GPU Programming 2020/21

Assignment 3

Due: 12/01/2021, 11:00

- Each student has to implement and submit the solution to the assignment her-/himself (it is admissible to discuss the assignment with other students but everyone has to implement their own solution). Plagiarism will be punished according to the study regulations.
- Base your solution on the skeleton code that is provided.
- Submit your implementation as well as the plot(s) before the deadline on the course site. Your code has to compile and run with the given CMake file on the machines in G29-426.

In this assignment, you will implement reduction in CUDA. It determines

$$r = a_0 \otimes a_1 \otimes \cdots \otimes a_{n-1} \tag{1}$$

for an input array $a = \{a_0, \cdots, a_{n-1}\}$ and a binary, associative operation $\otimes$. Reduction uses a tree-based algorithm that hierarchically combines two adjacent elements using $\otimes$ to efficiently determine Eq. 1. For simplicity, in our implementation the binary operation will be addition and $a_i \in \mathbb{Z}$, i.e. we will compute the sum of a list of integers. We furthermore assume that the result of the reduction is required on the host so that it is admissible to perform some computations there.

Your tasks are as follows:

1.) Download the skeleton code and generate the build system using `cmake`.

2.) Implement a serial version of Eq. 1 that computes a reference solution. (1 Point)

3.) Complete `version1()` and `reduction1()` that implement a version of reduction that directly operates on global memory and provides the correct result for at least $n = 16384$. Your implementation should use a number of threads that is appropriate for the chosen input size and that exploits the available parallelism on the device. (3 Points)

4.) Complete `version2()` and `reduction2()` that implement reduction using shared memory for at least $n = 16384$. Document the speed up that this version provides compared to `version1` with a graph where the performance of both is plotted as a function of n. (3 Points)

5.) Complete `version3()` and `reduction3()` that implement reduction using shared memory and that avoid divergent branches to the extent possible. Document the performance as in the previous question. (3 Points)

Bonus: Use templates to implement a version of reduction where all loops are unrolled and branching is minimized. Again document the performance as in the previous questions and if necessary document your implementation in the code. (3 Points)

Bonus: Use template meta-programming to implement a version of reduction that works for arbitrary admissible operations $\otimes$. (3 Points)