

From a shell to an HPC result

Introduction to supercomputing

SURF course materials

What you will be able to do

1. Find your way around the system and select software.
2. Submit a job with resources that match the program.
3. Check the answer before interpreting its performance.
4. Place files deliberately and express workflow dependencies.

One path through the exercises

Shell → modules → Slurm → results

Results → parallelism → I/O → workflows

Keep the first job small enough to understand. The backup slides hold full scripts and details; the module decks provide the extended reference.

Start with where you are

Identify your login node, home directory and course directory.

- The login node is for preparation and light interactive work.
- A batch job runs on resources selected by the scheduler.
- Home, shared scratch and job-local temporary storage serve different needs.

Explain which machine executes each command.

Select software before submitting work

- Modules select compilers, libraries and applications.
- Record the modules that the job requires inside its script.
- On the SRC teaching cluster, stack aliases select the EESSI environment.
- Check the site's instructions; an alias does not reproduce its hardware.

First result: run Python through Slurm

From course-module-fundamentals-of-hpc/hands-on:

```
SBATCH python.sh  
squeue -u "$USER"  
cat slurm-JOBID.out
```

Replace JOBID with the number returned by sbatch. Expect the compute hostname and sum: 499999500000. Find the actual runnable script in backup.

Resources describe how the program runs

Tasks Separate processes, commonly MPI ranks.

CPUs per task Cores available to each process, including OpenMP threads.

Nodes Machines across which tasks are placed.

One sequential process does not use extra CPUs simply because they were requested.

Threads: check pi, then compare runtimes

From the same hands-on directory:

```
bash compilepi  
sbatch --cpus-per-task=2 pi.sh
```

The script runs with one and two threads inside the allocation. Both answers should be approximately 3.141592654. Discuss correctness, elapsed time and overhead separately.

MPI: split data and combine the answer

- Use the hybrid-jobs AXPY and dot-product examples.
- Try a size not divisible by the number of ranks.
- AXPY prints $y[0] = 202$; the global dot product must match the serial result.
- Explain which process owns each element and where results are combined.

Use the SRC notes for small teaching allocations.

Files belong where their lifetime requires

Home Keep scripts and small persistent material.

Shared scratch Share active data across compute nodes, subject to site policy.

Job temporary directory Use `$TMPDIR` inside the compute job.

The filesystem exercise writes bounded data and reports actual bytes. Inspect free space before increasing the input size.

Many tasks need an explicit dependency

For the CSV exercise, compute each dataset's mean, then aggregate.

- Aggregation waits for every required producer to succeed.
- It consumes exactly the selected input set.
- A failed producer must make the workflow fail.

Compare the QCG teaching variants before using the workflow alternatives.

Compare engines through observable results

QCG-PilotJob, HyperQueue, Snakemake and FireWorks express dependencies differently.

- Use the same datasets and check the same CSV means.
- Repeat with fewer inputs to expose retained output.
- Inspect task states and the command's exit status.
- Compare performance only with the worker count and allocation stated.

Explain your result to a neighbour

1. Which machine ran the program, with which software?
2. Which tasks and threads did it use?
3. What expected value checked the answer?
4. Which files and dependencies did it need?

Backup: runnable scripts and reference

Python job: included from the runnable file

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=00:02:00
#SBATCH --partition=rome
set -euo pipefail
module load 2025
module load Python/3.13.1-GCCcore-14.2.0
srun python3 -c 'import platform; print("Python on", platform.node());
↳ print("sum:", sum(range(1000000)))'
```

Compile pi: included from the runnable file

```
#!/bin/bash
set -euo pipefail
module load 2025
module load GCC/14.2.0
gcc -O3 -fopenmp -Wall -Wextra pi.c -o pi
```

OpenMP job: included from the runnable file

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=32
#SBATCH --time=00:10:00
#SBATCH --partition=rome
set -euo pipefail
module load 2025
module load GCC/14.2.0
for ncores in 1 2 4 8 16 32; do
    if (( ncores <= SLURM_CPUS_PER_TASK ))
    then
        export OMP_NUM_THREADS=$ncores
        srun --cpu-bind=cores ./pi
    fi
done
```

The pi calculation

```
#include <omp.h>
#include <stdio.h>

int main(void) {
    const long steps = 1000000000;
    const double width = 1.0 / steps;
    double sum = 0.0;
    double start = omp_get_wtime();
#pragma omp parallel for reduction(+:sum)
    for (long i = 0; i < steps; ++i) {
        double x = (i + 0.5) * width;
        sum += 4.0 / (1.0 + x * x);
    }
    printf("pi = %.9f; threads = %d; elapsed = %.6f s\n",
        width * sum, omp_get_max_threads(), omp_get_wtime() - start);
    return 0;
}
```

Independent numerical checks

- AXPY: $y_0 = 202$ for the exercise's initial values.
- Dot product: $25 \sin(N) \sin(N - 1) / \sin(1)$ for the given initialization.
- Matrix multiplication: all entries are $2N$, so the checksum is $2N^3$.
- Binary filesystem output: $8N$ bytes for N doubles on the teaching platform.

Check uneven MPI partitions as well as sizes divisible by the rank count.

SRC resource examples

Exercise	Processes	CPUs per process
Sequential AXPY, dot, matrix	1	1
MPI AXPY	2	1
Hybrid dot (three nodes)	6	1
OpenMP pi	1	2

These are correctness allocations. Read the module's SRC notes for complete submission overrides and ensure the nodes have enough memory.

Workflow exercise versions are teaching steps

The QCG versions expose races, ordering and dependency choices. A finishing launcher alone does not establish a correct result.

- Explain the dependency graph in each version.
- Inspect the final version's aggregate against the selected inputs.
- Run alternatives in separate directories to preserve their outputs.

Extended module references

- Fundamentals: site architecture, accounting and Slurm directives.
- Hybrid jobs: serial, MPI and OpenMP source and submission scripts.
- Filesystems: navigation, permissions and I/O exercises.
- Performance analysis: profiling tools and interpretation.
- QCG-PilotJob and workflow alternatives: orchestration exercises.

The concise teaching path selects a first lesson; the extended decks remain available for site-specific questions and deeper classes.