High Performance Computing
Exercise Sheet 2

HS 25
Dr. Douglas Potter

http://www.astro.uzh.ch/
Teaching Assistants:
Cesare Cozza, cesare.cozza@uzh.ch
Mark Eberlein, mark.eberlein@uzh.ch

Issued: 26.09.2025
Due: 03.10.2025

In this exercise sheet, we will set up a git repository, which we will also be using for the rest of the course. In case you have any codes/files you want to hand in, you should push these into your git repository from now on and provide a link to the respective git folder via Teams App.

Exercise 0 [Bitbucket, GitLab or GitHub account and hand-in]
From now on, you will use git to hand-in your exercise sessions. First create a Bitbucket (https://bitbucket.org), GitLab account (https://gitlab.uzh.ch/) or a GitHub (https://github.com/) account using your academic email (if you already have your own git account, you can also use this). During each session, you will have to commit and push new changes to a repository named "`hpc_esc_401_solutions`". Make sure that this repository will contain each exercise session in a different folder. All the comments about your solutions or textual answers to questions can be put to `REAMDE.md` file inside the folder containing your solutions. Also, make your repository private and invite the teaching assistants (e.g. as collaborators) so we can check your hand-ins.

Exercise 1 [Version control with git]
In this exercise session, you will use an example code which can be found here: https://github.com/Mark-Eb/hpc_esc_401_2025. It contains a version of a code which is parallelized using `MPI` and one which uses `OpenMP`. Go to your `$HOME` directory on Eiger and clone the course repository with the following command:

`git clone https://github.com/Mark-Eb/hpc_esc_401_2025` If not done already, create your personal git repository, copy the folder `$HOME/hpc_esc_401_2025/exercise_session_02` in it 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 would happens if we change this to `-O0`? Hint: `man cc` might be helpful, one can search in manuals by typing `/somestring`.
Supercomputing centers work with modules that can be loaded to achieve different things. One can load a module using `module load <module_name>` (for example we loaded `cray-python` during last exercise).
Commit : Which programming environment are you currenlty using? Switch to `PrgEnv-gnu` and compile the two codes using `make` (see lecture slides). What are the results of commands `module list, module avail` and `module spider`?

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

- Run `sinfo -p debug`. What can you see? How can you print only your jobs? Or any other user?
- Now we want to set up a submission file, that tells the server how to run your programm. In the `exercise_session_02` folder you will find a file called `run.job` which is a basic submission file. Specify the job name and ask for 128 threads for 5 minutes on the debug partition. Replace `hostname` with the executable file created in exercise 2. Add a line where you specify the account (uzh8).
- What do you need to add to redirect your output to `output.log` and errors to `errors.log` (check SLURM documentation)?
- 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?
  (NOTE : in the omp submission script there may be the need to add after the srun command the flag "–cpu-bind=none" also if the script generator do not suggest it. What does this flag do? Run the code with 128 threads with and without it, what is the difference in the output?)

  Commit : Add the job scripts and your output logs to your git repository.
- `squeue` gives a printout which can be customised.
  Commit : 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 [Compilation and OpenMP setup]

Run the OpenMP version of `cpi.c` again with `OMP_NUM_THREADS=1` and `10`. What is happening?

Change `-O3` to `-O0` in the OpenMP version (Makefile) and run it again with 128 threads. What is happening? Can you see any difference?

Commit : Describe your observations from above experiments.

BONUS: Set the optimization flag to `-O3` and use again OpenMP version of the `cpi.c` code. In the job submission file set the value of `OMP_NUM_THREADS` to 128 and run the code for several different number of threads (e.g. `cpus_per_task` $= \{1, 2, 5, 10, 20, 50, 100, 128\}$). How does the scaling behave with varying number of threads? Make a plot showing the speedup of the code as a function of n_{threads}.