Technische Hochschule Ingolstadt
Prof. Dr.-Ing. Richard Membarth

17. April 2023

GPU Programming
Assignment 3

Submission deadline for the exercises: 24. April 2023

3.1 Memory Spaces

a) The following table lists all memory spaces available in CUDA as well properties such as

- scope (is it private to each thread, can all threads within a block access it, or is it globally accessible?);
- whether it is read-only in device code;
- whether it is cached;
- can it be dynamically allocated from host or device?

Unfortunately, the table is incomplete. Please fill in the missing information.

name	scope (thread/block/global)	read-only (y/n)	cached (y/n)	dyn. alloc. from	
				host (y/n)	device (y/n)
	global	n	y	y	y
constant					
texture					
	block				
register					
	thread				
system					

3.2 Gaussian Blur Filter

The purpose of this exercise is to get familiar with some of the different memory spaces in CUDA using a Gaussian blur filter on random data.

a) The `blur.cu` file contains the host and device code for the Gaussian blur filter. Implement the `blur_kernel` CUDA kernel that applies a given filter `filter` to an input image `in` and stores the result to `out`:

```
__global__ void blur_kernel(float* in, float* out, float* filter,
                            int size_x, int size_y, int width, int height) {
    // ...
}
```

The filter is of size $size\_x \times size\_y$ and the images have $width \times height$ pixels. We use a simplified boundary handling as shown in the `compute_reference` implementation: When we have an out-of-bounds memory access, we skip the computation for that filter coefficient. Add the missing CUDA API calls on the host side in order to launch the kernel:

- allocate device memory
- copy host memory to the device
- launch the kernel
- copy the device memory back to the host
- free device memory

Make sure that your code checks for CUDA API errors and works for different kernel input sizes.

b) Benchmark the Gaussian blur filter reporting the average execution time of 10 kernel invocations using either the CUDA event API or `nsys profile --stats=true` exploring different memory spaces:

- adding the `const` qualifier for kernel parameters where appropriate
- adding the `__restrict__` qualifier for kernel parameters where appropriate
- adding the `const` and `__restrict__` qualifier for kernel parameters where appropriate
- using `__constant__` memory for `filter`
- using `texture` memory for `in`

For the `texture` memory, define your `texture` using the `cudaReadModeElementType`:

```
texture<float, 2, cudaReadModeElementType> in_tex;
```

Allocate a CUDA array for texture and bind the CUDA array to the texture:

```
cudaChannelFormatDesc desc = cudaCreateChannelDesc<float>();
cudaArray* d_arr_in;
cudaMallocArray(&d_arr_in, &desc, width, height);
cudaMemcpy2DToArray(d_arr_in, ...);
cudaBindTextureToArray(&in_tex, d_arr_in, &desc);
```

For further documentation, lookup the functions of the CUDA API on https://docs.nvidia.com/cuda/cuda-runtime-api/index.html.