Performance Optimization of Transformation Technique - DCT using NVIDIA CUDA

[1]Daxa Vasoya , [2]Prof. Samir B. Patel, [3]Dr. S. N. Pradhan
Institute of Technology, Nirma University
Ahmedabad, Gujarat, India
[1]vasoyadaxa@gmail.com, [2]samir.patel@nirmauni.ac.in,
[3]snpradhan@nirmauni.ac.in
[1]Post Graduate Student,[2]Senior Associate Professor, CSE Department,
[3]Professor, CSE Department

***Abstract*—GPU- graphics processing unit play a major role in many computational environments, most notably those regarding real-time graphics applications, such as image processing. CUDA programmed GPUs are rapidly becoming a major choice in high performance computing and there are a growing number of applications which are being ported to the CUDA platform. However much less research has been carried out to evaluate the optimized performance, when CUDA is integrated with other parallel programming paradigms. The goal of this project is to explore the potential performance improvements that could be gained through the use of GPU processing techniques within the NVIDIA CUDA architecture. In this paper we have implemented DCT Compression algorithm in CUDA. Most of CPU-based implementations of DCT are firmly adjusted for operating using fixed point arithmetic but still appear to be rather costly as soon as blocks are processed in the sequential order by the single ALU. Performing DCT computation on GPU using NVIDIA CUDA technology gives significant performance boost even compared to a modern CPU.**

I. Introduction

A GPU provides platform to write application that can run on hundreds of cores. NVIDIA's CUDA - provides APIs to develop parallel application using the C programming language. CUDA is a software abstraction for the hardware called blocks. Blocks are a group of threads that can share memory. These blocks are then assigned to the many scalar processors that are available with the hardware.

There are several factors that affect the processing speed of the GPU. First one is the number of cores it has. The GPU, unlike the CPU uses less number of registers to store data temporally while they are processing. Therefore, GPU can use more registers to data processing and that is one of the major reason to have many execution cores inside the GPU. Second is the GPU has a faster memory bandwidth between device memory and the processing cores. However, any given algorithm won't gain the performance which is showed in the GPU specification because only the algorithm those are specially designed for the GPU environment will only enjoy the performance of the GPU. So that if the CUDA application has real parallel components, even a single GPU card is capable of delivering significant performance [5].

The DCT is used in JPEG compression and used in image and video encoding/decoding algorithms. In DCT image is divided into blocks and each block is process independently, so block wise transformation is performed in parallel. CUDA provides a natural extension of C language that allows a transparent implementation of GPU accelerated algorithms. Also, DCT greatly benefits from CUDA-specific features, such as shared memory and explicit synchronization points.

This paper illustrates the concept of DCT implementation using CUDA. Performing DCT computations on a GPU gives the significant performance boost even compared to a modern CPU. The given approach performs part of JPEG routines: forward DCT, quantization of each block, inverse DCT. Last section contains the comparison of execution speed performed for CPU and GPU implementations. The performance testing is done using Barbara image (Figure 1). The quality measure is done by means of PSNR, an objective visual quality metric.

Fig. 1. Barbara - test image

II. DCT Basics

Formally, the discrete cosine transform is an invertible function $F : \Re^N \rightarrow \Re^N$ or equivalently an invertible square

N x N matrix [1]. The formal definition for the DCT of two-dimensional sequence of length N is given by the following formula [2]:

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right] \quad (1)$$

The inverse of two-dimensional DCT for a sample of size N x N :

$$f(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \alpha(u)\alpha(v) C(u,v) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right] \quad (2)$$

Separability is an important feature of 2D DCT, and allows expressing equation (1) in the following form:

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \cos\left[\frac{\pi(2x+1)u}{2N}\right] \left\{ \sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \right\} \quad (3)$$

To perform the 2D DCT of length N, the cosine values are usually pre-computed offline. A 2D approach performs DCT on input sample X by subsequently applying DCT to rows and columns of the input signal, utilizing the Equation (3). So the whole 2D DCT process can be represented in matrix notation using the following formula:

$$C(u,v) = A^T X A \quad (4)$$

III. Implementation

Using NVIDIA CUDA technology it is possible to perform high-level program parallelization. Generally, DCT is a high-level parallelizable algorithm and thus can be easily programmed with CUDA.

In this paper we presents two different approaches to implement DCT using CUDA technology. The first approach only demonstrates CUDA programming model benefits and the second approach optimizes the outcome of the first one using the CUDA optimization techniques, which allows creation of fast optimized implementation. The second approach includes 2 methods: one for the floating point data and another for short integer data.

To perform DCT using equation (4) we need to first divide image into block, here our block size is 8x8 pixels as shown in fig 2a. To convert input block into transform domain, two matrix multiplications need to be performed. Each CUDA-block runs 64 threads that perform DCT for a single block. Every thread in a CUDA-block computes a single DCT coefficient. All waveforms are pre-computed beforehand and stored in the array located in constant memory. This array can be viewed as a two dimensional array containing values of basic functions A(x,u) one per column.

2D DCT is performed in following four steps:

1) One pixel from texture memory to shared memory is loaded by thread, then synchronization point to make sure the whole block is loaded at the moment.
2) The thread computes a dot product between two vectors: ThreadIdx.y column of cosine coefficients (which is actually the row of A^T with the same number) and ThreadIdx.x column of the input block. To ensure all coefficients of A^T X are calculated, the synchronization must be passed
3) The thread computes (A^T X A) in the same manner as specifies in step 2
4) Each thread works on single pixel. Once step 3 is completed the whole block is copied from shared memory to the global memory.

IV. Implementation results

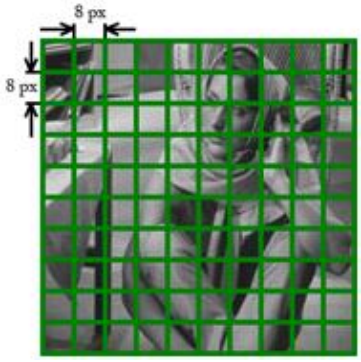
Once DCT is applied on image we need to perform analysis bases on output values. Also we apply quantization to reduce the amount of information that cannot be perceived by the human eye.

we have calculated the processing time for CPU and GPU version of Implementation. Below Table 1 and Figure 3a, 3b, 3c shows the comparison of CPU and GPU version of implementation in terms of processing time. Evolution is perform on different size of barbara image of type .bmp and .png.

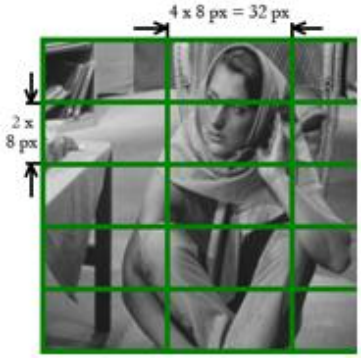
TABLE I
Execution time of each method (in Millisecond)

Method implemented on CPU and GPU	128x128	512x512	1024x728
CPU1	0.382300	6.005800	18.158300
CPU2	0.117700	1.808600	6.093700
GPU1	0.205200	2.109900	8.454901
GPU2	0.081000	0.558000	1.412000
GPU3	0.079000	0.525000	1.461000

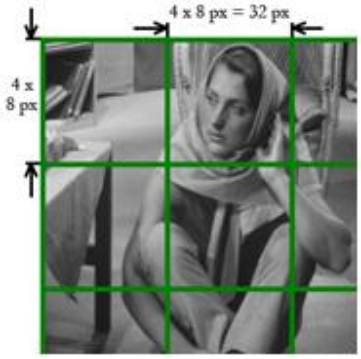
The evaluation can be done in many ways: the speedup rate analysis and consistency checking of CPU and CUDA implementations of the same approach. As for the first, each implementation outputs the pure processing timing to the console window as shown in Table 1 and graph in figure 3a, 3b, 3c. The speedup rate can be measured as the ratio of timings of reference CPU (this method refers as CPU1 and optimized method refers to as CPU2) implementation and of CUDA implementation(this method refers as GPU1, optimized method refers as GPU2 and otimization apply to short data type on GPU is refer as GPU3). The consistency checking is the assurance that both CPU and CUDA implementations of the same approach produce the same output given the same



(a) image split into 8x8 blocks (1^{st} implematation - CPU2)



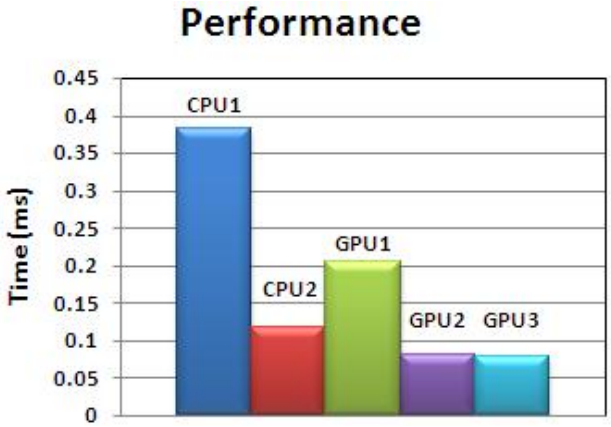
(b) image split into Macro blocks. (2^{nd} implementation - GPU2)



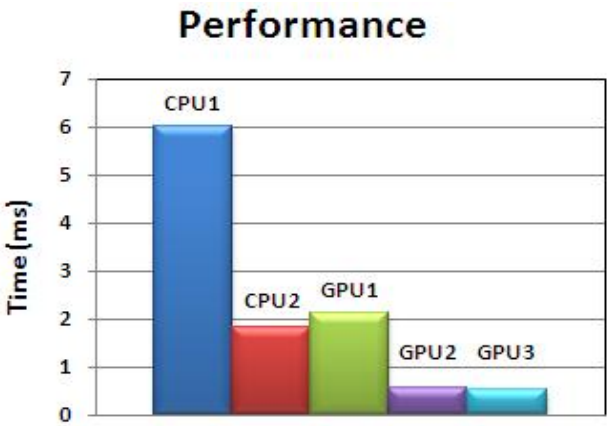
(c) image split into 16 blocks (3^{rd} implementation - GPU3)(short data type)

Fig. 2. image split into block

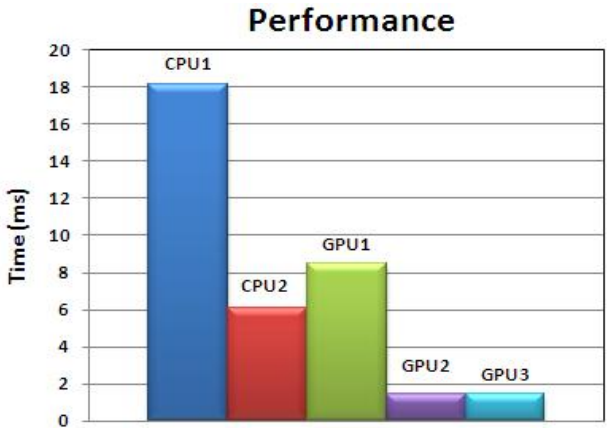
input. The bitwise check of results may fail here because of possible differences in floating point operations sequences in both implementations or due to differences in floating point units. Therefore the consistency checking is performed using the objective image similarity metric PSNR. We have chosen PSNR because it is commonly used to evaluate image degradation or reconstruction quality. PSNR stands for Peak Signal to Noise Ratio and is defined for two images I and K of size M x N as:



(a) Barbara_128 x 128



(b) Barbara_512 x 512



(c) Barbara_1024 x 728

Fig. 3. Performance gain of GPU over CPU for different DCT implementation

$$PSNR(I,K) = 20log_{10}\frac{MAX_I}{\sqrt{MSE(I,K)}} \quad (5)$$

Where I is the original image, K is a reconstructed or noisy approximation, MAX_I is the maximum pixel value in image I and MSE is a mean square error between I and K :

$$MSE(I,K) = \frac{1}{M}\frac{1}{N}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} \parallel I(i,j) - K(i,j) \parallel^2 \quad (6)$$

PSNR is expressed in decibel scale and takes on positive infinity for identical images. In image reconstruction typical values for PSNR vary within the range. PSNR of 50 and higher calculated from two images that were processed on diverse devices with the same algorithm says the results are practically identical.

The consistency checking of CPU (this method refer as CPU1, Optimized method refer as CPU2) and CUDA (this method refers as GPU1, optimized method refers as GPU2 and otimization apply to short data type on GPU is refer as GPU3) implementations of both approaches to DCT implementation is performed in two steps:

1) Table 2 shows PSNR values between the original image and the processed image. It is natural to expect that these values should be similar for both implementations.

2) Table 3 shows PSNR between images processed by CPU and CUDA(GPU) using the same algorithm. which is near about same in our implementation, so our CUDA implementation works properly.

TABLE II
PSNR BETWEEN ORIGINAL AND PROCESSED IMAGES

PSNR between original and processed image	128x128	512x512	1024x728
CPU1	31.272768	32.777073	39.375824
CPU2	31.273190	32.777050	39.375824
GPU1	31.273190	32.777027	39.375835
GPU2	31.272768	32.777061	39.375816
GPU3	31.246510	32.749447	39.372759

V. PERFORMANCE OPTIMIZATION

There are several optimization strategies which we have used here while implementing 2D DCT in CUDA which are as follows.[6]

- **Avoid bank conflicts occur while accessing shared memory.** This issue was resolved by padding each row of the macro block stored in shared memory with one element. The resulting amount of shared memory used per CUDA-block is (MACROBLOCK_WIDTH + 1) x MACROBLOCK_HEIGHT. Such configuration allows simultaneous accessing rows and columns without bank conflicts. This method refers as GPU2 .

- **coalesced global memory Access.** In second implementation the copying from global to shared memory is performed by the same threads that perform DCT. However, this approach doesn't work for short data type (this method refers as GPU3). It causes 2-way bank conflict and uncoalesced global memory access. This issue can be resolved if only half of threads in each block perform moving of 2 short elements as a single 4-byte element.

TABLE III
PSNR BETWEEN IMAGES PROCESSED ON DIVERSE ARCHITECTURES WITH THE SAME ALGORITHM.

PSNR between image Processed on CPU & GPU	128x128	512x512	1024x728
CPU1-GPU1	69.036484	58.073269	63.762222
CPU2-GPU2	69.036484	59.775074	64.196877
CPU2-GPU3	40.442123	42.258053	51.359547

VI. CONCLUSION

Here, we have implemented two new approaches to calculate 2D discrete cosine transform using CUDA technology. Both approaches were implemented for CPU and GPU. The GPU implementations utilize DCT separability, which yields the significant performance boost as compared to a modern CPU. Our implementation resulted in 70% speed up when implementing 2D DCT using CUDA specific performance optimization strategies. The performance testing was held for both approaches and they exhibited good speedup rates while preserving the image quality.Our future work will include implementation of 3D DCT with video sequences[7] and performance analysis of other transformation techniques including DWT.

VII. ACKNOWLEDGEMENT

We would like to thank all the members of NIRMA UNIVERSITY for providing continuous support and inspiration.

REFERENCES

[1] Syed Ali Khayam. *"The Discrete Cosine Transform (DCT): Theory and Application"*. ECE 802 - 602: Information Theory and Coding, March 10th 2003

[2] Tze-Yun Sung, Yaw-Shih Shieh, Chun-Wang Yu, Hsi-Chin Hsin. *"High-Efficiency and Low-Power Architectures for 2-D DCT and IDCT Based on CORDIC Rotation"*. Proceedings of the 7th ICPDC, pp. 191-196, 2006.

[3] R. Kresch and N. Merhav, *"Fast DCT domain filtering using the DCT and the DST"*. HPL Technical Report HPL-95-140, December 1995.

[4] N. P. Karunadasa & D. N. Ranasinghe, *"On the Comparative Performance of Parallel Algorithms on Small GPU/CUDA Clusters"* University of Colombo School of Computing, Sri Lanka,2008

[5] Uday Bondhugula, Chris J. P.Sadayappan,*"Towards Effective Automatic Parallelization for Multicore Systems"* , Department of Computer Engineering University of Toronto, 2008.

[6] Shane Ryoo, Christopher I. Rodrigue, Sara S. Baghsorkhi, *"Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA"*, University of Illinois at Urbana- Champaign,NVIDIA Corporation, 2007

[7] Prof. Samir B. Patel, Mr. Jeet R. Patanji, Mr. Nisarg H. Patel "An Im- plementation of 3 Dimensional DCT for Compression of Video Sequences" at 22 NUCONE-2009. A National conference on Current Trends In Technology, held at Nirma University.

[8] http://forums.nvidia.com/index.php?showtopic=181472

[9] http://www.nivdia.co.in/page/cuda.html