

SoftMax

CMP 202 2023/24 Term 2

GPU Mini-Project

Annie Place

2301241

Introduction.....	4
Application Overview.....	4
Parallelisation Technique.....	4
Performance Analysis.....	5
Additional Difficulties	5
Conclusion	6

Introduction

The demand for image identification software has only increased in recent years. With that, the need for these products to be efficient and consistently reliable has scaled just as steeply. As such, this need of reliability comes with the formula involved in calculating image recognition. Softmax is one of many possible neural network methods that could be used, and its effectiveness needs to be known.

Application Overview

Softmax is a formula developed by Ludwig Boltzmann long before modern neural networks were even conceptualized. The method calculates a probability using e to the power of some exponent over the sum of e to another constant.

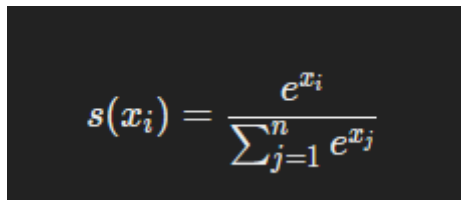
A screenshot of a dark background with the Softmax equation written in a light-colored, serif font. The equation is
$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

Fig 1. Screenshot of the Softmax equation.

The GPU project constructs three different implementations of this function, each in a different method in Unified Shared Memory: implicit, explicit, and buffers/accessors. These methods were then compared for their efficiencies running the softmax formula over a varying set of inputs. All the queues in each main method are wrapped inside a try/catch method that catches any exceptions that might crop up to display to the user.

Parallelisation Technique

The first program, named softmax1.cpp, utilizes buffers and accessors to calculate the Softmax function. The mathematical elements of the Softmax function are put into a separate class from the main function since those calculations cannot be done within the kernel. The kernel then calls a SoftmaxKernel object within the kernel and prints out its output afterwards.

The second program named softmax2.cpp utilizes explicit data movement as opposed to buffers and accessors. Similarly, the calculations are done outside of the kernel in a separate class. Instead of creating floating points with a buffer_array and an output_buffer, however, softmax2.cpp utilizes malloc_device and then manually frees the memory from these devices after the calculations have been made.

The third program, softmax3.cpp, utilizes implicit data movement. In this case, the calculations are done within the kernel due to the way the malloc_shared method is implemented. The kernel with implicit data movement can conduct the Softmax calculations however needs to have the memory freed manually by the programmer just the same as explicit data movement. Although a program with implicit data movement does not require memcpy to function, it was utilized to copy the data in the output_device array back into the output_array. With more time allotted for the project this would have been implemented differently.

Performance Analysis

The computer used for this test utilized an Intel Core i7-10700 CPU with 16.0 gb of RAM on a 64-bit operating system. The GPU used for the testing was an Intel(R) UHD Graphics 630.

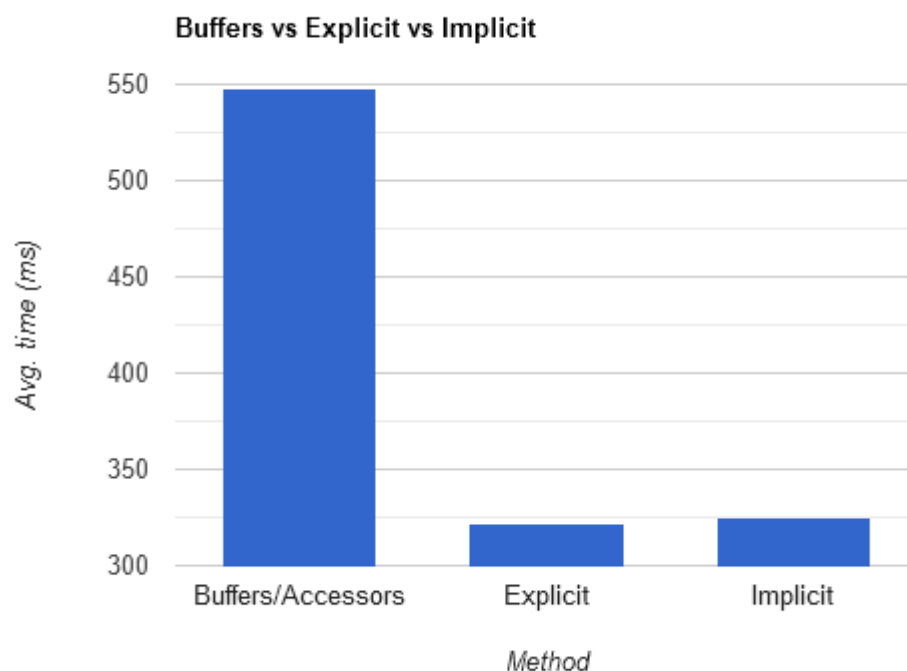


Fig 2. Bar graph showing the average times of each method.

There were several tests conducted on each method, each with a varying input array size. However, as the input array size increased each methods performance was equally as inefficient as one another. At lower variables, however, the buffer method was severally outperformed by both the explicit and implicit variants. Taking nearly twice as long as the other two methods, using buffers is highly discouraged for the Softmax equation.

Additional Difficulties

As stated previously, the logic of the kernel created several problems in the making of softmax1.cpp and softmax2.cpp. On top of this, by far the largest issue with this project was not being able to run DPC++ programs on home computers. Due to the lack of this convenience, there was a limited amount of time where the project could be effectively worked on. If the DPC++ was able to function with home systems, then there would be ample time to work out any issues that cropped out. Finally, as previously stated, the final major difficulty throughout the project was finding a way to implement implicit data movement without needing to use the memcpy method.

Conclusion

Softmax is a function with several use cases in the modern era, and as such needs to have proper implementation. Properly utilizing parallelisation techniques ensures that this can and will happen. The effectiveness of the products at play must utilize the correct Unified Shared Memory techniques if they are to be implemented onto a GPU, and the examinations in these programs are helpful in evaluating just that.