bwUniCluster

Lars Pastewka

Table of contents

Getting access

Accessing bwUniCluster is a multi-step procedure that is described in detail here: https://wiki.bwhpc.de/e/Registration/bwUniCluster. Please follow these steps as described. When being asked why you need access, please mention the "HPC with C++ class at the University of Freiburg".

NOTE: you must be connected to the eduroam to access the cluster. You can also use a VPN. The official instruction to set up the VPN can be found here: https://wiki.uni-freiburg.de/rz/doku.php?id=vpn.

Logging in and transferring data

You can then login to bwUniCluster with the `ssh` (*s*ecure *sh*ell) command:

```
ssh -Y <username>@uc3.scc.kit.edu
```

The option `-Y` tells ssh to forward X11 connections. This allows you to display plots while remotely logged in to bwUniCluster. This will not work if you are logging in from a Windows machine, unless you have set up a dedicated X11 server on your windows computer.

You can copy files to bwUniCluster with the `scp` (*s*ecure *cp*y) command:

```
scp my_interesting_file <username>@uc3.scc.kit.edu:
```

Don't forget the colon `:` after the machine name when running the `scp` command. Note that you can use the same command to get the data back from bwUniCluster. For example, to copy the file `result.npy` from your home directory on bwUniCluster to the current directory simply execute (on your local machine):

```
scp <username>@uc3.scc.kit.edu:result.npy .
```

The dot `.` refers to your current directory. You can also specify full paths on either remote or local machines when executing `scp`.

Setting up the software environment

After logging in to bwUniCluster at `uc3.scc.kit.edu`, you will need to set up your local environment to be able to compile and run parallel applications. The `module` command loads a specific environment: * You can see all possible modules with `module avail`. Note that the list changes when you load a module as there are dependencies. * You search for a specific module with `module spider`. * You can list the presently load modules with `module list`. * You can remove (unload) all presently load modules with `module purge`.

Please load the following modules (just execute these commands at the command line):

```
module load compiler/gnu mpi/openmpi
module list
```

If you need CUDA-aware MPI, please load:

```
module load mpi/openmpi/5.0-nvidia-25.1-nompi
```

Compiling your code

You will first need to checkout your code on bwUniCluster. To compile your code on the command line, create a directory for building the code. This could be a directory named `build` in your source directory, e.g.:

```
mkdir build
cd build
```

Run `CMake` to configure your build:

```
cmake -DCMAKE_BUILD_TYPE=Release ..
```

Compile the code:

```
cmake --build . -j $(nproc)
```

Running simulations

bwUniCluster has extensive documentation that can be found here: https://wiki.bwhpc.de/e/BwUniCluster3.0. Simulations are typically run as batch jobs. They have to be submitted through a batch or queueing system that takes care of assigning the actual hardware (compute node) to your job. A description of the queueing system can be found here. Please make sure you understand the concept of a *partition* (also called a *queue* on bwUniCluster).

IMPORTANT: never *ever* run a simulation on the front/login node (i.e. without submitting a job using the `sbatch` command, see below). Your access to bwUniCluster could be revoked if you do this.

Job scripts

To run you job, you need to write a job script. The job script is executed by the `bash` command which is the shell that you are using on Linux systems. (See the first tutorial here for an introduction to the shell.) The job script specifies how many CPUs you require for your job, how to set up the software environment and how to execute your job. An example job script (you can use this one almost as is) looks like this:

```
#!/bin/bash -x
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=20
#SBATCH --time=00:40:00
#SBATCH -J lbm
#SBATCH --mem=6gb
#SBATCH --export=ALL
#SBATCH --partition=cpu

module load compiler/gnu mpi/openmpi

echo "Running on ${SLURM_JOB_NUM_NODES} nodes with ${SLURM_JOB_CPUS_PER_NODE} cores each."
echo "Each node has ${SLURM_MEM_PER_NODE} of memory allocated to this job."
time mpirun ./lbm
```

Lines starting with a `#` are ignored by the shell. Lines starting with `#SBATCH` are ignored by the shell but are treated like command-line options to the `sbatch` command that you need to submit your job. You can see all options but running `man sbatch`.

The `--nodes` option specifies how many nodes you want to use. `--ntasks-per-node` specifies how many processors per node you need. The `--time` option specifies how long your job is allowed to run. You job is terminated if it exceeds this time! The `-J` option is simply the name of your job. It determines the names of output files generated by the batch system. Check the documentation linked above to understand the other options.

The `time` command in front of `mpirun` measures the execution time of the code and prints it to screen. Note that you do not tell `mpirun` how many cores to use. The batch system passes this information on automatically.

Submitting jobs

Assume the filename of the above script is `run.job`, you can submit this script with

```
sbatch run.job
```

Everything that is included via `#SBATCH` in the script above can also be specified on the command line. For example, to change the number of cores to 160 you can issue the command:

```
sbatch --nodes=4 --ntasks-per-node=40 run.job
```

Your job may need to wait for resources and will not run immediately. You can inquire the status of your job with

```
squeue
```

Once it ran, you will find a file that start with `job_` and has the extension `.out`. This file contains the output of the simulation, i.e. the output that is normally written to screen when you run from the command line. Note that `bash -x` (the first line in the job submission script above) instructs bash to print every command that is executed onto the screen. This makes debugging (of the job script, not your simulation code) easier.

`squeue` shows the `JOBID` of you job. You cancel a job by executing

```
scancel <JOBID>
```

More information on a certain job can be obtained by

```
scontrol show job <JOBID>
```

Interactive jobs

For testing purposes, in particular on GPUs, it is useful to run jobs interactively. To get a node with A100 GPUs, run

```
salloc --partition=dev_gpu_a100_il --gpus-per-node=1 --time=30
```

Please see the full list of available partitions for which partitions are available. The bwUniCluster documentation calls partitions also *queues*. The partitions starting with `dev_` are for development purposes and limited to 30 minutes wallclock time. You will be forcibly logged out of the node after that period.

To get the NVIDIA compilers, run

```
module load toolkit/nvidia-hpc-sdk/25.1
```