RESEARCH ARTICLE OPEN ACCESS

Performing DCT8x8 Computation on GPU Using NVIDIA CUDA Technology

Jagdamb Behari Srivastava, R. B. Singh, Jitendra Jain
Jawaharlal Nehru Krrishi Vishwavidyalaya Jabalpur

Abstract— In this paper, we have proposed sequential and parallel Discrete Cosine Transform (DCT) in compute unified device architecture (CUDA) libraries. The introduction of programmable pipeline in the graphics processing units (GPU) has enabled configurability. GPU which is available in every computer has a tremendous feat of highly parallel SIMD processing, but its capability is often under-utilized. The two-dimensional variation of the transform that operates on 8x8 blocks (DCT8x8) is widely used in image and video coding because it exhibits high signal de-correlation rates and can be easily implemented on the majority of contemporary computing architectures. Performing DCT8x8 computation on GPU using NVIDIA CUDA technology gives significant performance boost even compared to a modern CPU.
Keywords: - Discrete Cosine Transform, Graphics Processor unit (GPU), Computed unified device architecture (CUDA).

I. INTRODUCTION

The Discrete Cosine Transform (DCT) is a Fourier-like transform, which was first proposed by Ahmed *et al*. (1974). While the Fourier Transform represents a signal as the mixture of sines and cosines, the Cosine Transform performs only the cosine-series expansion. The purpose of DCT is to perform de-correlation of the input signal and to present the output in the frequency domain.

There are several types of DCT [2]. The most popular is two-dimensional symmetric variation of the transform that operates on 8x8 blocks (DCT8x8) and its inverse. The DCT8x8 is utilized in JPEG compression routines and has become a de-facto standard in image and video coding algorithms and other DSP-related areas. Most of the CPU implementations of DCT8x8 are well-optimized, which includes the transform reparability utilization on high-level and fixed point arithmetic, cache-targeted optimizations on low-level.

GPU acceleration of DCT8x8 computation has been possible since appearance of shader languages. Nevertheless, this required a specific setup to utilize common graphics API such as OpenGL or Direct3D. CUDA, on the other hand, provides a natural extension of C language that allows a transparent implementation of GPU accelerated algorithms. Also, DCT8x8 greatly benefits from CUDA-specific features, such as shared memory and explicit synchronization points.

The rest of the paper is organized as follows. In section 2, block diagram of graphic processing unit (GPU) is given. In section 3, explain the 1-D and 2-D discrete cosine transform (DCT). In section 4, Computation of discrete cosine transform on GPU using NVIDIA CUDA technology is introduced. Section 5 & 6 provides the simulation result and conclusion.

II. GRAPHIC PROCESSING UNIT (GPU)

The block-diagram of a modern programmable GPU is shown in Figure 1 [3]. The architecture of GPU offers a large degree of parallelism at a relatively low cost though the well-known vector processing model known as Single Instruction Multiple Data (SIMD). There are two type of process used in GPU system i.e. vertex and pixel processors. The vertex processor performs mathematical operations that transform a vertex into a screen position and the pixel processor performs the texturing operations.

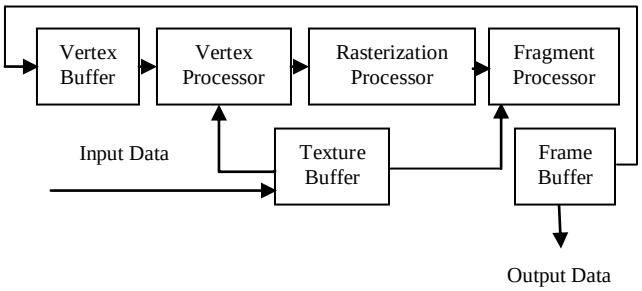

Figure 1: The Programmable Graphics Pipeline

In this sense, GPUs are stream processors. A stream is a set of records that require similar computation and provide data parallelism. The vertices and fragments are the elements in a stream. For each element one can only read from the input, perform operations on it, and write to the output. However, recent improvements in graphics cards have permitted to have multiple inputs, multiple

outputs, but never a piece of memory that is both readable and writable. It is important for general-purpose applications to have high arithmetic intensity; otherwise the memory access latency will limit computation speed. Recent developments in data transfer rate through PCI Express interface to an extent addressed this issue [1], [4]. Programming the GPU in a high level language can be done using Cg from NVidia, OpenGL, etc. [6].

III. DISCRETE COSINE TRANSFORM

Formally, the discrete cosine transform is an invertible function F: $R^N \to R^N$ or equivalently an invertible square $N \times N$ matrix [1]. The formal definition for the DCT of one-dimensional sequence of length N is given by the following formula:

$$C(u) = \alpha(u)\sum_{x=0}^{N-1} f(x)\cos[\frac{\pi(2x+1)u}{2N}].....(1)$$

The inverse transform is defined as:

$$f(x) = \sum_{u=0}^{N-1}\alpha(u)C(u)\cos[\frac{\pi(2x+1)u}{2N}].....(2)$$

The coefficients at the beginnings of formulae make the transform matrix orthogonal. For both equations (1) and (2) the coefficients are given by the following notation:

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} u = 0 \\ \sqrt{\frac{2}{N}} u \neq 0 \end{cases} .....(3)$$

To perform the DCT of length *N* effectively the cosine values are usually pre-computed offline. A 1D DCT of size *N* will require *N* vectors of *N* elements to store cosine values (matrix *A*). 1D cosine transform can be then represented as a sequence of dot products between the signal sample (vector *x*) and cosine coefficient (A^T), resulting in transformed vector ($A^T x$).

A 2D approach performs DCT on input sample *X* by subsequently applying DCT to rows and columns of the input signal, utilizing the separability property of the transform. In matrix notation this can be expressed using the following formula:

$$C(u,v) = A^T XA.....(4)$$

IV. COMPUTATION OF DISCRETE COSINE TRANSFORMATION (DCT)

With advent of CUDA technology it has become possible to perform SIMD (single instruction, multiple data) high-level program parallelization. Generally, DCT8x8 is a high-level parallelizable algorithm and thus can be easily computation by the CUDA.

In applications, such as JPEG and MPEG, DCT is a resource intensive task. In this section two different approaches to implementing DCT 8×8 block using CUDA. In the 1st approaches, demonstrate CUDA programming model benefits, a number of source textures, the associated vertex streams, the render target, the vertex shader and the pixel shader are specified. The source textures hold the input data. The vertex streams consist of vertices that contain the target position and the associated texture address information. The render target is a texture that holds the resulting DCT coefficients. The vertex shader calculates the target position. The 2nd approaches, which is triggered issuing the Draw Primitives call and creating really fast highly optimized kernels. Then the target texture is rasterized and the pixel shader is subsequently executed to perform pixel-wise calculations.

We used a gray scale image data (image size 64 × 64) to compute the DCT using OpenGL API (Application Programming Interface).

- Discrete Cosine Transform for 8x8 Blocks with CUDA in three projects are made
  - Project name 1.cpp (C++ code)
  - Project name2.cu (CUDA code)
  - Project name3.cpp (C++ code)

In the first project name1.cpp include the CImg.h header file; CImg.h defines the classes and methods to manage images in your own C++ code. CImg.h is self-contained and thus highly portable. It fully works on different operating system and is compatible with various C++ compilers. In this project, firstly save the image in .bmp format in project name1.cpp file. After than write the code (C++ code) and calculated the pixel value of the image.

In second project name 2, Copy the image pixel value in project name1.cpp and paste project name2.cu file. There are four kernels are used in this project.

In Equation (4) $C(u,v) = A^T XA$ used two kernel, the first kernel used in multiplier the transform of cosine coefficient and image pixel value $A^T X = k1$. The second kernel used in multiplier the k1 and cosine coefficient.

$$C(u,v) = A^T XA$$

Kernel 1

Kernel 2

Show the pixel value of the input image in Equation (5) and cosine coefficient of discrete cosine transform in Equation (6).

In the same process applied to the inverse discrete cosine transform.

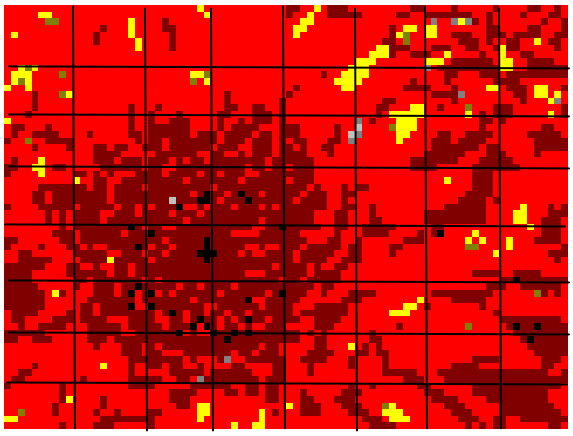

Figure 2: Input image (64 × 64). The input image divided into horizontal and vertical eight equal parts, each part have the dimension of 8 × 8.

One block of the input image pixel value is

X =

29	***38***	***52***	***53***	***33***	***29***	***33***	***33***
24	21	38	49	35	31	36	33
18	16	20	25	31	27	27	32
21	30	30	24	40	34	25	26
15	22	27	22	32	33	33	27
25	29	30	26	31	30	31	33
10	19	22	29	37	34	30	35
16	16	23	33	27	28	26	19

…… (5)

Cosine coefficient of the discrete cosine coefficient is

A=

.35	***.35***	***.35***	***.35***	***.35***	***.35***	***.35***	***.35***
.49	.41	.27	.09	.09	-.27	.41	-.49
.46	.19	-.19	-.46	-.46	-.19	.19	.46
.41	-.09	-.49	-.27	.27	.49	.09	-.41
.35	-.35	-.35	.35	.35	-.35	.35	.35
.27	-.49	.09	.41	-.41	-.09	.49	-.27
.19	-.46	.46	-.19	.19	.46	-.46	.19
.09	-.27	.41	-.49	.49	-.41	.27	-.09

..(6)

A kernel is the unit of work that the main program running on the host computer offloads to the GPU for computation on that device. In CUDA, launching a kernel requires specifying three things:

- The dimensions of the grid
- The dimensions of the blocks
- The kernel functions to run on the device.

The implementation of DCT8x8 by definition is performed using (7). To convert input 8x8 samples into the transform domain, two matrix multiplications need to be performed.

This solution is never used in practice when calculating DCT8x8 on CPU because it exhibits high computational complexity relatively to some separable methods. Things are different with CUDA; the described approach maps nicely to CUDA programming model and architecture specificity. Image is split into a set of blocks as shown on Figure 2, 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.

- Two-dimensional DCT is performed in four steps (considering thread-level):

In first step, a thread with coordinates (ThreadIdx.x, ThreadIdx.y) loads one pixel from a texture to shared memory. In order to make sure the whole block is loaded to the moment, all threads pass synchronization point. In second step, the thread computes a dot product between two vectors: ThreadIdx.y column of cosine coefficients (which is actually the row of $A^T XA$ with the same number) and ThreadIdx.x column of the input block. To ensure all coefficients of $A^T XA$ are calculated, the synchronization must be passed. In third step, the thread computes $A^T XA$ in the same manner as in step second. In four steps, the whole block is copied from shared memory to the output in global memory. Each thread works with the single pixel.

V. SIMULATION RESULT

There were two versions of algorithm sequential and parallel. Both of them was coded in C with using CUDA libraries and run on NVIDIA GeForce GT 630M with 2 GB dedicated VRAM (Video Random Access Memory) graphics card installed on Acer V3-571G, Intel Core i5-3210M 2.5GHz with Turbo Boost up to 3.1 GHz.

The computation of discrete cosine transform (DCT) GeForce GT 630M graphics card solved the day to day increasing demand for the massive parallel general purpose computing. The accelerated version on GPU was astonishingly fast, because it took less time compare to sequential implementation. However, this value is strictly limited to the execution of computation kernel itself and doesn't include any overhead cost.

VI. CONCLUSION

In this paper, we implemented and studied the performance of computing the discrete cosine transform for8×8 blocks with NVIDIA CUDA technology. Discrete cosine transform approaches were implemented for CPU and GPU. We have shown that GPU is an excellent candidate for performing data intensive discrete cosine transform. A direct application of this work is to perform discrete cosine transform in real time digital signal processing and image processing. The performance testing was held for both approaches and they exhibited good speedup rates while keeping objective result quality constant.

REFERENCES

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

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

[3] *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.*

[4] *Simon Green. Discrete Cosine Transform GPU implementation.http://developer.download.nvidia.com/SDK/9.5/Samples/vidimaging_samples.html#gpgpu_dct.*

[5] *K. Fatahalian and M. Houston, "GPUs: A Closer Look", ACM Queue, Vol. 6, No. 2, March/April 2008, pp. 18–28.*

[6] *S. P. Mohanty, N. Pati, and E. Kougianos, "A Watermarking Co-Processor for New Generation Graphics Processing Units", in Proc. 25th IEEE International Conference on Consumer Electronics, pp. 303-304, 2007.*

[7] *V. Galiano, O.Lopez and H. Migallon, "parallel Strategies for 2D Discrete Wavelet Transform in Shared Memory Systems and GPUs," Published Springer Science+ Business Media, LLC 2012.*

[8] *R. Qu and Chunhong Zhang ," High Performance Finite Impulse Response Filter on Graphics Processors," 3nd Int. Conf. on Intelligent Control and information Processing 2012.*

[9] *Wladimir J. van der Laan, Andrei C. Jalba, and Jos B.T.M. Roerdink ,"Accelerating Wavelet Lifting on Graphics Hardware Using CUDA," IEEE Transactions on Parallel and Distributed Systems, Vol. 22, No. 1, January 2011.*

[10] *Saraju P. mohanty, "GPU-CPU MULTI-Core for Real time Signal processing," 978-1-4244-2559-4/09 2009 IEEE.*

[11] *Ing. Vaclav Simek , "GPU Acceleration of 2-D DWT Image Compression in MATLAB with CUDA ," second UKSIM European Symposium on Computer Modeling and Simulation 2008.*