Accelerating the Retinex Algorithm with CUDA

Hyo-Seok Seo, Oh-Young Kwon, *Member, KIMICS*

***Abstract*— Recently, the television market trend is change to HD television and the need of the study on HD image enhancement is increased rapidly. To enhancement of image quality, the retinex algorithm is commonly used. That's why we studied how to accelerate the retinex algorithm with CUDA on GPGPU (general purpose graphics processing unit).**

Calculating average part in retinex algorithm is similar to pyramidal calculation. We parallelize this recursive pyramidal average calculating for all layers, map the average data into the 2D plane and reduce the calculating time dramatically. Sequential C code takes 8948ms to get the average values for all layers in 1024x1024 image, but proposed method takes only only about 0.9ms for the same image. We are going to study about the real-time HD video rendering and image enhancement.

***Index Terms*— CUDA, GPGPU, Image enhancement, Pyramidal calculation, Retinex**

I. INTRODUCTION

Parallel computing has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. As power consumption by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi core processors. The computing is evolving from "central processing" the CPU to "co-processing" on the CPU and GPU(Graphic Processing Units). GPU is a highly parallel machine. For instance, NVIDIA's G80 series boasts 128 processors. GPUs keep these processors busy by juggling thousands of parallel computational threads. GPUs can sustain hundreds of GFLOPS on a wide variety of computationally demanding general purpose problems. Especially SIMD problems well suited to get a high performance using GPU[1].

Manuscript received April 8, 2010; revised April 14, 2010; accepted April 28, 2010.

Hyo-Seok Seo is with the department of electrical and electronic engineering, Korea University of Technology and Education, Cheonan, 330-708, Korea (Email: bbluesky@kut.ac.kr)

Oh-Young Kwon is with the School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, 330-708, Korea (Email: oykwon@kut.ac.kr)

There are many interesting problems what is well suited to get a dramatic increase in computing performance with GPUs.

Digital camera's image size become bigger and bigger and HD Television is very popular in the market. That's why HD images and large size images are used frequently. Cause the need of real time HD image processing is increasing and it need high computing power.

In this paper, we chose the retinex algorithm to show the performance increase with GPUs. The retinex algorithm is the image processing algorithm to enhance image's contrast. It needs to compute with many pixels and compare with each other pixels in an image [2].

Calculating average part in retinex algorithm is similar to pyramidal calculation. We parallelize this recursive pyramidal average calculating for all layers, map the average data into the 2D plane and reduce the calculating time dramatically.

In the next section, we provide the related works (CUDA programming and retinex algorithm). In section 3, we provide how we accelerate retinex algorithm. In section 4 and 5, we provide experiment result and conclusion.

II. RELATED WORKS

A.CUDA programming

We chose NVIDIA's CUDA GPU Computing environment for our implementation. CUDA provides a direct, general purpose C language interface to the programmable processors on NVIDIA's 8-series GPUs, eliminating much of the complexity of writing GPGPU applications using graphics APIs such as OpenGL. CUDA makes single instruction computes multiple data possible using many cores in GPU and more fast calculation.

Fig. 1. shows the Grid of Thread Blocks in GPGPU and 1 grid has many thread blocks and each thread block has many threads[1].

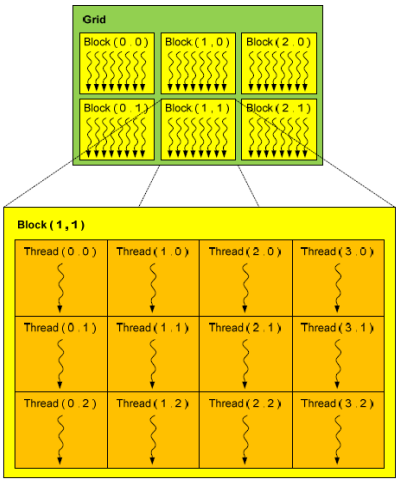

Fig. 1. Grid of Thread Blocks in GPGPU

CUDA's memory model has Texture memory, constant memory, global memory, shared memory, registers, local memory, each memory's access time and read, write limitations are different. Together with shared memory, this feature allows threads to cooperatively compute results[3]. CUDA structures GPU programs into parallel *thread blocks* of up to 512 SIMD-parallel threads. Programmers specify the number of thread blocks and threads per block, and the hardware and drivers map thread blocks to parallel multiprocessors on the GPU. Within a thread block, threads can communicate through shared memory and cooperate by combining shared memory with thread synchronization[4][5].

Efficient CUDA programs exploit both thread parallelism within a thread block and coarser block parallelism across thread blocks. Because only threads within the same block can cooperate via shared memory and thread synchronization, programmers must partition computation into multiple blocks.

B. RETINEX algorithm

The retinex model for the computation of lightness was introduced by Land and McCann. Since that time Land and his colleagues have described several variants on the original method. Many different descriptions of retinex methods of lightness computation exist.

The basic form of the Multi-scale retinex(MSR)[6] is given by

$$R_i(x_1,x_2) = \sum_{k=1}^{K} W_k\{\log I_i(x_1,x_2) - \log[F_k(x_1,x_2) * I_i(x_1,x_2)]\} \qquad i = 1,\ldots,N$$

where the sub-index i represents the i^{th} spectral band, N is the number of spectral bands $N = 1$ for grayscale images and $N = 3$ for typical color images. In the latter case, $i \in R; G; B$ I is the input image, R is the output of the MSR process, F_k represents the k^{th} surround function, W_k are the weights associated with F_k, K is the number of surround functions, or scales, and * represents the convolution operator. The surround functions, F_k are given as:

$$F_k(x_1,x_2) = k \exp[-(x_1^2 + x_2^2)/\sigma_k^2]$$

where σ_k are the scales that control the extent of the surround smaller values of σ_k lead to narrower surrounds and

$$k = 1/(\sum_{x_1}\sum_{x_2} F(x_1,x_2))$$

The basic principles of retinex are: (1) color is obtained from 3 'lightnesses' computed separately for each of the color channels; (2) the ratios of intensities from neighboring locations are assumed to be illumination invariant; (3) lightness in a given channel is computed over large regions based on combining evidence from local ratios; (4) the location with the highest lightness in each channel is assumed to have 100% reflectance within that channel's band. Lightness refers to the perceived (in the case of human perception) or estimated (in the case of computational methods) surface albedo(reflectance averaged over the channel's band).

The initial versions of retinex where based on combining the ratio information along random paths across the image. Multi-resolution versions of retinex where introduced for efficiency [7][8]. Horn [9] formalized retinex in terms of differentiation, threshold and reintegration in the logarithm domain. Kimmel [10] formulates the computation as a variational optimization problem. Two versions of retinex have been given standardized definitions in terms of Matlab code [2].

All of the retinex variants treat the input image as a spatial arrangement of colors and make no use of the 3-dimensional structure of the underlying scene. However, there are a number of psychophysical experiments indicating that the human lightness and color perception are influenced by information from several sources, including 3-dimensional scene geometry. In particular, Gilchrist's[11] early experiments showed that, in the black and white scenes, changing a surface's apparent 3-dimensional context affected the perception of its lightness. In experiments using computer graphics rendered 3- dimensional scenes, Boyaci et.al.[12] provide further evidence for the relationship between perceived orientation and the perceived lightness of matte surfaces. Yamauchi et al.[13] used stereoscopic stimuli to support the notion that surface color perception is strongly influenced by depth information. Bloj et.al.[14] illustrated the effect of spatial shape on chromatic recognition. Yang and Shevell[15] show that binocular disparity can improve color constancy. Adelson[16] argues that statistical and configurable information are combined for lightness perception.

In the retinex algorithm, calculating average of input image's what is divided to variety size grid blocks is frequently executed. In this steps take long time and up to

the image size, the executing time is linearly increased. So if we need to use the retinex algorithm to HD images, we need high performance computing system but it has a limitation. High performance computing system is too expensive. So we suggest the accelerate retinex algorithm with CUDA on GPGPU.

III. ACCELERATING RETINEX ALGORITHM

McCann99 retinex version that is implemented on Matlab[1], is a computer-based version described by McCann. McCann99 retinex creates a multi-resolution pyramid from the input by averaging image data. It begins the pixel comparisons at the most highly averaged, or top level of the pyramid. After computing lightness on the image at a reduced resolution, the resulting lightness values are propagated down, by pixel replication, to the pyramid's next level as initial lightness estimates at that level. Further pixel comparisons refine the lightness estimates at the higher resolution level and then those new lightness estimates are again propagated down a level in the pyramid. This process continues until New Products have been computed for the pyramid's bottom level.

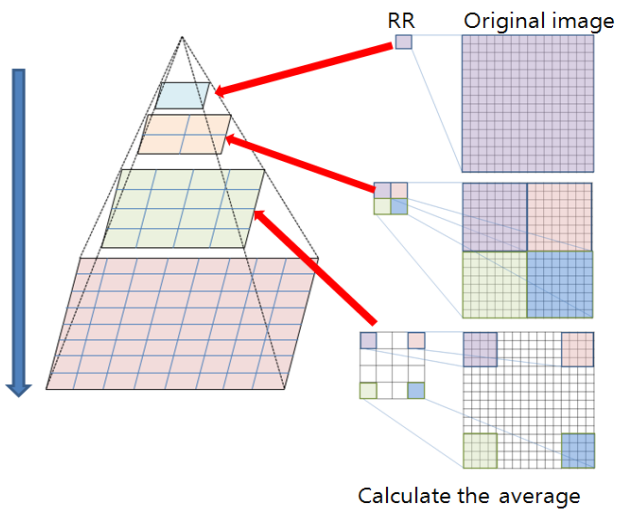

Fig. 2. Normal Steps to make RR Image (average)

Fig.2. show the pyramid to create RR image that is a average of divided block. Top of the pyramid is 1*1 image that is a calculated average of the original image's all pixels. Using this pixel (RR), calculate with original image's maximum data (OP) and each image, OP and RR, padded extra data and create the OPE and the RRE images to calculate easier. Create the IP image by "CompareWithNeighbor" function with these the RRE and the OPE images. In this function, it compares with neighbor pixels and create NP image. Then doubling all pixels in the NP image and save to the OP image. Next layer RR image has 4 pixels that is calculated average of divided original image. And calculate with RR and OP images as the order. These steps are continued until the bottom layer. And each step we have to calculate the average and save to RR. It spends long time and we target to parallelize to accelerate this part.

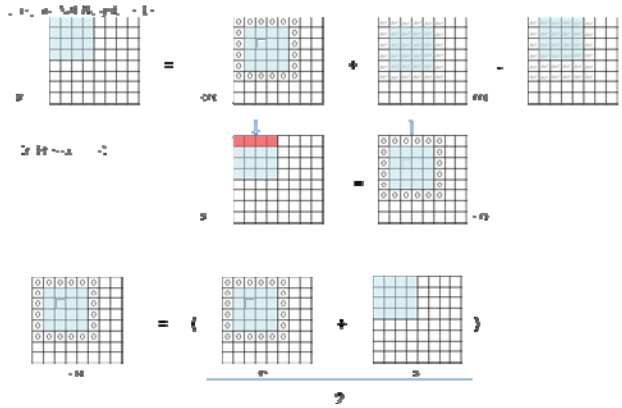

Fig. 3. Part of retinex algorithm

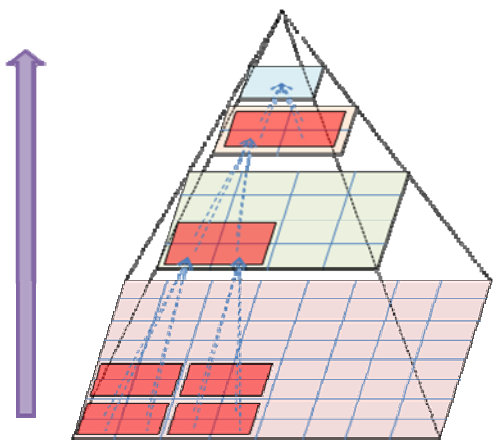

Fig. 4. Enhanced Pyramid

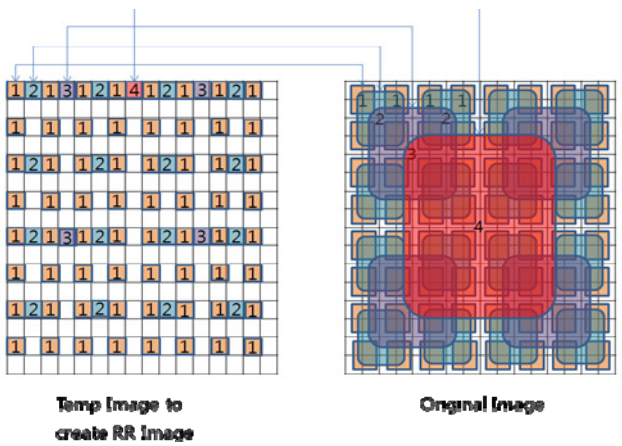

Fig. 5. Create Temp Image to create RR Image

At the Fig. 2. each layer's RR image is calculated with the Original image. Just divide original image and each grid block's average is calculated and store the average to the RR image. But Fig. 4. shows that bottom layer's divided grid blocks that have 4 pixels. The average of a grid block is become a pixel of next layer's RR image. As these steps, each 4 pixels average is became a pixel of upper layer's RR image. And this RR image's 4 pixels are became a pixel of next layer's RR image's pixel. Fig. 5. shows that the Original Image divided grid block and each grid block have 4 pixels. First layer's grid block calculated and save the result to the RR Image.

"AS(ty, tx)" is a shared memory and it copy input image(g_temp)'s data. In the thread block, the thread index is "tx" and "ty". The "blocksize" is thread block size. We set the thread block size as 16, cause to get the maximum performance. According to NVIDIA, using warp size (32) or half warp size (16) is the way to get a good performance. And "ImageSize" is the input image's size. "add" is distance of each grid blocks. At the layer 0 is 1, layer 1 is 2, layer 2 is 4. It is increased 2^n every layer. "add_save" is also a distance where is going to be saved and it is derived as (add -1)/2. Using these parameters, we saved average result temp image in Fig. 5, and next layer, load from temp image and calculate.

If the input image's size is 512*512, "imageSize" is 512 and the pyramid has 9 layers. And each layer are divided to grid blocks, at the layer 0, it can be divided 1024 (32*32) grid blocks and each grid block size is 16*16 and at the layer 4, it can be divided 16(4*4) grid blocks and each grid block size is 16*16. The reason, why we set the grid block size 16*16, is to map the thread block. If the data is smaller than 16*16, we reduced the thread block size and grid block size. There is 1 grid block that is reduced.

```
//thread index
unsigned int tx = threadIdx.x;
unsigned int ty = threadIdx.y;

//block index
unsigned int bx = blockIdx.x;
unsigned int by = blockIdx.y;

//shared memory
__shared__ float As[16][16];

//data copy from input image to shared memory
AS(ty, tx)
= g_temp[by*blocksize*imageSize + bx*blocksize
+ add*ty*imageSize + add*tx + add_save];
__syncthreads();

//calculating average
if(((tx)%(2))==0&&((ty)%(2)==0))
{
sum = AS(ty, tx) + AS(ty, tx+1)
+ AS(ty+1, tx) + AS(ty+1, tx+1);
__syncthreads();
g_temp[by*blocksize*imageSize + bx*blocksize
+ add*ty*imageSize + add*tx + add-1] = sum/4;
__syncthreads();
}
```

Fig. 6. Average Calculating CUDA code

Fig. 6. is a CUDA code to calculating the average of original image's grid block. It is executed each layers as Fig. 4. We use shared memory and all thread move data from original image to the shared memory. And the thread what is located in the (0,0), (0,2), (0,4)…(14, 10),(14,12),(14, 14), calculate with neighbor data((0,0) thread calculate with (0,0), (0,1), (1,0), (1,1)) and save to the 2d plane "g_temp" image as Fig. 5. Next step, collect these data from g_temp, move to shared memory and calculate average.

We try to accelerating this multi-resolution pyramid architecture. First, in the normal calculating pyramid, every layer need calculating the original image's all data, but our enhanced pyramid doesn't need to calculate all data. It just needs the output data from before layer. Second, we parallelized the average calculating step with CUDA on GPGPU. 16*16 threads in a thread block calculate at the same time.

IV. EXPERIMENTAL RESULT

We have tested the performance of the retinex algorithm that has implemented in C and CUDA on 1 PC. The CPU is Intel(®) Core™2 Quad CPU Q9400 @ 2.66Ghz and main memory is 5GB ,OS is Windows 7 64bit. The GPU is NVIDIA GeForce 9500GS and it has 512MB global memory and 4 multiprocessors and 32 cores, 16Kbyte shared memory per block, maximum number of threads per thread block is 512. Clock rate is 1.37GHz.

We tested and checked time C code and CUDA code. Not the all code's elapsed time. We tested 4 different size images, 256*256, 512*515, 1024*1024, 2048*2048. We just checked the time of Average calculating and create RR image. It cause we implemented this part to accelerate. As Fig. 7, there are dramatically improvements of performance. Normal graph shows that it takes long time to create RR image. It takes 22237036 microseconds and Accelerating graph show that it takes under 1000 microsecond when the input image size is 2048*2048. When the input image size is getting larger and larger, the performance efficiency is getting better and better. HD image is huge and it takes long time to image processing but we can reduce the time to process. It seems that it's possible that real time processing HD images and HD video.

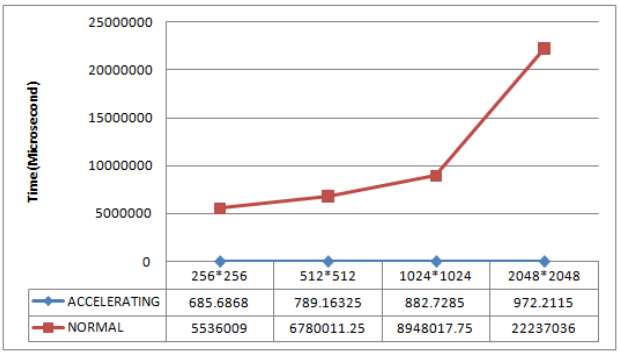

Fig. 7. Experimental Result

V. CONCLUSIONS AND FUTURE WORK

Real-time HD image and video processing need high computing power and times but as we provided the accelerating the retinex algorithm, it does not need the expensive computing systems (such as HPC or, fast clock speed CPUs). We can render and process the HD images and video with CUDA on GPGPU (cheaper than CPUs). Its more economic and efficient, as the result of our experiment.

We tried to improve the retinex algorithm with CUDA and we changed just the Pyramid what is the step to calculate average of original image and to create RR image. The result of accelerating the retinex algorithm, it shows dramatic performance increase.

There are more parts what can be enhanced and changed. We are going to enhance these parts (ex. Convolution with center and surround) and it will make more fast and efficient result. Also there are many way to optimize parallel program (loop unrolling, branch prediction...etc). After optimization, the performance also will be more increased. In the future, we are going to enhance the retinex algorithm and test HD video processing in real-time.

REFERENCES

[1] NVIDIA_CUDA_BestPracticeGuide_2.3
[2] B. Funt, F. Ciurea, and J. McCann, "Retinex in MATLAB" Journal of Electronic Imaging, vol. 13, no. 1, pp. 48-57, 2004
[3] Naga K. Govindaraju, Scott Larsen, Jim Gray, and Dinesh Manocha, "A Memory Model for Scientific Algorithms on Graphics Processors," Supercomputing 2006, Nov 2006, Florida, USA
[4] Mattson, Sanders, Massingill "Patterns for parallel Programming "
[5] Sain-Zee Ueng, Melvin Lathara, Sara Baghsorkhi, and Wen-mei Hwu, "CUDA-lite: Reducing GPU Programming Complexity,"LCPC 2008, Aug. 2008, Canada
[6] Z. Rahman, D. J. Jobson, and G. A. Woodell, "Retinex processing for automatic image enhancement," Journal of Electronic Imaging, vol. 13, no. 1, pp. 100-110, 2004
[7] Frank, J., McCann, J. "Method and Apparatus for Lightness Imaging," *US Patent*, No. 4,384,336 (1983)
[8] McCann, J. "Lessons Learned from Mondrians, Applied to Real Images and Color Gamut," *Proc. of IS&T/SID 7th Color Imaging Conference*. (1999), pp.1-8
[9] Horn, B.K.P. : Determining Lightness from an Image.*Computer Graphics and Image Processing*, Vol. 3 (1974) pp.277-299
[10] Kimmel, R., Elad M, Shaked A., Keshet R., Sobel I., "A Variational Framework for Retinex," International Journal of Computer Vision, Vol. 52, Issue 1, (2003), pp.7-23
[11] Gilchrist, A. L., "Perceived lightness depends on perceived spatial arrangement," Science, Vol. 195 (1977) pp.185-187
[12] Boyaci, H., Maloney, L. T. & Hersh, S., "The Effect of Perceived Surface Orientation on Perceived Surface Albedo in Binocularly Viewed Scenes," Journal of Vision, Vol. 3, (2003), pp.541-553.
[13] Yamauchi, Y., Uchikawa, K., "Depth Information Affects Judgment of the Surface-Color Mode Appearance," Journal of Vision, Vol. 5, (2005), pp.515-523
[14] Bloj. M.G., Kersten D., Hurlbert A.C., "Perception of Three-Dimensional Shape Influences Colour Perception through mutual Illumination," Nature, Vol. 42, (1999), pp.23-30
[15] Yang J. N., Shevell S. K. "Stereo Disparity Improves Color Constancy," Vision Research, Vol. 42, (2002). pp.1979-1989
[16] Adelson E.H.. "Lightness Perception and Lightness Illusions," New Cognitive Neuroscience , 2nd ed., MIT Press,(2000), pp.339-351

Hyo-Seok Seo
He is a master student in department of electrical and electronic engineering from Korea University of Technology and Education, Cheon-An city, Korea. He is researching parallel computing with CUDA on GPGPU.

Oh-Young Kwon received his PH.D degree in computer science from Yonsei University, Seoul, Korea in 1997. He had worked at ETRI as a senior researcher from 1997 to 2000. He is currently an associate professor of the School of Computer Science and Engineering at Korea University of Technology and Education, Cheon-An city, Korea. His research interests include high performance computing, cluster and grid computing and home networking middleware