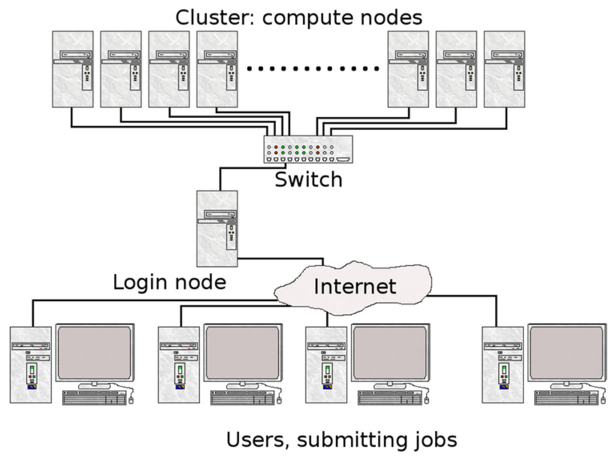
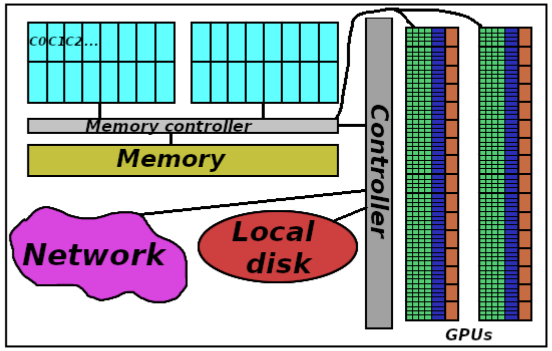
Slurm at NAISS

Overview

- What is a batch scheduler?
- Slurm commands
- Site specifics
- Serial, MPI, OpenMP, GPUs
- Dependencies
- Increasing the memory per task
- I/O intensive jobs
- Job arrays
- Job monitoring and efficiency
- Interactive work

Batch schedulers





- Provides a mechanism to submit programs (jobs) to be executed automatically.
- Usually associated with large compute clusters (Tetralith, Dardel, Alvis, Kebnekaise…)
- Many compute nodes (calculations)
- Login node (submission, compiling, …)
- Global disk (shared for all cluster, visible for all nodes)
- Node-local disk (best performance, only visible inside node during job)

Batch schedulers - job types

Many types of jobs can be handled by a job scheduler and made to run efficiently on a cluster. These are some typical examples:

- Tasks that can be split up into many serial jobs
- Many instances of the same task
- MPI jobs
- Shared memory
- GPU jobs

Batch scheduler key functions

- Keeps track of available system resources - it allocates to users, exclusive or non-exclusive access to resources for some period of time
- Enforces local system resource usage and job scheduling policies - provides a framework for starting, executing, and monitoring work
- Manages a job queue, distributing work across resources according to policies

NOTE:

You can either give all the commands on the command line or use a job script. Using a job script is often recommended (terminal is available, easy record of commands, etc.)

Batch schedulers - NAISS

- Large/long/parallel jobs **must** be run through the batch system.
    - Work done through OpenOnDemand where that is available is automatically run through the batch system
- The Swedish HPC centres under NAISS all use Slurm
- Slurm is an Open Source job scheduler which handles the three key functions mentioned previously

Slurm commands refresher

When you submit a job, the system returns a Job ID.

The Job ID is also found from `squeue --me`.

- **Submit job**: `sbatch JOBSCRIPT`
- **Get list of your jobs**: `squeue -u USERNAME` or `squeue --me`
- **Give Slurm commands on command line**: `srun <commands-for-your-job> program`
- **Check on a specific job**: `scontrol show job JOBID`
- **Delete a specific job**: `scancel JOBID`
- **Delete all your own jobs**: `scancel -u USERNAME`
- **Submit job**: `sbatch JOBSCRIPT`

Slurm - job states

Here I have submitted a jobscript `simple.sh` a few times:

```
b-an01 [~]$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          27774876  cpu_zen4, simple.s bbrydsoe  PD       0:00      1 (None)
          27774875  cpu_zen4, simple.s bbrydsoe  PD       0:00      1 (None)
          27774873  cpu_zen4  simple.s bbrydsoe  R        0:02      1 b-cn1702
          27774874  cpu_zen4  simple.s bbrydsoe  R        0:02      1 b-cn1702
          27774872  cpu_zen4  simple.s bbrydsoe  CG       0:04      1 b-cn1702
```

- **CA**: CANCELLED. Job was explicitly cancelled by the user or system administrator.
- **CF**: CONFIGURING. Job has been allocated resources which are not yet ready for use (e.g. booting).
- **CG**: COMPLETING. Job is completing, but some processes may still be active.
- **PD**: PENDING. Job is awaiting resource allocation.
- **R**: RUNNING. Job currently has an allocation.
- **S**: SUSPENDED. Job has an allocation, but execution is suspended and resources released for other jobs.

Slurm - job starting times

- the estimated start time can be seen from:
  - `scontrol show job JOBID`
  - or `squeue --me --start`
- the estimated start time **may change** depending on other jobs being submitted, starting, or ending (perhaps earlier than expected)

Slurm - Hands-on

1. Go to the sub-directory for your centre
2. Change the project ID in "simple.sh" if you are not at NSC, PDC, C3SE, or HPC2N
3. Submit the job script "simple.sh" a number of times
4. Type `squeue --me` to see a list of your jobs and their states
5. If the job ran too fast, then submit it several more times to see the output of `squeue --me`
6. You can also look at `scontrol show job JOBID` for more information about your job

Site specifics

There are minor differences between the HPC centres in Sweden.

- CPU/GPU types
- installed modules
- interactivity

Examples: Tetralith (NSC), Dardel (KTH), Alvis (C3SE), Kebnekaise (HPC2N)

Site specifics

Resource	cores/node	RAM/ node	GPUs, type (per node)
Tetralith	32	96-384 GB	Nvidia T4 GPUs (1)
Dardel	128	256-2048 GB	4 AMD Instinct™ MI250X (2)
Alvis	16 (skylake 2xV100), 32 (skylake 4xV100, 8xT4), 64 (icelake 4xA40, 4xA100)	256-1024 GB	Nvidia v100 (2), v100 (4), T4 (8), A40 (4), A100 (4)
Kebnekaise	28 (skylake), 72 (largemem), 128/256 (Zen3/Zen4)	128-3072 GB	NVidia v100 (2), a100 (2), a6000 (2), l40s (2 or 6), H100 (4), A40 (8), AMD MI100 (2)

- Alvis has a small number of nodes without GPUs, for heavy-duty pre- and postprocessing that does not require a GPU. To use, specify the constraint `-C NOGPU` in your Slurm script.

Batch Scripts

Serial job:

```
#!/bin/bash
#SBATCH -A <account>
#SBATCH -t HHH:MM:SS
#SBATCH -n 1

module load <modules>

./myserialprogram
```

- `-n` is number of tasks, where cores/task (`--cpus-per-task`) is 1 as default
- NOTE: Alvis is **only** used for GPU jobs

Batch scripts

MPI job:

```
#!/bin/bash
#SBATCH -A <account>
#SBATCH -t HHH:MM:SS
#SBATCH -n <tasks>

module load <modules>

srun ./mympiprogram
```

- Asking for whole nodes (`- N`) and possibly `--tasks-per-node`
- `srun` and `mpirun` should be interchangeable at HPC2N. Tetralith uses `mpprun` and Dardel uses `srun`
- Remember, you need to load modules with MPI
- At some centres `mpirun --bind-to-core` or `srun --cpu-bind=cores` is recommended for MPI jobs
- NOTE: Alvis is **only** used for GPU jobs

Note: Time/walltime

- the job **will** terminate when the time runs out, whether it has finished or not
- you will only be "charged" for the consumed time
- asking for more time than needed will generally make the job take longer to start
- short jobs can start quicker (backfill)
- if you have no idea how long your job takes, ask for "long" time
- Conclusion: Ask for "a bit" more time than needed, but not too much

Batch jobs

OpenMP job:

```
#!/bin/bash
#SBATCH -A <account>
#SBATCH -t HHH:MM:SS
#SBATCH -c <cores-per-task>

module load <modules>

# Set OMP_NUM_THREADS to the same value as -c with a fallback in case it isn't set.
# SLURM_CPUS_PER_TASK is set to the value of -c, but only if -c is explicitly set
if [ -n "$SLURM_CPUS_PER_TASK" ]; then
  omp_threads=$SLURM_CPUS_PER_TASK
else
  omp_threads=1
fi
export OMP_NUM_THREADS=$omp_threads

./myopenmpprogram
```

- `-c` is used to set cores per task and should be the same as `OMP_NUM_THREADS`
- Remember, Alvis is only for GPU jobs

Note: Binding job to cores, threads, sockets

- Binding can help performance
- The case of hybrid MPI+OpenMP jobs (both -n (and/or –ntasks-per-node) and -c/–cpus-per-task has been defined) is where binding jobs to core could help most
- At LUNARC it is recommended to use `mpirun --bind-to-core` or `srun --cpu-bind=cores` for MPI jobs
- On Alvis they recommend to let the batch system handle the binding by using "srun" instead of "mpirun"
- Manually, you can use the `srun --cpu-bind` option:
    - `--cpu-bin=cores` auto-generated masks bind to cores
    - `--cpu-bind=socket` auto-generated masks bind to sockets
    - `--cpu-bind=thread` auto-generated masks bind to threads
    - `--cpu-bind=verbose` verbosely report binding before task runs
    - `--cpu-bind=no` do not bind tasks (default)
- It is possible to give an explicit mask to bind with `--cpu-bind=mask_cpu:<list>`

Using GPUs

This is the most different of the Slurm settings, between centers.

Resource	batch settings	Comments

Tetralith	`#SBATCH -n 1`
	`#SBATCH -c 32`
	`#SBATCH --gpus-per-task=1`	

Dardel	`#SBATCH -N 1`
	`#SBATCH --ntasks-per-node=1`
	`#SBATCH -p gpu`	

Alvis	`#SBATCH -p alvis`
	`#SBATCH -N <nodes>`
	`#SBATCH --gpus-per-node=<type>:x`
		- no node-sharing on multi-node jobs
		(`--exclusive` is automatic)
		- Requesting -N 1 does not mean 1 full node

Kebnekaise	`#SBATCH --gpus=x`
	`#SBATCH -C <type>`	

Dependencies

Sometimes your workflow has jobs that depend on a previous job (a pipeline). This can be handled through Slurm (if many, make a script):

- Submit your first job: `sbatch my-job.sh`
- Wait for that job to finish before starting next job:
  `sbatch -d afterok:<prev-JOBID> my-next-job.sh`

Generally:

- **after:jobid[:jobid...]** begin after specified jobs have started
- **afterany:jobid[:jobid...]** begin after specified jobs have terminated
- **afternotok:jobid[:jobid...]** begin after specified jobs have failed
- **afterok:jobid[:jobid...]** begin after specified jobs have run to completion with exit code zero
- **singleton** begin execution after all previously launched jobs with the same name and user have ended

Increasing the memory per task

- Running out of memory ("OOM"):
  - usually the job stops ("crashes")
  - check the Slurm error/log files
  - check with sacct/seff/jobstats/job-usage depending on cluster
- Fixes:
  - use "fat" nodes
  - allocate more cores just for memory
  - tweak mem usage in app, if possible

Increasing memory per task - Tetralith

96 GB/node:

```
-C thin --exclusive
```

384 GB/node:

```
-C fat --exclusive
```

More cores/task, some just giving memory (MPI):

```
sbatch --ntasks-per-node=16 --cpus-per-task=2
```

You can write `--cpus-per-task=#num` in short form as `-c #num`.

Increasing memory per task - Dardel

Type	RAM	Partition	Available	Example flag
Thin	256 GB	main, shared, long	227328 MB	
Large	512 GB	main, memory	456704 MB	`--mem=440GB`
Huge	1 TB	main, memory	915456 MB	`--mem=880GB`
Giant	2 TB	memory	1832960 MB	`--mem=1760GB`
GPU	512 GB	gpu	456704 MB	`--mem=440GB`

On shared partitions you need to give number of cores and will get RAM equivalent for that

Increasing memory per task - Alvis

RAM	GPUs	Example flag

768	V100 (2)
	V100 (4)
	and a no GPU skylake
		`#SBATCH -C MEM768`
		`#SBATCH --gpus-per-node=V100:[1-4]`

576	T4 (8)	`#SBATCH -C MEM576`
		`#SBATCH --gpus-per-node=T4:[1-8]`

1536	T4 (8)	`#SBATCH -C MEM1536`
		`#SBATCH --gpus-per-node=A40:[1-4]`

512	A100 (4)
	and a no GPU icelake
		`#SBATCH -C mem512`
		`#SBATCH --gpus-per-node=A100:[1-4]`

256	A40 (4, no IB) A100 (4)	`#SBATCH -C mem256` and either `#SBATCH --gpus-per-node=A40[1-4]` or `#SBATCH --gpus-per-node=A100[1-4]`

1024	A100fat (4)	`#SBATCH -C mem1024`
		`#SBATCH --gpus-per-node=A100fat:[1-4]`

- **Note** though that you also need to ask for a GPU, as usual, unless you need the pre/post processing CPU nodes (`-C NOGPU`).
- You only really need to give the mem constraint for those bolded as the others follow from the GPU choice
- `sinfo -o "%20N %9P %4c %24f %50G"` will give you a full list of all nodes and features

Increasing memory per task - Kebnekaise

Type	RAM/core	cores/node	requesting flag
Intel Skylake	6785 MB	28	`-C skylake`
AMD Zen3	8020 MB	128	`-C zen3`
AMD Zen4	2516 MB	256	`-C zen4`
V100	6785 MB	28	`--gpus=<#num> -C v100`
A100	10600 MB	48	`--gpus=<#num> -C a100`
MI100	10600 MB	48	`--gpus=<#num> -C mi100`
A6000	6630 MB	48	`--gpus=<#num> -C a6000`
H100	6630 MB	96	`--gpus=<#num> -C h100`
L40s	11968 MB	64	`--gpus=<#num> -C l40s`
A40	11968 MB	64	`--gpus=<#num> -C a40`
Largemem	41666 MB	72	`-C largemem`

- Can also ask for more cores/task with `-c <#cores/task>` just to add memory

I/O intensive jobs

- In most cases, you should use the project storage
- Centre-dependent. If needed you can use node-local disk for **single-node** jobs
    - Remember you need to copy data to/from the node-local scratch (`$SNIC_TMP`)!
    - On some systems `$TMPDIR` also points to the node local disk
    - The environment variable `$SLURM_SUBMIT_DIR` is the directory you submitted from
- On Tetralith, the data access between /home or /proj and GPU/CPU compute nodes are **not** suitable for I/O intensive jobs => use /scratch/local (`$SNIC_TMP`)

I/O intensive jobs - example

```bash
#!/bin/bash
#SBATCH -A <account>
#SBATCH -t HHH:MM:SS
#SBATCH -n <cores>

module load <modules>

# Copy your data etc. to node local scratch disk
cp -p mydata.dat $SNIC_TMP
cp -p myprogram $SNIC_TMP

# Change to that directory
cd $SNIC_TMP

# Run your program
./myprogram

# Copy the results back to the submission directory
cp -p mynewdata.dat $SLURM_SUBMIT_DIR
```

When using node local disk it is important to remember to copy the output data back, since it will go away when the job ends!

Multiple jobs, same job script

- you have several (perhaps smaller) programs to run
- you can just have the jobs wait in queue once for all of them
- they are run from the same job script and have similar requirements
- if you have a few, just list all in the same script
- if you have many, job arrays or making a script to submit them are better

Multiple simultaneous jobs (serial or MPI)

In this example, 3 jobs each with 14 cores

```
#!/bin/bash
#SBATCH -A <account>
# Since the files run simultaneously I need enough cores for all of them to run
#SBATCH -n 56
# Remember to ask for enough time for all jobs to complete
#SBATCH --time=00:10:00

module load <modules>

srun -n 14 --exclusive ./mympiprogram data &
srun -n 14 --exclusive ./my2mpi data2 &
srun -n 14 --exclusive ./my3mpi data3 &
wait
```

Multiple sequential jobs (serial or parallel)

This example is for jobs where some are with 14 tasks with 2 cores per task and some are 4 tasks with 4 cores per task

```
#!/bin/bash
#SBATCH -A <account>
# Since the programs run sequentially I only need enough cores for the largest of th
#SBATCH -c 28
# Remember to ask for enough time for all jobs to complete
#SBATCH --time=HHH:MM:SS

module load <modules>

srun -n 14 -c 2 ./myprogram data
srun -n 4 -c 4 ./myotherprogram mydata
...
srun -n 14 -c 2 ./my2program data2
```

Job arrays

- Job arrays: a mechanism for submitting and managing collections of similar jobs.
- All jobs must have the same initial options (e.g. size, time limit, etc.)
- the execution times can vary depending on input data
- You create multiple jobs from one script, using the `-- array` directive.
- This requires very little BASH scripting abilities
- max number of jobs is restricted by max number of jobs/user
- More information here on the official Slurm documentation pages: https://slurm.schedmd.com/job_array.html.

Job arrays - simple example

- a small Python script `hello-world-array.py`

```python
# import sys library (we need this for the command line args)
import sys

# print task number
print('Hello world! from task number: ', sys.argv[1])
```

- a batch script `hello-world-array.sh`

```bash
#!/bin/bash
# A very simple example of how to run a Python script with a job array
#SBATCH -A <account>
#SBATCH --time=00:05:00 # Asking for 5 minutes
#SBATCH --array=1-10   # how many tasks in the array
#SBATCH -c 1 # Asking for 1 core    # one core per task
# Create specific output files for each task with the environment variable %j
# which contains the job id and %a for each step
#SBATCH -o hello-world-%j-%a.out

# Load any modules you need, here for Python 3.11.3 on Kebnekaise
module load GCC/12.3.0 Python/3.11.3

# Run your Python script
srun python hello-world-array.py $SLURM_ARRAY_TASK_ID
```

Job arrays - comments

- Default step of 1
  - Example: `#SBATCH --array=4-80`
- Give an index (here steps of 4)
  - Example: `#SBATCH --array=1-100:4`
- Give a list instead of a range
  - Example: `#SBATCH --array=5,8,33,38`
- Throttle jobs, so only a smaller number of jobs run at a time
  - Example: `#SBATCH --array1-400%4`
- Name output/error files so each job (`%j` or `%A`) and step (`%a`) gets own file
  - `#SBATCH -o process_%j_%a.out`
  - `#SBATCH -e process_%j_%a.err`
- There is an environment variable `$SLURM_ARRAY_TASK_ID` which can be used to check/query with

Job monitoring and efficiency - All centres

Command	**What**
`scontrol show job JOBID`	info about a job, including *estimated* start time
`squeue --me --start`	your running and queued jobs with *estimated* start time
`sacct -l JOBID`	info about job, pipe to `less -S` for scrolling side-ways (it is a wide output)
`projinfo`	usage of your project, adding `-vd` lists member usage
`sshare -l -A <proj-account>`	gives priority/fairshare (LevelIFS)

Most up-to-date project usage on a project's SUPR page, linked from here: https://supr.naiss.se/project/

Job monitoring and efficiency - specific centres

Command	What	Centre
`jobinfo`	wrapper around `squeue`	UPPMAX, LUNARC, C3SE
`jobstats -p JOBID`	CPU and memory use of finished job (> 5 min) in a plot	UPPMAX
`job_stats.py`	link to Grafana dashboard with overview of your running jobs. Add `JOBID` for real-time usage of a job	C3SE
`job-usage JOBID`	grafana graphics of resource use for job (> few minutes)	HPC2N
`jobload JOBID`	show cpu and memory usage in a job	NSC
`jobsh NODE`	login to node, run "top"	NSC
`seff JOBID`	displays memory and CPU usage from job run	NSC, PDC
`lastjobs`	lists 10 most recent job in recent 30 days	NSC
https://pdc-web.eecs.kth.se/cluster_usage/	Information about project usage	PDC

Why is a job ineffective?

- more threads than allocated cores
- not using all the cores you have allocated (unless on purpose/for memory)
- inefficient use of the file system (many small files, open/close many files)
- running job that could run on GPU on CPU instead

Interactive work

Cluster	`interactive`	`salloc`	OpenOnDemand	Other
Kebnekaise (HPC2N)	Works	Recommended	Recommended	
Pelle (UPPMAX)	Recommended	Works	N/A	
Cosmos (LUNARC)	Works	N/A	Recommended (GfxLauncher)	
Tetralith (NSC)	Recommended	N/A	N/A	
Dardel (PDC)	N/A	Recommended	Possible	
C3SE (Alvis)	N/A	N/A	Recommended	srun

Interactive work - examples

- `interactive -A [proj] [other options]`
- `salloc -A [proj] -t HHH:MM:SS [other options]`
    - time is required at HPC2N, but not at NSC
    - all commands must be run with `srun [command]` at HPC2N. Not "real" interactivity
    - at NSC you can login to the allocated compute node with `ssh [node]` and run there
- OpenOnDemand
    - Cosmos: Login to ThinLinc, start one of the apps listed under "Applications"
    - Kebnekaise: "https://portal.hpc2n.umu.se"
    - Dardel: Login to ThinLinc, start one of the apps listed under "Applications"
    - Alvis: "https://alvis.c3se.chalmers.se"
- Alvis: `srun -A [proj] --gpus-per-node=T4:1 -t 01:00:00 --pty=/bin/bash`

Hands-ons

1. Submit a non GPU job ("simple.sh", "MPI.sh", or "OpenMP.sh"). Check with `scontrol show job JOBID` which type of node was allocated.
2. Submit a GPU job ("GPU.sh") and check with `scontrol show job JOBID` which type of node was allocated.
3. Try some of the monitoring tools that are listed for your centre. You need a longer job to get data for the tools "job-usage" or "job_stats". Suggested to try and run "job-gpu.sh" (or "run_mmmult.sh" for a non-GPU job, particularly on PDC).
4. "run_mmmult-v2.sh" depends on "run_matrix-gen.sh". Submit "run_matrix-gen.sh" and then submit "run_mmmult-v2.sh" so that it does not start until "run_matrix-gen.sh" has run. Check that it works.
5. (Optional) Create a script that submits "run_matrix-gen.sh" and "run_mmmult-v2.sh", but so that "run_mmmult-v2.sh" does not start until "run_matrix-gen.sh" has run.
6. Make a jobscript that asks for more cores per task, but just for using the extra memory.
7. Try and run the "hello-world-array.sh" job. Change the size of the array or make other changes. See the difference in output files.