K. K. Institute of Engineering Education and Research
Department of Computer Engineering

Project Report

Convolution Operation using CuDA

Guided by:
Prof. Jyoti Mankar

By:
Shreyas Kalvankar (17)
Hrushikesh Pandit(18)
Pranav Parwate (19)
Atharva Patil (20)

A.Y. 2020-21 Sem I

Contents

1 Problem Statement **2**

1.1 Objectives 2

2 Introduction **3**

3 Requirements **4**

3.1 Software Requirements 4

3.2 Hardware Requirements 4

4 Methodology **5**

4.1 Algorithm 5

4.2 Experimental setup 6

5 Results **7**

1 Problem Statement

In mathematics (in particular, functional analysis), convolution is a mathematical operation on two functions (f and g) that produces a third function $f * g$ that expresses how the shape of one is modified by the other. The term convolution refers to both the result function and to the process of computing it. It is defined as the integral of the product of the two functions after one is reversed and shifted. And the integral is evaluated for all values of shift, producing the convolution function.

1.1 Objectives

- To implement a convolution operation over two 2-dimensional matrices using CuDA
- Understand the CuDA architecture and the CuDA kernel

2 Introduction

The convolution of f and g is written $f * g$, denoting the operator with the symbol *. It is defined as the integral of the product of the two functions after one is reversed and shifted. As such, it is a particular kind of integral transform:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)$$

The advent of powerful and versatile deep learning frameworks in recent years has made it possible to implement convolution layers into a deep learning model an extremely simple task, often achievable in a single line of code. However, understanding convolutions, especially for the first time can often feel a bit unnerving, with terms like kernels, filters, channels and so on all stacked onto each other. Yet, convolutions as a concept are fascinatingly powerful and highly extensible.

With such a versatile function at hand, implementing it in code by using simple regular sequential programming may hinder performance in case of large matrices. By virtue of the properties of integral transformation, we can divide the convolution operation into steps of fixed ranges. This divides the problem into smaller sub-problems, independent of each other, providing scope for a parallel algorithm.

3 Requirements

3.1 Software Requirements

Sr. no.	Parameter	Requirement	Justification
1	Compiler	NVCC	Nvidia CuDA Compiler
2	Binary Configuration	Microsoft Visual studio	Config file
3	C++	GNU C++ Compiler	C++ compiler
4	Nvidia Computing Toolkit	Nvidia Computing toolkit	CuDA programming

Table 1: Software Requirements

3.2 Hardware Requirements

Sr. no.	Parameter	Requirement	Justification
1	GPU	Nvidia GPU	CuDA Architecture
2	CPU	Any CPU chip	Comparative study

Table 2: Hardware Requirements

4 Methodology

4.1 Algorithm

- Select an appropriate kernel of size $n \times n$ where $n <$ size of matrix
- Pad the matrix with zeros to prevent out of bound access
- Slide the kernel onto the matrix
- Multiply the corresponding elements and add them
- Repeat until all values are calculated

We execute the code using CuDA thread blocks. A thread block is a programming abstraction that represents a group of threads that can be executed serially or in parallel. For better process and data mapping, threads are grouped into thread blocks. The number of threads varies with available shared memory. In CUDA, the kernel is executed with the aid of threads. The thread is an abstract entity that represents the execution of the kernel. A kernel is a small program or a function. Multi threaded applications use many such threads that are running at the same time, to organize parallel computation. Every thread has an index, which is used for calculating memory address locations and also for taking control decisions.

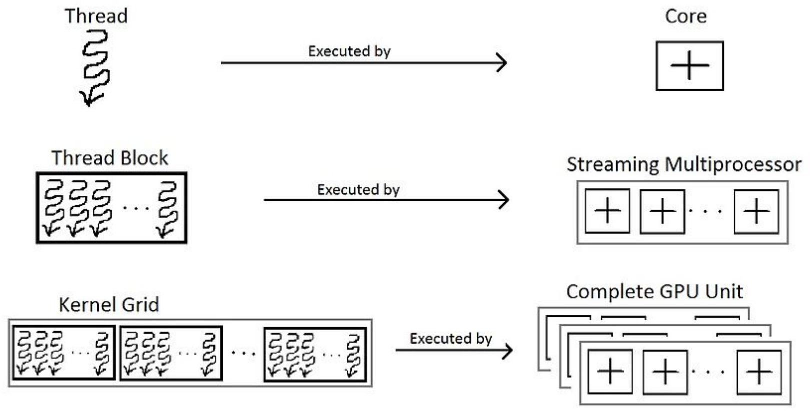


Figure 1: CuDA thread block

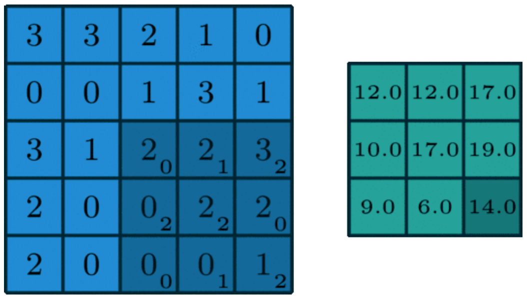


Figure 2: Convolution operation

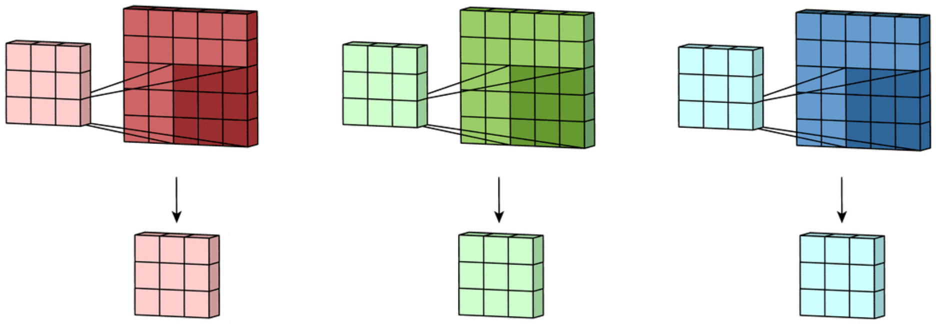


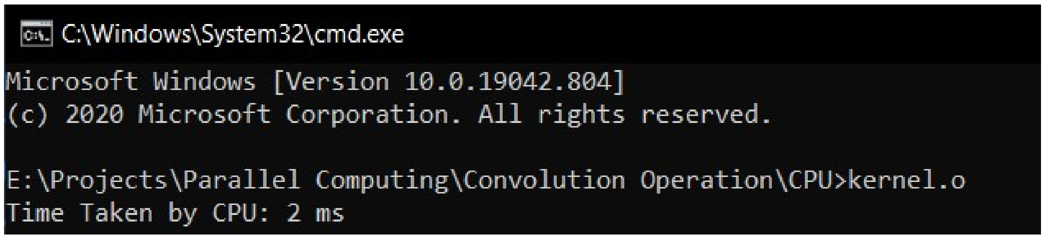
Figure 3: Per-channel Convolution

4.2 Experimental setup

We implement the code in C .The code is compiled using NVCC. The object is run on an Nvidia GTX 1060 GPU with Pascal architecture having 6 Gigabytes of memory.

5 Results

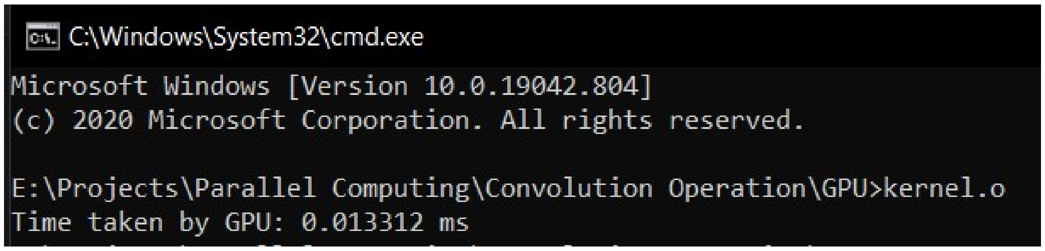
We run the algorithm on CPU as well as on GPU and we find that the algorithm runs exponentially faster when run on the GPU threads. The output is shown below:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.

E:\Projects\Parallel Computing\Convolution Operation\CPU>kernel.o
Time Taken by CPU: 2 ms
```

Figure 4: Time taken on CPU



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.

E:\Projects\Parallel Computing\Convolution Operation\GPU>kernel.o
Time taken by GPU: 0.013312 ms
```

Figure 5: Time taken on GPU

6 Conclusion

Thus we have successfully implemented convolution operation on two matrices using CuDA.