Image Steganography in CUDA

Chingyuan Yang  
*Computer Science*  
*Clemson University*Clemson, SC, USA  
Chingyy@g.clemson.edu

# Introduction

The image processing is a computational-intensive problem domain. Although CPU can also perform high computational power, the GPU can achieve the same goal at less execution time and light energy-consuming. GPU is a SIMD architecture and better for data parallelism. Therefore, to process image pixels is more suitable for the GPU instructions compare to CPU.

While I was searching for the idea of image processing using GPU, I found different image filters, such as grayscale, blurring, etc. In addition, I found the image steganography. I think it is an interesting topic. The image filter is a very common and widely used function build on many programs. Therefore, I choose image steganography for my main project idea. In this program domain, many algorithms have been proposed; for instance, the blindhid, hide seek, filter first, and battle steg[1].

In my project, I choose the blindhid algorithm for steganography. The reason is that this algorithm is more straightforward for GPU. There would not have too many judgment in the kernel code, such as if-else. Etc.

# Design Method

The blindhid algorithm starts at the top left corner of the image and works the image pixel by pixel[1]. Every pixel has red, green, blue three values. After the program converts them to binary, it has eight bits. We will choose the last bit of the RGB value to hide our secret message. Even we change the last bit. The image still looks the same from human eyes. So we say this is the least significant bits (LSB). Furthermore, each character also can convert to eight bits, in other words. One character needs at least 3 pixels to store it eight bits.

To decode the secret message hiding in the image, we also start from the same order and take the least significant bits from each pixel’s RGB value. Hence, we can combine every eight-bit to one character until we have eight bits zero, which means ‘\0’ the end of the file.

The CPU version of image steganography is developed by hitanshu-dhawan[2]. My GPU version of the program is based on his code to modify. I also reference the OpenCL code from Ghazanfar Abbas[3]. This code helps me to use an unsigned char array to store the OpenCV Mat data structure. So I can use threads to manipulate the RGB value easily.

# Implementation

First of all, the program loads the image and secret message text file to the buffer first. We use OpenCV to process image data. Secondly, we copy both data to the device memory. The GPU kernel starts to use bitwise operations to put the character’s bit to an RGB value. Each thread works on one bit, and every eight threads work on a character. After all the character has been finished, we will copy the image back to the host memory. Finally, output the image with a secret message and kernel execution time.

From the decoding point of view, the program also uses OpenCV to load images. Then it copies data and empty char array to the GPU device memory. The GPU’s each thread capture the last significant bits on every pixel RGB value and store them to the char array. After the kernel scan the whole image, the char array copy back to the host. The CPU converts the binary string to the string and prints out on the screen.

# Experiment

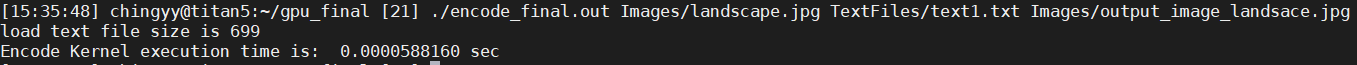
## Experiment environment

* System: Ubuntu 4.4.0-109-generic
* OpenCV: 2.4.9.1
* Cuda: V8.0.61
* Profile image: 400 H \* 309 W
* Landscape image: 768 H \* 1024 W
* Two text files: Text1.txt is 699 characters. Text2.txt is 540 characters. Both files are included null characters and newline characters.

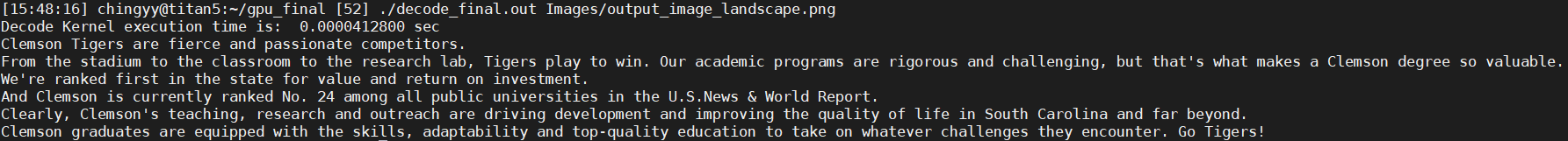
## Experiment result

From section IV A, I use two different sizes of images and two different text files to test my program. Also, I compare the CPU execution time [2] with GPU kernel execution time. However, I did not count the GPU memory copy time.

Figure 1 and figure 2 shows the actual output looks like for the GPU encoding and GPU decoding. When you encode the message, the output is an image and Encode Kernel execution time. When you decode the message, the output is decoded Kernel execution time and the hidden message that displayed on your console.



1. GPU encoding for text1.txt and landscape image.

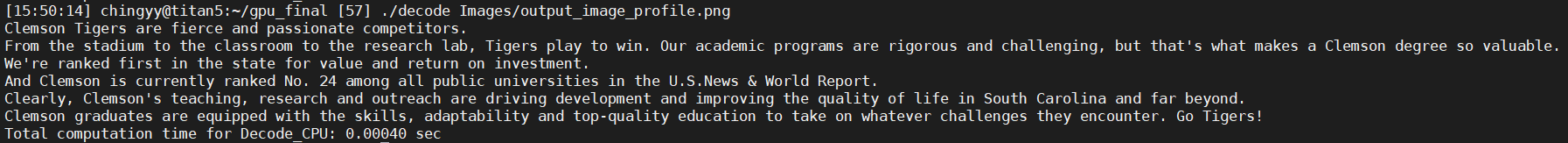


1. GPU decoding for text1.txt and landscape image.

Figure 3 and figure 4 shows the actual output looks like for the CPU encoding and CPU decoding. When you encode the message, the output is an image and Encode CPU computation time. When you decode the message, the output is decoded CPU computation time and the hidden message that displayed on your console.

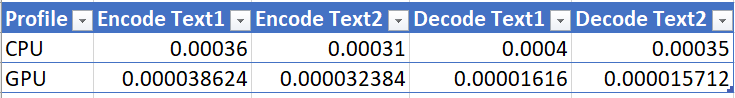


1. CPU encoding for text1.txt and profile image.

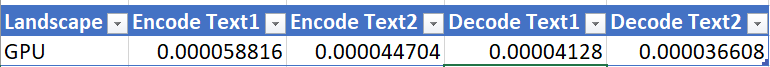


1. CPU decoding for text1.txt and profile image.

From figure 5, we can observe the experiment result that GPU is faster than CPU computation in all types of text files and image files. However, the CPU version of the program cannot process landscape jpg image type. So, I only test the profile png image type of CPU version. For the figure 6, we can see the GPU process the landscape image still fast.



1. CPU and GPU comparison for Profile image.



1. GPU kernel execution time for a landscape image.

# Conclusion

Although the GPU performance is great than CPU, the GPU computation time did not include the memory copy time. So we should do more experiments to verify this. In addition, the CPU version of program decoding contains the output of the character on the screen. The I/O operations take a lot of time in the computer, so we can modify the CPU version to remove this output part. Therefore, we can compare both versions in the same manner.

Furthermore, we only use blindhid algorithm in this program. We should try more secure algorithms in [1] to enhance the image steganography security.

##### References

1. M.Umamaheswari, S.Sivasubramanian, S.Pandiarajan, “Analysis of Different Steganographic Algorithms for Secured Data Hiding,” IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.8, August 2010.
2. Hitanshu Dhawan, ImageSteganography, <https://github.com/hitanshu-dhawan/ImageSteganography>.
3. Ghazanfar Abbas, How To Write A Custom CUDA Kernel With OpenCV As Host Library? <http://programmerfish.com/how-to-write-a-custom-cuda-kernel-with-opencv-as-host-library/>