School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

COMS4040A & COMS7045A Assignment 3 – Report

Marc Marsden
1437889
BDA Hons

May 29, 2020

1 Introduction

This report compares the performance of CUDA parallel computing and serial computing by applying each one to the convolution between an image and a kernel. Three images of sizes 512x512, 1080x1080, and 2160x2160 were used with four different kernels. The four kernels are: 3x3 edge detection, 3x3 box blur, 5x5 box blur, and 7x7 box blur. The results obtained in this report were achieved by making use of the CUDA Toolkit release 10.1, V10.1.243, running the code on Google Colab, which has the following specs:

- **CPU**: Intel Xeon @ 2.00GHz with 40MB L3 cache and 1 available core with 2 threads.
- **GPU**: Nvidia Tesla T4, with 2560 cores, 8.1 TFLOPS Single-Precision, 16GB GDDR6 VRAM.
- **RAM**: 13GB
- **Disk**: 34GB

2 Cuda Memory

This section discusses the different types of CUDA device memories used in this report. All information has been sourced from the CUDA C programming guide [NVIDIA 2018].

2.1 Global Memory

Global memory, found in Device RAM (DRAM), is memory which can be modified by both the host and the device. The memory transaction is limited to 32, 64, and 128 bytes at a time and memory addresses have to multiples of these sizes to be accessed. Global memory has a high latency due it being off-chip memory.

2.2 Shared Memory

Unlike global memory, shared memory is on-chip memory. This allows for higher bandwidth and lower latency than other memory types. Each thread block has access to an allocated shared memory, which other blocks can not access. The threads inside the their respective block all have access to the shared memory of the block. Shared memory must be implemented with synchronisation of threads due to the chance of race conditions occurring.

2.3 Constant Memory

Constant memory is a small memory with only 64KB, which is cached in a read-only “constant cache”. This read-only memory, which resides in the DRAM, can allow for memory accesses as fast as the on-chip registers, if the memory accesses are coalesced properly. Since it is read-only memory, the data stored can only be modified prior to the launch of the kernel.

2.4 Texture Memory

Texture memory, found in DRAM, is cached in the "texture cahce". If there is a failed attempt to read or write to the cache, there will only be one read access to global memory. This memory is optimised for 2D or 3D "spatial locality". This means that if the data is close to a 2D or 3D form, the performance will be improved.

3 Design Choice

3.1 Boundary

For this report, padding the image with zeros was chosen to prevent out of bounds access from occurring. Padding the images was chosen over boundary checking since the image would only have to be padded once, compared to having to check boundaries for every implementation.

3.2 Global Memory Implementation (Naive)

The global implementation is the most naive approach compared to the other CUDA approaches. Due to this, there was not much that could be considered when designing the implementation, except that a flattened matrix (1D array) was used for both the image and kernel to speed up memory access since 1D access is much faster than 2D.

3.3 Shared Memory Implementation

A flatten matrices were used to store the image and kernel to improved memory access. To minimise the amount of global memory accesses, a 2D shared memory array was used to tile and store the image in shared memory. The choice of a 2D array was to simplify the code.

3.4 Constant Memory Implementation

Due to the 64KB memory limit and that constant memory is read-only, the kernel was chosen to be stored in constant memory, because it will never be modified and is very small.

3.5 Texture Memory Implementation

A 2D texture reference was implemented with "Border" mode as the address mode. This allowed CUDA to treat the outer bounds as zero, if the program accesses out of bounds memory. Since the pixels of the image were already between 0 and 1, the normalised function was set to false.

4 Questions and Answers

4.1 How many floating point operations are performed in your convolution kernel using global memory? Explain

On average, the number of floating point operations performed in the naive implementation of the convolution kernel is 2×(Number of pixels in image)×(Number of elements in Filter). This is due to the fact that each pixel of the image must have each element of the filter applied to its surrounding elements. The "2" represents addition and multiplication that occurs.

4.2 How many global memory reads are performed by your kernel using global memory and kernel using shared memory, respectively?

For both the global and shared memory, the number of global memory reads are, on average, 2×(Number of pixels in image)×(Number of elements in Filter).

4.3 How many global memory writes are performed by your convolution kernel using shared memory?

The number of global writes which the shared memory implementation performs is the number of pixels that the image contains. i.e if the image is MxN, where M is the width and N is the height, then the number of writes is M×N.

4.4 What would happen to the performance of your kernel using shared and constant memory when the size of the 'averaging' mask increases (say, to a substantial large size) ?

The performance will decrease, to the point where it will take longer than the global constant memory implementation, until the filter becomes too big for the constant memory to store it. When this happens the kernel will fail.

5 Discussion

All figures referenced may be found in the Appendix section of the report.
All images used and outputted may be found in the Appendix section of the report

Throughout the various sizes of images and filters, the sequential program performed the worst in all cases by a substantial margin. In the smallest case (512x512 image and 3x3 filter) the sequential had a run time of 12ms 29 and in the largest case (2160x2160 image and 7x7 filter) the sequential had a run time of 1091ms 31. It is obvious from graphs 26, 27, and 28 that the sequential must have peaked for lower sized images and will only outperform the parallel implementations when the parallel overhead is too great.

Global memory performed the worst out of all the parallel implementations. With a minimum time of 0.15ms and a maximum time 10.7ms, it still greatly reduced the run time compared to the sequential implementation. As the number of threads increased, the performance of the global memory tends to the performance of the shared memory implementation, especially for small filters 27. When the global memory is implemented using constant memory the speed up is more than doubled and becomes more linear as the size of the data increases 28. This modified version of global memory outperforms all implementations, but shared constant memory.

Texture memory has a very similar performance to that of shared memory, which outperformed global memory, and texture memory even outperforms shared memory for very small data sizes 29. However, the implementation which outperformed all other parallel implementations is the shared constant memory. Compared to shared memory's minimum time of 0.122ms and maximum time of 7.64ms, shared constant dominates these times with a minimum time of 0.04ms and a maximum time of 1.479ms 29 31. Even though shared constant memory is the best in terms of performance, it is very limited by the amount of data which can be stored in its cache and so will fail for data sizes that exceed 64KB.

6 Appendix

6.1 Original Images

Figure 1: lena_bw.pgm (left), harvard1080.pgm (right), harvard2160.pgm (bottom)

6.2 Lena 512x512

6.2.1 Edge 3x3

Figure 2: Sequential (left), Global (centre), Global Constant (right)

Figure 3: Shared (left), Shared Constant (centre), Texture (right)

6.2.2 Blur 3x3

Figure 4: Sequential (left), Global (centre), Global Constant (right)

Figure 5: Shared (left), Shared Constant (centre), Texture (right)

6.2.3 Blur 5x5

Figure 6: Sequential (left), Global (centre), Global Constant (right)

Figure 7: Shared (left), Shared Constant (centre), Texture (right)

6.2.4 Blur 7x7

Figure 8: Sequential (left), Global (centre), Global Constant (right)

Figure 9: Shared (left), Shared Constant (centre), Texture (right)

6.3 Harvard 1080x1080

6.3.1 Edge 3x3

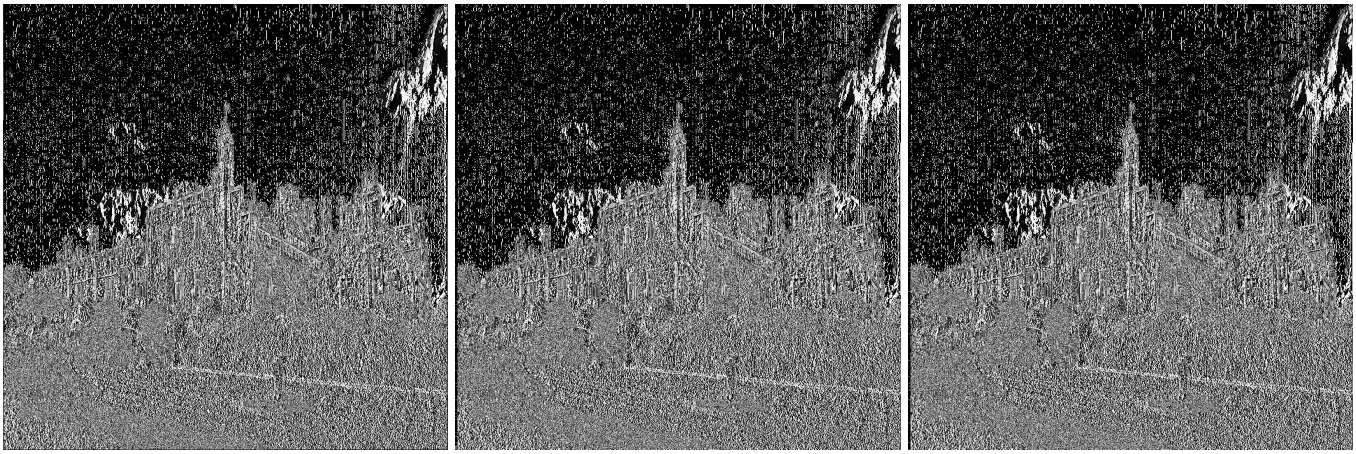

Figure 10: Sequential (left), Global (centre), Global Constant (right)

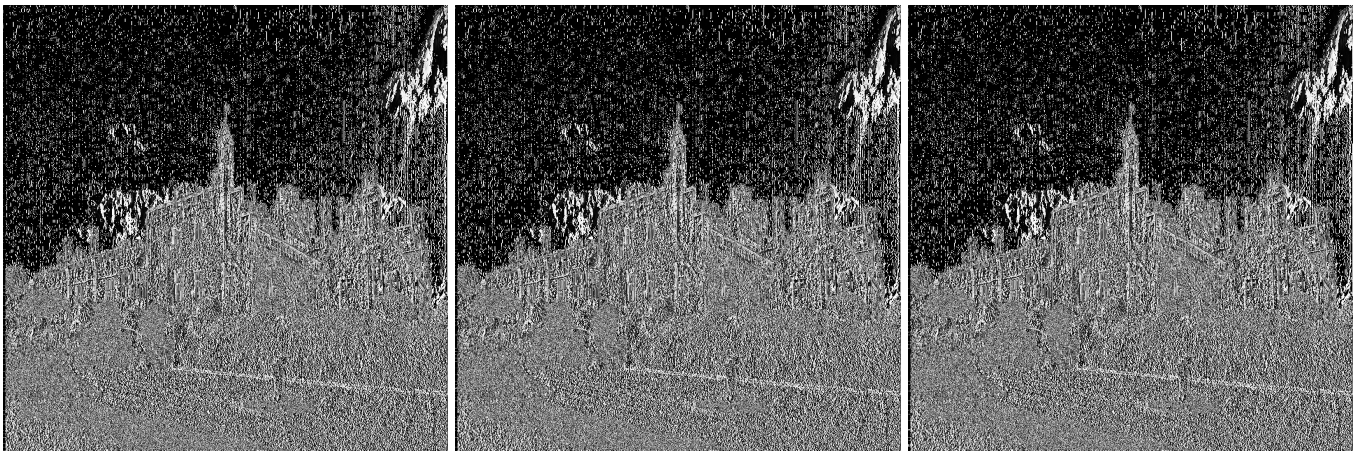

Figure 11: Shared (left), Shared Constant (centre), Texture (right)

6.3.2 Blur 3x3

Figure 12: Sequential (left), Global (centre), Global Constant (right)

Figure 13: Shared (left), Shared Constant (centre), Texture (right)

6.3.3 Blur 5x5

Figure 14: Sequential (left), Global (centre), Global Constant (right)

Figure 15: Shared (left), Shared Constant (centre), Texture (right)

6.3.4 Blur 7x7

Figure 16: Sequential (left), Global (centre), Global Constant (right)

Figure 17: Shared (left), Shared Constant (centre), Texture (right)

6.4 Harvard 2160x2160

6.4.1 Edge 3x3

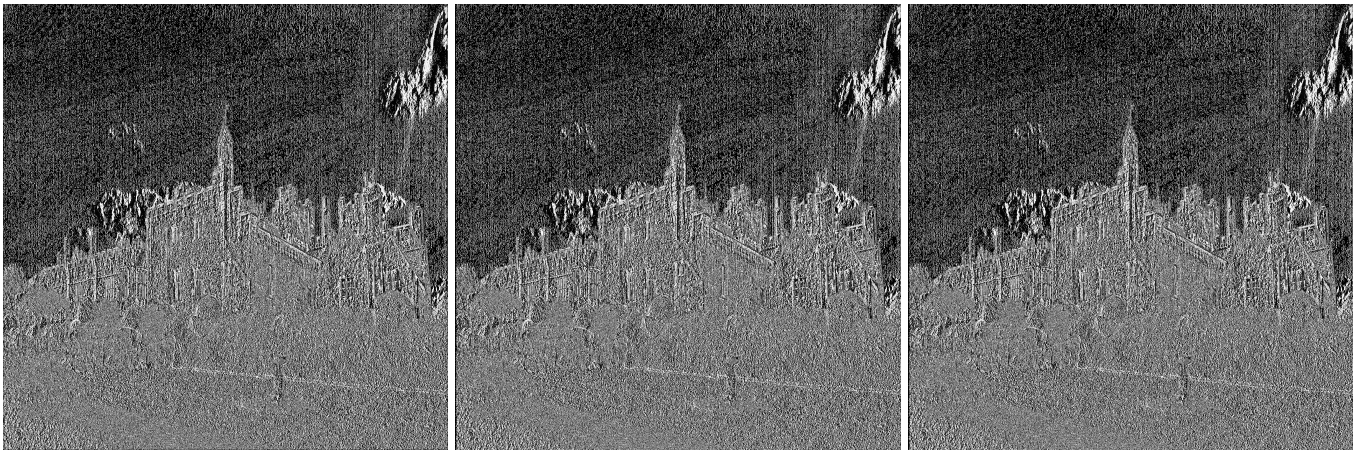

Figure 18: Sequential (left), Global (centre), Global Constant (right)

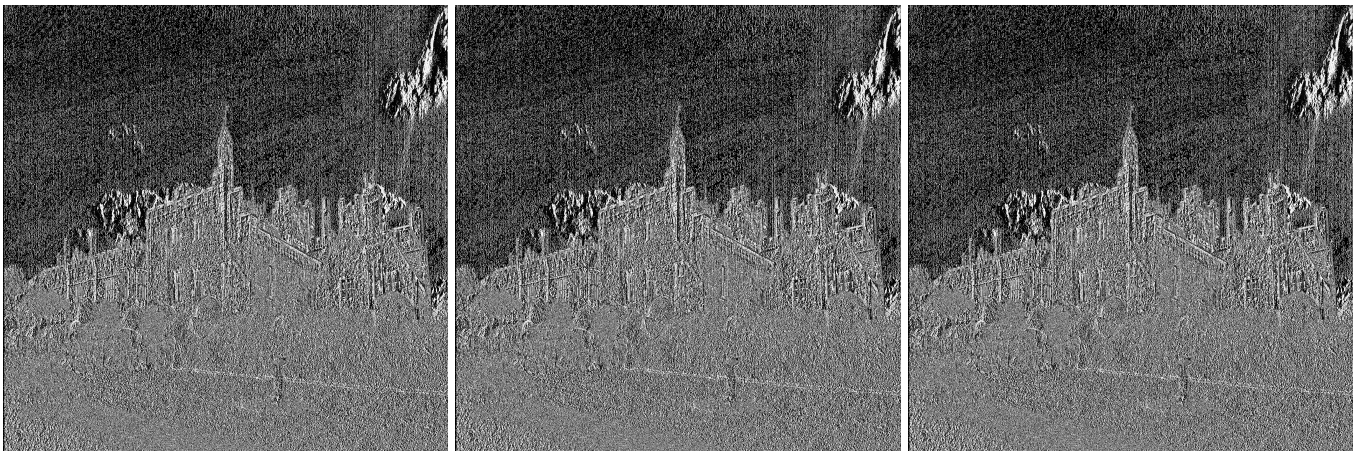

Figure 19: Shared (left), Shared Constant (centre), Texture (right)

6.4.2 Blur 3x3

Figure 20: Sequential (left), Global (centre), Global Constant (right)

Figure 21: Shared (left), Shared Constant (centre), Texture (right)

6.4.3 Blur 5x5

Figure 22: Sequential (left), Global (centre), Global Constant (right)

Figure 23: Shared (left), Shared Constant (centre), Texture (right)

6.4.4 Blur 7x7

Figure 24: Sequential (left), Global (centre), Global Constant (right)

Figure 25: Shared (left), Shared Constant (centre), Texture (right)

6.5 Graphs

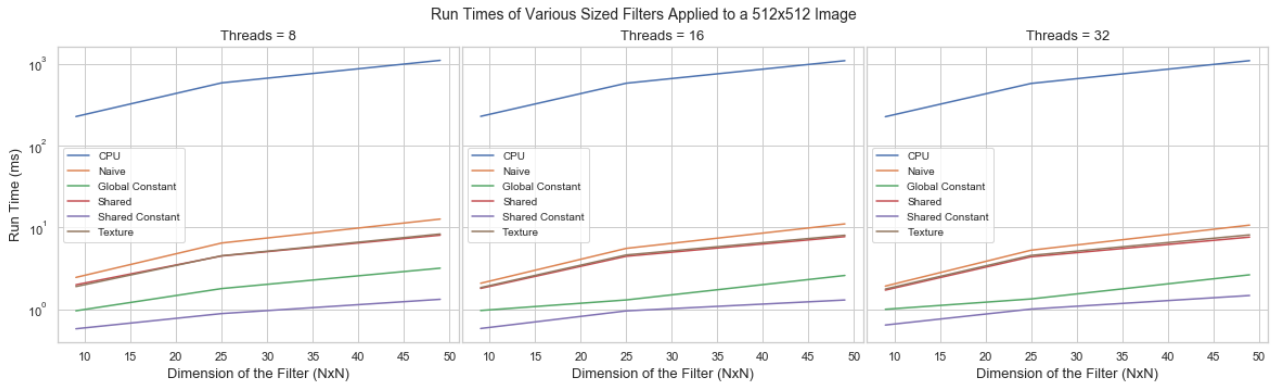

Figure 26: Filter Size vs Run Time of Lena 512x512

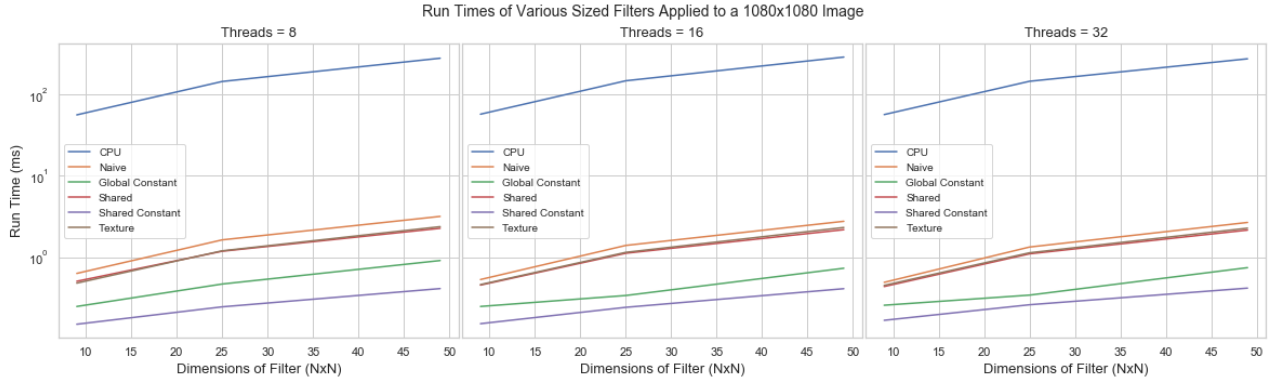

Figure 27: Filter Size vs Run Time of Harvard 1080x1080

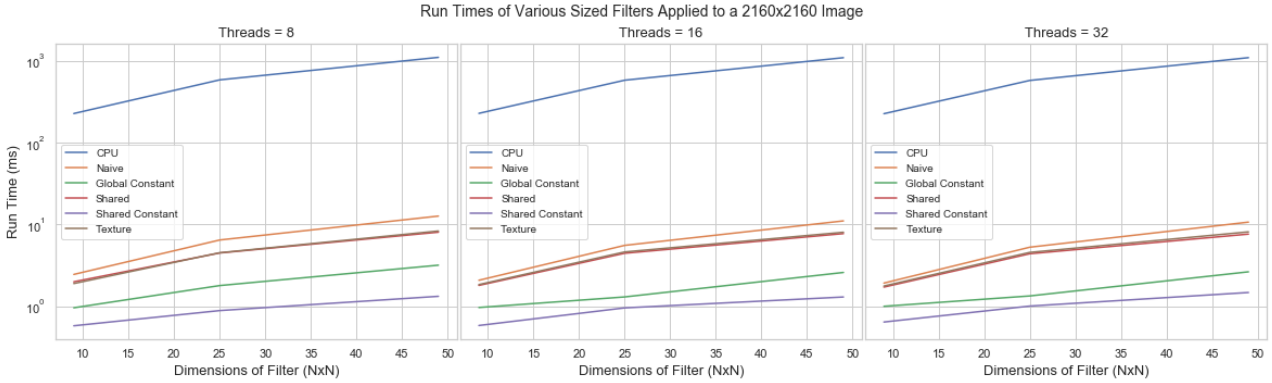

Figure 28: Filter Size vs Run Time of Harvard 2160x2160

6.6 Data

Thread	Filter	Type	Run Time
8	Edge 3x3	CPU	12.471398 ms
8	Edge 3x3	GPU(Global)	0.150122 ms
8	Edge 3x3	GPU(Global Constant)	0.066250 ms
8	Edge 3x3	GPU(Share) Tile Size = 10	0.122061 ms
8	Edge 3x3	GPU(Shared Constant) Tile Size = 10	0.042390 ms
8	Edge 3x3	GPU(Texture)	0.119088 ms
8	Blur 3x3	CPU	14.080716 ms
8	Blur 3x3	GPU(Global)	0.152883 ms
8	Blur 3x3	GPU(Global Constant)	0.063072 ms
8	Blur 3x3	GPU(Share) Tile Size = 10	0.121034 ms
8	Blur 3x3	GPU(Shared Constant) Tile Size = 10	0.041267 ms
8	Blur 3x3	GPU(Texture)	0.117651 ms
8	Blur 5x5	CPU	33.458992 ms
8	Blur 5x5	GPU(Global)	0.385536 ms
8	Blur 5x5	GPU(Global Constant)	0.113152 ms
8	Blur 5x5	GPU(Share) Tile Size = 12	0.276582 ms
8	Blur 5x5	GPU(Shared Constant) Tile Size = 12	0.062877 ms
8	Blur 5x5	GPU(Texture)	0.281392 ms
8	Blur 7x7	CPU	62.382488 ms
8	Blur 7x7	GPU(Global)	0.736256 ms
8	Blur 7x7	GPU(Global Constant)	0.210330 ms
8	Blur 7x7	GPU(Share) Tile Size = 14	0.515994 ms
8	Blur 7x7	GPU(Shared Constant) Tile Size = 14	0.101478 ms
8	Blur 7x7	GPU(Texture)	0.543434 ms
16	Edge 3x3	CPU	12.833796 ms
16	Edge 3x3	GPU(Global)	0.128413 ms
16	Edge 3x3	GPU(Global Constant)	0.063795 ms
16	Edge 3x3	GPU(Share) Tile Size = 18	0.111206 ms
16	Edge 3x3	GPU(Shared Constant) Tile Size = 18	0.043312 ms
16	Edge 3x3	GPU(Texture)	0.117043 ms
16	Blur 3x3	CPU	13.048215 ms
16	Blur 3x3	GPU(Global)	0.128512 ms
16	Blur 3x3	GPU(Global Constant)	0.064410 ms
16	Blur 3x3	GPU(Share) Tile Size = 18	0.111510 ms
16	Blur 3x3	GPU(Shared Constant) Tile Size = 18	0.043622 ms
16	Blur 3x3	GPU(Texture)	0.116736 ms
16	Blur 5x5	CPU	32.817047 ms
16	Blur 5x5	GPU(Global)	0.336589 ms
16	Blur 5x5	GPU(Global Constant)	0.087859 ms
16	Blur 5x5	GPU(Share) Tile Size = 20	0.270643 ms
16	Blur 5x5	GPU(Shared Constant) Tile Size = 20	0.065024 ms
16	Blur 5x5	GPU(Texture)	0.282931 ms
16	Blur 7x7	CPU	63.186840 ms
16	Blur 7x7	GPU(Global)	0.659661 ms
16	Blur 7x7	GPU(Global Constant)	0.182784 ms
16	Blur 7x7	GPU(Share) Tile Size = 22	0.515891 ms
16	Blur 7x7	GPU(Shared Constant) Tile Size = 22	0.104342 ms
16	Blur 7x7	GPU(Texture)	0.549376 ms
32	Edge 3x3	CPU	12.509799 ms
32	Edge 3x3	GPU(Global)	0.126051 ms
32	Edge 3x3	GPU(Global Constant)	0.070246 ms
32	Edge 3x3	GPU(Share) Tile Size = 34	0.115405 ms
32	Edge 3x3	GPU(Shared Constant) Tile Size = 34	0.048938 ms
32	Edge 3x3	GPU(Texture)	0.124515 ms
32	Blur 3x3	CPU	12.551270 ms
32	Blur 3x3	GPU(Global)	0.128918 ms
32	Blur 3x3	GPU(Global Constant)	0.073114 ms
32	Blur 3x3	GPU(Share) Tile Size = 34	0.114989 ms
32	Blur 3x3	GPU(Shared Constant) Tile Size = 34	0.048531 ms
32	Blur 3x3	GPU(Texture)	0.120422 ms
32	Blur 5x5	CPU	32.888931 ms
32	Blur 5x5	GPU(Global)	0.323997 ms
32	Blur 5x5	GPU(Global Constant)	0.092774 ms
32	Blur 5x5	GPU(Share) Tile Size = 36	0.282829 ms
32	Blur 5x5	GPU(Shared Constant) Tile Size = 36	0.073418 ms
32	Blur 5x5	GPU(Texture)	0.298803 ms
32	Blur 7x7	CPU	62.553387 ms
32	Blur 7x7	GPU(Global)	0.642662 ms
32	Blur 7x7	GPU(Global Constant)	0.195683 ms
32	Blur 7x7	GPU(Share) Tile Size = 38	0.542922 ms
32	Blur 7x7	GPU(Shared Constant) Tile Size = 38	0.114688 ms
32	Blur 7x7	GPU(Texture)	0.580096 ms

Figure 29: Data of Lena 512x512

Thread	Filter	Type	Run Time
8	Edge 3x3	CPU	56.453632 ms
8	Edge 3x3	GPU(Global)	0.624230 ms
8	Edge 3x3	GPU(Global Constant)	0.247296 ms
8	Edge 3x3	GPU(Share) Tile Size = 10	0.508925 ms
8	Edge 3x3	GPU(Shared Constant) Tile Size = 10	0.150934 ms
8	Edge 3x3	GPU(Texture)	0.478208 ms
8	Blur 3x3	CPU	56.072289 ms
8	Blur 3x3	GPU(Global)	0.634672 ms
8	Blur 3x3	GPU(Global Constant)	0.249446 ms
8	Blur 3x3	GPU(Share) Tile Size = 10	0.508422 ms
8	Blur 3x3	GPU(Shared Constant) Tile Size = 10	0.150938 ms
8	Blur 3x3	GPU(Texture)	0.482506 ms
8	Blur 5x5	CPU	144.078949 ms
8	Blur 5x5	GPU(Global)	1.638093 ms
8	Blur 5x5	GPU(Global Constant)	0.470118 ms
8	Blur 5x5	GPU(Share) Tile Size = 12	1.188042 ms
8	Blur 5x5	GPU(Shared Constant) Tile Size = 12	0.246784 ms
8	Blur 5x5	GPU(Texture)	1.202170 ms
8	Blur 7x7	CPU	277.993256 ms
8	Blur 7x7	GPU(Global)	3.180237 ms
8	Blur 7x7	GPU(Global Constant)	0.913306 ms
8	Blur 7x7	GPU(Share) Tile Size = 14	2.271846 ms
8	Blur 7x7	GPU(Shared Constant) Tile Size = 14	0.413389 ms
8	Blur 7x7	GPU(Texture)	2.379059 ms
16	Edge 3x3	CPU	57.671577 ms
16	Edge 3x3	GPU(Global)	0.529606 ms
16	Edge 3x3	GPU(Global Constant)	0.249242 ms
16	Edge 3x3	GPU(Share) Tile Size = 18	0.455066 ms
16	Edge 3x3	GPU(Shared Constant) Tile Size = 18	0.150627 ms
16	Edge 3x3	GPU(Texture)	0.461722 ms
16	Blur 3x3	CPU	57.140015 ms
16	Blur 3x3	GPU(Global)	0.533914 ms
16	Blur 3x3	GPU(Global Constant)	0.249450 ms
16	Blur 3x3	GPU(Share) Tile Size = 18	0.456294 ms
16	Blur 3x3	GPU(Shared Constant) Tile Size = 18	0.153290 ms
16	Blur 3x3	GPU(Texture)	0.462848 ms
16	Blur 5x5	CPU	147.084183 ms
16	Blur 5x5	GPU(Global)	1.400218 ms
16	Blur 5x5	GPU(Global Constant)	0.340070 ms
16	Blur 5x5	GPU(Share) Tile Size = 20	1.122918 ms
16	Blur 5x5	GPU(Shared Constant) Tile Size = 20	0.243507 ms
16	Blur 5x5	GPU(Texture)	1.151898 ms
16	Blur 7x7	CPU	287.285645 ms
16	Blur 7x7	GPU(Global)	2.770227 ms
16	Blur 7x7	GPU(Global Constant)	0.734618 ms
16	Blur 7x7	GPU(Share) Tile Size = 22	2.185113 ms
16	Blur 7x7	GPU(Shared Constant) Tile Size = 22	0.412262 ms
16	Blur 7x7	GPU(Texture)	2.324787 ms
32	Edge 3x3	CPU	58.717285 ms
32	Edge 3x3	GPU(Global)	0.496534 ms
32	Edge 3x3	GPU(Global Constant)	0.260301 ms
32	Edge 3x3	GPU(Share) Tile Size = 34	0.437760 ms
32	Edge 3x3	GPU(Shared Constant) Tile Size = 34	0.168243 ms
32	Edge 3x3	GPU(Texture)	0.452915 ms
32	Blur 3x3	CPU	56.529102 ms
32	Blur 3x3	GPU(Global)	0.492851 ms
32	Blur 3x3	GPU(Global Constant)	0.258256 ms
32	Blur 3x3	GPU(Share) Tile Size = 34	0.438173 ms
32	Blur 3x3	GPU(Shared Constant) Tile Size = 34	0.168653 ms
32	Blur 3x3	GPU(Texture)	0.452403 ms
32	Blur 5x5	CPU	144.911346 ms
32	Blur 5x5	GPU(Global)	1.337344 ms
32	Blur 5x5	GPU(Global Constant)	0.343757 ms
32	Blur 5x5	GPU(Share) Tile Size = 36	1.104384 ms
32	Blur 5x5	GPU(Shared Constant) Tile Size = 36	0.261837 ms
32	Blur 5x5	GPU(Texture)	1.137050 ms
32	Blur 7x7	CPU	273.537537 ms
32	Blur 7x7	GPU(Global)	2.681136 ms
32	Blur 7x7	GPU(Global Constant)	0.746899 ms
32	Blur 7x7	GPU(Share) Tile Size = 38	2.159719 ms
32	Blur 7x7	GPU(Shared Constant) Tile Size = 38	0.418816 ms
32	Blur 7x7	GPU(Texture)	2.279421 ms

Figure 30: Data of Harvard 1080x1080

Thread	Filter	Type	Run Time
8	Edge 3x3	CPU	227.593628 ms
8	Edge 3x3	GPU(Global)	2.458931 ms
8	Edge 3x3	GPU(Global Constant)	0.958877 ms
8	Edge 3x3	GPU(Share) Tile Size = 10	2.004787 ms
8	Edge 3x3	GPU(Shared Constant) Tile Size = 10	0.579584 ms
8	Edge 3x3	GPU(Texture)	1.890816 ms
8	Blur 3x3	CPU	226.883179 ms
8	Blur 3x3	GPU(Global)	2.460775 ms
8	Blur 3x3	GPU(Global Constant)	0.962662 ms
8	Blur 3x3	GPU(Share) Tile Size = 10	1.999977 ms
8	Blur 3x3	GPU(Shared Constant) Tile Size = 10	0.580403 ms
8	Blur 3x3	GPU(Texture)	1.903206 ms
8	Blur 5x5	CPU	584.276184 ms
8	Blur 5x5	GPU(Global)	6.474748 ms
8	Blur 5x5	GPU(Global Constant)	1.797325 ms
8	Blur 5x5	GPU(Share) Tile Size = 12	4.496691 ms
8	Blur 5x5	GPU(Shared Constant) Tile Size = 12	0.889754 ms
8	Blur 5x5	GPU(Texture)	4.523620 ms
8	Blur 7x7	CPU	1100.611328 ms
8	Blur 7x7	GPU(Global)	12.688589 ms
8	Blur 7x7	GPU(Global Constant)	3.190272 ms
8	Blur 7x7	GPU(Share) Tile Size = 14	8.068505 ms
8	Blur 7x7	GPU(Shared Constant) Tile Size = 14	1.329046 ms
8	Blur 7x7	GPU(Texture)	8.337919 ms
16	Edge 3x3	CPU	225.497711 ms
16	Edge 3x3	GPU(Global)	2.090496 ms
16	Edge 3x3	GPU(Global Constant)	0.972896 ms
16	Edge 3x3	GPU(Share) Tile Size = 18	1.806951 ms
16	Edge 3x3	GPU(Shared Constant) Tile Size = 18	0.584598 ms
16	Edge 3x3	GPU(Texture)	1.841968 ms
16	Blur 3x3	CPU	227.951614 ms
16	Blur 3x3	GPU(Global)	2.085785 ms
16	Blur 3x3	GPU(Global Constant)	0.970342 ms
16	Blur 3x3	GPU(Share) Tile Size = 18	1.806131 ms
16	Blur 3x3	GPU(Shared Constant) Tile Size = 18	0.585110 ms
16	Blur 3x3	GPU(Texture)	1.847402 ms
16	Blur 5x5	CPU	578.791382 ms
16	Blur 5x5	GPU(Global)	5.566768 ms
16	Blur 5x5	GPU(Global Constant)	1.306109 ms
16	Blur 5x5	GPU(Share) Tile Size = 20	4.461568 ms
16	Blur 5x5	GPU(Shared Constant) Tile Size = 20	0.959693 ms
16	Blur 5x5	GPU(Texture)	4.628480 ms
16	Blur 7x7	CPU	1090.283813 ms
16	Blur 7x7	GPU(Global)	11.079376 ms
16	Blur 7x7	GPU(Global Constant)	2.599721 ms
16	Blur 7x7	GPU(Share) Tile Size = 22	7.747476 ms
16	Blur 7x7	GPU(Shared Constant) Tile Size = 22	1.303757 ms
16	Blur 7x7	GPU(Texture)	8.045978 ms
32	Edge 3x3	CPU	229.637238 ms
32	Edge 3x3	GPU(Global)	1.924506 ms
32	Edge 3x3	GPU(Global Constant)	1.003318 ms
32	Edge 3x3	GPU(Share) Tile Size = 34	1.722368 ms
32	Edge 3x3	GPU(Shared Constant) Tile Size = 34	0.643686 ms
32	Edge 3x3	GPU(Texture)	1.776336 ms
32	Blur 3x3	CPU	225.741302 ms
32	Blur 3x3	GPU(Global)	1.926656 ms
32	Blur 3x3	GPU(Global Constant)	1.003721 ms
32	Blur 3x3	GPU(Share) Tile Size = 34	1.722368 ms
32	Blur 3x3	GPU(Shared Constant) Tile Size = 34	0.644813 ms
32	Blur 3x3	GPU(Texture)	1.774592 ms
32	Blur 5x5	CPU	574.954468 ms
32	Blur 5x5	GPU(Global)	5.285376 ms
32	Blur 5x5	GPU(Global Constant)	1.342566 ms
32	Blur 5x5	GPU(Share) Tile Size = 36	4.399616 ms
32	Blur 5x5	GPU(Shared Constant) Tile Size = 36	1.009968 ms
32	Blur 5x5	GPU(Texture)	4.568371 ms
32	Blur 7x7	CPU	1091.225586 ms
32	Blur 7x7	GPU(Global)	10.706226 ms
32	Blur 7x7	GPU(Global Constant)	2.646838 ms
32	Blur 7x7	GPU(Share) Tile Size = 38	7.640780 ms
32	Blur 7x7	GPU(Shared Constant) Tile Size = 38	1.479373 ms
32	Blur 7x7	GPU(Texture)	8.139363 ms

Figure 31: Data of Harvard 2160x2160

References

[NVIDIA 2018] NVIDIA. *CUDA C PROGRAMMING GUIDE*, 2018. https://docs.nvidia.com/cuda/archive/9.1/pdf/CUDA_C_Programming_Guide.pdf.