High Performance Computing Exercise Sheet 9

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: 14.11.2025
Due: 21.11.2025

Remark
To use the GPU part of daint, you need to use the account uzh8 when submitting jobs.
`#SBATCH --account=uzh8 --constraint=gpu`

General
Download the exercise codes from the course git repository. The directory contains several pieces of code that need to be adjusted to make use of the GPU. Each subdirectory also contains a makefile to compile the code. For use on Daint, you can switch compiler (NVC, PGI, CRAY or GNU) by properly setting 1 and 0 in the beginning of the makefile. Preferably use PGI, as it is up-to-date with the most recent openACC standard. Don't forget to load the appropriate modules first (`cray, nvhpc`). If not on daint, you need to change the command which is calling the compiler. The exercise codes contain a serial (openmp) and openACC version of the critical function. The places where you need to modify or add something are marked with `TODO`.

Exercise 1 [axpy]
This code performs a simple mathematical operation $y_i + \alpha x_i$ for n couples of (x_i, y_i) and store the result back in y_i. Modify the function `axpy_gpu` so that the work is done by the gpu. You also need to make sure the gpu has access to the arrays and that the result is copied back to the host. Compile and run the code with one CPU core. The command in your job script is
```
srun -n 1 ./axpy.openacc N
```
where `N` indicates the size of the problem (2^N). Try a few different sizes. What can you say about the execution time of the GPU compared to the CPU version?

Commit: Upload you source code and comment on the execution time.

Exercise 2 [basics/blur]

This code performs a two-step bluring of a 1D array by calling the function `blur` on the array elements. This procedure is done `nstep` times. The most straightforward way of parallellizing this for the GPU, is to offload the loops over the array elements (n) to the GPU. Implement this in `blur_twice_gpu_naive`. Remark that the function `blur` is called within the parallel region.

- What is the problem with this approach?

The function `blur_twice_gpu_nocopies` shows you a better way of handeling the loops. Fill the openACC directives instead of the `TODO`s.

- What is the difference with the naive implementation?

Compile and run both versions (change the function in `main` to test the different implementations). The executable accepts two input parameter: the size of the main vector `N` (again the size is 2^N) and the number of blurring iterations `nstep`.

- What is the difference in execution time?

Commit: Upload your source code and answers.

Exercise 3 [basics/dot]
This code performs the dot product of two n-dimensional vectors.

- Why is this function prone to race conditions?
- What keyword is used to prevent this?

Fill in the appropriate openACC directive. Compile and run the code on one core.

- How well does the GPU code perform compared to the CPU version?

Commit: Upload your source code and answers.