Advanced Parallel Algorithms

Exercise 1

- **Submit electronically (MOODLE) as one archive named after you until Thursday, 17.11.2022 9:00**
- **Archive should encompass a PDF and your code**
- **Code should be compilable (include Makefile or similar)**
- **Include names on the top of the sheets.**
- **A maximum of two students is allowed to work jointly on the exercises.**

Reading assignment

Chapter 8, CUDA toolkit example:

- `6_Advanced/FDTD3d`

1.1 Kernel instead of memcpy (10 points)

Modify the example `1_Utilities/bandwidthTest`, such that when the data is on the device or pinned, instead of the `cudaMemcpy` function a `copyKernel` is used to copy the data. The access pattern of the `copyKernel` should be the same as in `0_Simple/vectorAdd`. Do not optimize the `copyKernel` yet (this comes in 1.2), but do compare the bandwidth of the simple `copyKernel` against the bandwidth of `cudaMemcpy` for a large transfer like 100 MiB in all directions D2H, H2D, D2D.

1.2 Amount of work per thread (25 points)

The `copyKernel` can be parametrized by the amount of data that each thread transfers per instruction (4 (`int`), 8 (`int2`), 16 (`int4`) bytes) and by the number of transfer instructions per thread (1, 2, 4, 8 instructions). For the former it is best to use the predefined types (`int`, `int2`, `int4`) because they have appropriate alignment, in contrast to normal structs. For the latter use an elegant range similar to `6_Advanced/c++11_cuda`. For 100 MiB of data, with which parameters do you obtain the best performance? Information on data alignment you can find in

- http://en.cppreference.com/w/cpp/types/aligned_storage
- CUDA C Programming Guide 5.3.2
- CUDA toolkit example `6_Advanced/alignedTypes`

1.3 On-the-fly computation (25 points)

Rather than operating with unsigned char pointers, make a template version of `copyKernel` that works for various numerical types (`char`, `short`, `int`, `half`, `float`, `double`) and allows to scale the data by a certain constant factor passed in to the kernel as the same type. For fast operations on `half`, see `0_Simple/fp16ScalarProduct`. Given the template type, make sure that the kernel chooses the best parameters from 1.2 automatically, such that you can still achieve the same performance as in 1.2.