Advanced computer architecture
Lab2: Striding and warps

by Steven IMPENS & Nino SEGERS

MELICT
2020-2021

Mentor: Alexander MARINSEK

Contents

1 Introduction

In this lab we will find out what striding and warps is all about. We learn how striding is useful in the code execution, why it should be implemented and what the benefits are. The next part of the lab is about warps where we will try to determine the impact of the thread size and limitations of CUDA.

The code repository can be found at:
https://github.com/imstevenxyz/geavanceerde-computerarch

2 Striding

To see the benefits of striding an image is generated using the given formula. This formula is implemented in two kernel functions, one that executes with striding and one without. Both are run with an amount of threads, the total pixels, a quarter of the total pixels and with more than the total pixels.

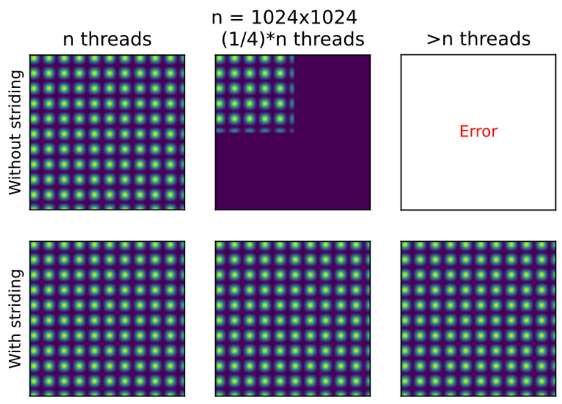

Figure 2.1: *Timing performance semi parallel kernel*

It is directly obvious in figure 2.1 that without striding we get three different results while with striding we get the wanted result for all variations. The outstanding in the figure is when we use more threads than pixel. In this variation the code will return an error because otherwise memory that does not belong to the image is accessed, this is either impossible or very dangerous (unwanted memory access, data theft, etc). When using a quarter of the pixels, only a quarter of the image is generated because the other pixels will not be calculated since there are not enough threads.

From this we can see that striding actually helps organize the threads in two ways:

- Not enough threads? $\rightarrow$ Run threads sequentially.
- More than needed threads? $\rightarrow$ Do not run the calculations for non-existing memory locations.

3 Warps

A warp is a set of threads that work at the same time and execute the same code. The threads all share resources, they start and stop at the same time. This allows for better performance when the code is executed in warps. The size of a warp is depended on the architecture of the GPU and on the GPU we work on the warp size is 32 threads.

In figure 3.1 the execution time is plotted in function of the amount of threads used. This to show the effect of warp. The Y-axis is the execution time and X-axis is the amount of threads used.

The names in the legends mean:

- B: Amount of blocks allocated (singular*).
- T: Amount of threads allocated per block (singular).
- I: Amount of pixels in the image (cubic)

(*the amount of blocks is actually cubic in the code, but we display it singular on the plot. This is to keep the numbers lower on the plot, so it's easier to see them, while reducing the computation time to get all the results. If interested in the real numbers you can open "plot_actual_threads.html" on the git repository.)

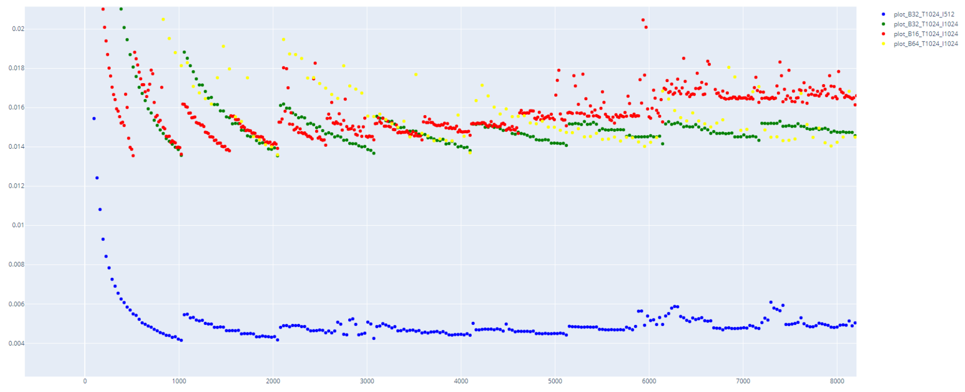

Figure 3.1: *Execution time in function of amount of threads*

In figure 3.1 we can see several interesting features pop up. Let's start with the 'waves'. You can see for all cases that there form something like 'waves'. The time drops the more threads are allocated and then suddenly jump up, and this pattern repeats itself on regular interval. The repetition of the wave is because of the amount of blocks and the warp size. For B_16 it repeats every 512 threads, 16*32=512. For B_32 it repeats every 1024 threads, 32*32=1024. For B_64 it repeats every 2048 threads, 64*32=2048. The overall time increases because of the overhead of the large amount of threads that are used. The blue plot B_32_T1024_I512 is faster just because the image is smaller so fewer need to be calculated. The warp is still there and the pattern is on the same interval as the green plot, the green plot has the same amount of blocks.

The waves (drop, jump, repeat) are there because:
The drop is there because in the interval the warps are not full, so more threads do not require more resources, warp will execute anyway. So any more threads can only reduce the amount of time that is required.
The jump is because once all the warps are full another set of warp needs to run. Another warp will require a lot more time even though there are more threads.
The repetition is there because the last warp, the unfilled one, is the bottleneck. And this only stops when you run out of threads.

4 Conclusion

With striding the code doesn't need to know beforehand how many blocks and threads needs to be allocated. The striding will work out if threads have to run the code multiple times or if threads don't have to run at all. And it will have somewhat of an even division of the workload. The warp has an impact on the time. More threads doesn't mean faster work. The threads should fill an entire warp, if not it will be slower and cost more resources.