

HPC Assignment

by Rhiannon Thomas

FILE	4475_RHIANNON_THOMAS_HPC_ASSIGNMENT_107698_1588103989.DO CX (96.45K)		
TIME SUBMITTED	08-MAY-2019 10:40PM (UTC+0100)	WORD COUNT	2234
SUBMISSION ID	106701019	CHARACTER COUNT	11478

Rhiannon Thomas

P149985

High Performance Computing for Games and Visualisation

Lecturers: Gordon Dickers & Andrew Baker

ABSTRACT

In this project a comparison between using the CPU and GPU will be implemented to see which method is faster when changing the brightness of an image. Firstly, an algorithm will be implemented to solve the problem of an overexposed image and then this algorithm will run on C++ and CUDA and will evaluate the results by looking at the performance tab on Visual Basic. There will be graphs and explanations provided to show which one is faster than the other.

CONTENTS

1	Introduction and review	4
2	My algorithm.....	4
3	Parallelization of algorithm.....	4
3.1	Kernels & Threads	5
3.2	Examples	5
4	Results	6
4.1	CPU Results	6
4.1.1	Init	6
4.1.2	LoadBitmapFile	6
4.1.3	Display	7
4.1.4	Reshape.....	7
4.1.5	DrawBox.....	7
4.2	Conclusion.....	7
4.3	GPU Results.....	7
5	Evaluation	7
6	Conclusions and Future work.....	8
7	References	9
8	Bibliography	10

1 INTRODUCTION AND REVIEW

In this project a comparison between using the CPU and GPU will be implemented to see which method is faster when changing the brightness of an image. Firstly, an algorithm will be implemented to solve the problem of an overexposed image and then this algorithm will run on C++ and CUDA and will evaluate the results by looking at the performance tab on Visual Basic. There will be graphs and explanations provided to show which one is faster than the other. Lastly, conclusions between the CPU and GPU will be made and future work will be added.

When an image is overexposed or underexposed this means that image has not been normalized. Normalisation is a part of Digital Image Processing, where an image is defined as a two-dimensional function $f(x, y)$, where x and y are plane coordinates and the magnitude of f at any pair coordinates (x, y) is called the intensity values of the image at the specific point [1]. In this project, the (x, y) values of each pixel of the image will be used to change its intensity values.

Normalization is the process of transforming the image function $f(x, y)$ into another function $f_2(x, y)$ so this can retain all relevant information of the original image and satisfies a set of conditions which is called normalization criteria [2]. Also, normalization is the process that changes the range of pixel intensity values, which is sometimes called contrast stretching or histogram stretching [3]. In general fields of data processing normalization is usually referred to as dynamic range expansion [3].

The purpose of image normalization is usually either to make features of an image familiar to the human senses and more visually appealing [4]. Also, in terms of Machine Vision applications, normalization is used to increase contrast for improved feature extraction or image segmentation [4]. Linear Normalization will stretch the intensity range of an image to equalize the values so the data in each pixel will be scaled from minimum to maximum values.

2 MY ALGORITHM

The purpose of dynamic range expansion in applications is usually to bring the image into range that is more familiar or normal to the eye [3]. The algorithm which is going to be used allows a change from the original pixel values from the image to new pixel values which will then change the intensity values. The algorithm which is going to be performed is the linear normalization formula:

$$I_N = (I - Min) \frac{newMax - newMin}{Max - Min} + newMin$$

For example, if the intensity values of the image ranges from 30 to 180 and the desired range is 0 to 255 (Min and Max values) the process includes subtracting 30 from each pixel intensity, making the range from 0 to 150, then each pixel is multiplied by 255/150 making the range from 0 to 255 [4].

3 PARALLELIZATION OF ALGORITHM

In this section a discussion about the parallelization of the normalized algorithm will be created and examples of parallelization techniques will be shown including screenshots of the segments of code which has been implemented in CUDA.

In order to parallelize the programme there will be two types of Visual Studio applications. The CPU will be implemented in Win32 and the GPU will be implemented CUDA. There are differences between using the CPU and GPU, this is due to the calculation. In the CPU, for each RGB intensity value, each pixel will have to be calculated on at a time however, in the GPU, for each RGB intensity value, all pixels can be calculated all at once. The section below include segments of code in CUDA which allows the programme to be parallelized.

3.1 Kernels & Threads

The CUDA programming model is a heterogeneous model which can be used in the CPU and GPU. In CUDA, the host refers to the CPU and its memory, while the device refers to the GPU and its memory [6]. Code which is compiled on the host can manage memory on both the host and device, also launches kernels which are functions executed on the device, these kernels are executed by multiple GPU threads in parallel [6].

CUDA gives each thread a unique ThreadID to differentiate between each other, however kernel instructions are similar. A BlockID holds multiple threads and a ThreadID holds each individual thread to make it unique. Each thread runs a kernel, which will then ensure that the code is parallelized.

3.2 Examples

Below is an Example which has been used in CUDA which shows the amount of blocks used in the programme, the thread size and the elements used in the Kernel function:

```
// Launch a kernel on the GPU with one thread for each element.
loadBitmapFileKernel <<<1, size>>>(width, height, bitmapData, newmax, newmin, normalize);
```

The loadBitmapFileKernel is the name of the kernel which has been used, this is where the Normalization calculation has been implemented. Then, the 1 represents the amount of blocks that are being used, the blocks have been defined just above this code, the Size represents the amount of threads used in the programme and then in the brackets passes in all elements used in the kernel. Many values have been passed in because CUDA is unable to find the values from an outside function.

Also, a device has been set up for bitmapData, so in order to access the data from the kernel it must be implemented into a device first. Data will then be allocated into GPU buffers and then copy the bitmap data to the GPU buffer. Below is an example of coding devices:

```
// sets up a device for the bitmapData
unsigned char *dev_bitmapData = 0;

cudaError_t cudaStatus;

// Choose which GPU to run on, change this on a multi-GPU system.
cudaStatus = cudaSetDevice(0);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaSetDevice failed! Do you have a CUDA-capable GPU installed?");
    goto Error;
}

// Allocate GPU buffers for bitmap data .
cudaStatus = cudaMalloc((void**)&dev_bitmapData, sizeof(bitmapData));
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}

// Allocate GPU buffers for three vectors (two input, one output) .
cudaStatus = cudaMalloc((void**)&dev_c, size * sizeof(int));
if (cudaStatus != cudaSuccess) { ... }

cudaStatus = cudaMalloc((void**)&dev_a, size * sizeof(int));
if (cudaStatus != cudaSuccess) { ... }

cudaStatus = cudaMalloc((void**)&dev_b, size * sizeof(int));
if (cudaStatus != cudaSuccess) { ... }

// Copy bitmap data to the GPU buffer
cudaStatus = cudaMemcpy(dev_bitmapData, bitmapData, sizeof(bitmapData), cudaMemcpyHostToDevice);
if (cudaStatus != cudaSuccess) {
```

Lastly, the normalization algorithm has been implemented using a Kernel, by creating the code in the kernel it will parallelize it. Below is a segment of code used to create a kernel function:

```
__global__ void loadBitmapFileKernel(int width, int height, unsigned char *bitmapData, float newmax, float newmin, float normalize)
{
    int i = threadIdx.x;

    int min = width;
    int max = height;

    normalize = (min)*(newmax - newmin) / (max - min) + newmin;

    bitmapData[i] = bitmapData[i] * 0.8;
    bitmapData[i + 1] = bitmapData[i + 1] * 0.8;
    bitmapData[i + 2] = bitmapData[i + 2] * 0.8;
}
```

In order to make this algorithm to work, all functions declared outside this function must be passed into the kernel so the kernel will then be able to calculate the normalization algorithm. This kernel function will then be launched onto the GPU with one thread per element.

4 RESULTS

In this section the results of the CPU and GPU will be discussed and an evaluation of the data that has been collected. In order to collect results from both the CPU and GPU, an analysis of the time that the algorithm has been calculated must be shown and then compare the times together to create an evaluation.

4.1 CPU Results

Fortunately, the normalization algorithm has been correctly implemented into Visual Studio Win32. The results of this process is shown below:

Function Name	Number of Calls	Elapsed Inclusive Time %	Elapsed Exclusive Time %	Average Elapsed Inclusive Time	Average Elapsed Exclusive Time
Init	1	2.24	1.10	26.23	12.87
LoadBitmapFile	2	0.80	0.44	4.70	2.57
Display	1	5.46	0.02	64.03	0.24
Reshape	1	0.00	0.00	0.02	0.01
DrawBox	1	0.00	0.00	0.01	0.01

4.1.1 Init

This function has taken the longest average time to calculate everything in its function. The Init function initializes the bitmap file and also includes the normalization algorithm. This function also gets the image from the file explorer as well.

4.1.2 LoadBitmapFile

This function is used as a file loader, this enables the programme to load a bitmap image from a file from the file explorer.

4.1.3 Display

This function is used to draw the image into the scene, this includes the lighting and the material which is being used and positioning the image onto the screen correctly using the x and y axis.

4.1.4 Reshape

This function is used so when the user resizes the window application the image will be able to be resized with it and also will not lose any quality of data in the image.

4.1.5 DrawBox

This function will draw a 2-dimensional box where the image will be placed. This function also is adjusted so the box is located at the centre of the screen.

4.2 Conclusion

Looking at the data that has been received, it is predictable that the initialize function would be the longest to calculate due to the algorithm which is in the function among other bitmap loading elements.

4.3 GPU Results

Unfortunately, the normalization algorithm is not working in CUDA due to time constraints. However, the code does compile but does not output any image onto the screen. The problem is due to not being able to send information back out to the kernel. In the implementation, the information going into the kernel was successful however sending the information back out was not implemented due to time constraints. However, the implementation of the information into the kernel has been coded, the next step would have been to send each individual element of the kernel back to the kernel. In order to make the CUDA code successful, information must be sent in kernel and then information must be received back to output the data onto the screen. The Parallelism section includes segments of code which has been implemented in CUDA showing the different types of techniques that has been used.

To predict, the GPU would have been the fastest to calculate the normalization algorithm because CUDA provides parallelism which allows access to each pixel all at the same time.

5 EVALUATION

To evaluate, even though the results have not been clear using the GPU a prediction has been made stating that the GPU would have been faster to calculate the normalization algorithm compared to the CPU. As shown above, the data from the CPU shows that the function which creates the algorithm was the longest of Average Elapsed Inclusive Time of 26.23 and an Average Elapsed Exclusive Time of 12.87, this is due to the CPU going into each pixel of the image and changing its intensity values to normalize the overall image. However, whilst implementing the GPU the normalization algorithm was created in the kernel function which allows the GPU to go into all pixels at the same time and changing their intensity values. Also, an attempt has been made in the GPU which allows the normalization algorithm to send its information to the kernel however it is unable to output the data onto the screen due to no data being passed back into the kernel.

6 CONCLUSIONS AND FUTURE WORK

To conclude, the overall project has not been successful due to the GPU code which was unable to work but does compile when the user runs the code showing that there are no errors in the code. However, the CPU code was able to work and was able to receive data from the code which shows the statistics of how well the CPU performance is and shows which functions have been slow and which functions have been fast, in the data which has been collected it shows that the initialize function was the slowest to calculate as predicted.

Throughout the research of this project there have been resources which have been looked at including Websites and Books, all resources are shown in the References and Bibliography section. The main area of research was mainly about what is normalization and how does it work. Also, an Algorithm section was created stating which algorithm that has been chosen and how it can be implemented into this project. The main criteria of this report have been the results from the CPU and results from the GPU. Unfortunately there were no results collected from the GPU, however results have been collected from the CPU. 

Considering that the implementation was not a success, further work would include research more on CUDA and how the normalization algorithm would be implemented into CUDA correctly, then the results would be received so that this project can then be successfully implemented.

7 REFERENCES

- [1] R. E. W. Rafael C. Gonzalez, Digital Image Processing.
- [2] D. P. Yaser S. Abu-Mostafa, "Image Normalization by Complex Moments," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, 1985.
- [3] eClass, "Normalization (Image Processing)," [Online]. Available:
<https://eclass.teicrete.gr/modules/document/file.php/TP283/Lab/03.%20Lab/lesson3Notes.pdf>.
- [4] C. Walker, "Chris Walker's Tech Blog," 18th August 2012. [Online]. Available:
<http://chriswalkertechblog.blogspot.com/>.
- [5] Revolv, "Normalization (image processing)," [Online]. Available:
<https://www.revolv.com/page/Normalization-%28image-processing%29>.
- [6] M. Harris, "An Easy Introduction to CUDA C and C++," 31 October 2012. [Online]. Available:
<https://devblogs.nvidia.com/easy-introduction-cuda-c-and-c/>.

8 BIBLIOGRAPHY

- [1] R. E. W. Rafael C. Gonzalez, Digital Image Processing.
- [2] D. P. Yaser S. Abu-Mostafa, "Image Normalization by Complex Moments," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, 1985.
- [3] eClass, "Normalization (Image Processing)," [Online]. Available:
<https://eclass.teicrete.gr/modules/document/file.php/TP283/Lab/03.%20Lab/lesson3Notes.pdf>.
- [4] C. Walker, "Chris Walker's Tech Blog," 18th August 2012. [Online]. Available:
<http://chriswalkertechblog.blogspot.com/>.
- [5] Revolv, "Normalization (image processing)," [Online]. Available:
<https://www.revolv.com/page/Normalization-%28image-processing%29>.
- [6] M. Harris, "An Easy Introduction to CUDA C and C++," 31 October 2012. [Online]. Available:
<https://devblogs.nvidia.com/easy-introduction-cuda-c-and-c/>.

HPC Assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

53/100

PAGE 1

PAGE 2


Comment 1
Visual studio!

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8


Comment 2
no comment

PAGE 9

PAGE 10

GRADING FORM: SMU GENERIC_GJ

RESEARCH-INFORMED LITERATURE



Summarise results in abstract. Refs generally appropriate - avoid too many online refs. Check formatting errors of refs. Bibliography included.

5

KNOWLEDGE AND UNDERSTANDING



Identified relevant techniques for image normalisation - other methods could have been identified and discussed. Looked at Cuda technologies too.

7.5

ANALYSIS



Identification of algorithm to parallelise in evidence. You have provided CPU and some GPU evaluation - conclusions contradict about this. GPU results limited. CPU evaluation (timings) provided.

4.5

PRACTICAL APPLICATION AND DEP.



CPU version produces reasonable results. GPU - limited results- why not evaluate without normalisation code?

35

SKILLS FOR PROFESSIONAL PRAC.



Report structured. Refs included. See comment re abstract above.

3