GPU Programming with Directives Exercise
Approximating Pi

1 Introduction

The purpose of this exercise is to investigate the performance of a piece of code with GPU offloading. The provided code uses a `for`/`do` loop to mathematically approximate the value of pi. It then prints out the approximated value, as well as the time taken for the approximation.

2 Connecting to ARCHER2

Log on to ARCHER2: `ssh username@login.archer2.ac.uk`.
Note: for this exercise, it is important to use the work file system,
i.e. `/work/project/project/username`, and not the home file system.

3 Compiling and Running

The code is contained in `pi.tar`. The tar file can be fetched from GitHub by cloning the course repository with the following commands:

```
git clone https://github.com/EPCCed/archer2-GPU-directives.git
git checkout 2025-03-12
```

Alternatively, the file can be found on ARCHER2 and copied into your `/work/` directory with the command:
'

```
cp /work/z19/shared/GPUdir/pi.tar .
```

Unpack the file with the command: `tar -xvf pi.tar`. There are equivalent C and Fortran versions available in `pi/C` and `pi/Fortran`, respectively.

Since this exercise will involve offloading to GPUs with OpenMP directives, certain modules must be loaded prior to compiling the code:

```
module load PrgEnv-amd
module load rocm
module load craype-accel-amd-gfx90a
module load craype-x86-milan
```

Makefiles are provided to assist with compilation. In order to compile the code, simply run `make`, and to remove the generated executable, run `make clean`.

Slurm scripts are also provided and should be submitted with the command `sbatch jobscript.slurm`. Before the first submission, the script must be edited to include the correct budget code, which is the project code for the course, e.g. `ta192`.

4 Offloading

Begin by compiling and running the code. Initially, it will only run on the CPU since the directives used to offload the code to the GPU have not been added yet.

Then, experiment with the following:

- offload the loop to the GPU with the `target` construct. Ensure that data is properly mapped to and from the GPU with the `map` clause. Futhermore, remember that the code inside the target region will execute sequentially.
- create teams and distribute the iterations of the loop across them with the `target teams distribute` construct. Experiment with the number of teams with the `num_teams` clause.
- add a parallel worksharing loop construct with the `target teams distribute parallel for/do` construct. Experiment with the number of iterations per chunk with the `dist_schedule` clause.