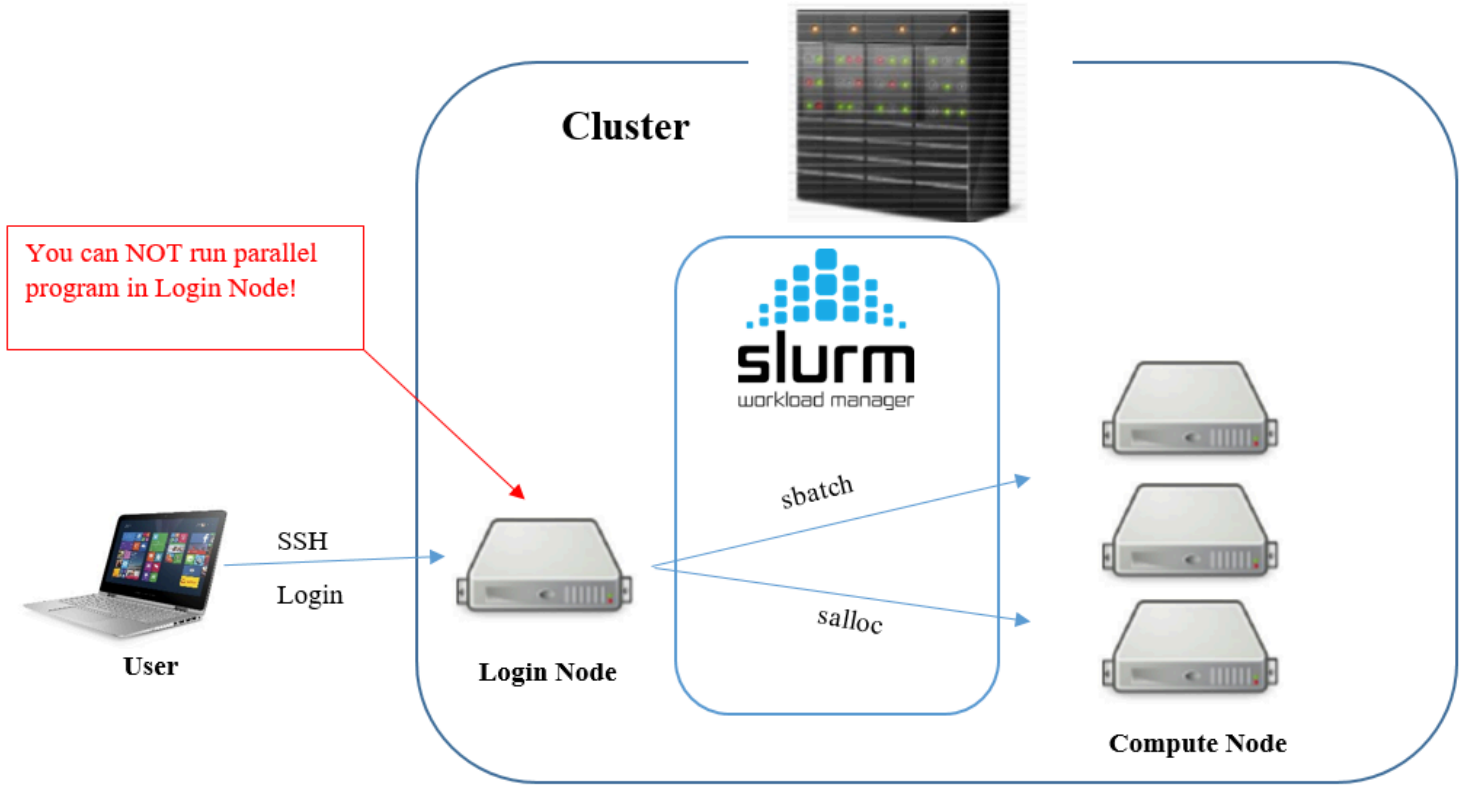
Running jobs with efficient utilization of hardware

Arash Banaei

How do we run jobs on a Supercomputer ?

On the login node you:

- Edit files
- Compile programs
- Submit jobs
- **Do not** run computations on the login node!



What is SLURM ?

Simple Linux Utility for Resource Management

- Open source, fault-tolerant, and highly scalable cluster management and job scheduling system
  - Allocates access to resources
  - Provides a framework for work monitoring on the set of allocated nodes
  - Arbitrates contention for resources

How jobs are prioritized?

- **Age**: the time the job has been in the queue
- **Job size**: number of nodes or cores requested
- **Partition**: a factor associated with each node configuration
- **Fair-share**: the difference between the computing resources promissed and the amount of resources used

Types of jobs?

- **Batch jobs**
  - the user writes a job script indicating the number of nodes, cores, time needed, etc.
  - the script is submitted to the batch queue
  - The user can monitor the output files during the runtime
- **Interactive jobs:**
  - the user runs a command that allocates interactive resources on a number of cores
  - this creates an interactive job that awaits in the queue as any other job
  - when the job reaches the front of the queue, the user gets access to the resources

SLURM basic commands

Requests resources for interactive use:

```
salloc <options>
```

Submit a job to the queue:

```
sbatch <script>
```

List queued/running jobs belonging to a user:

```
squeue -u <username>
```

Cancel a job:

```
scancel <job-id>
```

Which allocation I am a member of

projinfo

```
$ projinfo -h
Usage: projinfo [-u <username>] [-c <clustername>] [-a] [-o] [-m] [-c <cluster>] [-d] [-p <DNR>] [-h]
-u [user] : print information about specific user
-o : print information about all (old) projects, not just current
-m : print usage of all months of the project
-c [cluster] : only print allocations on specific cluster
-a : Only print membership in projects
-d : Usage by all project members
-p [DNR] : only print information about this project
-h : prints this help
```

Shows information about membership, allocation use, storage paths, and stored quota

Usage statistics for every allocation are also available at…
https://support.pdc.kth.se/doc/stats

Partitions

- Nodes are logically grouped into partitions
- There are several partitions that can be used on Dardel
  - **main**: 128 CPU cores, 256 GB -512 GB- 1 TB RAM, exclusive access, maximum job time 24 hours
  - **long**: 128 CPU cores, 256 GB - 512 GB RAM, exclusive access, maximum job time 7 days
  - **shared**: 128 CPU cores, 256 GB RAM, job shares node with other jobs, maximum job time 7 days
  - **memory**: 128 CPU cores, 512 GB - 1 TB - 2 TB RAM, exclusive access, maximum job time 7 days
  - **gpu**: 64 CPU cores, GPU nodes, 512 GB RAM, and 4 AMD MI250X GPU cards, exclusive access, maximum job time 24 hours

Important note about memory

- A certain amount of the memory on the nodes is used by system operations

Total memory	Approximate available memory for user job
256 GB	230 GB
512 GB	480 GB
1 TB	980 GB
2 TB	1760 GB

Core/Memory allocation on the shared partition

If requesting number of cores without any memory specification:

- Memory is allocated proportional to the requested cores
- Maximum allocated memory is 111G (~ half of available memory on a node)
- For example if requesting 64 cores, you will get around 115G memory

If requesting specific amount of memory without specifying number of cores:

- The cores are allocated based on the **half of available memory on a node**
- For example if requesting 55G, you will be given 64 cores
- If requesting 111G, you will be given 128 cores (the entire node)
- If requesting >111G, you will be given 128 cores (the entire node)

Job submission with job scripts

Create a file containing details of the resource allocation and running your software

```
#!/bin/bash -l

#SBATCH -J myjob

#SBATCH -A naissXXXX-Y-ZZ

#SBATCH -p shared

#SBATCH -t 1-00:00:00

module load X
module load Y

srun myexe > my_output_file
```

```
sbatch <name of the script>
```

Job scripts (serial)

```
#!/bin/bash -l
# The -l above is required to get the full environment with modules

# The name of the job is myjob
#SBATCH -J myjob

# Set the allocation to be charged for this job
#SBATCH -A naissXXXX-Y-ZZ

# The partition
#SBATCH -p memory

# Required memory

#SBATCH --mem 480G

# 7 days time will be given to this job
#SBATCH -t 7-00:00:00

# Run the executable named myexe
srun myexe > my_output_file
```

Job scripts (OpenMP)

```
#!/bin/bash -l
# The -l above is required to get the full environment with modules

# The name of the job is myjob
#SBATCH -J myjob

# Set the allocation to be charged for this job
#SBATCH -A naissXXXX-Y-ZZ

# Number of cores
#SBATCH -c 10

# binding threads to physical cores

#SBATCH --hint=nomultithread

# The partition
#SBATCH -p shared

# Required memory

#SBATCH --mem 50G

# 10 hours wall-clock time will be given to this job
#SBATCH -t 10:00:00

export SRUN_CPUS_PER_TASK=$SLURM_CPUS_PER_TASK

# Run the executable named myexe
srun myexe > my_output_file
```

Job scripts (MPI)

```
#!/bin/bash -l
# The -l above is required to get the full environment with modules

# The name of the job is myjob
#SBATCH -J myjob

# Set the allocation to be charged for this job
#SBATCH -A naissXXXX-Y-ZZ

# Number of nodes

#SBATCH --nodes=4

# Number of MPI tasks
#SBATCH -n 512

# Number of MPI tasks per node

#SBATCH --ntasks-per-node=128

# The partition
#SBATCH -p main

# Required memory

#SBATCH --mem 480G

# 10 hours wall-clock time will be given to this job
#SBATCH -t 10:00:00

# Run the executable named myexe
srun myexe > my_output_file
```

Job scripts (packed jobs running serially)

Several short jobs can be packed together into a single job where they run serially

```
#!/bin/bash -l
#SBATCH -A naissXXXX-Y-ZZ
#SBATCH -N 1
#SBATCH -n 4
#SBATCH -p shared
#SBATCH -t 00:10:00

for arg in "$@"; do
        srun myexe $arg
done
```

The job is submitted with:

```
sbatch packed_job.sh x0 x1 x2 x3 x4 x5 x6 x7 x8 x9
```

Job scripts (packed jobs running in parallel)

```
#!/bin/bash -l
#SBATCH -A naissXXXX-Y-ZZ
#SBATCH -N 1
#SBATCH -p main
#SBATCH -t 00:08:00

for arg in "$@"; do
  srun -n 4 myexe $arg &
done
wait
```

The job is submitted with:

```
sbatch packed_job.sh x0 x1 x2 x3 x4 x5 x6 x7 x8 x9
```

Job scripts (GPU)

```
#!/bin/bash -l
# The -l above is required to get the full environment with modules

# The name of the job is myjob
#SBATCH -J myjob

# Set the allocation to be charged for this job
#SBATCH -A naissXXXX-Y-ZZ

# Number of nodes

#SBATCH --nodes=1

# Number of MPI tasks per node

#SBATCH --ntasks-per-node=8

# The partition
#SBATCH -p gpu

# GPUs per MPI task

#SBATCH --gpus-per-task=1

# 10 hours wall-clock time will be given to this job
#SBATCH -t 10:00:00

module load rocm/6.0.0
module load craype-accel-amd-gfx90a


# Run the executable named myexe
srun myexe > my_output_file
```

Submitting job to the queue

```
sbatch  <script_name>
```

Check status of your pending/running job

```
squeue -u <username> or squeue --me
```

Cancel your job:

```
scancel <job-id>
```

SSH to a compute node

```
ssh nid<XXXXXX>
```

Interactive jobs (MPI)

Request an interactive allocation

```
salloc -A <allocation> -t <d-hh:mm:ss> -p <partition> -N <nodes> -n <tasks>
```

Once the allocation is granted, a new terminal session starts (typing exit will stop the interactive session).
You will still be on the login node. Make sure you use srun to launch your job.

```
srun mybinary.x
```

It is also possible to ssh into one of the allocated nodes.

Interactive jobs (OpenMP)

Request an interactive allocation

```
salloc -A <allocation> -t <d-hh:mm:ss> -p <partition> -c <cores> --hint=nomultithread
```

```
srun mybinary.x
```

Interactive jobs

We can check what nodes have been granted with:

```
squeue -u <username> or squeue --me
```

or inspecting the environment variable:

```
echo  $SLURM_NODELIST
```

SLURM advanced commands

Get detailed information of a running job:

```
sstat --jobs=<your_job-id>
```

Filter the information provided by sstat

```
sstat --jobs=your_job-id --format=jobid,maxrss,maxrssnode,ntasks
```

Tip: Use *sstat -e* to see all possible fields for the *format* flag

SLURM advanced commands

Get information about finished/failed jobs:

```
sacct
```

Get detailed information of a certain job:

```
sacct --jobs=<your_job-id> --starttime=YYYY-MM-DD
```

```
sacct --starttime=2025-08-13 --format=jobid,jobname,nodelist,maxrss,stat,exitcode
```

Tip: Use *sacct -e* to see all possible fields for the *format* flag

SLURM advanced commands

Quick performance summary for a finished job:

```
seff <jobid>
```

Important note: due to multi-threaded feature of Dardel CPUs, the number for CPU efficiency should be multiplied by 2 to get the real one

SLURM good practices

- Avoid too many short jobs
- Avoid massive output to STDOUT
- Try to provide a good estimate of the job duration before submitting
- For very long jobs use checkpoints to restart the simulation

Common reasons for inefficient jobs

- Not all nodes allocated are used
- Not all cores within a node are used (if it's not intentional)
- Many more cores than the available are used
- Inefficient use of the file system

Exercise 1: Basic SLURM commands

- In this exercise you are going to test some basic SLURM commands.
- You will:
  - Create and compile a simple MPI program
  - Submit it to the queues
  - Inspect the queues
  - Inspect the partitions
  - Check the output of the job

MPI Hello world

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
  int world_size,world_rank;

  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Get the number of processes
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  // Get the rank of the process
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  // Print off a hello world message
  printf("Hello world from rank %d out of %d process\n", world_rank, world_size);
  // Finalize the MPI environment.
  MPI_Finalize();
  return 0;
  }
```

Exercise 1

- You can find the code from the previous slide here
- Save the file on Dardel, compile the code and generate a binary called *hello_mpi.x*

```
cc -o hello_mpi.x hello_mpi.c
```

Exercise 1

- Take the job script that you can find here and modify it accordingly to:
  - Use one node for the job
  - Use 4 cores from that node

Exercise 1

- Submit this script using **sbatch**
- Check the queue using **squeue --me**
  - What's the ID of the job?
  - Is it already running? If so, which node was allocated for the job?
- Once the job finishes check the job output. Where is it saved?

Note:

Notice that you run our program with just:

```
srun hello_mpi.x
```

It would be also possible to run our program with:

```
srun -N 1 -n 4 hello_mpi.x
```

However, we don't need to specify those flags because SLURM takes the *-N* and *-n* values from the *#SBATCH* directives in the script

Exercise 1

- Use **sinfo** to check the partitions
  - How many different partitions are defined? What are their names?
  - What's the partition with the highest number of nodes?
  - Name 1-2 nodes included in that same partition

Exercise 2

- In this exercise we are going to use some more advanced SLURM commands to explore job performance.
- You will:
  - Create and compile a simple MPI program
  - Submit it to the queues
  - Check job data using sacct

Exercise 2

- You can find the sample code for this exercise here
- Compile the code and generate a binary called *vector_mpi.x*

Exercise 2

- You can find the job script for this exercise here
- Save the script on Dardel and submit the job. Once the job finishes check its output.
- Inspect the job performance data using **sacct**
    - Use: -j --format=jobid,jobname,elapsed,reqmem,maxrss

      Tip: use flag "--unit=M" to see memory units in MB

Exercise 2

- Use the **seff** command for quick job efficiency overview.

```
Job ID: 211499
Cluster: dardel
User/Group: xaguilar/xaguilar
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 8
CPU Utilized: 00:00:01
CPU Efficiency: 0.37% of 00:04:32 core-walltime
Job Wall-clock time: 00:00:34
Memory Utilized: 549.25 MB (estimated maximum)
Memory Efficiency: 7.73% of 6.94 GB (888.00 MB/core)
```