Prof. G. Zachmann
Maximilian Kaluschke (mxkl@uni-bremen.de)
Thomas Hudcovic (hudo@uni-bremen.de)

University of Bremen
School of Computer Science
CGVR Group
April 17, 2024

Summer Semester 2024

# Assignment on Massively Parallel Algorithms - Sheet 2

Due Date April 28, 2024

## Exercise 1 (Amdahl's law, *4 + 1 Credits*)

Given a single core processor A and a multi-core processor B with N cores. Additionally, assume that all cores of A and B are identical.

a) Given a program that runs 1.7 times faster on processor B than on processor A. Compute the parallel portion of the program i.e. $f = P/(P + S)$ with $P$ = execution time of parallizable part on single processor and $S$ = execution time of inherently serial part on single processor (see Slide on Amdahl's Law (the "Pessimist") in the Introduction Chapter).

b) Suppose parallel portion $f$ is 0.5, how many processor cores are needed to achieve an overall speed up of 1.6?

Bonus) You have also been introduced to Gustafson's Law. Briefly describe the differences in the approach of Amdahl and Gustafson. What do both focus on and why can Gustafson's Law be considered "optimistic" compared to Amdahl's Law?

## Exercise 2 (CUDA: Atomic operations, *6 Credits*)

Start from the framework `dotProduct`. We implemented a CUDA kernel with the name `dotProduct(int *in1, int *in2, int *out)` which calculates the dot product of the two integer vectors `in1` and `in2` with the length of `numElements` and writes the result at the integer pointer `out`.

There is a thread for each vector element of `in1`, and each thread multiplies it's component with the corresponding component of `in2`. Aftewards each thread adds the local result to the overall result `out` (which is initialized with 0) by using the atomic function `atomicAdd(...)`. `atomicAdd` takes an int pointer and an int value and adds the value to the integer at the pointer and synchronizes simultaneous modifications by serializing them.

a) CUDA kernels cannot return a value. What could be the reason for this?

b) What is likely the bottleneck of the given CUDA kernel? Give a reason.

c) What problems could arise from encoding the length of the vectors `in1` and `in2` in the CUDA kernel grid size like we did in the framework?

d) The kernel `dotProduct` computes $\sum_i^N v_i w_i$ with input vectors $v, w$ of length $N$ with $v_i$ being the $i$-th vector component of $v$ with integer value. Modify `dotProduct`, such that it computes

$$max(v_0 + w_0, v_1 + w_1, ..., v_N + w_N)$$

e) What problems would arise if you were to implement the previous task for float vectors?