash --login
#SBATCH -p serial      # 1-core job
#SBATCH -t 30          # 30 minute wallclock for each task
#SBATCH -a 1-1000      # Job array with 1000 tasks
someapp data.${SLURM_ARRAY_TASK_ID}.xyz  data.${SLURM_ARRAY_TASK_ID}.dat
```

arrayjob2.txt

```
#!/bin/bash --login
#SBATCH -p multicore   # Multicore job
#SBATCH -n 4           # Each task can use 4 cores.
#SBATCH -t 60          # 60 minute wallclock for each task
#SBATCH -a 1-1000      # Job array with 1000 tasks
otherapp data.${SLURM_ARRAY_TASK_ID}.dat result.${SLURM_ARRAY_TASK_ID}.dat
```

Add a job dependency so that second job array waits until entire first job-array has finished.




```
JID=$(sbatch --parsable arrayjob1.txt)
sbatch -d afterok:$JID arrayjob2.txt
```

Job-Array Dependencies (3)

- A *job array* can wait for a *job array* to finish
  - Each task in *second* job array waits for *corresponding* task in *first* job array to finish

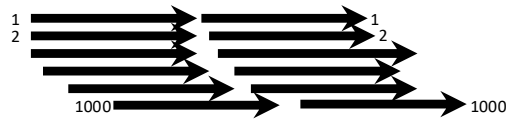
arrayjob1.txt

```
#!/bin/bash --login
#SBATCH -p serial      # 1-core job
#SBATCH -t 30          # 30 minute wallclock for each task
#SBATCH -a 1-1000      # Job array with 1000 tasks
someapp data.${SLURM_ARRAY_TASK_ID}.xyz  data.${SLURM_ARRAY_TASK_ID}.dat
```

arrayjob2.txt

```
#!/bin/bash --login
#SBATCH -p multicore   # Multicore job
#SBATCH -n 4           # Each task can use 4 cores.
#SBATCH -t 60          # 60 minute wallclock for each task
#SBATCH -a 1-1000      # Job array with 1000 tasks
otherapp data.${SLURM_ARRAY_TASK_ID}.dat result.${SLURM_ARRAY_TASK_ID}.dat
```

arrayjob1.txt tasks running then arrayjob2.txt tasks



Add a job dependency so that second job array tasks wait for corresponding tasks in first job-array to finish.

```
JID=$(sbatch --parsable arrayjob1.txt)
sbatch -d aftercorr:$JID arrayjob2.txt
```


INTERACTIVE JOBS

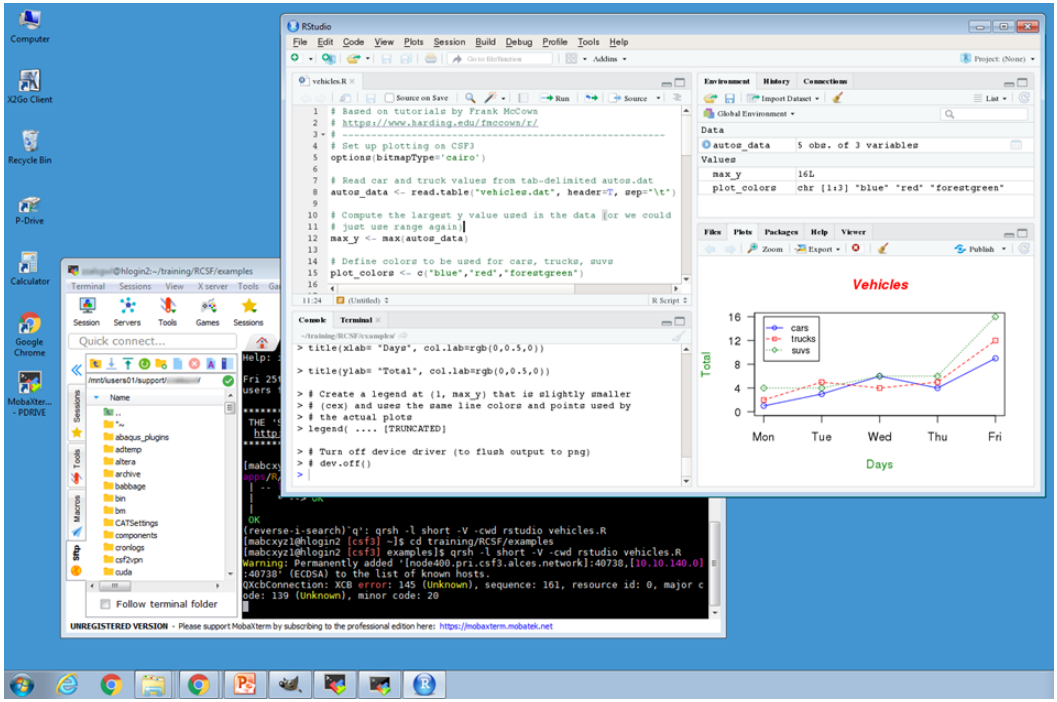
Compute apps with GUIs

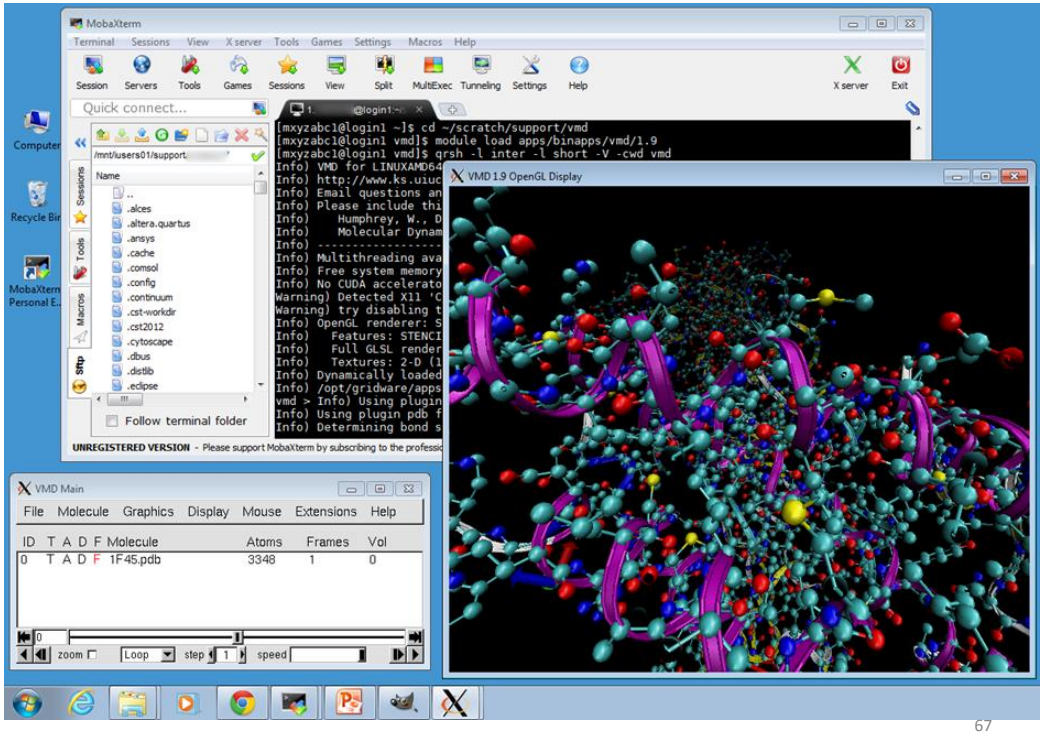
Interactive work

- Some apps (eg Rstudio, VMD, molden, paraview) may have a GUI but should not be run on the login node.
  - The GUI itself might be quite a light-weight program, but if you then load a huge dataset in, or set off some long-running multi-process computation, via the GUI, you'll "hammer" the login node.
- Use the **`srun`** command to get an *interactive session* on a compute node

```
# We are on the login node here
module purge
module load apps/gcc/R/4.5.0
module load apps/binapps/rstudio/2025.05.0-any-r
srun -p interactive -t 60 -n 1 rstudio vehicles.R
```

- No dedicated resource, priority to batch jobs.
- Runs on an AMD 168-core node, hence 8GB per core.
- Max 60 minutes wallclock limit.
- Remember - a GUI app, as with `gedit`, needs an X-server running on your PC (as provided by MobaXTerm, or X-Quartz, or a Linux desktop)
- Remember to exit your GUI app when you have finished so the resource is made available for others.





GPU Jobs

Accessing the GPUs

Nvidia GPUs

- CSF3 has 204 x Nvidia GPUs
  - A100(80GB), L40S are available to all users. The A100(40GB) GPUs are for a specific research group. The H200 GPUs are for specific projects / faculty.

84 x Ampere A100 GPUs in total – 4 GPUs/node
80GB(**76x**) or 40GB(**8x**) GPU mem, Mem b/w 2TB/s
6912 CUDA cores (108 Multiprocessors, 64 cores/MP)
432 Tensor cores
Peak FP64 9.7 TFLOPS
48-core AMD Epyc "Milan"
512GB RAM host node
40GB A100 nodes restricted to one research group

88 x Ada Lovelace L40S GPUs in total – 4 GPUs/node
48GB GPU memory, Mem bandwidth 864GB/s
18176 CUDA cores (142 Multiprocessors, 128 cores/MP)
142 RT cores
568 Tensor cores
Peak SP 91.6 TFLOPS
48-core Intel Xeon Gold "Sapphire Rapids"
512GB RAM host node

GPU Jobscript

gpuA, gpuL, or **gpuA40GB** is the **partition** name. It means: app will use a GPU compute node containing the indicated type of GPU - A100, L40S or gpuA40GB (this one has restricted access.)

-t wallclock time limit for the job. Max in **gpuX** is 4 days.

```
#!/bin/bash --login
#SBATCH -p gpuX      # Partition (A,L)
#SBATCH -G 2         # GPUs: 1-4
#SBATCH -n 8         # Cores: 1-8,12
#SBATCH -t 4-0       # Max is 4-0 here
# Set up to use the CUDA toolkit
module purge
module load libs/cuda/12.4.1

# Run a GPU app. Slurm will ensure no
# other jobs can use your GPUs.
deviceQuery
```

-G (**--gpus=**) **2** is the total number of **GPUs** we want to *reserve* in the system.

-n (**--ntasks=**) **8** is the total number of **host CPU cores** we want to *reserve* in the system.

Slurm will set some environment variables for use in your jobscript:
$CUDA_VISIBLE_DEVICES gives the device IDs (**0** or **0,1** or **0,1,2** or **0,1,2,3**) depending number of GPUs.
$SLURM_GPUS gives the number of GPUs you request on the -G line.
$SLURM_NTASKS as previous jobs, the number on the –n (host CPU cores).

GPU Limits

- Most users get access to **up to four GPUs** from the gpuA and gpuL partitions, in use at any one time with a max of 4 overall (e.g., 4xA100 or 2xA100+2xL40S or 4xL40S ....)
  - This is "free at point of use access", funded by the Research Lifecycle Programme, managed by Research IT.
  - Users from some contributing groups who have funded GPUs may get access to more GPUs.
  - All GPU nodes contain 4 GPUs. Multi-node (>4 GPU jobs) are NOT possible.
  - H200 nodes are restricted an require an application process to be complete – see https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/gpu-jobs-slurm/h200/
- CPU host cores are limited by number of GPUs in job and by node-type
  - Note that unless the jobscript contains the **-n** flag, jobs will only have one host CPU core.

GPU Partition (GPU type)	Host CPU type	Max host CPU cores per GPU	Host RAM per CPU core (GB)	Max host RAM per GPU (GB)
gpuA (A100 80GB)	48-core AMD EPYC "Milan"	12	10.4	125
gpuL (L40s)	48-core Intel Xeon "Sapphire Rapids"	12	10.4	125
gpuA40GB (A100 40GB)	48-core AMD EPYC "Milan"	12	10.4	125

Other GPU Notes

- GPUs are run in **DEFAULT** compute mode (not **EXCLUSIVE_PROCESS**.)
  - You can run multiple processes / apps on the same GPU – e.g., several small chemistry simulations.
- You can monitor your GPU jobs by accessing the compute node and GPU once your job as started.
  - Use the **srun** command on the login node to "login" to the compute node and resource container where your job is running:
    srun --jobid=*JOBID* --pty bash
    (wait until you are logged into the compute node where you job is running. You'll see the same GPUs.)
    nvidia-smi
    or, for example:
    module load libs/cuda/12.4.1
    ncu-ui or **nvvp** (or other Nvidia tool)
- A quick exercise 6:
  - If you are logged-in to the CSF, try editing the **~/training/RCSF/examples/dq_gpu.sbatch** jobscript (change the **-p** line to request one of the **gpuA** or **gpuL** partitions.) Then *submit* the job.
  - It runs the Nvidia **deviceQuery** utility to return some stats about the GPU assigned to your job.

High Memory Jobs

Accessing the high memory nodes

High Memory Jobs

- So far, our serial and parallel jobs have access to a fixed amount of memory
  - The partitions provide a specific amount of memory per core.
    - e.g. the multicore partition provides 8GB/core
  - If you want more memory, you need to request more cores.
  - Max in **`multicore`** is ~ 1.5 TB if all 168 cores used.
  - Can be very wasteful of CPU resources
- The **`himem`** and **`vhimem`** partitions provide access to more memory (up to 2TB and 4TB.)
  - Memory is a "consumable" - the amount of memory can be requested independently of the number of cores.

Do you need more memory?

- If a job fails and reports an "OOM" error in the **`slurm-`*`JOBID`*`.out`** file, you should request more memory.
- You can also check a previous job to see how much memory it used with "**`seff`**"

```
[mabcxyz1@login1[csf3] ~]$ seff 12345
Job ID: 12345
Cluster: csf3.man.alces.network
User/Group: username/xy01
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 2
CPU Utilized: 00:04:13
CPU Efficiency: 49.41% of 00:08:32 core-walltime
Job Wall-clock time: 00:04:16
Memory Utilized: 21.45 GB          # Peak memory usage
Memory Efficiency:33.5% of 64.00 GB  # A low memory efficiency means this job did NOT need
                                     # to use the himem partition. You should check this.
```


High-memory Compute Nodes

- The 2TB (**`himem`**) partition contains various Intel compute nodes.
  - Slurm will place your job on any of the node-types that have enough resources (memory, cores) as requested by your job.
- The 4TB (**`vhimem`**) partition is restricted
  - Please request access via the Connect Portal
  - We will need evidence from previous jobs that you need >2TB of RAM.

Partition (required)	Default job mem-per-core if memory not requested (GB)	Max job size (cores)	Max job memory (GB)	Arch Flag (optional, but will activate specific limits shown in the next columns)	Max job size (cores)	Max job memory (GB)	Has SSD storage
himem	31	32	2000	`-C skylake`	32	1472	No
				`-C cascadelake`	32	1472	No
				`-C icelake` (also `ssd`)	32	2000	Yes
vhimem	125	32	4000	`-C icelake` (also `ssd`)	32	4000	Yes

High Memory Jobscripts

```
#!/bin/bash --login
#SBATCH -p himem            # Partition
#SBATCH -n 2                # Cores: Can be 1-32
#SBATCH --mem=1200G         # Total memory for the job
#SBATCH -t 4-0              # Wallclock (max is 7-0 here)

# Set up to use your app
module purge
module load apps/gcc/something/1.2.3

# Run a GPU app. Slurm will ensure no
someApp -in hugh_dataset.dat -out results.dat
```

```
#!/bin/bash --login
#SBATCH -p himem            # Partition
#SBATCH -n 2                # Cores: Can be 1-32
#SBATCH --mem-per-cpu=600G # OR specify the Memory per Core
                            # (note: name of flag uses 'cpu')
#SBATCH -t 4-0              # Wallclock (max is 7-0 here)
```


OTHER PARALLEL HARDWARE

What else is available?

HPC Pool

- Dedicated pool for HPC jobs
  - ~8700 cores of Infiniband connected CascadeLake shared with the **multinode** partition
  - Minimum 120-core job size, maximum 1000
  - Frontend shared with CSF3
    - You just submit HPC jobs like any other CSF job (with a different "partition" name and an **account code**.)
  - Lightweight application process – must be made by PI
  - Available to free-at-point-of-use (fatpou01) users

https://ri.itservices.manchester.ac.uk/csf3/hpc-pool

Parallel Jobscript – HPC Pool

#! line from serial jobscript earlier.

hpcpool is the **partition** name. It means: app will use multiple compute nodes (Intel 40-core nodes, all 40 cores **must** be used on each) and has fast InfiniBand networking between the nodes.

-A gives an *account code*, needed for this restricted partition.

-t wallclock time limit for the job. Max in hpcpool is 4 days.

-N (--nodes=) **3** is the total number of **nodes** we want to *reserve* in the system.

-n (--ntasks=) **120** is the total number of **cores for MPI processes** we want to *reserve* in the system.

```
#!/bin/bash --login
#SBATCH -p hpcpool
#SBATCH -N 4                # Nodes: 3-25
#SBATCH -n 120              # Cores: 120-1000
#SBATCH -A hpc-projectcode
#SBATCH -t 4-0              # Max is 4-0 here
# Example: Set up to use MrBayes
module purge
module load apps/gcc/mrbayes/3.2.6

# App uses MPI to run across nodes
mpirun -n $SLURM_NTASKS pmb in.nex
```

The commands we run in our job. They will execute on the compute nodes assigned to the job. `pmb` is the app name and is started via `mpirun` (very common for multi-node apps).

Must inform the app how many cores we *reserved*. `$SLURM_NTASKS` is automatically set to number (**120**) given on -n line. `mpirun` accepts a "`-n numcores`" flag (which is optional – `mpirun` reads the `-n` number from Slurm if you don't supply it!!)

N8 Bede

- 32 IBM Power 9 dual-CPU nodes
  - Each node comprises 4 NVIDIA V100 GPUs and high performance interconnect.
- 5 Nvidia GH200 Grace Hopper nodes
  - Each node comprises 1x NVIDIA H100 96GB with 900 GB/s NVLink-C2C and 1x NVIDIA Grace aarch64 CPU @ 3.483 GHz (72 Arm Neoverse V2 cores)
- Same architecture as the US government's SUMMIT and SIERRA supercomputers which occupied the top two places in a recently published list of the world's fastest supercomputers.
- Contact Research IT for advice
- https://n8cir.org.uk/supporting-research/facilities/bede/docs/

ARCHER2

- National supercomputer funded by UK Research Councils
  - Service will end 20 November 2026
  - 5,848 compute nodes, each with dual AMD EPYC Zen2 (Rome) 64 core CPUs at 2.2GHz, giving 748,544 cores in total.
  - Estimated peak performance of 28 PFLOP/s
- Mostly open source / free HPC software
- See https://www.archer2.ac.uk/
  - Info for how to apply for access
    - Applications assessed for suitability
- IT Services can help you apply for access

FINAL POINTS

Further info

News

- MOTD when you log into the CSF - please read it
- Problems e.g. system down, can't log in, minor changes to the service (and other services - e.g storage):
  https://ri.itservices.manchester.ac.uk/services-news/
- Prolonged problems or major changes emailed to all users

https://ri.itservices.manchester.ac.uk/csf3/help/

- CSF Slurm documentation
  https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/
- Job Arrays - multiple similar jobs from a single submission script
  https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/job-arrays-slurm/
- SSHFS - another means of file transfer
  https://ri.itservices.manchester.ac.uk/userdocs/file-transfer/
  Virtual Desktop Service – another means of connecting and running GUIs and logging in from off campus
  https://ri.itservices.manchester.ac.uk/virtual-desktop-service/
- Please give feedback: Quick form at https://goo.gl/forms/zfZyTLw4DDaySnCF3 (choose "*Introduction to HPC (Using CSF)*")

Thank you!