

**KAUNAS UNIVERSITY OF TECHNOLOGY**  
**Faculty of Informatics**  
**Department of Information Systems**

## **Image Processing – Application of data Concurrency Tools**

December 21, 2021

Student: Leo Onyema Sochi Muoghara

Teacher: Karolis Ryselis

Kaunas, 2021

## Table of Contents

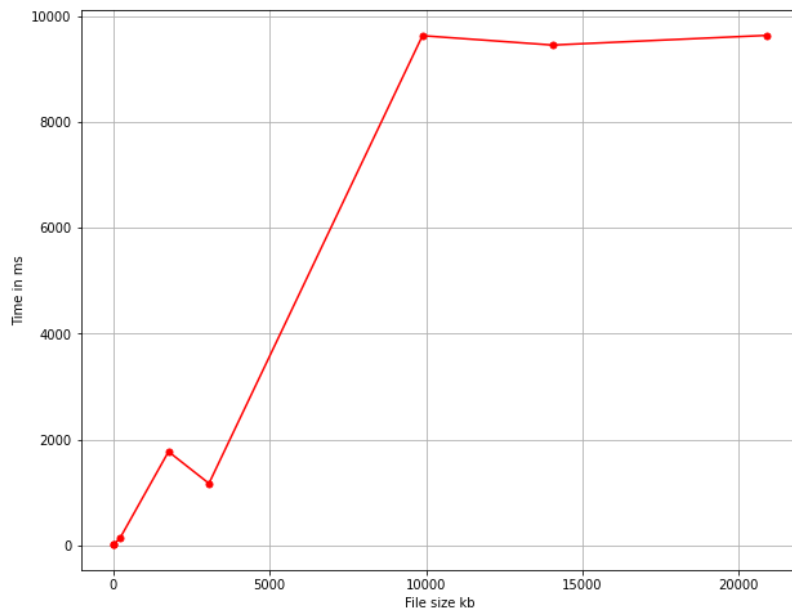
1. Task analysis and solution.....	3
1.1. Problem Description.....	3
1.2. Problem Solution and Tool Implementation .....	4
2. Testing and instructions.....	5
2.1. Program Results .....	5
2.2. Program Run Instruction .....	6
3. Performance analysis.....	7
Conclusions .....	9

# 1. Task analysis and solution

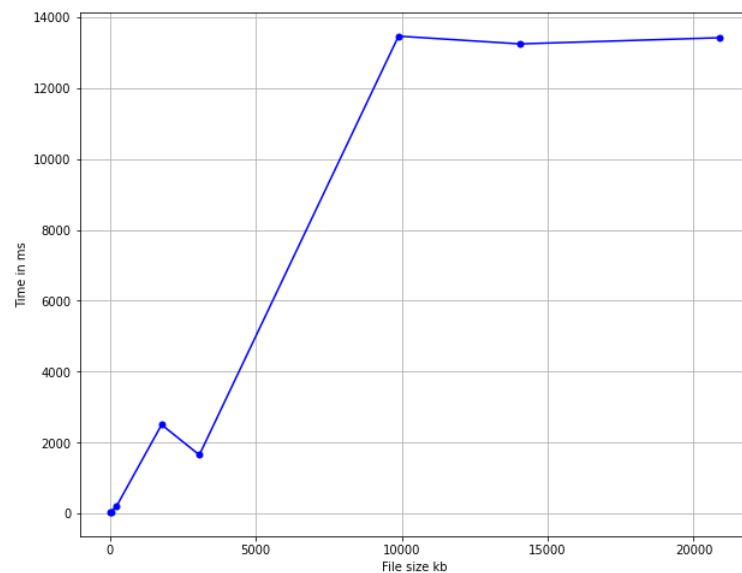
## 1.1. Problem Description

A control test was done to transform 8 digital images by applying the Gaussian Blur. The control test was performed in a single thread on the CPU with no concurrent tool running.

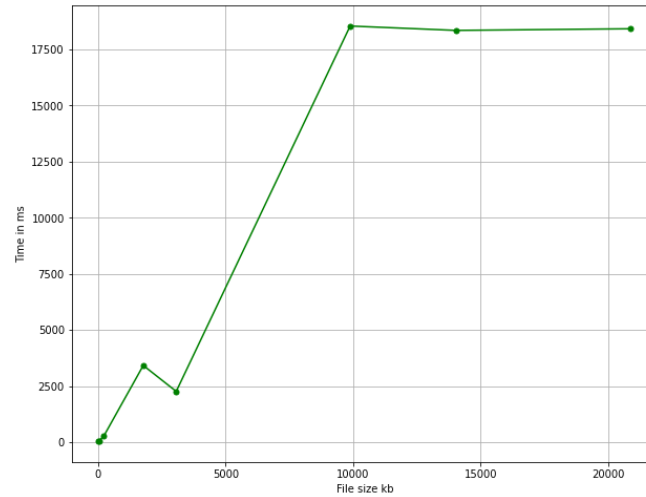
The following results were obtained.



**Fig 1 Time Taken in ms to Apply Gaussian Blur of Kernel size 3x3 Images of different file Sizes in kb**



**Fig 2 Time Taken in ms to Apply Gaussian Blur of Kernel size 5x5 Images of different file Sizes in kb**



**Fig 3 Time Taken in ms to Apply Gausssian Blur of Kernel size 7x7 Images of different file Sizes in kb**

**Table 1 Result of a Single Thread applying Gaussian Blur of Varied sizes**

File Size (kb)	Kernel 3x3 (milliseconds)	5x5 (milliseconds)	7x7 (milliseconds)
10	25	35	50
30	33	46	62
215	137	189	263
1,770	1775	2503	3422
3,063	1175	1650	2266
9,887	9625	13462	18544
14,045	9448	13243	18349
20,900	9631	13417	18422

We can see that the trend in all three graphs is somewhat consistent. In the **Table.1**, we see that the time taken increases as the kernel size increases because the number of operation increases.

The goal of this project is to utilize CUDA (c++) concurrent tool to reduce the time required to perform these transformations.

## 1.2. Problem Solution and Tool Implementation

Using CUDA, I was able to take an x number of blocks and assign a particular amount of data to it to process, as well as allocate the number of channels to threads, resulting in the algorithm just having to execute once rather than three times, as the number of supplied channels requires. Furthermore, allocating memory for the output Image allows the program to save memory by not creating an additional temporary array that will hold the processed pixels until all pixels have been processed; this is because when performing Gaussian blur, the original image being processed must remain unchanged until the processing is complete, after which the temporary array will overwrite the original image. This also saves runtime because the copying is not performed. This theoretically should reduce the time required to perform these transformations.

## 2. Testing and instructions

### 2.1. Program Results



**Fig 4. Original Image**

Using the single thread CPU with a 7x7 Gaussian kernel, we get Fig 5.



**Fig 5. Single Thread CPU with a 7x7 Gaussian Kernel**


Using CUDA concurrency tool with a 7x7 Gaussian Kernel, we get Fig 6.



**Fig 6. CUDA Concurrency Tool with a 7x7 Gaussian Kernel**

As one can observe, both Fig 5. and Fig 6. Images are the same.

## 2.2. Program Run Instruction

1. To run the program, you must first select the kernel you want to use by entering it to **cudaMalloc** and **cudaMemcpy**:
  - a. 7x7 – ker7x7
  - b. 5x5 – ker5x5
  - c. 3x3 – ker3x3
2. Select which image which you want to use by entering it to **stbi\_load**, **cudaMalloc** and **cudaMemcpy**: **img1.jpg** – **img8.jpg**.
3. Click on Debug  Start Without Debugging.

### 3. Performance analysis

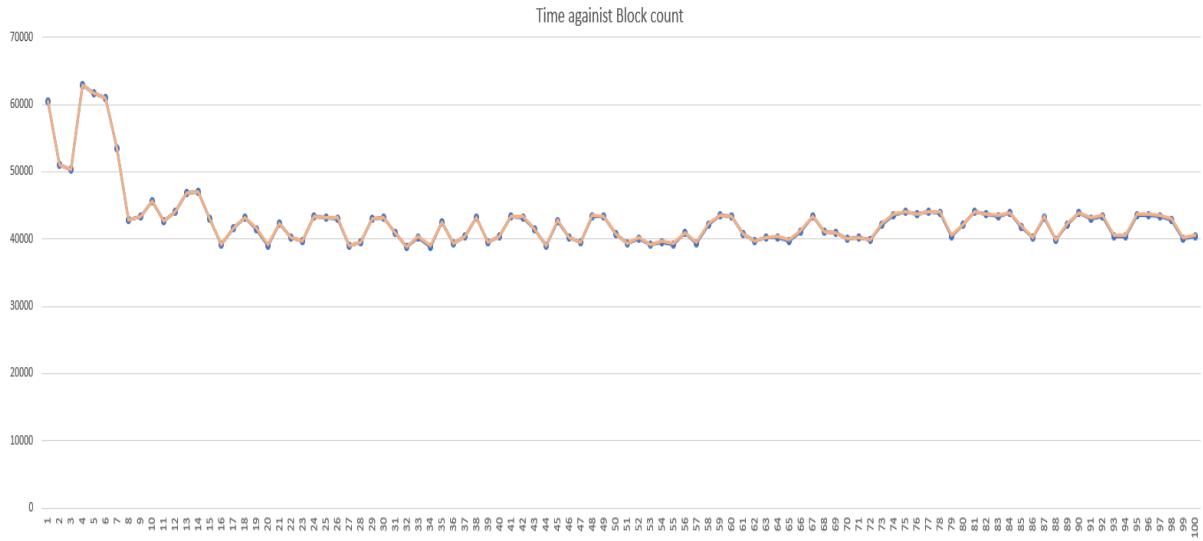


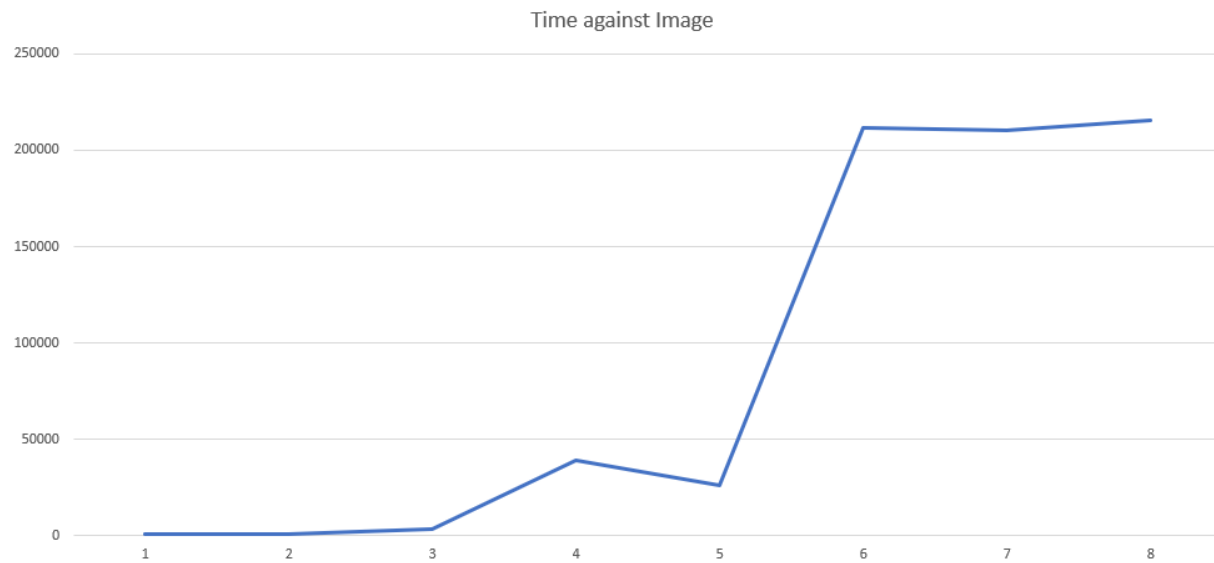
Fig 7. Graph of Time against number of Block



Fig 8. Image 4 in the Dataset

The Fig 7. above depicts a graph in which the x-axis represents the number of blocks, and the y-axis represents the time taken. The analysis was performed on the fourth picture in the dataset (out of eight photographs) (this was chosen because the smallest image is img1.jpg[10kb] and the largest is img8.jpg [20.9mb]). The time and block count were collected in excel and using the excel formula Min. I found the minimum time taken was **38821 milliseconds**, which was obtained by **32** blocks out of 100. Because of the **WDDM TDR**, which detects response problems from a graphics card and recovers to a functional desktop by resetting the card, 100 blocks were chosen as the maximum and not the height of the image (*critical point of the algorithm is to divide the height of the image by the number of blocks to get the number of rows each block will have to calculate, therefore, if the number of blocks is larger than the height then we get an invalid operation*).

Computing the average time from the data we got to graph Fig 7., using excel average formular, we get **42875 milliseconds**. And, getting the average of the time taken using **32 blocks**, we get **40170 milliseconds**.



**Fig 9. Time against Image**

File Size (kb)	Kernel 7x7 (milliseconds)
10	784
30	716
215	2985
1,770	38781
3,063	26016
9,887	211719
14,045	210451
20,900	215445

Here we can see how time taken is dependent on the size of the image. This was performed using 32 blocks. The results are expected because as the image size increases, so does the number of pixels to process.



## Conclusions

All images were successfully transformed using both single and multi-thread processes. Some problems to overcome where:

1. finding the best way to divide the data according to the number of threads or blocks.
2. Finding the maximum size of an array inside the local CUDA kernel.
3. If to processes the image block-wise or thread-wise.

Having a larger runtime for processing the images using CUDA than the single thread runtime was unexpected but could be due to:

- Poorly implement gaussian blur algorithm or
- Putting into out, allocating memory to the GPU, copying the data to GPU and copying the results back to the CPU.

But, with further investigation and a more optimised algorithm, a better result can be obtained.

CUDA concurrency is (in my opinion) the better tool to use in this problem because of the computational brute force it provides.