Volume 5, Issue 7, July 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Parallel Image Compression using CUDA with Median Filter

Bhavesh Vekariya
Department of Computer Engineering
PG Student, RK University, India

Prof. Chhaya Patel
Department of Computer Engineering
Assistant Professor, RK University, India

Abstract— Images are becoming bulkier in size. This size is not physible for use with the many of the application. To make usable this bulkier image size must be reduced by image compression algorithm. Image Compression algorithms are resource and time conservative. Time can be reduced by the providing more resources to the compression algorithm. By Parallel execution this goal can be achieved. CUDA (Compute Unified Device Architecture) provides a platform for the parallel implementation of the algorithm. With the help of GPU (Graphical Processing Unit) parallel execution with CUDA is possible. With the Help multicore support GPU provides a parallel execution environment. JPEG2000 is an efficient algorithm for the image compression, and it is a wavelet based transformation technique. Lossy image compression reduces the quality of the image. By applying the filter the image noise can be reduced and quality will be enhanced. Median filter before the compression will reduces color noise. Using filter and parallel compression all over the efficiency of the image compression algorithm is improved approximately 10% in time and about 1% in quality measure PSNR and MSE.

Keywords— GPU, CUDA, Multicore, Shared Memory, Multithreading, DCT, DWT, BWT, EBCOT, MTF, DAST, JPEG2000, Median Filter

I. INTRODUCTION

JPEG2000 is a wavelet transform based compression algorithm. It is successor version of the JPEG. Quality of image after compression is better improvement in JPEG2000 compare to the old version. Before the process of compression image is divided in to tiles, and this all tiles are processed independently. It will make the every tile independently process able. So, due to this task can be also possible to divide among the multiple threads and it becomes helpful in parallel execution of the algorithm. High- resolution images are easily divided and compressed in parallel. [6]

Flow of the JPEG2000 algorithm is as shown in Fig1. It starts from MCT (Multi Color Transform) which is required because the JPEG2000 processes color components independently, this will remove color dependency between the components before the actual process of the compression. Then the DWT (Discrete Wavelet Transform) is applied on the image data, it is the actual process of the compression. Compressed data is scalar quantized in lossy mode, after data is divided into the code block. Each code block is independently processed by the EBCOT (Embedded Block Coding with Optimized Truncation). EBCOT is process of two stages known as Tier-1 for the BPC (Bit Plane Coding) and context adaptive AE (Arithmetic Encoding), Tier-2 is for the rate-distortion optimization and bit stream layer formation. Execution time of JPEG2000 for serial implementation over the single core processor is very high due to all above processes are executes in serial mode. [2][7]

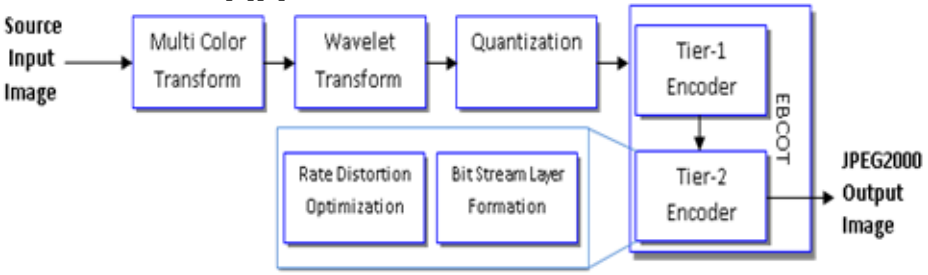


Fig. 1 JPEG2000 Encoding Flow

II. GPU COMPUTING USING CUDA

CUDA provides software as well hardware platform for general purpose computing on NVIDIA`s GPUs. Large numbers of threads are run on GPU using parallel architecture of the CUDA. These threads are grouped into the thread blocks. These numbers of blocks are grouped into the grid. Shared memory is used by threads for sharing data between the threads. Shared memory is fast memory then the single global memory this will increases the execution performance. Shared memory is available to all threads under the same thread block. Flow of the work is to copy data from host RAM to global memory of the GPU, then executes the parallel code over the GPU and copy result data back in the host memory. [9]

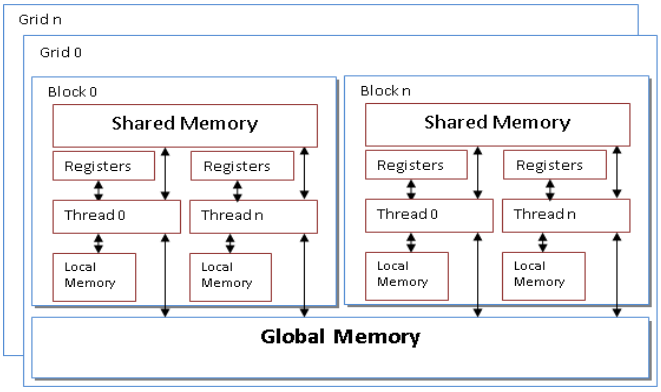


Fig. 2 CUDA GPU Architecture

III. RELATED WORK

In image compression, there are also the work done under the parallel compression algorithm. All though there is no completely parallel algorithm for the image compression. Some of the part of algorithm runs in CPU as serial execution and the part which can be possible to divide task will executes on the GPU in parallel. Most parallel algorithm applied on directly without removing any type of noises in the image. To enhance the image quality by reducing the noise in image by applying the filter. [1][3][4][5] DCT parallel execution is a complex and creates a problem of memory access due to the dependency in data during the processing. [8][14]

IV. PARALLEL BIT PLANE CODER

Bit plane coder processes code block bit plane by bit plane from MSB (Most Significant Bit) to LSB (Least Significant Bit). On every bit plane there are the bits lines which are further grouped into the strips of four lines, these lines are scanned in columns from left to right. There are the three arrays of state variables called σ, σ' and η during coding process by the BPC. All states variables are updated during run time of bit plane scanning by the BPC. During coding process from MSB bit plane to LSB bit plane σ and σ' states are maintained, and after each bit plane η is reset. Fig3 shows the scanning pattern in BPC.

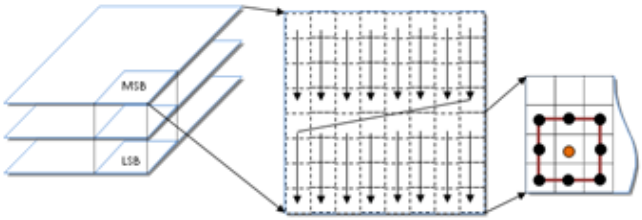


Fig. 3 Architecture Context formation process of BPC in JPEG2000

Fig4 shows the scanning pattern of the BPC. Consider that BPC is processing sample which is known as current sample and rest of all are divided into the two parts visited samples and not visited samples. Already scanned by BPC samples are visited and which are remaining to process are the not visited samples. When BPC starts codding bit plane p is called pre arrays (i.e., σpre, p, σ'pre, p, ηpre, p), and after finishing of BPC it called post arrays (i.e. σpost, p, σ'post, p, ηpost, p). When the BPC codes sample s[x,y], that time context window is formed out by the four neighbors with state information from σpost,p, σ'post,p, ηpost,p and four not-visited neighbors with state information from σpre,p, σ'pre,p, ηpre, p. For example samples s[x,y] in Fig4 can be computed as below:

$$\sum_{H,p}[x,y] = \sigma_{pos,p}[x-1,y] + \sigma_{pre,p}[x+1,y]$$

$$\sum_{V,p}[x,y] = \sigma_{pos,p}[x,y-1] + \sigma_{pre,p}[x,y+1]$$

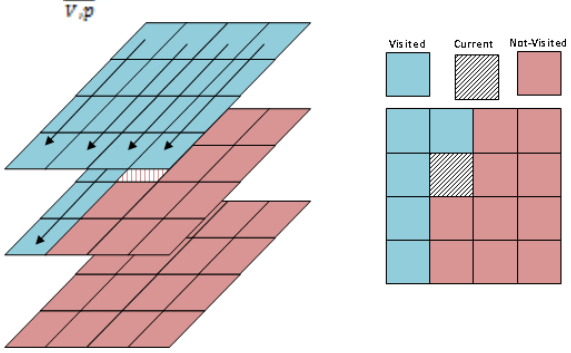


Fig. 4 Scanning pattern in bit plane. Each sample has 4 visited neighbors and 4 not-visited neighbors

V. MEDIAN FILTERING

The median filter is one of the nonlinear digital filtering methods, used for remove noise in the image. Median filter is very use full in image noise removing method because it preserves edges during noise removing task. [13] It is sliding-window spatial filter. It replaces the value of the center pixel with median of the intensity value of neighborhood of that pixel. If considering 3 x 3 neighbors with centered pixel then median filter will replaces the center pixel value with 3 x 3 neighbor's values. [10] [11]

Pseudo code for median filter

```
allocate outputPixelValue[width of  image][height of image]
allocate window[width of window * height of window]
edgex := (width of window / 2) rounded  down
edgey := (width of window / 2) rounded down
for x from edgex to width of image -  edgex
for y from edgey to height of image – edgey
        i = 0
for fx from 0 to width of image
for fy from 0 to height of image
window[i] := inputPixelValue[x + fx – edgex][y + fy - edgey]
           i := i + 1
          sort entries in window[]
outputPixelValue[x][y] := window[window  width * window  height / 2]
```

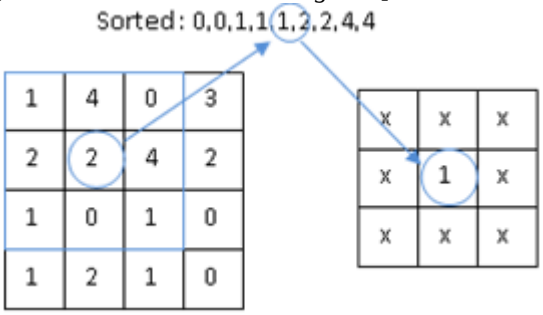


Fig. 5 Median Filtering on 3 x 3 sampling window

By applying median filter the quality measure PSNR (Peak signal-to-noise ratio) is improved and AE (Absolute Error) can be reduced. PSNR calculated from the original and compressed image as below. MSE (Mean Squared) error is used for define the PSNR. It gives a noise from m x n image and MSE is defined as: [12]

$$\text{MSE} = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

The PSNR is expressed in dB and it is defined as:

$$\text{PSNR} = 10 \cdot log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

$$= 20 \cdot log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

$$= 20 \cdot log_{10}(MAX_I) - 10 \cdot log_{10}(MSE)$$

VI. TIER-2 ENCODER

In process of encoding tiles the output array required to initialize with all zeros and this task is being done by the iterating loop up to the total size of the code block. This looping execution takes more time to make all elements 0. This can be avoided by initializing elements in parallel by each thread in the CUDA kernel execution. After successful execution result will transferred back to host memory. During kernel launch configuring it is set to be create the threads equal to the size of the code block. By this ideal thread condition can be avoid and maximum thread utilization can be achieved.

Packet header encoding process is time consuming due to the looping instruction in the execution process. This part of the algorithm is based on differentiated by the indexing in the structure element by the looping variables. This makes possible to implement this part in parallel to execute over the GPU and reduce time consumption of overall algorithm. On CPU process will consume more time because of the repeated tasks on the structure and same can be avoid by executing one instruction on structure by one thread. Each threads are parallel executes. In Calculation of Quantization Defaults (QCD) also the more time consumption is there due to the same problem, and also it can be overcome by parallel execution.

JPEG2000 uses an embedded structure known as a "tag tree" to take benefit from residual redundancy between code-blocks. It is convenient to describe tag tree coding. The Tag Tree is calculated through using function of calculate next level until the root node is reached. This can be also further enhanced to execute over GPU in parallel using CUDA.

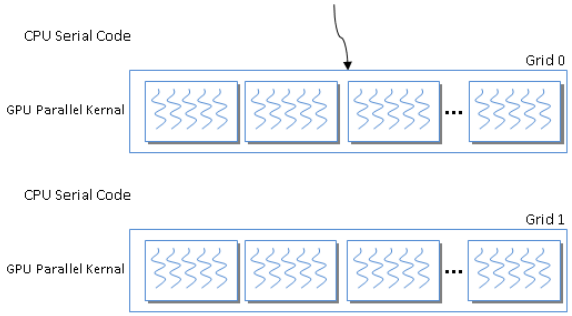


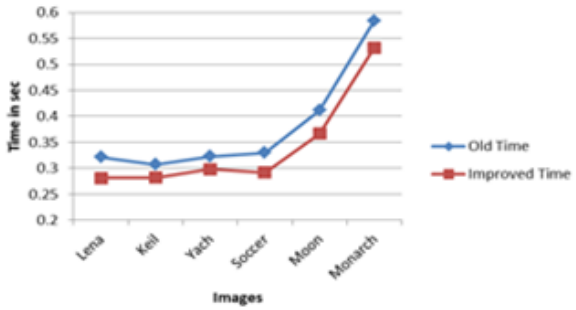
Fig. 6 Thread execution pattern in Kernel

VII. RESULT

JPEG2000 is implemented using Nvidia SDK3.2 running on GEFORCE GT320M GPU. And the host configuration is Intel Core 2 Duo CPU running at 2.10GHz with 4GB of RAM. For the compare old running version of the JPEG2000 in parallel using the same version CUDA. Table1 Show the Time comparison between the old versions of parallel JPEG2000 and improved by implementing non parallel function in parallel over the GPU using CUDA.

TABLE I TIME COMPARISON TABLE FOR IMAGES

Time Comparison in sec		
	Old Time	**Improved Time**
Lena	0.3212	0.281
Keil	0.3074	0.282
Yach	0.3228	0.2978
Soccer	0.33	0.2916
Moon	0.4118	0.3666
Monarch	0.5836	0.5316



\Fig. 7 Time Comparison Graph

TABLE II PSNR COMPARISON BETWEEN ORIGINAL AND COMPRESSED IMAGE

PSNR Comparison in dB		
	Old PSNR	Improved PSNR
Lena	37.615	38.836
Keil	43.037	43.356
Yach	38.878	39.544
Soccer	38.196	38.568
Moon	46.624	46.651
Monarch	40.523	40.848

TABLE III MSE COMPARISON BETWEEN ORIGINAL AND COMPRESSED IMAGE

MSE Comparison		
	Old MSE	Improved MSE
Lena	261588	261269
Keil	200671	194613
Yach	244720	244423
Soccer	245127	244958
Moon	105855	105641
Monarch	389798	388544

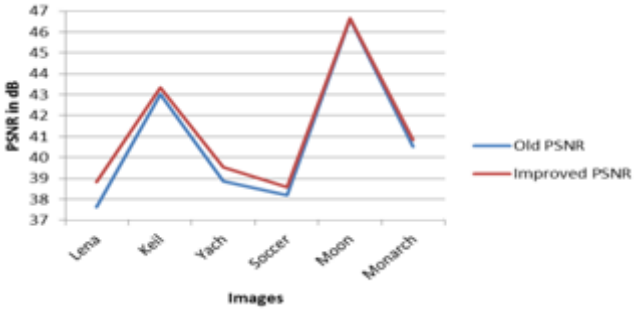


Fig. 8 PSNR Comparison Graph

Number of CUDA cores in processors is the most effective factor for time consumption of JPEG2000 algorithm. Memory utilization during the image compression process is depends on the run time parameters of the algorithm.

VIII. CONCLUSION AND FUTURE WORK

Using the power of parallel processing on CUDA the algorithm can be implemented such that it can be reduce time for processing with help available numbers of threads on the GPU which are very large in number as compare to CPU. And by applying the filter to remove noise in image the quality is being improved and size of image is being further reduces. The process apart from the TIER-2 is being made to run parallel in this paper.

Further more efficient and less time consuming implementation of the algorithm can be possible by applying Unified memory over the CUDA. It will deduct the time which is required for transferring data from Host to Device and further back transferring result after completion of process from Device to Host. So, enhancement on memory access is being possible.

REFERENCES

[1] Kgotlaetsile Mathews Modieginyane, Zenzo Polite Ncube, and Naison Gasela, "CUDA based performance evaluation of the computational efficiency of the DCT image compression technique on both the CPU and GPU", Advanced Computing: An International Journal (ACIJ), Vol.4, No.3, May 2013.
[2] Roto Le, Iris R. Bahar, Joseph L. Mundy, "A Novel Parallel Tier-1 Coder for JPEG2000 using GPUs", IEEE, 2011.
[3] Ritesh A. Patel, Yao Zhang, Jason Mak, Andrew Davidson, John D. Owens, "Parallel Lossless Data Compression on the GPU", IEEE, 2012.
[4] Axel Eirola, "Lossless data compression on GPGPU architectures", 2011.
[5] Mohammad H. Asghari, "Discrete Anamorphic Transform for Image Compression", IEEE Signal Processing Letters, VOL. 21, NO. 7, JULY 2014.
[6] Joaquín Franco, Gregorio Bernabé, Juan Fernández and Manuel E. Acacio, "A Parallel Implementation of the 2D Wavelet Transform Using CUDA", 1066-6192/09 IEEE 2009.
[7] Jiri Matela, "GPU-Based DWT Acceleration for JPEG2000", Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, 2009.
[8] R Pranit Patel, Jeff Wong, Manisha Tatikonda, and Jarek Marczewski, "JPEG Compression Algorithm Using CUDA".
[9] Lih-Jen Kau, Chih-Shen Chen, "Speeding up the Runtime Performance for Lossless Image Coding on GPUs with CUDA", IEEE 2013.
[10] Prateek Kumar Garg, Pushpneel Verma, Ankur Bhardwaz, "A Survey Paper on Various Median Filtering Techniques for Noise Removal from Digital Images", AIJRFANS 2014.
[11] Prabhdeep Singh, Aman arora, "Analytical Analysis of Image Filtering Techniques", IJEIT 2013.
[12] Mamta Juneja,Rajni Mohana, "An Improved Adaptive Median Filtering Method for Impulse Noise Detection", ACEEE 2009.
[13] Pinaki Pratim Acharjya, Soumya Mukherjee, Dibyendu Ghoshal, "Digital Image Segmentation Using Median Filtering and Morphological Approach", IJARCSSE 2014.
[14] Rongyang Shan, Chengyou Wang, Wei Huang and Xiao Zhou1, "DCT-JPEG Image Coding Based on GPU", IJHIT 2015

