Matrix Toolkit Report

Paweł Lipiór, Michał Żoczek

February 9, 2020

Abstract

The main goal of this application is to present different techniques used in CUDA development, as well as to see how multi-threaded approach compares to conventional way of writing programs. To achive this we use 5 different operations on matrices (addition, subtraction, multiplying, transposition and calculating sum of a vector).

1 Program overview

The main section of the program works as a loop allowing user to choose different operations. Then, before doing calculations user is asked to input matrix (or matrices) size. Each operation work in pretty similar fashion - first input and output matrices are crated and initialized. Next, calculations are done using both CPU and GPU. The results are then compared (both execution time and operation result) and printed. The grid size for kernels is based on number of SMs checked while creating MatrixCalculator object.

2 CUDA-specific techniques used in this project

Achieving good performance with CUDA programs requires use of some techniques and tools not commonly(or, at all) found in the CPU programming. For this project we used those listed below:

1. Unified Memory - When using both CPU and GPU in the same program, transferring data between the two is often unavoidable. Using unified memory makes it easier as the task is done automatically.

2. Memory Prefetching - While unified memory is responsible for transferring data, it has no way of knowing when that data will be needed on specific device, resulting in page faulting when device tries to use data that wasn't yet moved. Prefetching makes it possible to transfer data before it's actually used greatly improving performance.

3. Grid-Stride Loops - As size of the data can be larger than maximum possible grid size the kernel should have some way of working around that limitation. This is done by using stride loops, where each iteration works on grid-sized data.

4. Grid definition - To work correctly, kernels need to be provided a grid. There are two things worth remembering when creating it: first - the threads should be grouped in multiples of warp size, and two - blocks should be grouped in multiples of Streaming Multiprocessors.

5. Shared memory - When device needs to access the same data from multiple threads, it is good practice to use much faster shared memory.

6. Reduction - In this project used for finding a sum of vector. It works iteratively reducing array size by half in each iteration.