High Performance Computing
Exercise Sheet 2

FS 21
dr. Douglas Potter

`http://www.ics.uzh.ch/`

Teaching Assistants:
Jozef Bucko, jozef.bucko@uzh.ch
Hugues de Laroussilhe, hugues@ics.uzh.ch

Issued: 05.03.2020
Due: 10.03.2020

Exercise 0 [Bitbucket or Gitlab account and hand-in]
From now on, you will use git to hand-in your exercise sessions. First create a Bitbucket (`https://bitbucket.org`) or a Gitlab account (`https://gitlab.uzh.ch/`) using your academic email. During each session, you will have to commit and push new changes to a repository named "`hpc_esc401_solutions`". Make sure that this repository will contain each exercise session in a different folder.

Exercise 1 [Version control with git]
In this exercise session, you will use an example code which can be found here: `https://github.com/hlasco/hpc_esc_401`. It contains a version of a code which is parallelized using `MPI` and one which uses `OpenMP`. Go to your `$HOME` directory on daint and clone the course repository with the following command:

```
git clone https://github.com/hlasco/hpc_esc_401.git
```

If not done already, create your personal git repository, copy in it the folder `$HOME/hpc_esc_401/exercise_session_02` and make the initial commit (slides from lecture 02 may help). Check the status and the log of your new repository. Did you encounter any problems? If yes, how did you solve them?

Exercise 2 [Compilation]
Compiling a simple code can be done by calling the appropriate compiler with the necessary flags. Compilation of bigger codes, consisting of multiple files, is usually done with the help of a Makefile. There is such a makefile included in the directory you just copied. Open it and see if you can make sense of the content. What is `cc`? What does the flag `-O3` do? What happens if we change this to `-O0`?
Supercomputing centers work with modules that can be loaded to achieve different things. Load the multicore partition of daint. Which programming environment are you currenlty using? Switch to `PrgEnv-gnu` and compile the two codes using `make`.

Exercise 3 [Submitting jobs]
`SLURM` is the job queueing system used on daint.

- Run `sinfo -p debug`. What can you see? How can you print only your jobs? Or any other user?
- Use the job script generator (`https://user.cscs.ch/access/running/jobscript_generator/`) to generate a script which will submit a job on the Piz Daint multicore system. Specify the job name, ask for 36 threads for 5 minutes on the debug partition. Use `hostname` as the program. Add a line where you specify the account (uzh8). Move the job script to your `$SCRATCH` directory, submit the job and check if it is running.
- What do you need to add to redirect your output to `output.log` and errors to `errors.log`?
- Generate a job script for both the MPI and OpenMP version of the `cpi` code from exercise 1. Run both on a single node using all cores. (HINT: the job scripts are different). What is the output?

  Commit : Add the job scripts and your output logs to your git repository.
- BONUS: `squeue` gives a printout which can be customised. What command would you run in order to get the output with following fields

```
JOBID  USER  PRIORITY ACCOUNT  NAME NODES ST  REASON   START_TIME  TIME_LEFT PRIORITY
```

Exercise 4 [Serial and MPI/OpenMP versions]
Write a serial version of the code "cpi.c" and edit the Makefile.

Commit : Update your git repository.

BONUS: Can you write a hybrid version of the code, that uses both MPI and OpenMP ?