Gaussian Filter Accelerator

EE5332 - Course Project

Arun D A (EE19B071), Sakthi Harish (EE19B054)

December 2021

Git Repository: https://github.com/Sakthi-Harish/Gaussian_filter
Video Presentation:
https://drive.google.com/drive/folders/1IheSvjDFP9_QrOqrwVA8WA-xIBAJmYw6

1 Introduction

Gaussian filtering is a smoothing filter used in various tasks to reduce the Gaussian noise and enhance the image quality. One of the main drawbacks of the filter is its $O(R^2)$ dependency on its kernel radius and we have used techniques like Sliding-DCT based Gaussian filtering, separable filtering, FFT to optimize the computation.

2 Project Description

For this project, we have initially implemented a naive Gaussian filtering algorithm based on 2D convolution and measure the performance on the CPU and GPU, and we have also implemented it on HLS for the Artix7 board to measure the performance increase as we optimize it. We first try unrolling the inner loops in the computation and it was discovered that due to the large number of computations inside the loop, unrolling more than 10 iterations would consume a large amount of resources on the FPGA. We have also written a python script that converts any given image into a grey scale image of size 480*360 and prints it as a 2D matrix to a text file and another script to read a 2D matrix from a text file and display the output image. These were used for verifying the correctness of our code and also used to measure the accuracy of the algorithm by comparing the output to the inbuilt python function.

$$
\begin{aligned}
G_\sigma(x,y) &= \frac{1}{2\pi\sigma^2}\exp^{-\frac{x^2+y^2}{2\sigma^2}} \\
&= \left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{x^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{y^2}{2\sigma^2}}\right)
\end{aligned}
$$

Figure 1: Separable Gaussian Filter

Proceeding this we have implemented a separable Gaussian filter and have again measured the time taken on CPU, GPU and latency reported by HLS. It was observed that due to reduction in the complexity of computation in the inner loops we can unroll the loops by a larger factor. We have then optimized the code for GPU computation making use of shared memory.

To further optimize the HLS implementation, we have used unsigned fixed point instead of floats for the computations. Though this step didn't provide a significant speedup it reduced the usage of LUTs and FFs by a factor of 11 and this allowed us to unroll the loops by an even larger factor. Upon using the scheduling viewer it was observed that the majority of the clock cycles were used in reading and writing data to the arrays stored in BRAMs. To optimize this we have decided to partition and reshape the arrays in a cyclic type by a factor greater than the unrolling factor to allow for parallel access to the adjacent elements, this provided a large speedup. Again visiting the schedule viewer it was observed that the bottleneck in the computation was now the read and store operations to the main memory, this

can't be optimized further unless multiple ports are available for the main memory. The DCT3 method was used to implement the sliding DCT algorithm. The FPGA was designed to run at a clock speed of 10ns. The CPU used is Intel i5 9300H and the GPU used is NVIDIA GTX 1050.

3 Results

Computation unit	**Total Time**
CPU unoptimized	105838940 ns
CPU separable	20408333 ns
CPU Sliding DCT	28246660 ns
GPU unoptimized	1405246 ns
GPU separable	233577 ns
GPU separable(optimized for shared memory)	182645 ns
FPGA unoptimized	1252800110 ns
FPGA optimized	1028678760 ns
FPGA unoptimized separable	160704900 ns
FPGA optimized separable with fixed points	864460 ns
MATLAB convolution based Gaussian filter	6046000ns
MATLAB FFT based Gaussian filter	13792000ns

Table 1: Performance of various algorithms on different devices

It can be seen that moving from the unoptimized CPU version to the separable filter gives a speedup of around 5, whereas it gives a speedup of 6 in the GPU, this is due to fact that latency of the GPU memory is more than that of the CPU memory and shifting to a 1D array from a 2D for the kernel provides some additional speedup. In the FPGA it provides a speedup of around 8, this is due to reduction in computations in the inner loop and HLS can optimize the code better because of that. It can also be seen that there is a speedup of around 19x in the FPGA implementation on using fixed points and partitioning the arrays. This indicates the huge amount of resources being used for floating point operations and the latency caused by the memory reads and writes.

It is observed that the sliding-DCT implementation causes a decrease in accuracy of the results due to the algorithm transforming the image and Gaussian kernel into a sum of cosines, since the range of frequencies can't be infinite in a real world scenario some amount of data is lost. The accuracy can be improved by using a larger number of frequencies but this results in larger memory usage and latency. It can also be observed that the sliding-DCT algorithm results in a slowdown, this is due to its time complexity of 6K compared to a time complexity of 2R for the separable Gaussian filter. This can be attributed to the fact the the approximation step in equation 6 of the $2^n d$ reference paper gives rise to the speedup claimed by the paper. Since we have used a kernel of radius 2, we will need to set K = R to get an output with an acceptable accuracy.

4 Conclusion

As is evident from the results, the bottleneck in the hardware implementation is caused by the read and write operations from and to the main memory. A speedup of 23.6x was achieved when compared to code executed on a modern CPU, but the FPGA implementation was 4.7x slower when compared to code executed on a modern GPU. This makes the hardware implementation useful in low power image processing applications since it consumes much lower power compared to a GPU but gives a comparable speed. The implementation was also verified by running the C/RTL co-simulation. The sliding-DCT algorithm doesn't give any speedup for smaller kernel sizes and since the output produced by a small Gaussian kernel is enough for most applications, the sliding-DCT algorithm cannot be used here, but it is extremely useful in convolution based algorithms using a separable kernel where the kernel radius is large. It is also observed that a convolution based filter is faster than a FFT based one for small kernel sizes.

5 Contributions

I was responsible for writing the python scripts for reading the images and displaying the output, developing the FPGA implementation and optimizing it using various optimization directives, developing the CUDA code for various algorithms, implementing the sliding DCT algorithm and partly responsible for implementing the separable CPU code

6 References

1. https://www.researchgate.net/publication/305820879_FPGA_implementation_of_filtered_image_using_2D_Gaussian_filter
2. https://ieeexplore.ieee.org/document/9301775