GPU Based Accelerated JPEG Decoder

Mr. Rushikesh Tade
PG Student
Department of Electronic & Telecommunication Engineering
D.Y.P.S.O.E. Pune

Prof. Saniya Ansari
Professor
Department of Electronic & Telecommunication Engineering
D.Y.P.S.O.E. Pune

Abstract

In this work authors have implemented efficient JPEG (Joint Photographic experts group) Decoder on nVidia GPU (Graphic Processing Unit) using CUDA Technology. This decoder is capable of decoding images of Ultra HD resolution with superfast speed GPU is used to assist the CPU for time consuming tasks. In this paper IDCT module which consumes 70 to 80 percent of computation time is implemented on GPU. An asynchronous parallel execution between the CPU and the GPU is used at a same time to improve the JPEG decoder acceleration rate. In the experiment, the JPEG decoder based on the CUDA performs decompression of 2560 x 1600 pixels and below images. Finally the results are shown with respect to different sized images and consumed time for decoding. The results show that this decoder faster in multiple times than the decoder in CPU.

Keywords: CUDA, HUFFMAN Decoding JPEG, RLE

I. INTRODUCTION

In recent year many with the development of new technology data storage cost is also increased. The more familiar example would be the screen resolution of the TV sets. We can see how the popularity of high resolution TV sets is been increasing over the years. Since the data is increased and so do the quality of material. This has resulted in more data to be stored and transferred via communication media. Which has lead us to the various compression techniques for the data to be transferred and stored. Now a days we can find many standards for the image compression. JPEG (Joint Photographic Experts Group) is one them which has very popular for photo- graphic image decoding [7]. In computing, JPEG is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography. The degree of compression can be adjusted in JPEG, allowing a selectable trade- off between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in quality of image. Since it’s very popular in image compression many software and hardware techniques have been developed to improve the speed of the JPEG decoding process. To the hardware resources for the JPEG decoder, Some common efficient ways can be obtained using the ,FPGA (Field Programmable Gate Array) or other ASIC (Application Specific Integrated Circuit) resources, DSP (Digital Signal Processor) [4].With increasing computations speed of GPU along with the CUDA framework a software decoder can be efficiently implemented.

II. CUDA TECHNOLOGY

General-Purpose computing on Graphics Processing Units (GPGPU) is referred to techniques where calculations traditionally done by the Central Processing Unit (CPU) handed over to the Graphics Processing Unit (GPU). Earlier, the GPU was used only to accelerate certain parts of the graphics pipeline, but now it can reduce the CPU load and/or increase the processing throughput for general purpose scientific and engineering computing. A GPU acceleration model is shown in above figure 1.GPUaccelerated computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user’s perspective, applications simply run significantly faster. In order to ease the usage of GPUs for programmers not familiar with the graphics pipeline, the CUDA language was created by NVIDIA. CUDA is a programming interface to parallel architecture for general purpose computing. It is an extension to the C language, with a programming model easily understood by programmers already familiar with threaded applications. This interface is a set of library functions which can be coded as ex- tension of the C language.

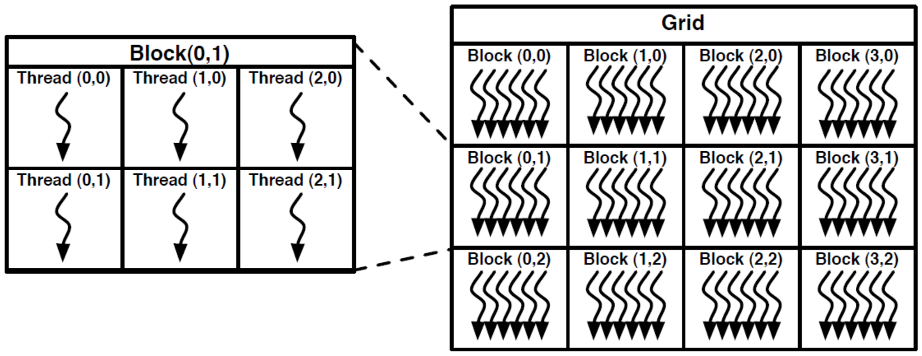

Fig. 1: CUDA Execution Model

A compiler generates executable code for CUDA device. The CPU sees a CUDA device as multi-core co- processor. The CUDA design does not have memory restrictions of GPGPU. One can access all memory available on the device using CUDA with no restriction on its representation though the access times vary for different types of memory[6].The GPU works with a very high amount of threads which run at the simultaneously. Threads are run in different batches called thread blocks [8]. And the CUDA kernel function is essentially based on the block as a unit. In the same block, there are maximally 512 threads. As the threads of the same block are executed into the same streaming multiprocessors (SM), they can share data with each other by shared memory. The number of active blocks is not more than 8 in each SM. Generally, the number of active threads is not more than 768. There are six types of memory in the CUDA programming model: Global Memory, Local Memory, Shared Memory, Register, Constant Memory and Texture Memory [4], as show in Fig. 2. The global memory is located in the memory and it can be both read and write by devices, but it has large memory access latency. However constant memory and texture memory are cached, the access speed is faster than global memory. But they are only read by devices. The register are on the GPU caches, execution units of GPU can access to register with very low latency, the register is the basic unit of the register file and each register file only has 32 bit. When the program has many private variables, it can use the local memory. When the program need data communicate in a block, the shared memory is a good choice, it is also on cache and the access speed of shared memory almost as fast as the register.

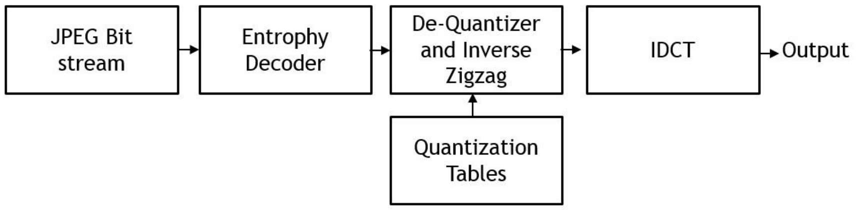

Fig. 2: JPEG Decoding Process on CUDA

So properly using the shared memory to reduce other kinds of memory access is a good way to reduce delay caused by memory access. [5]

A. NVidia Performance Primitives (NPP) Library:

The NVIDIA Performance Primitives library (NPP) is a collection of GPU-accelerated image, video, and signal processing functions that deliver 5x to 10x faster performance than comparable Cupola implementations. Using NPP, developers can take advantage of over 1900 image processing and approx. 600 signal processing primitives to achieve significant improvements in application performance in a matter of hours. Whether there is application are simply replacing CPU primitives with GPU-accelerated versions or integrating NPP primitives with your existing GPU-accelerated pipeline, NPP delivers high performance while reducing development time. [3] We have used this library to boost our performance.
Fig. 5. JPEG decoding process on CUDA

III. JPEG DECODING PROCESS AND IMPLEMENTATION ON GPU

JPEG decoder usually consist of entropy decoder which usually consist of Huffman decoder. .Before the start of the actual image in the decoder's stream of input data, the encoder has placed the tables it used for encoding the image. This means that the decoder can first extract the required tables from its input data stream, in order to process the rest of the input data with the help of these tables. The first step is to reverse the entropy encoding process, which produces the quantized DCT coefficients. With the help of the quantization tables, the DCT coefficients can be DE quantized and finally be transformed back via the IDCT process. Figure 2 shows the de- coding process of the JPEG.[5][13] The Huffman decoder includes the decoding of the run-length encoding (RLE), the entropy coding of the DC(Direct Cur- rent) coefficients and AC(Alternating Current) coefficients.

At the same time, in order to decode the DC coefficients, the current DC variable is to add the former DC variable of a color component unit, and it involves a large number of logical operations. So the Huffman decoding is more suitable to use the CPU to complete [8][12].In this paper, the de-quantizer and the inverse zig- zag operation are done with the Huffman decoder module. The de- quantizer is accomplished by multiplying the value with the quantization table when the coefficients is finished by the Huffman de- coder model. Then the inverse zig-zag is implemented with the help of look up table. Since the hardware configuration of the GPU is not suitable for logic operations. The GPU will bring large overhead in the implementation of the control flow instruction. At the same time, this model is also so simply done in the Huffman model, so this module is more efficiently done by the CPU than by the GPU with the CUDA. The whole 2-D IDCT have more data computation in the JPEG decoder than any other block. The 2-D IDCT in the decoded image is no correlation between each sub-graph, so which makes it computable independently. Therefore, there are different levels of parallelism in the step of the IDCT transform. So we can use the CUDA to finish it. In this work have used CUDA NPP library for the processing of the data in CUDA is been used. This gives users advantage for the processing of the most of the data into the GPU memory. Which has resulted as boost of performance as discussed in the Results. First a bit stream is processed in the CPU and different markers are detected from it. Using this bit stream we have separated out the data such as Huffman tables, Quantization tables and data bits. After that we have performed De-quantization and Huffman Decoding of the data. First, we use the texture memory to store the Huffman decoder data for the input of IDCT module. With this way we can reduce the access latency comparing to global memory. Then we move with the constant memory to store some constant data. Such as: cos matrix and it makes the constant data access latency as fast as the register. Secondly, in the same block, we use the shared memory to store the output data from the row 1-D IDCTs. Since there are eight sub IDCTs executing in the GPU, there are 512 intermediate data for the next 1-D IDCT step. As a result, we put the 512 intermediate data into the shared memory and it will also re- duce the access latency of data. Which has resulted into producing the the DCT data. Preceding that we have used IDCT module for which outputted the pixel values of the data.

IV. RESULTS

We have used images of three size for the decoding process. Input image is shown in 4. The three resolutions which are used in for experimentation are 600 x 522, 1920 x 1080, and 3240 x 2160. The Results are calculated with by calculating the time required for the execution a module. We have also compared the results with the method proposed by Ke Yan [14]. Ke Yan has used two methods for for the implementation of the CUDA decoder it. Synchronous and Asynchronous execution of the CUDA decoder.

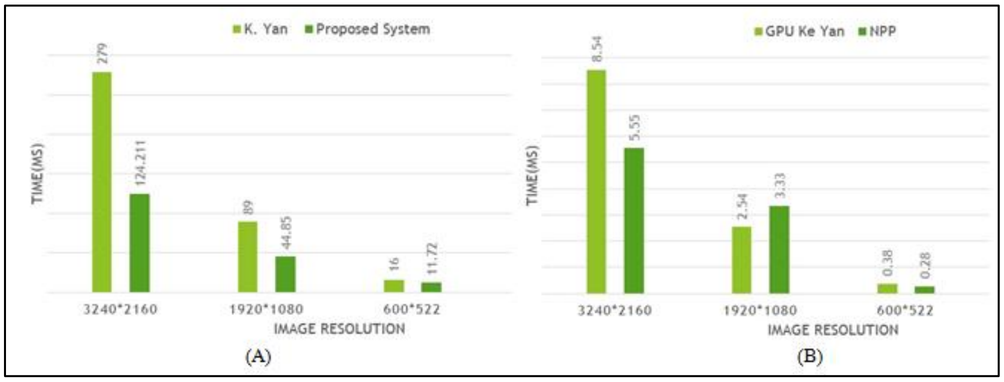

Fig. 3: Performance Evaluation by comparing timings with the previous method for JPEG Decoding Process on CUDA
a) Image decoding b) IDCT module

These results show that with this method we get around 26x more speedup that the previous method and also around 3x more speedup in decoding process. This is mainly because most of our operations are performed in GPU memory.

V. CONCLUSION

In this paper, we introduced the CUDA technology and analyzed the basic JPEG decoder models. By taking advantage of the CUDA technology and NPP libraries, we can speed up the JPEG decoding by much more that speed than the traditional CUDA based JPEG Decoders. The experimental results show that the CUDA- based JPEG decoder can save about 70% time than the CPU-based realization and the model of the IDCT decoder realized on the GPU can be 49 times faster than the CPU-based realization. Our future work will further more improve the JPEG decoder and we will move towards optimization of JPEG re-sizing process.

REFERENCES

[1] Areppim AG, stats of screen resolution eu. http://stats.areppim.com/stats/stats_screenresxtime_eu.htm. Accessed: 2015-04-12.

[2] NVIDIA, gpu accelerated computing. http://www.nvidia.com/object/what-is-gpu-computing.html. Accessed: 2015-04-25.
[3] NVIDIA, nvidia performance primitives. https://developer.nvidia.com/NPP. Accessed: 2015-04-25.
[4] Jahanzeb Ahmad, Kamran Raza, Mansoor Ebrahim, and Umar Talha. Fpga based implementation of baseline jpeg de- coder. In Proceedings of the 7th International Conference on Frontiers of Information Technology, page 29. ACM, 2009.
[5] Jingqi Ao, Sunanda Mitra, and Brian Nutter. Fast and efficient lossless image compression based on cuda parallel wavelet tree encoding. In Image Analysis and Interpretation (SSIAI), 2014 IEEE Southwest Symposium on, pages 21–24. IEEE, 2014.
[6] Shane Cook. CUDA programming: a developer's guide to parallel computing with GPUs. Newnes, 2012.
[7] Pawan Harish and PJ Narayanan. Accelerating large graph algorithms on the gpu using cuda. In High performance computing–HiPC 2007, pages 197–208. Springer, 2007.
[8] Jeong-Woo Lee, Bumho Kim, Jungsoo Lee, and Ki-Song Yoon. Gpu-based jpeg2000 decoding scheme for digital cinema. In Advanced Communication Technology (ICACT), 2014 16th International Conference on, pages 601–604. IEEE, 2014.
[9] Kun-Bin Lee and Chi-Cheng Ju. A memory-efficient pro- gressive jpeg decoder. In VLSI Design, Automation and Test, 2007. VLSI-DAT 2007. International Symposium on, pages 1–4. IEEE, 2007.
[10] Hong Biao Li. A new efficient method for dct8x8 with cuda. In Applied Mechanics and Materials, volume 681, pages 231– 234. Trans Tech Publ, 2014.
[11] CUDA, Nvidia. Compute unified device architecture programming guide. 2007.
[12] Bart Pieters, Charles-Frederik Hollemeersch, Jan De Cock, Peter Lambert, and Rik Van de Walle.Data-parallel intra de- coding for block-based image and video coding on massively parallel architectures. Signal Processing: Image Communication, 27(3):220–237, 2012.
[13] KS Priyadarshini, GS Sharvani, and SB Prapulla. A survey on parallel computing of image compression algorithms jpeg and fractal image compression. IJITR, pages 78–83, 2015.
[14] Ke Yan, Junming Shan, and Eryan Yang. Cuda-based accel- eration of the jpeg decoder. In Natural Computation (ICNC), 2013 Ninth International Conference on,IEEE, 2013.