International Journal of Latest Trends in Engineering and Technology
Special Issue SACAIM 2017, pp. 252-255
e-ISSN:2278-621X

COMPARISON OF SERIAL AND PARALLEL IMAGE PROCESSING USING HISTOGRAM EQUALIZATION ALGORITHMS

Nithin C[1], Sagar Naik[2] & Nausheeda B S[3]

Abstract- For preparing on picture, operations must be performed on every pixel. On the off chance, the operations are performed consecutively it will require excessively investment. This paper give an effective study on serial as well as parallel image processing. So to diminish the time, there is need of parallel handling on every one of the pixels. So that rather than working on every pixel one by one, operations on all the pixels is done parallel at once. By performing Parallel operations speed of preparing is expanded essentially as contrasted with successive one. So it will likewise perform video handling in speedier way. For parallel handling NVIDIA Illustrations card is utilized. Parallel calculation is performed on CUDAC stage. In this paper we have studied the work done by Hawkins et al. and executed the algorithm with different dataset.
Keywords- CUDA (Computer Unified Device Architecture), GPU, Histogram Equalization, Parallel computing.

1. INTRODUCTION

Ongoing picture handling methods/issues imperative a huge degree of handling matchless quality, ability in registering and huge assets to play out the calculation and operations. These requirements for the most part Show up on the framework because of the mammoth measurement, nature of the pictures to be prepared and extreme expansion of information from kilo to terabyte in most recent couple of years. Parallel preparing of the pictures is observed to be the best way to deal with handle this issue. For the effective and fast usage of picture handling parallel. There are a few devices and methods are accessible, for example, CUDA, GPU, PCT of MATLABTM, Open-CV, and Open-CL. In the display day circumstance GPU (Graphics Processing Unit) has long been used to animate PC helped plan, PC liquid flow, computational basic mechanics, electronic outline computerization, 3D gaming, applications, superior picture handling and some different applications

The proposed calculation is kept an eye on NVIDIA GTX 480 card. The CUDA Architecture incorporates pipeline, permitting every single number-crunching rationale unit (ALU) on the chip to be marshaled by a program planning to perform universally useful calculations. Since NVIDIA proposed this new group of illustrations processors to be utilized for universally useful registering these ALUs were worked to agree to IEEE necessities for single-accuracy skimming point number juggling and were intended to utilize a guideline set custom fitted for general calculation rather than particularly for illustrations. Besides, the execution units on the GPU were permitted subjective perused and compose access to memory and in addition access to a product oversaw reserve known as shared memory

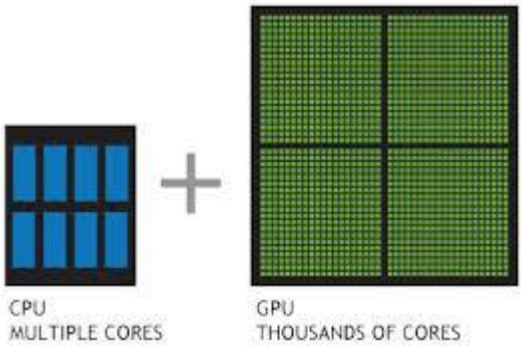

Fig .Gives the view of cores in CPU and GPU

[1] Department of Information Technology & Bioinformatics, AIMIT, St. Aloysius College, Mangalore, Karnataka, India
[2] Department of Information Technology & Bioinformatics, AIMIT, St. Aloysius College, Mangalore, Karnataka, India
[3] St. Aloysius Institute of Management and Information Technology, Mangalore, Karnataka, India

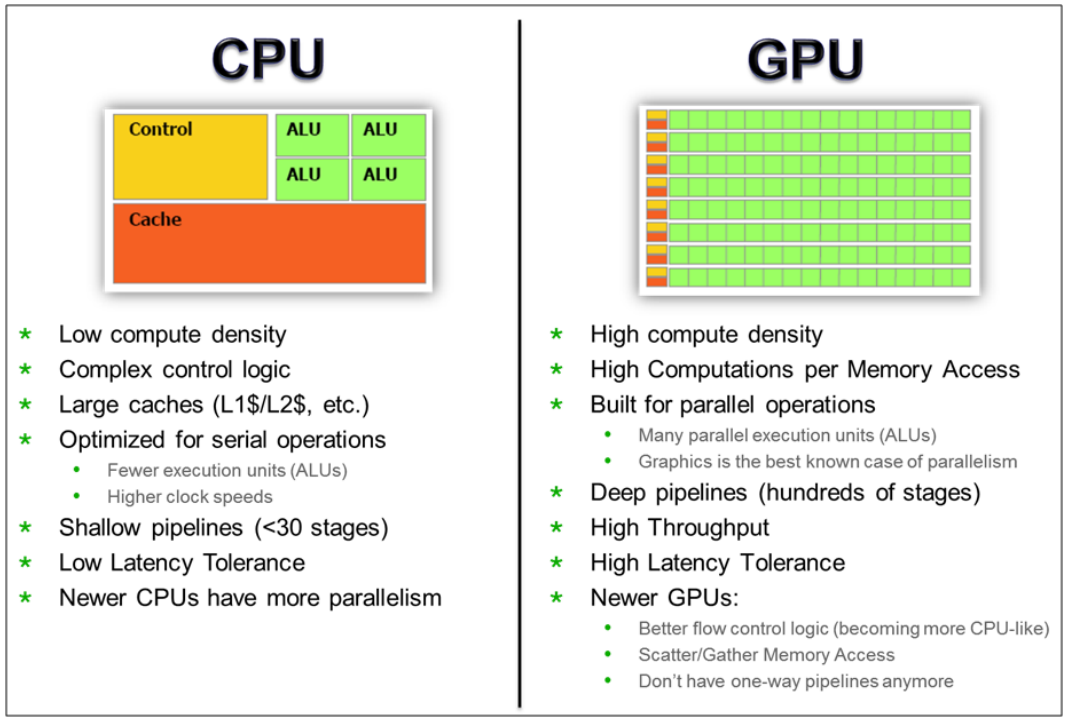

Fig .1 shows the difference between the CPU and GPU.

The models accessible are Fermi, Kepler and Tesla. On Fermi engineering, each time amid calling a part, the factors ought to be replicated from have memory to gadget memory, and again from gadget memory to have a great many consummation of the capacity. In kepler design, there is no need to duplicate factors inevitably. They can be exchanged on gadget as it were. For this work Fermi engineering is utilized

In CPU architecture, there is limited number of cores. So the ALU operations are done one after another. But in case of GPU, lacs of threads can be launched which can perform ALU operations simultaneously. As shown in Figure 2.and Figure 3, number of parallel blocks are launched in grid and a number of threads are launched in each block. Maximum no. of threads per

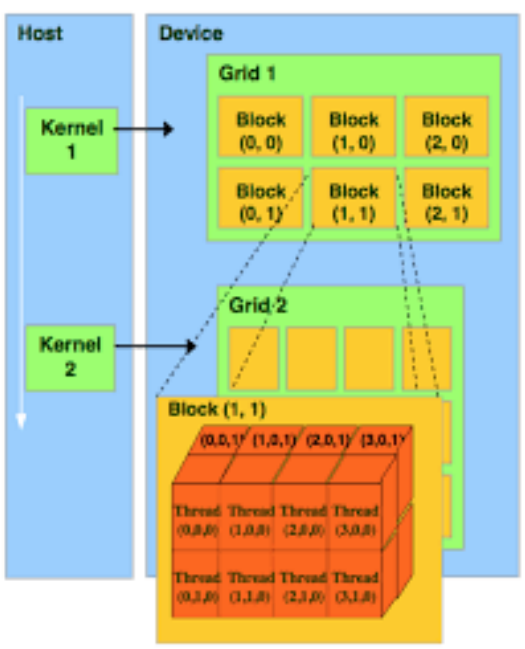

Fig.2 Hierarchy of block of thread in GPU

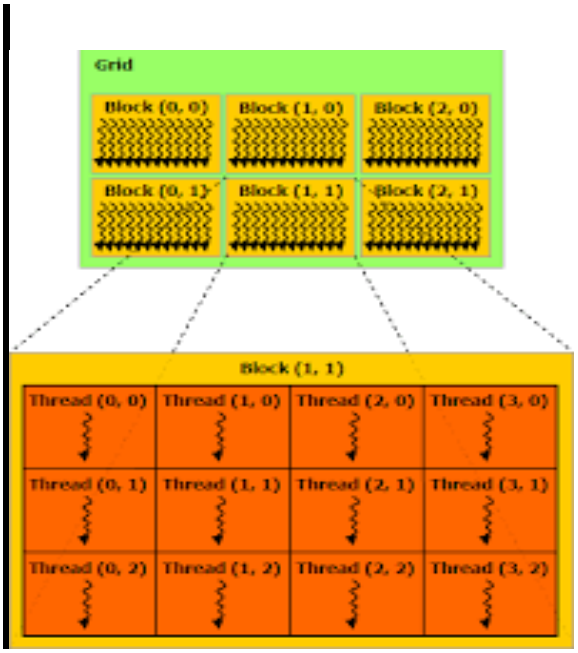

Fig.3 hierarchy of block of thread in CPU

In CPU engineering, there is set number of centers. So the ALU operations are done in a steady progression. Be that as it may, if there should be an occurrence of GPU, lack of strings can be propelled which can perform ALU operations all the while. As appeared in Figure 2.and Figure , number of parallel squares are propelled in lattice and various strings are propelled in each square. Greatest no. of strings per square is equivalents to 512. Most extreme size of x, y, z measurements of strings are 512, 512, 64 separately. Most extreme size of each measurement of lattice is 65536 squares. So strings and pieces are orchestrated according to our benefit to address the information factors what's more, perform operations

2. EXISTING ALGORITHM

2.1 Histogram equalization algorithm –

The algorithm which is to be performed for sequential and parallel process (for histogram equalization) is as follows:

a) To convert colour image to grey scale image:
1. Calculate average of Red, Green and Blue values for each pixel. Store this average at each Red, Green and Blue value of that pixel.
2. Store this average at each Red, Green and Blue value of that pixel.

b) To calculate histogram of colour image
1. Initialize all the three histogram arrays for Red, Green and Blue to zero.
2. For each pixel, take the Blue value in variable ch , increment the ch element of array of histogram for Blue value by one.
3. For each pixel, take the Green value in variable ch, increment the ch th element of array of histogram for Green value by one.
4. For each pixel, take the Red value in variable ch, increment the chth element of array of histogram for Red value by one.

c) To calculate histogram of grey scale image
1. Initialize array of histogram for grey scale image to zero.
2. For each pixel, calculate average of all the three values Red, Green and Blue and store it in variable ch.
3. Increment the ch th element of array of histogram for grey scale image by one.

d) To calculate CDF
1. Copy the first element of histogram array to first element of cdf array.
2. For i=0 to i

e) To normalize CDF
1. Find minimum and maximum values in cdf.
2. For i=0 to i

f) To perform histogram equalization of colour image
1. Copy the value of cdf of intensity of each pixel in input image to the value of intensity of that pixel in output image.
2. Copy the value of Hue of each pixel in input image to the value of Hue of that pixel in output image.
3. Copy the value of saturation of each pixel in input image to the value of saturation of that pixel in output image.

g) To perform histogram equalization of grey scale image
1. Copy the value of cdf of blue value of each pixel in input image to the blue value of that pixel in output image.
2. Copy the value of cdf of green value of each pixel in input image to the green value of that pixel in output image.
3. Copy the value of cdf of red value of each pixel in input image to the red value of that pixel in output image.

3. EXPERMENTATION AND RESULTS

Fig 4.colour image

Fig 5.histogram Equalization image

Histogram of grey scale image and its Histogram equalization image

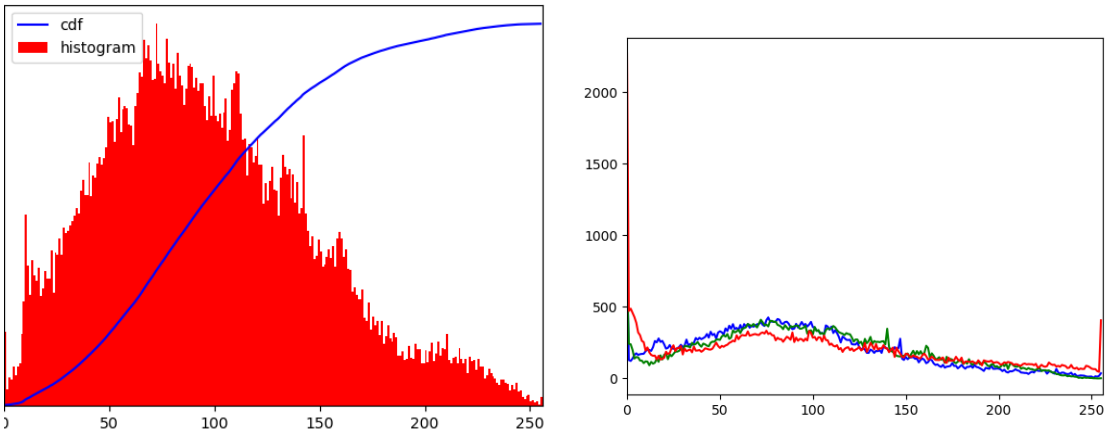

Figure5. Shows the time required for the image processing and there equivalent pixel formation and RGB values.

From the current observation the parallel image processing gives the fast response as compared to serial image processing

4. CONCLUSION

The histogram balance calculation has been actualized in open c v, python which is executed consecutively, and in CUDA C code which is executed parallel. As a result of serial code, in C the operation is performed on pixels one by one .There are lacs of pixel in one picture. So additional time is required for serial code, for parallel calculation, operation on every one of the pixels is performed all the while at once. So time required for preparing lessens immensely

5. REFERENCES

[1] J. K. Hawkins and C. J. Munsey, "A parallel computer organization and mechanizations", IEEE Trans. Electron. Comput., vol. EC-12, pp.251- 262, 1963.

[2] https://developer.nvidia.com/embedded/learn/tutorials

[3] https://in.udacity.com/course/intro-to-parallel-programming--cs344

[4] https://devblogs.nvidia.com/parallelforall/even-easier-introduction-cuda/

[5] Sanjay Saxena, Shiru Sharma, Neeraj Sharma Study of Parallel Image Processing with the Implementation of vHGW Algorithm using CUDA on NVIDIA'S GPU Framework

[6] P. Kaur, ―Implementation of image processing algorithms on the parallel platform using MATLAB‖ in Int. J. Comp. Sci. Eng. Technol., vol. 4(6), 2013, pp. 696-706.

[7] M. Wadpalliwar, M. Bhutani, M. M. Deshpande, ―Implementation of an image filtering technique for image processing using CUDA‖ in Proceedings of 20th IRF International Conference, 1st March 2015, Chennai, India, ISBN: 978-93-84209-01-8

[8] J. Tse, ―Image Processing with CUDA,‖ Master's Thesis, University Of Nevada, Las Vegas, August 2012.

[9] Placido Salvatore Battiato, ―High Performance Median Filtering Algorithm Based on NVIDIA GPU Computing‖, 2016.

[10] J. Ghorpade, J. Parande, M. Kulkarni, ―GPGPU Processing in CUDA Architecture‖ in arXiv preprint arXiv:1202.4347, 2012