KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Warps, Striding and Thread Divergence

GPU vs CPU

Benjamin Rübenkamp
R0793577
MEIsw
Master industrieel ingenieur Elektronica-ICT
Academiejaar 2023-2024

** There are no code examples in this text, please look in github repository under session1: https://github.com/Benjamin4-23/Geavanc_comp_arch/tree/main/session3 **

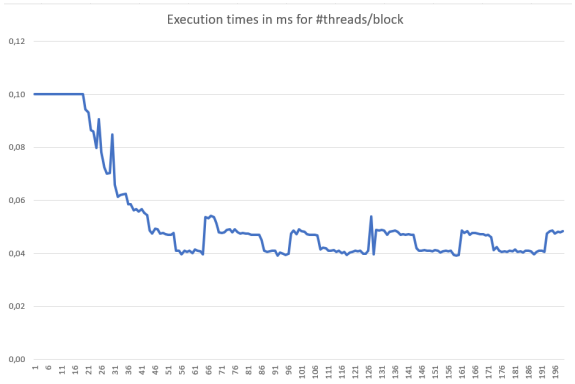
Introduction:
For this exercise, an image is read into memory (RGB values). On the values, we can then perform operations such as color inversion. I first used the GPU to invert all values of the image array. (x = 255 - x) Then I added striding. I implemented a 1:16 ratio of threads: elements.

Loading the image:
First, the image is opened and all RGB values are kept in 1 large array. This array is transferred to the GPU memory where it can be used for calculations.

Processing the image:
There is a kernel function that when called performs the requested operations on the values of the picture. First, this was invert all values (x = 255 - x). Later another operation was used for the red values to demonstrate thread divergence. (Not all threads will perform equally long because they perform different operations). The operation for the red-values is x = (x % 25) * 10.

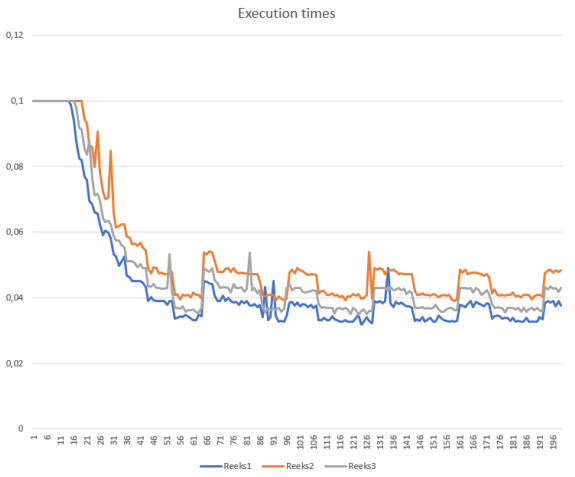
This kernel function is called repeatedly from the main method with the difference that each time the number of threads per block changes. The number of required blocks also changes. (The 1:16 ratio is taken into account.) A record is kept of how long the calculations take before the result (the new image values) is written to the output file. It is tested with threads_per_block-values from 1 to 200. You can clearly see in the times that it optimizes up to a multiple of 32 (also #threads in a warp) after which the time it takes becomes a lot longer again. (An additional warp (32 threads) has to be scheduled for just 1 element. The GPU always schedules entire warps, so it can't execute for example 48 threads. It can schedule and execute 2 warps which contain 64 threads, but then the last 16 threads of the second warp will not be used. Scheduling an extra warp when the threads per block is just 1 more than a multiple of 32, for each time a block is executed, will end up causing a lot of overhead.

In a chart of execution times, we see clearly that every time it goes above a multiple of 32 the execution time increases again.



Figuur 1: Execution times

Finally, I also took a look at what difference there was in execution time due to thread divergence. First, I tested by performing the simple operation (reeks 1) on all values, then the complex computation only on the red values (reeks 2), and finally the complex computation (sequence 3) on all values. The result surprised me. Of course, the sequence with all easy operations was the fastest, but the half-and-half sequence took longer than the sequence where they all did the difficult operations. This is probably due to the overhead of the if statement to check if the value is red.



Figuur 2: Comparison execution times