pawsey

Supercomputing User Training

Module 9: Running Jobs

Pawsey Training Series

Supercomputing User Training

1. Supercomputing Introduction
2. Logging In
3. Filesystems Overview
4. Moving Data In and Out
5. Using Software Modules
6. Using Software Containers
7. Accounting Model Overview
8. Job Scheduling Overview
9. Running Jobs
10. Testing Job Runs
11. Managing Project Data

Outcomes for this Module

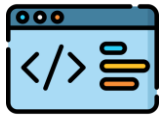
- Describe what a batch script is
- Submit, monitor and cancel jobs
- Open an interactive job session
- List the key resource requirements in a batch script
- Run a parallel program

✓ Prerequisite knowledge:

- ✓ **Bash shell basics**
- ✓ **User Training 02: Logging In**
- ✓ **User Training 07: Accounting Model Overview**
- ✓ **User Training 08: Job Scheduling Overview**

Watch for These Signs!

Definition of new concepts



Hands-on coding (demo)

Best practices

Exercises and solutions

Warnings (bad practices)

Links to user documentation

Icons made by Freepik from www.flaticon.com

Pawsey

Australian Government

NCRIS National Research Infrastructure for Australia

CSIRO

Curtin University

Murdoch University

Government of Western Australia

ECU Edith Cowan University

The University of Western Australia

Submitting and Monitoring Jobs

What is a Batch Script?

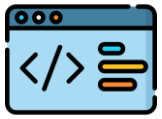
Batch Script
A shell script used to submit computational jobs to the scheduler. It contains both instructions on commands to execute, and information on compute resource requirements for the scheduler.

Features of a batch script:

- Commands to execute are written in the shell script language
  - In this module we are showing batch scripts written in Bash
- Compute resource requirements are expressed as header comments
  - The Slurm scheduler reads comment lines starting with `#SBATCH`
  - Any `#SBATCH` directive can alternatively be provided as an option to the `sbatch` command

For improved reproducibility, specify resource requirements within the batch script rather than via command line.
Writing resource requirements in the batch script is more reproducible, for yourself, your research team, and the support staff.

Submitting and Monitoring Jobs



DEMO on Setonix: `sbatch` and `squeue` – let's do this together

NOTE: we are going to discuss the contents of batch scripts later in this module

- Use `sbatch` *`batchscript-name`* to submit batch scripts to the scheduler
  - At some user trainings, Pawsey provides compute resource reservations to attendees
  - Use a reservation via the `sbatch` flag `--reservation=`*`reservation-id`*
  - This is a rare good case for adding the command-line option to `sbatch` rather than the directive in the batch script (once-off use, with no impact on reproducibility)
  - The command outputs the submitted job ID, as assigned by Slurm
- As seen in User Training 08, use `squeue --me` to query your queue of submitted jobs
  - If many jobs in the queue, you can find the one you just submitted via its job ID
- If you need to cancel a submitted job, use `scancel` *`jobid`*
  - See next Exercise

OUTPUTS: Submitting and Monitoring Jobs

```
$ cd $MYSCRATCH
$ ls
..
$ # run the command below, only if the repository has not been downloaded yet:
$ git clone https://github.com/PawseySC/supercomputing-user-training

$ cd supercomputing-user-training/running-jobs

$ sbatch submit_monitor.sh  # if instructed in training, add: --reservation=reserv-id
Submitted batch job 159534

$ squeue --me
 JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
159534      work first_jo username  R       0:25      1 nid001053

$ squeue --me
 JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)

$ # no more squeue output when job finished
```

EXERCISE: Job Output and Cancelling Jobs

From the directory: `$MYSCRATCH/supercomputing-user-training/running-jobs`
(If needed, get it with `cd $MYSCRATCH ; git clone https://github.com/PawseySC/supercomputing-user-training`)

1. Inspect the output of the previous job (tip: use Linux `cat` over a file called like `slurm-123456.out`)
2. What's the hosthame in the output? (tip: it appears in multiple lines)
3. What's the current hostname in your shell? (tip: use the command `hostname`)

4. Submit the batch script `cancel.sh`
5. Check it is in your queue, note the job ID, and wait until it goes to Running (R) status
6. Now cancel that job from the queue (tip: you need the command `scancel`)
7. Inspect the output of this job: do you see any error messages?

OUTPUTS: Job Output and Cancelling Jobs

```
$ cd $MYSCRATCH/supercomputing-user-training/running-jobs

$ cat slurm-159534.out
The date is: Thu Aug 11 10:40:43 AWST 2022
The hostname is: nid001053
The hostname is: nid001053
The date is: Thu Aug 11 10:41:13 AWST 2022
$ hostname
setonix-01

$ sbatch cancel.sh  # if instructed, add: --reservation=reserv-id
Submitted batch job 159535
$ squeue --me
 JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
159535      work first_jo username  R       0:20      1 nid001053
$ scancel 159535

$ cat slurm-159535.out
The date is: Thu Aug 11 10:42:55 AWST 2022
The hostname is: nid001053
The hostname is: nid001053
slurmstepd: error: *** JOB 159535 ON nid001053 CANCELLED AT 2022-08-11T10:43:12 ***
```

- By default, output of a Slurm job goes in a file called `slurm-jobid.out`
- Hostname convention for
  - compute nodes: `nidXXXXXX`
  - login nodes: `setonix-XX`
- Slurm outputs a CANCELLATION error message when a job is cancelled

EXERCISE: Job Hitting the Time Limit

From the directory: `$MYSCRATCH/supercomputing-user-training/running-jobs`

1. Submit the batch script `time_limit.sh`
2. Check if it is in your queue, and note the job ID
3. Wait for about 1-2 minutes (this job has a 1 minute maximum wall time)
4. Double-check the job is not in the queue any more
5. Inspect the output of this job: do you see any error messages?

OUTPUTS: Job Hitting the Time Limit

```
$ cd $MYSCRATCH/supercomputing-user-training/running-jobs

$ sbatch time_limit.sh  # if instructed, add: --reservation=reserv-id
Submitted batch job 159539
$ squeue --me
 JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
159539      work first_jo username  R       0:02      1 nid001053

$squeue --me
 JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)

$ cat slurm-159539.out
The date is: Thu Aug 11 10:46:58 AWST 2022
The hostname is: nid001053
The hostname is: nid001053
The hostname is: nid001053
The hostname is: nid001053
The hostname is: nid001053
The hostname is: nid001053
The hostname is: nid001053
slurmstepd: error: *** JOB 159539 ON nid001053 CANCELLED AT 2022-08-11T10:48:14 DUE TO TIME LIMIT ***
```

- Slurm outputs a CANCELLATION DUE TO TIME LIMIT error message when the jobs hits its maximum wall time

Other Common Error Messages in a Job Output

- Segmentation Fault:
  - Typically coming from program error (invalid memory access), or insufficient memory

```
forrtl: severe (174): SIGSEGV, segmentation fault occurred
slurmstepd: error: *** STEP 4677820.0 ON nid00846 CANCELLED AT 2020-02-20T11:34:34 ***
```

- Out of Memory (OOM) Errors:
  - Various possible error messages of this type; for instance:

```
slurmstepd: error: Detected 1 oom-kill event(s) in StepId=38342.0. Some of your processes may have been killed
by the cgroup out-of-memory handler.
srun: error: nid001073: task 0: Out Of Memory
```

When something goes wrong with a job, check the output file first.
The output file of a job may contain error/warning messages, or other useful information that can help characterise an issue.

More details @
- Troubleshooting Articles

Getting Details About Queued and Past Jobs

Some useful Slurm commands to inspect job information (see user documentation):

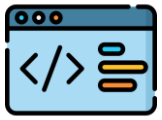
- Use `scontrol show job jobid` to get detailed information on jobs currently in the queue
  - Includes resource request, (projected) start/end time, work directory, batch script path, ..
- Use `sacct` to list your history of previous jobs
  - Select a specific job with `-j jobid`
  - Select time window with `-S timeformat` / `-E timeformat`
  - Change output format with `--format`, or the `SACCT_FORMAT` shell variable
- Use `seff jobid` to display the resource usage of a job in a more human-readable way

More details @
- Job Scheduling

Example of Output from `sacct`

```
$ sacct
JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
159405        first_job       work pawsey0001          2  COMPLETED      0:0
159405.batch      batch            pawsey0001          2  COMPLETED      0:0
159405.exte+     extern            pawsey0001          2  COMPLETED      0:0
159534        first_job       work pawsey0001          2  COMPLETED      0:0
159534.batch      batch            pawsey0001          2  COMPLETED      0:0
159534.exte+     extern            pawsey0001          2  COMPLETED      0:0
159535        first_job       work pawsey0001          2 CANCELLED+      0:0
159535.batch      batch            pawsey0001          2  CANCELLED     0:15
159535.exte+     extern            pawsey0001          2  COMPLETED      0:0
159539        first_job       work pawsey0001          2    TIMEOUT      0:0
159539.batch      batch            pawsey0001          2  CANCELLED     0:15
159539.exte+     extern            pawsey0001          2  COMPLETED      0:0
```

Interactive Job Session



DEMO on Setonix: interactive session with `salloc`

- Use `salloc` to start an interactive job session
- `salloc` takes the same options as `sbatch` (see later in this module)
- Use an interactive session for debugging, compiling, pre-/post-processing, ..

```
$ salloc  # if applicable, add: --reservation=reserv-id
salloc: Granted job allocation 160198
salloc: Waiting for resource configuration
salloc: Nodes nid001155 are ready for job

$ hostname
nid001155

$ exit
exit
salloc: Relinquishing job allocation 160198
```

You may need to wait, if the requested resources are not available

```
$ salloc -N 100
salloc: Pending job allocation 160205
salloc: job 160205 queued and waiting for resources
```

Q & A Session

Supercomputing User Training

1. Supercomputing Introduction
2. Logging In
3. Filesystems Overview
4. Moving Data In and Out
5. Using Software Modules
6. Using Software Containers
7. Accounting Model Overview
8. Job Scheduling Overview
9. Running Jobs
10. Testing Job Runs
11. Managing Project Data

Pawsey

Australian Government

NCRIS National Research Infrastructure for Australia

CSIRO

Curtin University

Murdoch University

Government of Western Australia

ECU Edith Cowan University

The University of Western Australia

Writing Batch Scripts

Quick Aside: Parallel Programming

Process (in computing)
A running instance of a program, with its own private space in memory. It can be executed as one or many threads.

Thread (in computing)
Within a process, the smallest sequence of instructions that can be executed by the operating system of the computer.

Distributed memory parallelism

- Multiple processes execute at the same time to divide and conquer a computational workflow. They can exchange information with each other via so called messages, that travel through the interconnect network.
- Most common specification: MPI (Message Passing Interface), www.mpi-forum.org

Shared memory parallelism

- The more computationally intensive portions of a process execute via multiple concurrent threads. Being part of the same process, threads have access to and share the same memory space, and hence information.
- Very common specification: OpenMP (Open Multi-Processing), www.openmp.org

Quick Aside: Parallel Programming

GPU-accelerated parallelism

- A heterogeneous, multi-device approach, where the CPU offloads the most computationally intensive portions to a GPU.
- A GPU is a specialised hardware, designed to process large blocks of data in parallel.
- Various programming interfaces available:
  - HIP (by AMD)
  - CUDA (by Nvidia)
  - OpenMP for target offloading
  - OpenACC
  - OpenCL
  - ..

Slurm Batch Script Overview

Let's start by having a look at the batch script `submit_monitor.sh` from a previous example

```
$ cd $MYSCRATCH
$ cd supercomputing-user-training/running-jobs
$ cat submit_monitor.sh

#!/bin/bash -l

#SBATCH --job-name=submit_monitor
#SBATCH --partition=work
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=00:01:00

# print date and time
echo "The date is: $(date)"

# print host name multiple times
# wait 15 seconds in between
for i in $(seq 1 2); do
   echo "The hostname is: $(hostname)"
   sleep 15s
done

# print date and time again
echo "The date is: $(date)"
```

More details @

- Job Scheduling
- Example Batch Scripts for Setonix
- Slurm Documentation (external link)

Slurm Batch Script Overview

Always use `-l` or `--login` in the initial shell definition (the "shebang") of a batch script.
This option ensures bash fully configures the shell environment for the job and avoids undesirable behaviours.

The batch script may contain any legal bash syntax, as any other shell script, including:

- Commands to configure the software environments, e.g.
  - Loading modules (see User Training 05)
  - Configuring containers (see User Training 06)
- Variable definitions
- File manipulations
- Other pre-/post-processing tasks
- Commands to execute applications (see later)

```
#!/bin/bash -l

#SBATCH --job-name=submit_monitor
#SBATCH --partition=work
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=00:01:00

# print date and time
echo "The date is: $(date)"

# print host name multiple times
# wait 15 seconds in between
for i in $(seq 1 2); do
   echo "The hostname is: $(hostname)"
   sleep 15s
done

# print date and time again
echo "The date is: $(date)"
```

Common sbatch Directives

- `job-name` (optional): assign a name to the job
  - Useful to better identify jobs in the scheduler queue
  - Default: filename of the batch script
- `account` (optional): Pawsey project
  - Useful for users that are in multiple projects
- `partition`: scheduler partition
  - See User Training 08: Job Scheduling Overview
  - Default: `work`
- `time`: maximum wall time (duration) for the job
  - Default: 1 hour
- `qos` (optional, not shown): scheduler quality-of-service
  - See User Training 08: Job Scheduling Overview
  - Default: `normal`

```
#!/bin/bash -l

#SBATCH --job-name=useful-name
#SBATCH --account=project-id
#SBATCH --partition=partition
#SBATCH --ntasks=processes
#SBATCH --ntasks-per-node=procs-per-node
#SBATCH --cpus-per-task=threads-per-proc
#SBATCH --mem=memory-per-node
#SBATCH --time=max-duration

..

srun ./program
```

Common `sbatch` Directives

- `ntasks`: maps to total number of parallel processes (typically using MPI)
  - Default: 1
- `ntasks-per-node`: number of processes per node
  - Default: 1
  - Combined with the above, enables Slurm to figure out how many nodes are needed
  - Always specify it, to avoid Slurm spreading your tasks over more nodes than needed
  - In alternative, can also use `nodes` option (default: 1)
- `cpus-per-task`: maps to number of threads per process (typically using OpenMP)
  - Default: 1
- On Setonix, `ntasks-per-node`*`cpus-per-task` ≤ 128

```
#!/bin/bash -l

#SBATCH --job-name=useful-name
#SBATCH --account=project-id
#SBATCH --partition=partition
#SBATCH --ntasks=processes
#SBATCH --ntasks-per-node=procs-per-node
#SBATCH --cpus-per-task=threads-per-proc
#SBATCH --mem=memory-per-node
#SBATCH --time=max-duration

..

srun ./program
```

- `mem` (optional): total memory per node
  - Default: ntasks-per-node*cpus-per-task / 128 * 230G
  - Useful if standard amount causes out of memory errors

Shared Node Access

- **Default mode**
- Useful if the program cannot scale to the entire node
- Allocation charged for partial node usage
- Other jobs may run on the node at the same time
- Less predictable performance
- NOTE: currently sub-optimal process/thread pinning

- Use `distribution=block:block:block`, to enforce packing of processes/threads
- IF using a socket or less (i.e. 64 cores or less), also define `ntasks-per-socket`, to pack requested resources as much as possible
- Try to use processes/threads in multiples of 8, to optimise L3 cache utilisation, by your job and by those of other users

```
#!/bin/bash -l

#SBATCH --job-name=useful-name
#SBATCH --account=project-id
#SBATCH --partition=partition
#SBATCH --ntasks=processes
#SBATCH --ntasks-per-node=procs-per-node
#SBATCH --ntasks-per-socket=procs-per-socket
#SBATCH --distribution=block:block:block
#SBATCH --cpus-per-task=threads-per-proc
#SBATCH --mem=memory-per-node
#SBATCH --time=max-duration

..

srun ./program
```

Exclusive Node Access

- Activated via `exclusive` option
  - Useful for
    - Programs using one or more full nodes
    - Programs needing all the node memory
    - Benchmarks (need predictable performance)
  - Provides all available cores and memory on the node
  - More predictable performance
  - Allocation charged for full usage of nodes
  - NOTE: process/thread pinning fully supported

```
#!/bin/bash -l

#SBATCH --job-name=useful-name
#SBATCH --account=project-id
#SBATCH --partition=partition
#SBATCH --ntasks=processes
#SBATCH --ntasks-per-node=procs-per-node
#SBATCH --cpus-per-task=threads-per-proc
#SBATCH --mem=memory-per-node
#SBATCH --time=max-duration
#SBATCH --exclusive

..

srun ./program
```

Tips for Batch Scripts

For improved batch script reproducibility, be explicit and do not rely on defaults for key requirements.
Always specify at least the following requirements: `partition`, `ntasks`, `ntasks-per-node` (or `nodes`), `cpus-per-task`, `time`.

The first time you use a new batch script, perform a test run.
This is to ensure resource requests are correct and the script executes what you need.

```
#!/bin/bash -l

#SBATCH --job-name=useful-name
#SBATCH --account=project-id
#SBATCH --partition=partition
#SBATCH --ntasks=processes
#SBATCH --ntasks-per-node=procs-per-node
#SBATCH --cpus-per-task=threads-per-proc
#SBATCH --mem=memory-per-node
#SBATCH --time=max-duration

..

srun ./program
```

**Tips on test runs and scaling tests in User Training 10: Testing Job Runs*

Executing your Program

- Use `srun`, a Slurm tool that ensures the program is using the assigned compute resources at best
- Without options, `srun` will use resources as defined via the `sbatch` directives
- If part of the workflow requires a subset of all requested resources, specify them using the same option flags
  - You must provide the `exact` option, too, to ensure the subset of resources is used and not the whole job pool
  - Note the available shortcut options `-n` and `-c`; there is also `-N` for `--nodes`
- Serial and multi-threaded programs do not strictly need `srun`; use it anyway to help form good habits

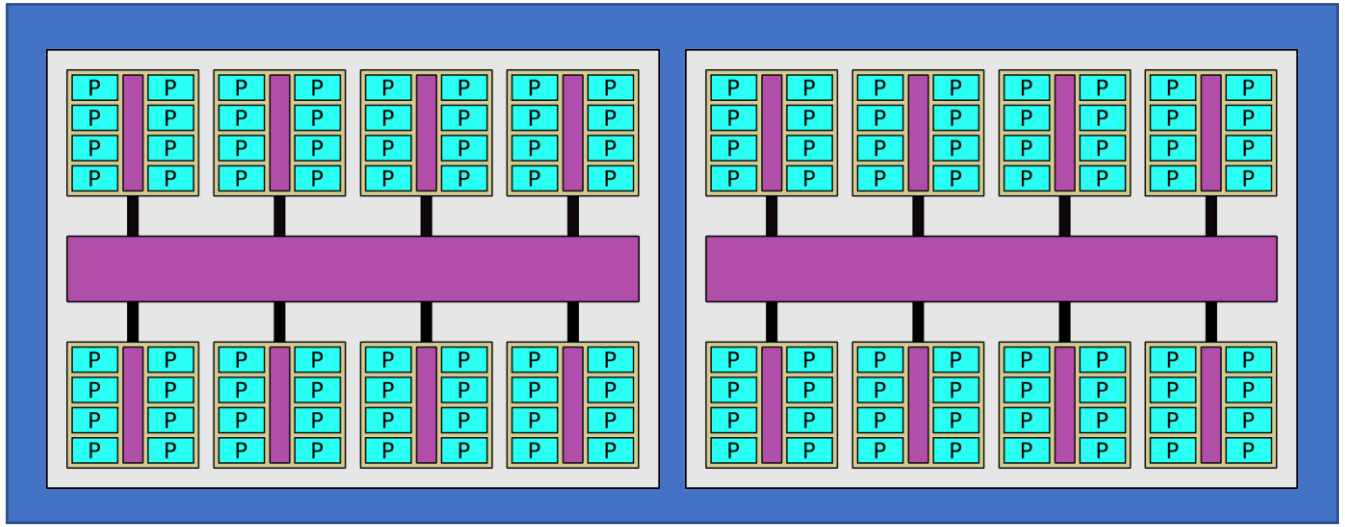
```
#!/bin/bash -l

#SBATCH --job-name=useful-name
#SBATCH --account=project-id
#SBATCH --partition=partition
#SBATCH --ntasks=processes
#SBATCH --ntasks-per-node=procs-per-node
#SBATCH --cpus-per-task=threads-per-proc
#SBATCH --mem=memory-per-node
#SBATCH --time=max-duration

..

srun ./program
# or
srun --exact -n processes2 \
--ntasks-per-node procs-per-node2 \
  -c threads-per-proc2 \
  ./program
```

Job Example: MPI with Exclusive Access



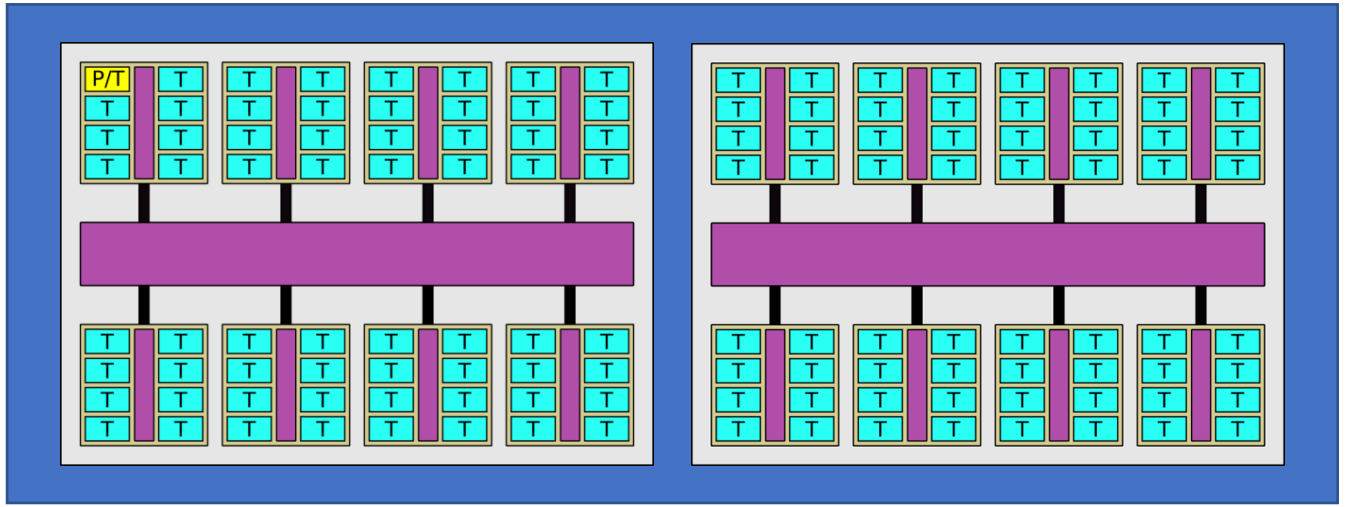

```
#!/bin/bash --login

#SBATCH --partition=work
#SBATCH --ntasks=256
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=1
#SBATCH --time=00:01:00
#SBATCH --exclusive

srun ./hello.x
```

- The script requests 2 nodes (256/128).
- For clarity, of the 2 nodes requested, only one node is shown, made of 2 sockets.
- One process (P) is launched per core (blue box).

Job Example: OpenMP with Exclusive Access



- One process (P) in a node.
- The process spawns 128 OpenMP threads (T)
- Value for the `OMP_PROC_BIND` variable
  - `close` for improved cache/memory sharing
  - `spread` if program is bound by memory bandwidth

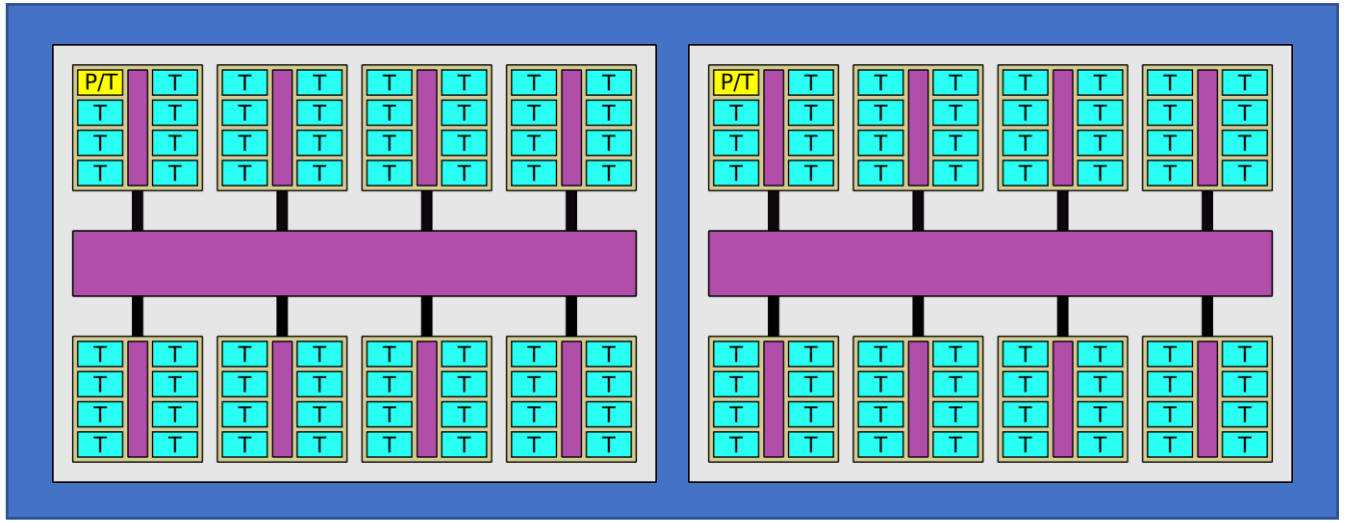
```
#!/bin/bash --login

#SBATCH --partition=work
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=128
#SBATCH --time=00:01:00
#SBATCH --exclusive

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
export OMP_PLACES=cores
export OMP_PROC_BIND=close

srun ./hello.x
```

Job Example: Hybrid MPI/OpenMP with Exclusive Access



- One process per socket, each spawning 64 OpenMP threads.
- The script requests 2 nodes (4/2). For clarity, one node is shown.

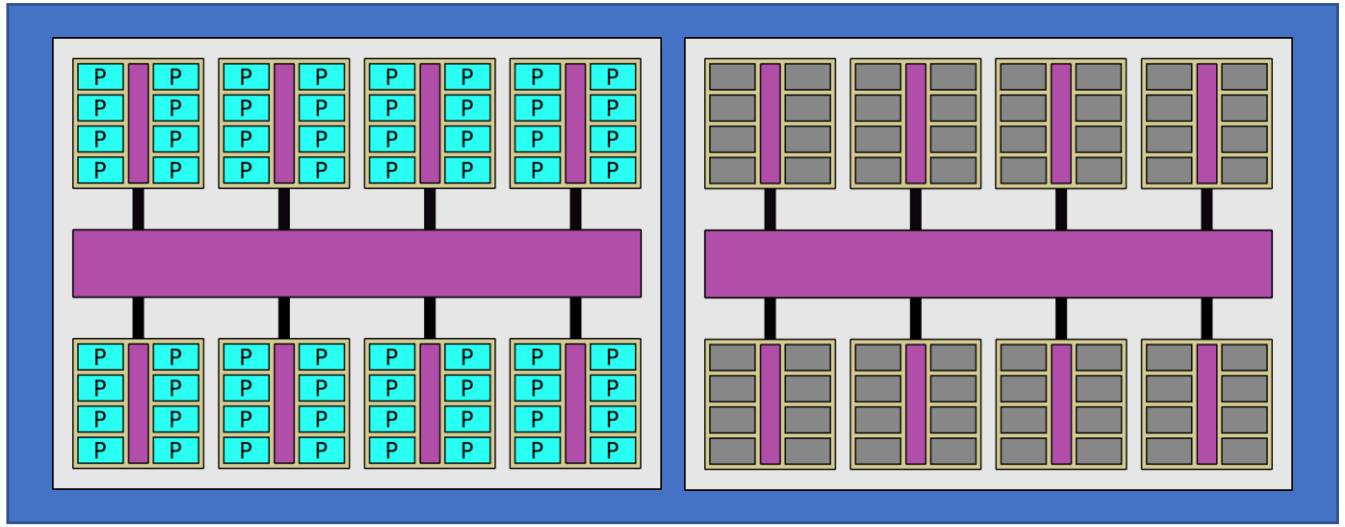
```
#!/bin/bash --login

#SBATCH --partition=work
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=64
#SBATCH --time=00:01:00
#SBATCH --exclusive

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
export OMP_PLACES=cores
export OMP_PROC_BIND=close

srun ./hello.x
```

Job Example: MPI with Shared Access



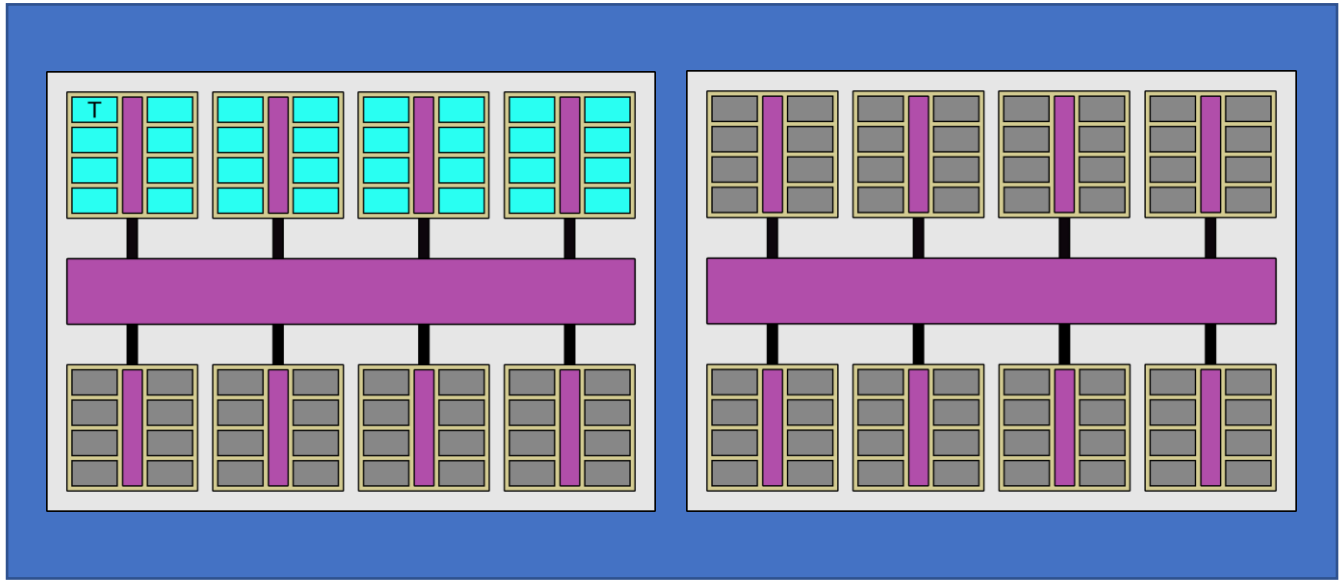

```
#!/bin/bash --login

#SBATCH --partition=work
#SBATCH --ntasks=64
#SBATCH --ntasks-per-node=64
#SBATCH --ntasks-per-socket=64
#SBATCH --distribution=block:block:block
#SBATCH --cpus-per-task=1
#SBATCH --time=00:01:00

srun ./hello.x
```

- Using `ntasks-per-socket=64` confines the job to a single socket, which is optimal when sharing a node.
- The `distribution=block:block:block` ensures tasks are packed together.
- Try to use multiples of 8 MPI processes for best L3 cache utilisation.

Job Example: Memory Request with Shared Access




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

#SBATCH --partition=work
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks-per-socket=1
#SBATCH --distribution=block:block:block
#SBATCH --cpus-per-task=1
#SBATCH --mem=58G
#SBATCH --time=00:01:00

srun ./program
```

- Using `mem=58G` amounts to requesting 1/4 of the total RAM of the node.
- Memory requests are pinned to cores, resulting in the allocation of more cores than requested.

Job Example: Requesting a GPU

- Specify how many GPUs per node you need by means of the option `gres=gpu:`*`gpus-per-node`*

- Relevant Pawsey supercomputers
  - Topaz
  - Garrawarla
  - Setonix Phase 2 [future]

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

#SBATCH --partition=gpuq
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:1
#SBATCH --time=00:01:00

srun ./program
```

More details @
- Example Batch Scripts for Topaz
- Garrawarla User Guide

EXERCISE: Run a Parallel Job

1. Go to the directory: `$MYSCRATCH/supercomputing-user-training/running-jobs/examples`
2. In the directory there is a source file, `hello.c`, and a corresponding `Makefile`. Run the command `make`, to build the program executable `hello.x`

3. Now have a look at the available batch scripts in the directory (prefix `job_`). Pick one and submit it to the scheduler queue (we recommend `job_hybrid_mpi_openmp_exclusive.sh`)
4. Monitor the job in the queue. When finished, search for output files (extension `.out`)
5. Can you find how this output filename was set in the batch script?
6. Now have a look at the contents of the output file

OUTPUTS: Run a Parallel Job

```
$ cd $MYSCRATCH/supercomputing-user-training/running-jobs/examples
$ make
cc -fopenmp -g -O3 -o hello.x hello.c

$ ls job_*
job_hybrid_mpi_openmp_exclusive.sh  job_mpi_exclusive.sh  job_openmp_exclusive.sh
job_Memory_shared.sh                job_mpi_shared.sh
$ sbatch job_hybrid_mpi_openmp_exclusive.sh
Submitted batch job 164989

$ ls *.out
job_hybrid_mpi_openmp_exclusive.sh.out
$ grep out job_hybrid_mpi_openmp_exclusive.sh
#SBATCH --output=%x.out
```

- Here, the Slurm `output` directive is used to specify the filename for the output file
- The wildcard `%x` is substituted with the job name (more on this in the Slurm Documentation)

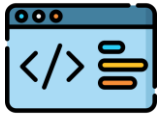
OUTPUTS: Run a Parallel Job

```
$ cat job_hybrid_mpi_openmp_exclusive.sh.out
Hello world from processor nid001260, rank 3 out of 4 processors, and here within from thread 0 out of 64 threads
Hello world from processor nid001260, rank 3 out of 4 processors, and here within from thread 41 out of 64 threads
Hello world from processor nid001260, rank 3 out of 4 processors, and here within from thread 59 out of 64 threads
Hello world from processor nid001260, rank 3 out of 4 processors, and here within from thread 31 out of 64 threads
Hello world from processor nid001260, rank 3 out of 4 processors, and here within from thread 43 out of 64 threads
Hello world from processor nid001260, rank 3 out of 4 processors, and here within from thread 47 out of 64 threads
Hello world from processor nid001260, rank 3 out of 4 processors, and here within from thread 44 out of 64 threads
Hello world from processor nid001260, rank 3 out of 4 processors, and here within from thread 42 out of 64 threads
Hello world from processor nid001260, rank 3 out of 4 processors, and here within from thread 39 out of 64 threads
Hello world from processor nid001260, rank 3 out of 4 processors, and here within from thread 36 out of 64 threads
..
```

- For each process and thread, this program outputs
  - node hostname
  - process ID and total process count
  - thread ID and total thread count

Summary - 1

- Terms we learnt
  - Batch Script
  - Process (in computing)
  - Thread (in computing)



- Tasks we learnt
  - Submit, monitor and cancel batch jobs: `sbatch`, `squeue` and `scancel`
  - Start an interactive job session: `salloc`

- Syntax basics for batch scripts

Summary - 2

- For improved reproducibility, specify resource requirements within the batch script rather than via command line.
- When something goes wrong with a job, check the output file first.
- Always use **`-l`** or **`--login`** in the initial shell definition (the “shebang”) of a batch script.
- For improved batch script reproducibility, be explicit and do not rely on defaults for key requirements.
- The first time you use a new batch script, perform a test run.

Pawsey

Australian Government

NCRIS National Research Infrastructure for Australia

CSIRO

Curtin University

Murdoch University

Government of Western Australia

ECU Edith Cowan University

The University of Western Australia

Getting Help

Getting Help

https://support.pawsey.org.au

Pawsey has extensive User Support Documentation.

Areas covered include:

- System user guides
- Knowledge Base
- Pawsey-supported software list
- Maintenance logs
- Policies and terms of use

For further assistance, contact the help desk, via User Support Portal.

Help us to help you by providing details, such as:

- Which resource
- Error messages
- Location of files
- SLURM job id
- Your username if having login issues
- Never tell us (or anyone) your password!

Become a Pawsey Friend and receive our Newsletter:
https://pawsey.org.au/pawsey-friends/

Q & A Session