International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 04, April-2017

Using OPENCV over MATLAB for Implementing Image Processing Application on CUDA GPU to Achieve Better Execution Speedup

Shraddha Oza[1],
E&Tc Department,
Army Institute of Technology,
Pune, India

Dr. (Mrs.) K. R. Joshi[2]
E & Tc Department,
PES Modern College of Engineering,
Pune, India

***Abstract* - Digital Image Processing is significant for correct interpretation, analysis and enhancement of digital images. It has varied applications in the domain of computer vision, medical imaging, astronomical imaging, photography. Matlab and OpenCV are the two most popularly used toolkits for building the image processing applications. The purpose of the work presented here is to compare and analyse performance of these two platforms in the context of execution speed. A color (RGB color space) to gray converter was implemented using Matlab as well as OpenCV with CPU at the backend executing the code sequentially. The conversion speed was found to be much higher in case of OpenCV. The same converter was implemented using CUDA GPU, which gave higher speed up over its CPU version due to its extensively parallel architecture. The work highlights use of OpenCv library to be used alongside CUDA C for pre and post image processing functions executed by CPU, to achieve maximum speed up. In future, different optimization techniques for CUDA may be used to enhance the speed up.**

Keywords : Matlab, OpenCV, CUDA, Color to gray converter

INTRODUCTION

Digital image processing has a wide range of applications in the domain of medical imaging, telemedicine, acoustic imaging, and video surveillance. Typical Image Processing operations include image segmentation, image denoising and image enhancement. Image enhancement implies processing the input image such that information in the image is restored and is more useful for information analysis. Color to gray conversion is one of the commonly used image enhancement techniques. The color to gray scale image conversion process should retain the information alongside gray scale output.

There exist many conversion algorithms, each one with its pros and cons [2][4]. The work here presents implementation of color to gray converter which computes luminance from Red, Green and Blue colors in the image by applying different weights to each color [1][3][4].

In the work presented, the color to gray scale image converter was implemented separately using Matlab toolkit (version 2010) and OpenCv library (2.4.10) on Windows 7, i5 quad core platform. As these two platforms are distinct in their approach [8][9] and are popularly used, the purpose was to compare their performance in terms of speed up. It was observed that, as the image resolution was increased from 255X255 to 4096X4096 OpenCV proved to be much faster than Matlab.

Considering the performance efficiency of Graphics processor for data parallel computations [5][6][10], in the proposed work, the color to gray converter was implemented also using CUDA C with GPU at backend. The speed up achieved was much higher as compared to CPU OpenCV version.

The work is presented in seven sections. The first three sections explain the basis for Matlab, OpenCV platforms and their comparison. Section IV briefly describes CUDA architecture. Section V gives details of the mathematical model used to implement color to gray converter. Section VI describes the experimental set up and section VII elaborately discusses the observations and conclusions drawn.

I. MATLAB

MATLAB stands for MATrixLABoratory and the software is built around vectors and matrices. It has tool boxes which can be used for developing signal processing and image processing applications.

Matlab is an interpreted language. The Matlab code is translated piece by piece into an equivalent machine code during the runtime of the program and immediately executed. It is a relatively simple language and easy to use**.** The typical programming issues such as including libraries, memory allocation, and variable declarations need not be accounted for. Thus, it is extensively used for developing image processing applications [9].

II. Open CV

OpenCV is an acronym which stands for Open Source Computer Vision Library. It was originally developed by Intel to provide access to image processing functions required to build computer vision applications. It is a library of different inbuilt functions which are mainly written to achieve real time performance. Being open source, the library functions get continuously improvised and optimized. The inbuilt library functions have the capability to exploit multi-core processing. It is free for commercial as well as noncommercial use and is supported by Willow Garage.

OpenCV is written in optimized C/C++ thus, as against Matlab, the function is compiled by a C/C++ compiler and not interpreted which makes its performance much faster. [8]

III. MATLAB VS OPENCV

The comparison of the two popularly used development platforms can be done with following parameters –

- Resources Required: As Matlab is a high levelscripting language, it uses a lot of system resources as against OpenCV. Also, OpenCV being open source, optimization is a continuous process.
- Execution Speed: The Matlab code is interpreted and executed while an OpenCV code is compiled and executed. Thus, program in OpenCV executes much faster than similar program in Matlab.
- Development Cost: Matlab is a commercial software and is costly while OpenCV being open source is free.
- Portability: Matlab and OpenCV both support Windows, Linux and Mac Operating system. The application developed using OpenCV can very well be ported to any device or machine that can run c.
- Support to parallel architecture: Matlab and OpenCV both equally provide good library support to parallel architecture.

IV. CUDA

In 2007, Nvidia launched its CUDA (Compute Unified Device Architecture) compliant GPU devices for general purpose applications. These devices have proved to be much superior especially for image processing applications [5][6].

CUDA GPU is collection of SMPs (Symmetric Multiprocessors), each one with a capacity to launch large number of threads executing concurrently. Thus, CUDA avails a very powerful environment of highly parallel data computation. In recent years, the CUDA devices are being used to develop a variety of general purpose applications especially in the domain of image processing, it being a typical single instruction multiple data computation domain. CUDA uses a language which is largely an extension to C language which makes it programmer friendly.

The CUDA programmer needs to define the threads to be used or instantiated in terms of block size and grid size. A bock is defined as group of threads and it can hold maximum 512 or 1024 threads. A grid is a group of blocks and a unique multiprocessor is assigned to each block for execution. (Fig 1)

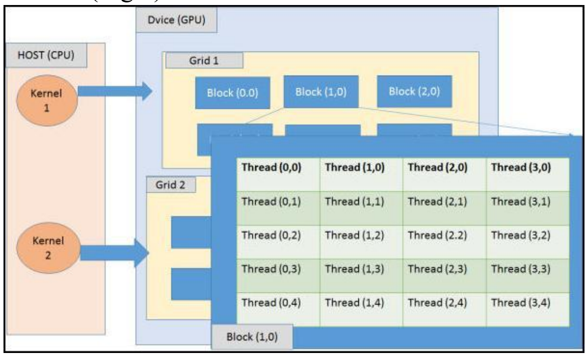

Fig 1: CUDA Programming Model [10]

In CUDA C, parallel execution is indicated by launching a kernel function. The kernel function is in turn executed by a set of concurrent threads. The concurrency factor provides the intended speed up. The syntax for a kernel call in the context of the color to gray converter is

```
ctgr_to_gray_kernel<<<grid,block>>>(d_input,d_o
utput,input.cols,input.rows,input.step,output.s
tep);
```

Here ctgr_to_gray is name of the kernel function as used in the work presented here, grid and block values together define maximum threads executed and rest are parameters required for the function to execute properly.

V. THE COLOR TO GRAY CONVERTER

RGB color space is defined by the three basicadditive primary chromaticity: the red, the green, and the blue (refer Fig 1). Using these three colors and by changing the weights associated, any chromaticity can be produced. The human RGB is close to the way human visual system works. Thus it is a convenient color model for computer graphics. Humans do not perceive all colors equally. Thus, a good gray scale conversion is to weigh each color (Red, Green, Blue) based on the way human eye perceives it. The color to gray conversion is based on the formula given below [1][3][4]-

$$\text{Gray} = (\text{Red} * 0.3 + \text{Green} * 0.59 + \text{Blue} * 0.11) \quad (1)$$

The above expression forms a weighted sum of the red, green, and blue components. This function transforms a 24-bit, three-channel, color image to an 8-bit, single-channel gray scale image. The weighted sum is also known as luminance. The weights used to compute luminance are related to the monitor's phosphors. The basis for the expression is that for equal amounts of blue and green light, the green will be perceived as brighter. Though the resulting gray scale image is dynamic, the formula requires computations.

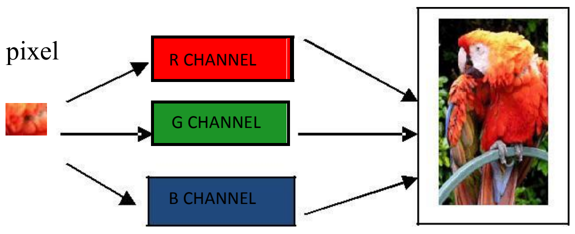

Fig. 2: RGB Image

VI. EXPERIMENTATION

To analyse the performance of Matlab and OpenCV CPU version as against CUDA GPU version, six color (PNG type) images of different resolution were selected ranging from 255x255 to 4096x4096. The color to gray converter was applied to each of the images and the speed of conversion was noted down (Refer to table 1). The sample RGB image and the converted gray scale image is given in fig 3 and fig 4 respectively.

Image	Resolution (pixels)	Matlab (CPU) (msec)	Opencv (CPU) (msec)	CUDA (GPU) (msec)
I1	225x225	1.167	1.7411	0.211456
I2	512x512	4.078	5.972	0.5744
I3	1100 x 1099	18.314	8.704	2.3767
I4	1344 x 1008	16.458	10.3393	3.8707
I5	2048 x 1536	50.859	15.317	6.0096
I6	4096 x 4096	199.384	51.971	31.4265

Table1: Conversion time of color to gray converter in Matlab(CPU), OpenCV(CPU) and CUDA GPU

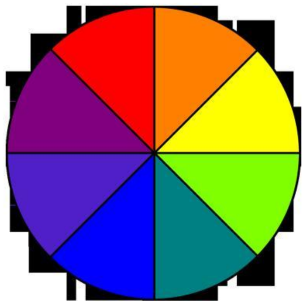
Fig 3: Sample Colored Image (RGB

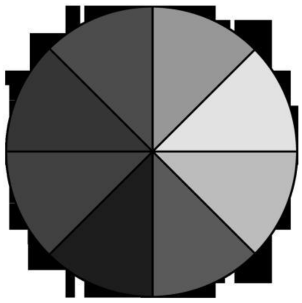
Fig 4: Output Gray Image

The experimentation for Matlab(ver 2010) and OpenCV(ver 2.4.10 integrated with visual studio 2010) was done on i5 quad core processor machine with windows 7 OS.

The CUDA implementation was done using GEFORCE 830M. The kernel function was defined for color to gray conversion as given in section III. The grid dimension was calculated for each image size using following formula:

Grid dim = [(Image Width + (block size in x direction – 1) / Threads per Block in x direction), (Image Height + (block size in y direction – 1) / Threads per Block in y direction) [7][10]

Here, Block size was defined to be (16, 16) i.e. 256 threads per block. Thus, for an image of 225 X 225, total threads instantiated for computation were 15/block to cover the whole image while it was 256 threads /block for the image size of 4096X4096.

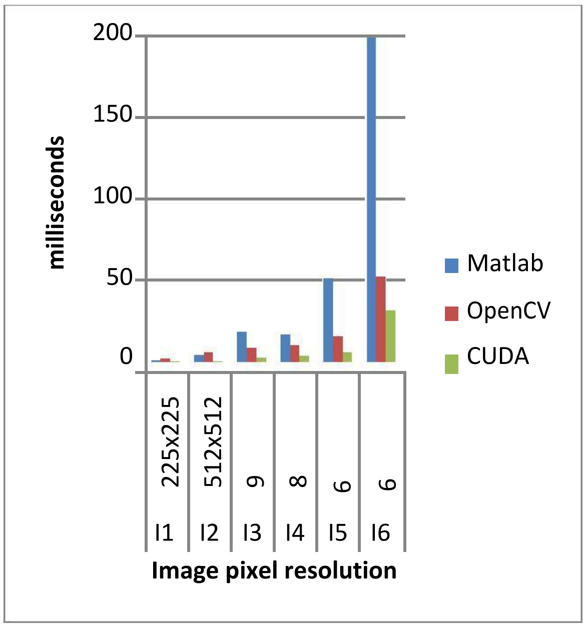

Fig 5: Conversion Time Vs Images with different pixel resolution

VII. OBSERVATIONS AND CONCLUSION

As can be seen from table 1and fig 5, Matlab proved to be faster than OpenCV for image size 225X225 with execution time of 1.167 msecs and 512X512 with execution time of 4.078 msecs. But as the image size increased beyond IK resolution, OpenCV started giving better results. For maximum size of 4096X4096 resolution Matlab version conversion was completed in 199.384 msecs while that in OpenCV finished in 51.971msecs.

The parallel domain CUDA GPU version of the converter proved to be fastest for all the image sizes. The conversion time for 225X225 was found to be 0.21146 msecs and that for 4096X4096 was found to be 31.4265 msecs.

The color to gray conversion is done by applying the equation (1) given in section IV. As this equation is applied to every pixel in image per channel, the process obviously involves for/while loop running with repetition count equal to image size per channel. OpenCV being a compiled language, the "for / while" loop bodies are transformed into machine code only once at compile time. However, in interpreted languages like Matlab, the loop body is translated each time a loop executes making the conversion much slower. As can be seen from the graph in Fig 4, the converter when applied to the I6 image with resolution of 4096X 4096 pixels, Opencv proved to be much faster than its Matlab counterpart by 20.5445 msecs.

The CUDA implementation was done using CUDA tool chain ver 6.5 integrated with visual studio 2010. The image preprocessing was done using OpenCV. In future different optimization techniques may be implemented such as tiling, separable approach, and use of shared memory [10].

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 04, April-2017

There exists different color to gray algorithms. These algorithms may be implemented and the gray image obtained can be analysed to quantify performance of algorithm. [3][4]

REFERENCES

[1] http://gimp- savvy.com/BOOK/index.html?node54.html
[2] Mark Grundland et al,“Decolorize: fast, contrastenhancing, color to grayscale conversion”, ComputerLaboratory, University of Cambridge
[3] M. ˇCadík et al, “Perceptual Evaluation of Color-to-Grayscale Image Conversions”, Pacific Graphics2008, Volume 27 (2008), Number 7
[4] www.tannerhelland.com/3643/grayscale-
image-algorithm-vb6/Oct 1, 2011
[5] ShuaiChe et al., “A performance study of general-purpose applications on graphics processors using CUDA”, Journal of Parallel and Distributed
[6] Computing, Elsevier, Volume 68, Issue 10, October2008
[7] J D Owens et al., “GPU Computing”, Proceedings of the IEEE(Volume:96, Issue: 5),ISSN - 0018-9219, May 2008
[8] Cliff Woolley, NVIDIA Developer TechnologyGroup www.opencv.org
[9] http://in.mathworks.com
[10] https://developer.nvidia.com/opencv
[11] www.mathworks.com/discovery/matlab- gpu.html