GCA Session 5 - Lab Report

Maxime Rombaut
Master of Electronics and ICT Engineering Technology

Abstract: In this report, we present our work on developing a CUDA-based application that compares the execution times of 4 kernels. This happens in 2 approaches: synchronous and asynchronous.
Keywords: CUDA, GPU, CPU, C++

1 Introduction

There are 4 kernels implemented: sum, product, min and max. All of those get an array with 1024 elements. Before execution those kernels, the arrays will have to be initialized. This can happen in different sequences. In this lab we want to compare the differences in execution time as a result of those sequences.

2 Synchronous

In the synchronous approach the CPU does all its work first. When it is done, the GPU can start doing its work. We have implemented this in 2 ways.
In the first, we use the regular memcpy to copy the input array from the host to the device. Memcpy will block the calling thread until the copy operation is complete.
In the second, we use the memcpyAsync. This copy method uses a cudaStream. In this implementation we don't explicitly give one so the default stream will be used. MemcpyAsync works asynchronously and doesn't block the calling thread. By using this, the CPU can continue doing work while the GPU is working. Because of the structure of the implementation we would think that this doesn't have that much effect because the CPU doesn't have to do that much work after the initialisation of the arrays.

3 Asynchronous

In this approach, we initialise the arrays just before they are needed in their kernels. Because of this, the CPU could initialise the array of the next kernel while the previous kernel is still running.
We have implemented this approach with and without defining streams. In the implementation with difined streams, every kernel has its own stream. In the one without defined streams, the default stream will be used.

4 Comparison

In this section we compare the execution times of the different implementations. To make a representative comparison, we ran every implementation a 1000 times. We then took the average of those times. We have also experimented with adding additional work for the CPU in the implementations. The extra work was initialising an array of a 1000 elements for every kernel.

We can see that with the extra work, the execution times of all implementations are about the same. Without it however, the synchronous approach with the regular copy takes a while longer than the other implementations. With the asynchronous copy, the execution time is as fast and sometimes even faster than the asynchronous implementations.

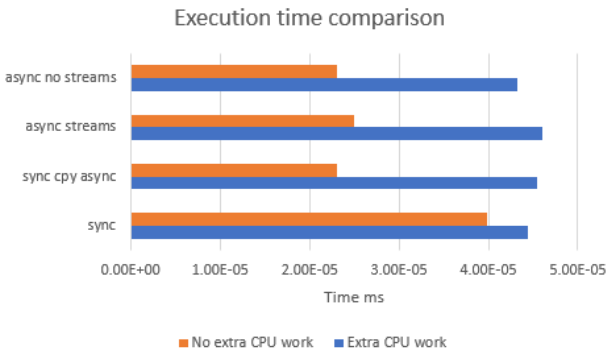

Figure 1: *Execution times of the different implementations*

At the moment we think that this could be because the kernels can execute at the same time, without waiting for the previous ones. While with the regular memcpy, the CPU would wait on the first kernel to finish to call the next one. When running the experiment a few times, we saw that the

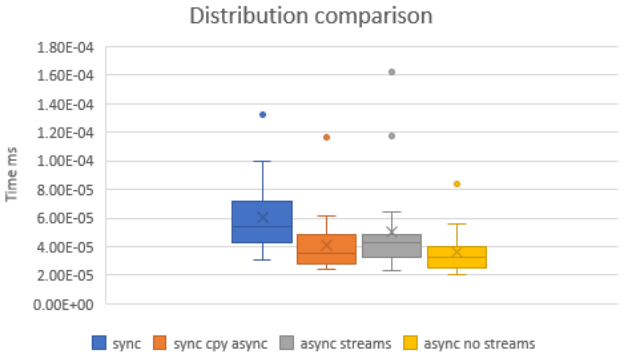

Figure 2: *Spread of the averages per implementation*

averages were a lot different every time. We can see that the sync implementation has the most spread, not considering the outliers. The differences between the others are about the same.

5 Github link

`https://github.com/MaximeRombaut1412/GCA_labs.git`