HPC introduction Spring'19

Exercise block 1: Building an environment

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

- Log on to the cluster
- Go to your data directory
  `cd $VSC_DATA`
- Copy the examples for the tutorial to that directory: type
  `cp -r /apps/antwerpen/tutorials/Intro-HPC/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 leibniz/2018b 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`?
      module av shows fewer modules than module spider as the latter also shows modules that are not enabled by the Leibniz/2017a module. The output format is also very different as module spider shows some information about what's in the module.
3. Which module offers support for VNC? Which command does that module offer?
   Both module spider vnc or module keyword vnc wil show the vsc-vnc module on the system. The information shown by module spider will however also show some help that tells the module provides the vnc-xfce command. module spider vsc-vnc/0.1 shows that same information, but would be needed if there were more than one vsc-vnc module.
4. Which module(s) offer the CMake tool?
   module spider cmake does not show all modules that provide CMake as we have some modules that provide CMake bundled with other tools, and those modules are hence not called CMake/…. However, module keyword cmake will show all modules as we have made sure that that information is in the part of the module files searched by module keyword. In particular, recent versions of CMake are provided by the buildtools package.
   "module keyword" is often the more powerful way to search for software.
5. What is the difference between the HDF5/1.8.20-intel-2018b-MPI and HDF5/1.8.20-intel-2018b-noMPI (well, you can guess it from the name but how can you find more information?)
   Use module spider HDF5/1.8.20-intel-2018b-MPI or module spider HDF5/1.8.20-intel-2018b-noMPI (or module help with the name of the module, but then you'll need to load leibniz/2018b or leibniz/supported first if it is not yet loaded already) to see all the help information about these modules.

Exercise block 2: Job basics

1. Tutorial §4.3: examples/Running-batch-jobs contains a short bash script (fibo.pbs) that will print Fibonacci numbers. Extend the script to request 1 core on 1 node with 2Gb of (virtual) memory and with a wall time of 1 minute in the job script and run the job. Note which files are generated

   NUMA-syntax:

   ```
   #PBS -L tasks=1:lprocs=1:swap=2gb
   #PBS -l walltime=1:00
   ```

   Old syntax:

   ```
   #PBS -l nodes=1:ppn=1
   #PBS -l vmem=2gb
   #PBS -l walltime=1:00
   ```

2. Extend the script to give the job a name of your choice. Run and note which files are generated.

   Add

   ```
   #PBS -N ThisIsMyName
   ```

3. While the job is running, try some of the q-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.

   Example job script:

   ```
   #!/bin/bash -l
   #PBS -L tasks=1:lprocs=1:swap=2gb
   #PBS -l walltime=10:00
   #PBS -N ThisIsMyName
   cd $PBS_O_WORKDIR
   ./fibo.pl
   sleep 300
   ```

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/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)

   Use

   ```
   qsub -I -L tasks=1:lprocs=1:swap=4gb
   ```

   Then go to the example directory ans simply run. You can use the system python, it is not necessary to load a Python mode for this simple example:

   ```
   ./primes/py
   ```

2. If you have an X-server on your PC:
   a. How would you run message.py on one of the login nodes?

      Don't forget the -X option with the ssh command, or select the appropriate option for X tunneling in your GUI-based ssh-client.

b. Run message.py in an interactive job (so on one of the compute nodes)
Log in to a login node as in a, then start the job by adding the -X option to the command line from 1.

Starting parallel applications

The module vsc-tutorial (2018b Intel toolchain) 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 7 cores.

```
#! /bin/bash
#PBS -L tasks=1:lprocs=7:swap=4gb
#PBS -l walltime=5:00
#PBS -N omp-hello-test

cd $PBS_O_WORKDIR

module load $VSC_INSTITUTE_CLUSTER/supported
module load vsc-tutorial/201810-intel-2018b
module load torque-tools

export OMP_NUM_THREADS=$(torque-lprocs 0)
omp_hello
```

2. Write a job script that runs the mpi_hello command on all cores of 2 nodes.

```
#! /bin/bash
#PBS -L tasks=2:lprocs=all:place=node:swap=4gb
#PBS -l walltime=5:00
#PBS -N MPI-hello-test

cd $PBS_O_WORKDIR

module load $VSC_INSTITUTE_CLUSTER/supported
module load vsc-tutorial/201810-intel-2018b
mpirun mpi_hello
```

3. Write a job script that starts the mpi_omp_hello hybrid application using 8 MPI processes with 7 cores each. Use the torque-tools module in your job script.

```
#! /bin/bash
#PBS -L tasks=8:lprocs=7:swap=4gb
#PBS -l walltime=5:00
#PBS -N hybrid-hello-test

cd $PBS_O_WORKDIR

module load $VSC_INSTITUTE_CLUSTER/supported
module load vsc-tutorial/201810-intel-2018b
module load torque-tools

export OMP_NUM_THREADS=$(torque-lprocs 0)
torque-host-per-line >machinefile.$PBS_JOBID
mpirun -machinefile machinefile.$PBS_JOBID mpi_omp_hello
/bin/rm machinefile.$PBS_JOBID
```

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”.

Job script for the first job: workflow_job_first.pbs.sh:

```
#!/bin/bash
#PBS -L tasks=1:lprocs=1:swap=1gb
#PBS -l walltime=2:00

cd "$PBS_O_WORKDIR"

# Run the "simulation"
echo "10" >sim1.txt
sleep 30
```

Job script for the second job: workflow_job_second.pbs.sh:

```
#!/bin/bash
#PBS -L tasks=1:lprocs=1:swap=1gb
#PBS -l walltime=2:00

cd "$PBS_O_WORKDIR"

# Read the input from the previous "simulation"
number=$(cat sim1.txt)
# Run the new "simulation" and write the result
echo $((2*$number)) >sim2.txt
sleep 30
```

Job script for each of the perturbations: workflow_job_perturbation.pbs.sh

```
#!/bin/bash
#PBS -L tasks=1:lprocs=1:swap=1gb
#PBS -l walltime=2:00

cd "$PBS_O_WORKDIR"

# Read the result from a previous simulation
number=$(cat sim2.txt)
# Apply the perturbation
number=$(($number+$perturbation))
# Run the "simulation" and write the result
echo $((2*$number)) >sim3_pert_$perturbation.txt
sleep 30
```

Starting the jobs:

```
first=$(qsub -N job_sim1 workflow_job_first.pbs.sh)
second=$(qsub -N job_sim2 -W depend=afterany:$first workflow_job_second.pbs.sh)
qsub -N job_pert_1 -v perturbation=1 -W depend=afterany:$second workflow_job_perturbation.pbs.sh
```

```
qsub -N job_pert_2 -v perturbation=2 -W depend=afterany:$second workflow_job_perturbation.pbs.sh
```

Multi-job submission

1. Parameter sweep: Have a look at the example in `examples/Multi-job-submission/par_sweep` (and tutorial §12.1) and run this job through the Worker framework
   Just copy and run:
   ```
   module load Leibniz/supported worker
   wsub -data data.csv -batch weather.pbs
   ```
2. Job array: Have a look at the example in `examples/Multi-job-submission/job_array` (and tutorial §12.2) and run this job through the Worker framework
   Launching as a Torque array job:
   ```
   qsub -t 1-100 job_array.pbs
   ```
   Launching as a Worker array job:
   ```
   module load leibniz/supported worker
   wsub -t 1-100 -batch test_set.pbs
   ```
   Note the differences in the resource specifications
   - *Torque array job: Resources for a single job of the array*
   - *Worker: Resources for the overall worker process, so all array jobs together.*
3. Advanced feature of worker: Map-Reduce: Have a look at the tutorial text §12.3 and the code in `examples/Multi-job-submission/map_reduce` and try to run this code through the Worker framework
   Run with:
   ```
   module load $VSC_INSTITUTE_CLUSTER/supported worker
   wsub -prolog pre.sh -batch test_set.pbs.sh -epilog post.sh -t 1-100
   ```