HPC introduction Spring ’22

The exercises follow the tutorial which you can download from https://hpc.uantwerpen.be/, User Support, Documentation and tutorials. Look for the line “HPC tutorials for Windows / Mac / Linux”.

Exercise block 1: Building an environment

Set up some example programs (see also Tutorial §3.1):

- Log on to Vaughan (login1-vaughan.hpc.uantwerpen.be or login2-vaughan.uantwerpen.be)
- Go to your scratch directory
  `cd $VSC_SCRATCH`
- Copy the examples for the tutorial to that directory: type
  `cp -r /apps/antwerpen/examples .`

Exercises:

1. Follow the text from the tutorial in §4.1 “Modules” and try out the module command. Note that `module spider` and `module keyword` are still missing from this text as those are not available on all VSC clusters.
2. Load the cluster module calcua/2020a and check the difference between `module av` and `module spider`.
   a. Try to find modules for the HDF5 library. Did you notice the difference between `module av` and `module spider`?
3. Which module offers support for VNC? Which command does that module offer?
4. Which module(s) offer the CMake tool?
5. What is the difference between the HDF5/1.10.6-intel-2020a-MPI and HDF5/1.10.6-intel-2020a -noMPI (well, you can guess it from the name but how can you find more information?)

Exercise block 2: Job basics

1. Tutorial §4.3: `examples/Slurm/04_Running-batch-jobs` contains a short bash script (fibo.slurm) that will print Fibonacci numbers. Extend the script to request 1 core on 1 node with 1 Gb of (virtual) memory and with a wall time of 1 minute in the job script and run the job. Note which files are generated.
2. Extend the script to give the job a name of your choice. Run and note which files are generated.
3. While the job is running, try some of the squeue and sstat commands. To do this, change your job script first to extend the walltime to 10 minutes and put a sleep 300 at the end because otherwise the job will finish so quickly that you don’t get the chance to see anything.

Exercise block 3: Starting various job types

Think for yourself what material might apply best to the research that you will be doing, and select some of the exercises below:

Starting an interactive job

With the help of the example scripts in `examples/Slurm/05_Running-interactive-jobs`:

1. Start an interactive session with 1 core on 1 node and run the primes.py example (end of §5.2 in the tutorial)
2. If you have an X-server on your PC:
    a. How would you run message.py on one of the login nodes?
    b. Run message.py in an interactive job (so on one of the compute nodes)

Starting parallel applications

The module vsc-tutorial (2019b and 2020a Intel toolchains) contains some compiled "Hello, world!"-style programs to demonstrate starting OpenMP, MPI or hybrid MPI/OpenMP-programs: omp_hello, mpi_hello and mpi_omp_hello. These commands are documented through man pages. These commands require very little memory and execution time, but are excellent to see where your regular OpenMP, MPI or hybrid application would run.

1. Write a job script that runs the omp_hello command on 8 cores.
2. Write a job script that runs the mpi_hello command on 32 cores.
3. Write a job script that starts the mpi_omp_hello hybrid application using 8 MPI processes with 16 cores each.

Workflows

1. Try the example from the slides for yourself.
2. Extend the example:
    a. A first job writes the number 10 to the file "sim1.txt"
    b. Job two executes after the first. It reads the number from "sim1.txt", multiplies it by 2 and writes it to "sim2.txt"
    c. To result of the second simulation, two different perturbations are applied: add 1 and 2 to the number in sim2.txt, and the third "simulation" that starts from that perturbed value multiplies the result by 2 and writes it in a file with the name sim3_pert_<…>.txt with <…> the perturbation.

    Use dependent jobs to make sure that the second "simulation" runs after the first one and that both "simulations" for the perturbations can run concurrently and after the second "simulation".

Multi-job submission

1. Job array: Have a look at the example in `examples/Slurm/12_Multi-job-submission/1_job_array` and run `job_array.slurm` as a Slurm array job
2. Parameter sweep: Have a look at the example in `examples/Slurm/12_Multi-job-submission/2_par_sweep` and run this job using atools.