Simple batch script examples

Objectives

- See and try out different types of simple batch script examples.
  - Serial
  - MPI
  - GPU
- Try using constraints: how to allocate specific CPUs.
- Try using constraints: how to allocate specific GPUs.

For consistency, I have given all the example batch scripts the suffix `.sh` even though it is not required. Another commonly used suffix is `.batch`, but any or none will work.

You need to compile any programs mentioned in a batch script in order to run the examples, except for `compile-run.sh` and the `CUDA` examples, which includes compilation.

Important

- The course project has the following project ID: hpc2n2025-014
- In order to use it in a batch job, add this to the batch script: `#SBATCH -A hpc2n2025-014`
- We have a storage project linked to the compute project: **kebnekaise-intro**.
  - You find it in `/proj/nobackup/kebnekaise-intro`.
  - Remember to create your own directory under it.

Hint

Try to change the C programs, add different programs, and in general play around with the examples!

Note

1. For these test examples I would suggest using the `foss` compiler toolchain, version 2022b, unless otherwise specified, as it is available on both the Intel and AMD nodes. If you decide to use a different one, you will have to make changes to some of the batch scripts.
2. To submit a job script, do `sbatch JOBSCRIPT`

3. In most of the examples, I name the executable when I compile. The flag `-o` tells the compiler you want to name the executable. If you don't include that and a name, you will get an executable named `a.out`. Of course, you do not have to name the executable as I do. It is just an example. In general, I have named all the executables the same as the program (without the suffix). If you change the name, remember to make the change in the submit script as well.

Serial batch job

To compile a serial program, like `hello.c` with gcc do:

```
gcc hello.c -o hello
```

Sample batch script (hello.sh)

```
#!/bin/bash
# Project id - change to your own after the course!
#SBATCH -A hpc2n2025-014
# Asking for 1 core
#SBATCH -n 1
# Asking for a walltime of 1 min
#SBATCH --time=00:01:00

# Purge modules before loading new ones in a script.
ml purge  > /dev/null 2>&1
ml foss/2022b

./hello
```

Exercise: serial job

Submit the job with `sbatch`. Check on it with `squeue --me`. Take a look at the output (`slurm-JOBID.out`) with `nano` or your favourite editor.

MPI batch job

To compile an MPI program, like `mpi_hello.c` (and create an executable named `mpi_hello`) with gcc, do:

```
mpicc mpi_hello.c -o mpi_hello
```

Sample batch script (mpi_hello.sh)

```
#!/bin/bash
# Remember to change this to your own Project ID after the course!
#SBATCH -A hpc2n2025-014
# Number of tasks - default is 1 core per task
#SBATCH -n 14
#SBATCH --time=00:05:00

# It is always a good idea to do ml purge before loading other modules
ml purge > /dev/null 2>&1

ml add foss/2022b

# Use srun since this is an MPI program
srun ./mpi_hello
```

Exercise: MPI job

Submit the job with `sbatch`. Check on it with `squeue --me`. Take a look at the output (`slurm-JOBID.out`) with `nano` or your favourite editor. Try running it more than once to see that the order of the tasks are random.

OpenMP batch job

To compile an OpenMP program, like `omp_hello.c` (and create an executable named `omp_hello`) with gcc, do:

```
gcc -fopenmp omp_hello.c -o omp_hello
```

Sample batch script (omp_hello.sh)

```
#!/bin/bash
#SBATCH -A hpc2n2025-014
# Number of cores per task
#SBATCH -c 28
#SBATCH --time=00:05:00

# It is always a good idea to do ml purge before loading other modules
ml purge > /dev/null 2>&1

ml add foss/2022b

# 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

./omp_hello
```

Exercise: OpenMP job

Set `OMP_NUM_THREADS` to some value between 1 and 28 (`export OMP_NUM_THREADS=value`). Submit the job with `sbatch`. Take a look at the output (`slurm-JOBID.out`) with nano or your favourite editor. Change the value of `OMP_NUM_THREADS`). Submit it again and check on the output to see the change.

NOTE

The reason we are setting `OMP_NUM_THREADS` to at most 28 is that we in the batch script said we were asking for that number of cores. 28 is the number of cores on the Intel Skylake nodes, which has the fewest number of cores. You can see the list of nodes here: https://docs.hpc2n.umu.se/documentation/batchsystem/resources/#nodes.

Multiple serial jobs from same submit file

This submit file shows one way of running several programs from inside the same submit file.

To run this example, you need to compile the following serial C programs:

```
hello.c
Greeting.c
Adding2.c
Mult2.c
```

When the C programs have been compiled, submit the `multiple-serial.sh` program:

▸ multiple-serial.sh

Exercise: multiple serial jobs

Compile the above mentioned programs. Submit the batch script with

```
sbatch multiple-serial.sh
```

If you run it several times you will notice that the order is random.

Job arrays

Job arrays offer a mechanism for submitting and managing collections of similar jobs. All jobs must have the same initial options (e.g. size, time limit, etc.), however it is possible to change some of these options after the job has begun execution using the scontrol command specifying the JobID of the array or individual ArrayJobID.

More information here on the official Slurm documentation pages.

To try an example, we have included a small Python script `hello-world-array.py` and a batch script `hello-world-array.sh`. Both can also be found in the `exercises/simple` directory you have cloned.

hello-world-array.py

```
# 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])
```

hello-world-array.sh

```
#!/bin/bash
# This is a very simple example of how to run a Python script with a job array
#SBATCH -A hpc2n2025-014 # Change to your own after the course!
#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
#SBATCH -o hello-world-%j-%a.out

# Load any modules you need, here for Python 3.11.3
ml GCC/12.3.0 Python/3.11.3

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

Exercise: job arrays

Submit the batch script. Look at the output files. Change the number of tasks in the array. Rerun. See the change.

Multiple parallel jobs sequentially

To run this example, you need to compile the following parallel C programs:

```
mpi_hello.c
mpi_greeting.c
mpi_hi.c
```

When the MPI C programs have been compiled, submit the `multiple-parallel-sequential.sh` program:

```
#!/bin/bash
#SBATCH -A hpc2n2025-014
# Since the files are run sequentially I only need enough cores for the largest of them t
#SBATCH -c 28
# Remember to ask for enough time for all jobs to complete
#SBATCH --time=00:10:00

module purge > /dev/null 2>&1
ml foss/2022b

# Here 14 tasks with 2 cores per task. Output to file - not needed if your job creates ou
# In this example I also copy the output somewhere else and then run another executable.

srun -n 14 -c 2 ./mpi_hello > myoutput1 2>&1
cp myoutput1 mydatadir
srun -n 14 -c 2 ./mpi_greeting > myoutput2 2>&1
cp myoutput2 mydatadir
srun -n 14 -c 2 ./mpi_hi > myoutput3 2>&1
cp myoutput3 mydatadir
```

```
sbatch multiple-parallel-sequential.sh
```

Exercise: multiple parallel jobs sequentially

Submit the job:

```
sbatch multiple-parallel-sequential.sh
```

See that output data are thrown to files and copied to the directory `mydatadir`.

Multiple parallel jobs simultaneously

To run this example, you need to compile the following parallel C programs:

```
mpi_hello.c
mpi_greeting.c
mpi_hi.c
```

As before, we recommend using the `foss/2022b` module for this. If you use a different one you need to change it in the `multiple-parallel-simultaneous.sh` batch script.

When the MPI C programs have been compiled, submit the `multiple-parallel-simultaneous.sh` program:

```
#!/bin/bash
#SBATCH -A hpc2n2025-014
# 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 purge > /dev/null 2>&1
ml foss/2022b

srun -n 14 --exclusive ./mpi_hello &
srun -n 14 --exclusive ./mpi_greeting &
srun -n 14 --exclusive ./mpi_hi &
wait
```

Just like for the multiple serial jobs simultaneously example, you need to add `wait` to make sure the batch job will not finish when the first of the jobs in it finishes.

Exercise: multiple parallel jobs simultaneously

When you have compiled the needed programs, as mentioned above, submit with

```
sbatch multiple-parallel-simultaneous.sh
```

Compiling and running in the batch job

Sometimes you have a program that takes a long time to compile, or that you need to recompile before each run. It could also be that you are optimizing for the architecture in your compilation and that you don't know which architecture your job will end up running on (for instance, Kebnekaise has Intel and AMD CPUs, as well as Nvidia and AMD GPUs).

To see a simple example of compiling and running from the batch job, look at the batch script `compile-run.sh`.

In this case it compiles and runs the `mpi_hello.c` program.

compile-run.sh

```
#!/bin/bash
# CHANGE THE PROJECT ID TO YOUR OWN PROJECT ID AFTER THE COURSE!
#SBATCH -A hpc2n2025-014
#Name the job, for easier finding in the list
#SBATCH -J compiler-run
#SBATCH -t 00:10:00
#SBATCH -n 12

ml purge > /dev/null 2>&1

ml foss/2022b

mpicc mpi_hello.c -o mpi_hello
mpirun ./mpi_hello
```

Exercise: compile and run in a batch job

This batch script can be submitted directly, without compiling anything first, as that happens in the batch script. Try submitting it with `sbatch` and see what happens. Which files are created? You could try changing the program it compiles and runs to a different one. Remember to change the compiler if you are not using an MPI program.

Getting errors and outputs in separate files

As a default, Slurm throws both errors and other output to the same file, named `slurm-JOBID.out`. If you want the errors and other output to separate files, you can do as in the example `separate-err-out.sh`:

```
#!/bin/bash
# Remember to change this to your own Project ID after the course!
#SBATCH -A hpc2n2025-014
#SBATCH -n 8
#SBATCH --time=00:05:00

# Putting the output in a separate output file and the errors in an
# error file instead of putting it all in slurm-JOBID.out
# Note the environment variable %J, which contains the job ID. It is handy to
# avoid naming the files the same for different runs, and thus overwriting them.
#SBATCH --error=job.%J.err
#SBATCH --output=job.%J.out

ml purge > /dev/null 2>&1
ml foss/2022b

mpirun ./mpi_hello
```

You need the `mpi_hello.c` file compiled (and the executable named `mpi_hello`) for this to run without changes. Of course, you can also just add your own programs.

Exercise: errors and outputs in separate files

Compile the file `mpi_hello.s` after loading the module `foss/2022b`. Submit the job script with `sbatch`. See that separate output and error files are created.

CUDA/GPU programs

To run programs/software that uses GPUs, you need to allocated GPUs in the job script. They will not be allocated by your program.

To compile a cuda program, like `hello-world.cu` you need to load a toolchain containing CUDA compilers/load CUDA compilers.

To run a piece of software that uses GPUs, you need to load a module version which is GPU aware. In many cases there are several versions of a module, only some of which are for running on GPUs.

Important

Remember to check the modules, versions, and prerequisites! Also make sure you check for the correct node type. Some of the GPUs are on Intel nodes (check modules on `kebnekaise.hpc2n.umu.se`), some on AMD nodes (check modules on `kebnekaise-amd.hpc2n.umu.se`).

V100 - Intel Skylake

This example runs a small CUDA code.

We recommend `fosscuda/2020b` (contains `GCC`, `OpenMPI`, `OpenBLAS`/`LAPACK`, `FFTW`, `ScaLAPACK`, and `CUDA`) or `intelcuda/2019a` (contains `icc`, `ifort`, `IntelMPI`, `IntelMKL`, and `CUDA`)

Sample batch script `gpu-skylake.sh`

```
#!/bin/bash
# This job script is for running on 1 V100 GPU.
# Remember to change this to your own project ID after the course!
#SBATCH -A hpc2n2025-014
#SBATCH --time=00:05:00
#SBATCH --gpus=1
#SBATCH -C v100

ml purge > /dev/null 2>&1
ml fosscuda/2020b

nvcc hello-world.cu -o hello
./hello
```

The batch script gpu.sh compiles and runs a small cuda program called `hello-world.cu`.

Exercise: V100 GPU job

To submit it, just do:

```
sbatch gpu.sh
```

Use `squeue --me` or `scontrol show job JOBID` to see that the job runs in the correct partition/node types.

Important

Instead of

```
#SBATCH --gpus=1
#SBATCH -C v100
```

you could have used

`#SBATCH --gpus=v100:1`

or

```
#SBATCH --gpus-per-node=1
#SBATCH -C v100
```

A100 - AMD Zen3

Remember, in order to find the correct modules, as well as compile a program if you need that, you must login to one of the AMD login nodes with either SSH (`kebnekaise-amd.hpc2n.umu.se`) or ThinLinc (`kebnekaise-amd-tl.kebnekaise.hpc2n.umu.se`). The job can be submitted from the regular login node, though.

Exercise: login to the AMD login node and find a suitable module

If you are logged in to the regular Kebnekaise login node, then you can easiest login to the AMD login node by typing this in a terminal window:

```
ssh kebnekaise-amd.hpc2n.umu.se
```

After that, you check for a suitable CUDA toolchain: `ml spider CUDA`.

You can then load it (here `CUDA/11.7.0`) and use `nvcc` to compile the program `hello-world.cu`:

```
ml CUDA/11.7.0

nvcc hello-world.cu -o hello
```

Now logout from the AMD login node again.

The batch script `gpu-a100.sh` compiles and runs a small cuda program called `hello-world.cu`.

Sample A100 GPU job script: gpu-a100.sh

```
#!/bin/bash
# Remember to change this to your own project ID after the course!
#SBATCH -A hpc2n2025-014
#SBATCH --time=00:05:00
#SBATCH --gpus=a100:1

ml purge > /dev/null 2>&1
ml CUDA/11.7.0

nvcc hello-world.cu -o hello
./hello
```

Exercise: A100 GPU batch jobs

The above script is found in the same directory as the other exercises (`intro-course/exercises/simple`). You can submit it directly in that directory:

```
sbatch gpu-a100.sh
```

Like for the A100, you are encouraged to use `squeue --me` and/or `scontrol show job JOBID` to see that the job gets the correct partition/node type allocated.

A40 - Intel broadwell

Kebnekaise also has a few of the A40 GPUs. These are placed on Intel broadwell nodes.

In order to run on these, you add this to your batch script:

```
#SBATCH --gpus=number
#SBATCH -C a40
```

where `number` is 1 or 2 (the number of GPU cards).

You can find the available modules on the regular login node, `kebnekaise.hpc2n.umu.se`.

L40s - AMD Zen4

Since these GPUs are located on AMD Zen4 nodes, you need to login to `kebnekaise-amd.hpc2n.umu.se` to check available modules.

Then, to ask for these nodes in your batch script, you add:

```
#SBATCH --gpus=number
#SBATCH -C l40s
```

where `number` is 1 or 2 (the number of GPU cards).

H100 - AMD Zen4

The H100 GPUs are located on AMD Zen4 nodes. You can find the available modules by logging in to `kebnekaise-amd.hpc2n.umu.se`.

You ask for these GPUs in your batch script by adding:

```
#SBATCH --gpus=number
#SBATCH -C h100
```

where `number` is 1, 2, 3, or 4 (the number of GPU cards you want to allocate).

A6000 - AMD Zen4

The A6000 GPUs are placed on AMD Zen4 nodes. That means you can find the available modules by logging in to `kebnekaise-amd.hpc2n.umu.se`.

To run on these GPUs, add this to your batch script:

```
#SBATCH --gpus=number
#SBATCH -C a6000
```

where `number` is 1 or 2 (the number of GPU cards you want to allocated).

MI100 - AMD Zen3

The MI100 GPUs are located on AMD Zen3 nodes. You can find the available modules by logging in to `kebnekaise-amd.hpc2n.umu.se`.

To allocate MI100 GPUs, add this to your batch script:

```
#SBATCH --gpus=number
#SBATCH -C mi100
```

where `number` is 1 or 2 (the number of GPU cards).

GPU features

Sample batch script for allocating any AMD GPU

```
#!/bin/bash
# Remember to change this to your own project ID after the course!
#SBATCH -A hpc2n2025-014
#SBATCH --time=00:05:00
#SBATCH --gpus=1
#SBATCH -C amd_gpu

ml purge > /dev/null 2>&1
ml CUDA/11.7.0

./myGPUcode
```

Sample batch script for allocating any Nvidia GPU

```
#!/bin/bash
# Remember to change this to your own project ID after the course!
#SBATCH -A hpc2n2025-014
#SBATCH --time=00:05:00
#SBATCH --gpus=1
#SBATCH -C nvidia_gpu

ml purge > /dev/null 2>&1
ml CUDA/11.7.0

./myGPUcode
```

Sample batch script for allocating any Nvidia GPU on an Intel node

```
#!/bin/bash
# Remember to change this to your own project ID after the course!
#SBATCH -A hpc2n2025-014
#SBATCH --time=00:05:00
#SBATCH --gpus=1
#SBATCH -C 'nvidia_gpu&intel_cpu'

ml purge > /dev/null 2>&1
ml CUDA/11.7.0

./myGPUcode
```

Sample batch script for allocating any GPU with AI features and on a Zen node

```
#!/bin/bash
# Remember to change this to your own project ID after the course!
#SBATCH -A hpc2n2025-014
#SBATCH --time=00:05:00
#SBATCH --gpus=1
#SBATCH -C ''zen3|zen4'&GPU_AI'

ml purge > /dev/null 2>&1
ml CUDA/11.7.0

./myGPUcode
```

Exercise: GPU features

In order to run these examples, you can change `./myGPUcode` to

```
nvcc hello-world.cu -o hello
./hello
```

or any other GPU program of your choice.

The `gpu-features.sh` example script in the `exercises/simple` directory is prepared for the "any GPU with AI features and on a Zen node". You can either run it as is, or make changes to it and try any of the other combinations here (or try new combinations yourself).

Check with `squeue --me` which partition/node type the job ends up in, and that it fits. More information can be found with `scontrol show job JOBID`.

Starting JupyterLab

On Kebnekaise, it is possible to run JupyterLab. This is done through a batch job, and is described in detail on our "Jupyter on Kebnekaise" documentation.

Exercise

Try starting Jupyter as shown on the link above.

Keypoints

Keypoints

- How to run serial, MPI, OpenMP, and GPU jobs
- How to use GPU features
- How to run several jobs from inside one batch job