Introduction to the CSF

Practical session 4: Using Slurm to submit a parallel job and to check scalability.

Overview

We are going to use a parallel matrix-multiply application - pmm.exe (a distributed memory app, written in C using the Message Passing Interface (MPI) library):

Run a parallel job using 2 cores.

1. Modify the jobscript to run with different numbers of cores.
2. Calculate the speed-up for the different runs to see whether the application *scales* with more cores.

More information on Slurm on the CSF can be found at
https://ri.itservices.manchester.ac.uk/csf/batch-slurm/

Instructions

1. Connect to the CSF using `ssh` if you don't already have a shell on the login node (see practical 1 for how to do this if you can't remember).

2. (This step won't do much if you did it in exercise 1, but won't harm if repeated)
   Install the necessary files by running the following on the CSF login node:

   ```
   module load training/RCSF
   ```

   (enter your University IT `password` – same as used for CSF login - when asked)

3. Ensure you are in the directory that contains the files for today's training

   ```
   cd ~/training/RCSF/examples
   ```

   Linux is case-sensitive so ensure upper and lowercase letters are correct. The `~` character is shorthand for "your home directory".

   Tip: You can press the [Tab] key while typing folder names to see if they'll auto-complete – this can save you a lot of typing!

   We are now going to run an application named *`pmm.exe`* as a job on the CSF. This is a program that does a matrix multiplication calculation in parallel – it can use multiple CPU cores.

   Matrix multiplication is a common operation found in a lot of codes – e.g., engineering codes doing finite element analysis or chemistry codes doing molecular dynamics.

   As always, we describe the job – the resources it needs and the commands we want it to run – in a "jobscript", which is a small text file.

4. Let's examine the text file `pmm_jobscript`. You can do this using:

   `gedit pmm_jobscript` (this will open the text-editor from exercise 1)
   or
   `cat pmm_jobscript` (this will just print the file to screen)

   This is the *jobscript* and it tells Slurm (the batch system):

   1. The jobscript is written using the BASH script language.
   2. To run the job on the compute nodes (hardware) dedicated to multi-core "parallel" jobs. This partition will use the (new, fast) AMD 168-core nodes.
   3. To use 2 CPU cores.
   4. To allow a maximum of 10 minutes for the job to complete, once it starts (we know this program doesn't take long to do its computation.)

5. We ask Slurm to change the name of the output file from **slurm-*JOBID*.out** to **pmm_2_*JOBID*.out**. This will help you to identify your output files.
6. For today, we have some reserved compute nodes, so we use those.
7. We load a modulefile needed by the program to provide access to the "MPI" library – the software often used to develop parallel applications.
8. The command(s) that we want the job to run. In this case it's a program named `./pmm.exe` (the `./` at the start means the program is in the folder where we submit the job from.)

   As it's a parallel "MPI" program, we start it using the `mpirun` command. Notice that we tell mpirun how many cores the job is allowed to use, using mpirun -n `$SLURM_NTASKS`. This is actually an optional flag – the `mpirun` command will automatically read the `$SLURM_NTASKS` number if you don't use the `-n` flag to give `mpirun` the number.

5. Submit the job to the batch system (i.e., submit it to the "queue")

   ```
   sbatch pmm_jobscript
   ```

   it will return a unique *JOBID* number to you. **Make a note of it**.

6. Check the status of your job in the batch queue by running:

   ```
   squeue
   ```

   to see just *your* jobs. The *ST* column (short for STATUS) should be either `PD` (if your job is pending - i.e., waiting) or `R` (when your job starts running). If you see nothing, your job has finished!

7. When the job has finished, examine the output file. Remember that the job has run on a backend "compute node" and so you don't see output that is normally written to the screen, like when you run commands on the login node.

   Instead, any output that the application would normally print to screen, is captured into a file called **`pmm_2_`*`JOBID`*`.out`** (remember that we renamed the output file via the "`#SBATCH -o`" flag.)

   To list your files and then to display the contents of one of the output files (change the number at the end appropriately):

   ```
   ls -ltr
   ```

   ```
   cat pmm_2_123456.out
   ```

   Notice that the most-recently written files appear at the bottom of the list when you run `ls -ltr` . This makes it easier to see which files were last updated by your jobs.

   The output from the `pmm.exe` includes some timing information – it times itself (a lot of apps do this.)

Q1: How long did the pmm.exe program take when using 2 cores?
IF YOU CANNOT ANSWER THIS QUESTION, PLEASE ASK FOR HELP NOW.

8. You are now going to run the job again but with a different number of cores. Edit the jobscript using:

   ```
   gedit pmm_jobscript
   ```

   and change the number of cores to **4** (then **8**, **16**, **32**, **64**.) Note that the output filename includes the number of cores. You will need to change that name too

   ```
   #SBATCH -o pmm_4_%j.out
   ```

   (You cannot use the $SLURM_NTASKS variable in the -o line unfortunately.)

   Submit the jobscript *each time you change the number of cores* using

   ```
   sbatch pmm_jobscript
   ```

9. You also need to submit a "serial" (**1-core**) job. If you try to do this with the "multicore" partition, you'll see your job is rejected.

   You could submit to the "serial" partition.
   Q: Why would this not be a good idea when running timing tests?

   Instead, use this page to determine which partition will accept "serial" (1-core) jobs on the AMD 168-core compute nodes.
   https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/partitions/
   (**if you are stuck with this, please ask for help!**)

10. Once you've run all of the jobs (1, 2, 4, 8, 16, 32, 64 cores), by looking at the timing information for each job, you can determine whether the job scales. Do this by plugging the numbers into the "Speed-up" formula in the slides (slide 35.)

    Q: Does it scale?

    Summary – You've now submitted parallel jobs to the multicore partition so that they run on the AMD 168-core compute nodes. The job reserved a number of cores and then ran an parallel program using that number of cores.

    If the application scales with the number of cores, then we know we'll get the results sooner by running the job with more cores. If the application doesn't scale (beyond a certain number of cores), then there is no point submitting jobs that ask for a lot of cores – you'll only wait longer in the queue without any gain in performance.