Multicore Programming

Filip Trela

Application of concurrent GPU kernel execution for vector addition

Introduction

The point of the report is to compare the performance of CPU execution, GPU parallel kernels and GPU concurrent kernels when tasked with operation of vector scaling and addition.

Short description of the algorithms

Source code used for the project can be found at:
https://github.com/Faramour/FT_MCP_Project

The algorithm used to test performance is a simple operation of scaling values at all indices of an input vector by a scalar float number and then adding another vector to the outcome of scaling:

$$\mathbf{Y} = a \cdot \mathbf{X} + \mathbf{B},$$

where **X** is an input vector, *a* is some scalar float number and **B** is a vector we add to the result of scaling **X** by *a*.

3 approaches to executing that algorithm will be tested:

- CPU execution - algorithm is launched on the host device,
- GPU parallel – algorithm is launched on a single GPU kernel (default stream),
- GPU concurrent – algorithm is launched on several GPU kernels at once (non-default streams).

To measure algorithm's execution time, I used std::chrono library.

Measurements

All listed times are averages from multiple operations. CPU algorithm has been run 50 times, both GPU implementations were ran 1000 times each. Concurrent algorithm was run on 4 streams and 8 streams to compare differences in execution time.

Table 1. Measurements of execution time for different implementations of the algorithm and input/output vector size, in milliseconds.

Vector size	CPU	GPU parallel	GPU conc. 4 streams	GPU conc. 8 streams
2^{26}	119.1	2.111	2.129	2.134
2^{27}	233.6	4.222	4.261	4.266
2^{28}	476.2	8.464	8.516	8.528
2^{29}	950.8	16.91	16.988	17.01

Summary

As expected, GPU kernel execution times are far superior compared to those of the algorithm launched on CPU. Unexpectedly, parallel and concurrent approaches are basically the same in execution time. The concurrent execution time doesn’t scale with increased number of streams either, which suggests that something is wrong in the program. My guess for that mistake is that I’ve incorrectly written the code for concurrent execution, which causes it to bottleneck in performance, dropping down to execution times similar to parallel approach.

If I were to try writing the concurrent version of the algorithm again, I’d make use of the Nvidia Visual Profiler (nvvp) tool. This program could show if the kernels are actually running concurrently or if there is some bottleneck that causes loss of performance. The reason for not using this software in the first place is that it was not installed on machines which were used to run the code (and I believe that students are not allowed to install software on them).