

## Excercise Sheet 4

(To be processed until 06.01.)

## Lecture Parallel Processing Winter Term 2025/26

### Compulsory Exercise 1: GPU-Parallelization of the Jacobi method using OpenMP Submit until Tuesday, January 6<sup>th</sup>, 10:00 via moodle

In this exercise, you will examine the use of GPUs for parallel processing. Be sure to first have a look at Sect. 3 of the [tutorial slides](#) and the [accompanying screen cast](#).

- a) Parallelize the Jacobi method (see Exercise 1 on [Exercise Sheet 3](#)) using OpenMP, such that the computation is offloaded to the GPU. Note that you should process this exercise in the lab room H-A 4111, where the computers are equipped with a Nvidia RTX 4060 GPU.

Since the OpenMP `target` environment cannot deal with the variable-sized 2-dimensional arrays used in the solver, for this exercise the arrays have been linearized. Thus, instead of accessing `a[i][j]`, the modified code uses a macro to compute the index in the linearized array, which then reads `a[INDEX(i, j, n)]`. You will find the modified code in the archive [u04eFiles.zip](#)<sup>1</sup> on the lecture's web page.

Be sure to review Sect. 3.6.3 of the lecture, before you start, esp. slide 289. To allocate the auxiliary array `b`, use the OpenMP library routines `omp_target_alloc()` and `omp_target_free()`, as mentioned in the notes for slide 284:

```
int device = omp_get_default_device();  
fp_number *b = (fp_number *)omp_target_alloc(n*n * sizeof(fp_number), device);  
...  
omp_target_free(b, device);
```

You should use the `nvc++` compiler to compile your program. To do that, you have to load the `nvhpc` module first, using the following shell command:

```
module load nvhpc
```

**Document to be submitted:** modified C++ file `'solver-jacobi.cpp'`.

- b) Examine the performance for small matrix sizes (e.g., 1000) and large matrix sizes (e.g., 20000) for both 64-bit and 32-bit floating point arithmetic and compare to the performance on the CPU (see exercise 1 on [Exercise Sheet 3](#)). In order to use 32-bit floating point arithmetic, change the typedef in `heat.h` to

```
typedef float fp_number;
```

Use the Nvidia Nsight tools to get an insight into the reasons for the observed behavior. A short tutorial for the Nsight tools is provided in Sect. 3.2 of the [tutorial slides](#) and the [accompanying screen cast](#).

What is limiting the performance of the solver for 64-bit arithmetic? Why is the performance with 32-bit arithmetic only about twice as high, although the theoretical performance of the RTX 4060 is 64 times higher for 32-bit than for 64-bit arithmetic?

**Documents to be submitted:** PDF file showing

- screen dump of Nsight Compute showing the relevant analysis results of the compute kernel for a small matrix,
- screen dump of Nsight Compute showing the relevant analysis results of the compute kernel for a large matrix,
- answers to the questions.

---

<sup>1</sup><http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/pv/u04eFiles.zip>

## Compulsory Exercise 2: Task Parallelism with OpenMP

Submit until Tuesday, January 6<sup>th</sup>, 10:00 via moodle

In this exercise, we revisit the optimization problem presented in Exercise 3 on [Exercise Sheet 2](#).

However, this time the program contained in `optimize.cpp` (in the archive [u04eFiles.zip](#)<sup>2</sup> on the lecture's web page) doesn't use a loop. Instead it just calls four functions, each examining a different configuration, and then computes the minimum cost. To parallelize this code, you can use either

- the `single` directive (see Sect. 3.3.6 of the lecture),
- the `sections` directive (see Sect. 3.5.1), or
- the `task` directive (see Sect. 3.5.2)

to execute the four calls to `computeCostForConfig...()` in parallel.

- Parallelize the program using one of the approaches given above. Do not modify any code in `functions.cpp`!  
**Document to be submitted:** modified C++ file `'optimize.cpp'`.
- Determine the speed-up of your program. In addition, think about the difference between the three approaches mentioned above, especially, how the computational work is distributed to the existing threads. Shortly describe these differences.  
**Document to be submitted:** PDF file summarizing speed-up and differences.

---

<sup>2</sup><http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/pv/u04eFiles.zip>