ULTRA HIGH DEFINITION VIDEO DECODING WITH MOTION JPEG XR USING THE GPU

Bart Pieters, Jan De Cock, Charles Hollemeersch
Jeroen Wielandt, Peter Lambert, and Rik Van de Walle

Ghent University – IBBT,
Department of Electronics and Information Systems, Multimedia Lab,
Gaston Crommenlaan 8 bus 201, B-9050 Ledeberg-Ghent, Belgium

Index Terms— GPU, JPEG XR, massively-parallel, NVIDIA CUDA, ultra high definition

ABSTRACT

Many applications require real-time decoding of high-resolution video pictures, for example, quick editing of video sequences in video editing applications. To increase decoding speed, parallelism can be exploited, yet, block-based image and video coding standards are difficult to decode in parallel because of the high number of dependencies between blocks. This paper investigates the parallel decoding capabilities of the new JPEG XR image coding standard for use on the massively-parallel architecture of the GPU. The potential of parallelism of the hierarchical frequency coding scheme used in the standard is addressed and a parallel decoding scheme is described suitable for real-time decoding of Ultra High Definition (4320p) Motion JPEG XR video sequences. Our results show a decoding speed of up to 46 frames per second for Ultra High Definition (4320p) sequences with high-dynamic range (32-bit/4:2:0) luma and chroma components.

1. INTRODUCTION

JPEG XR [1], formerly known as Windows Media Photo and HD Photo, is a relatively new still-image compression standard, originally developed by Microsoft Corporation and standardized in June 2009. It specifically targets digital photography and features amongst others state-of-the-art compression capability, high dynamic range support, lossless coding support, and full-format 4:4:4 color coding. The compression efficiency of JPEG XR is comparable to that of H.264/AVC intra, yet the standard allows for less complex implementations and is amenable to parallelized implementation using SIMD instructions. The Motion JPEG XR specification, standardized in March 2010 introduced the use of JPEG XR as a video compressor. According to this standard, each video picture is to be separately coded using the intra prediction techniques available by JPEG XR such as the spatial or frequency coding hierarchy and overlapped transforms. One application of Motion JPEG XR can be found in production environments as it offers typical features required for those applications such as high-dynamic range, lossless coding, and random-access to any frame. Furthermore, it allows for degradation-free compressed domain cropping, flipping, and rotation operations [2].

Modern day workstations contain GPUs with hundreds to thousands of processing cores at their disposal. These GPUs are capable of addressing fast on-board memory, typically in the order of gigabytes in size, at rates up to 175 gigabytes per second (NVIDIA GeForce GTX 480 card). The GPU architecture targets data-level parallelism, as current generation hardware only has limited support for task-level parallelism because of the underlying SIMD architecture. Hence, enough parallel tasks are required in order to take advantage of the processing power of the GPU [3].

With high-definition images, a high number of processing cores suggests a high-number of samples or blocks that can be decoded concurrently. Yet current block-based image and video coding standards are not amenable to data-parallel processing. This paper investigates the potential of parallelism of the hierarchical frequency coding scheme used in the JPEG XR standard and describes a parallel decoding scheme suitable for real-time Ultra High Definition (4320p) Motion JPEG XR video sequences on the massively-parallel architecture of the GPU. An overview of the coding tools is given with special attention to maximizing data-parallel processing capabilities while minimizing synchronization. Finally, the parallel decoding scheme is evaluated and decoding speed is presented as the amount of frames per second (f/s) and megapixel per second (mp/s) the decoder can process.

This paper is organized as follows. Section 2 briefly describes previous work done in the field of parallel decoding of image and video coding standards. Section 3 discusses the JPEG XR standard and investigates concurrent processing of the decoding tools. Special attention is given to our solution for concurrent prediction of coefficients in the hierarchical prediction scheme. Section 4 shows and discusses our experimental results. Finally, Section 5 concludes this paper and proposes future work.

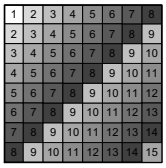

Fig. 1. Prediction of blocks using a wavefront algorithm.

2. PREVIOUS WORK

To the authors' knowledge, no previous work targets the use of the GPU for decoding UHD Motion JPEG XR video sequences. However, an extensive number of publications has investigated decoding the H.264/AVC standard on the GPU [4–6]. These show how data dependencies introduced in the H.264/AVC decoding scheme (especially with the intra prediction and deblocking tools) introduce a high number of synchronization points and do not allow a sufficient amount of parallelism for massively-parallel architectures. A typical way of dealing with dependencies to the left and top is using a wavefront algorithm [4–6], as visualized in Fig. 1. Each block in a certain wave is independent of the other blocks in the wave. Therefore, all coefficients in a wave can be processed in parallel, yet synchronization points are introduced between each wave. Also, parallelism within a wave is limited by the image dimensions, and is typically not enough for massively-parallel architectures. As an example, Pieters *et al.* [5] show that the wavefront algorithm used for parallel intra prediction barely allows for real-time deblocking of a 1080p video picture. The next section shows how JPEG XR eliminates the use of wavefront decoding in all but one decoding tool thanks to its frequency hierarchy and transform coefficient predictions, thereby enabling a more data-parallel approach to decoding than other block-based standards.

3. PARALLEL DECODING OF MOTION JPEG XR VIDEO PICTURES

In this section, all JPEG XR decoding steps are discussed with respect to their optimal parallel decoding algorithm. JPEG XR supports both a spatial and frequency coding mode. The frequency mode is more suited for parallel processing as it groups all coefficients per band per tile. It is this prediction mode that is targeted in this paper.

3.1. Tile Configuration and Entropy Decoding

In JPEG XR, the image is divided into tiles, two-dimensional groups of adjacent macroblocks (MBs), which can be decoded largely independent from other tiles. One way to enable parallel decoding is to limit the amount of dependencies within each tile by decreasing its size. Each processing core can now target a single tile and process the tile, if required, with further limited parallelism. However, Fig. 2 shows this

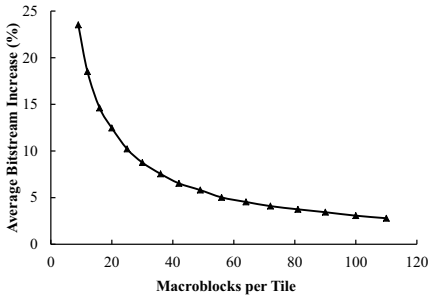

Fig. 2. Impact of tile size on coding rate.

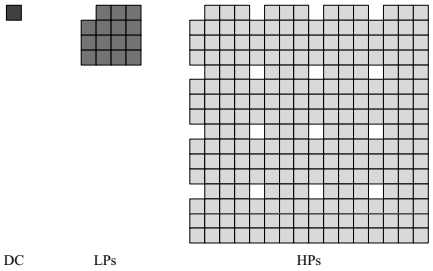

Fig. 3. Frequency hierarchy of the luma component of a MB.

to decrease compression efficiency. For example, the bitstream increases in size with up to 23.5% in case of a small tile configuration of nine MBs per tile. Occupying 480 cores to decode an image with size 1080p on the latest NVIDIA GTX480 graphics cards require tiles of sixteen MBs according to Fig. 2, which creates a bitstream increase of 14.6%. Some GPUs require even more parallism, typically in the order of thousands of threads. It is clear that there is a need to process large tiles efficiently in parallel.

Because of the context adaptivity, concurrent entropy decoding and coefficient scanning is only possible on the tile level. This way, each processor core can be assigned a tile for entropy decoding, coefficient scanning, and can prepare the output data for further GPU processing. Next in the decoding phase, coefficients are predicted and transformed.

3.2. Inter-coefficient Prediction

JPEG XR employs a frequency hierarchy as depicted in Fig. 3. Here, coefficients are predicted in three bands, starting from DC coefficients (one per MB), to LP (Low Pass Coefficients – 16 per MB), and ending with HP (High Pass – 256 per MB) coefficients. As the number of coefficients to calculate increases from DC to HP, so does the opportunity for parallel processing. In other words, the more calculation dependencies, the less deep the sequential calculation path is. This reflects the parallel processing ability inherent to the JPEG XR standard and stands in contrast to other coding standards such as H.264/AVC and VP8 intra. Each coefficient is predicted using its surrounding coefficients in the same co-

efficient band and those of the higher band. Afterwards, these coefficients are reconstructed using the stored coefficient deltas from the bitstream. Next, coefficient dequantization is done on each sample independently. Afterwards, all dequantized coefficients are transformed using the inverse PCT (Photo Core Transform) transformation. A second optional inverse transformation, inverse POT (Photo Overlap Transform), is done on edges of MBs to prevent blocking artifacts.

Each DC coefficient is predicted using either the DC above the current DC, the DC left to the current DC or a combination of the top and left DC. The prediction mode is not described in the bitstream, but is derived from the direction of the similarity of the previously-decoded DC values. Therefore, dependencies are introduced in the DC prediction algorithm as to predict a DC value, all previous DC values need to be decoded. For DC prediction, we propose the use of a wavefront algorithm as visualized in Fig. 1. Each DC coefficient in a wave is independent of the other DC coefficients in the wave and can therefore be processed in parallel on the GPU. As there is only one DC coefficient per MB, the DC coefficient plane is limited in size, e.g. 64×64 for a one megapixel picture. The number of waves is determined by $w + h - 1$, where w and h stand for the width and height of the tile in MBs respectively. For a tile of 1024×1024 in size, this calculates to 127 waves.

The NVIDIA CUDA architecture exposes two levels of synchronization barriers, global and local synchronization. Local synchronization is fast as this is mapped onto operations on a single Streaming Multiprocessor (SM). Each SM exposes 32 hardware threads through 8 Streaming Processors (SPs). Because of the amount of synchronization and the limited concurrency possibilities, DC prediction of one tile is mapped on a single SM. Hence, each SP of the SM will target one 32th of a wave.

As with DC prediction, prediction of LP coefficients uses previously-decoded coefficients, in this case LP coefficients to the left or top of the current LP coefficients. Fig. 4 shows this prediction scheme. It can be seen that every horizontally-aligned LP coefficient (LPa) is predicted from its left neighbor. Alternatively, the coefficient can be not predicted at all. Vertically-aligned LP coefficients (LPb) are predicted using LP coefficients directly above. LPa and LPb calculations share no dependencies and can therefore be predicted in parallel. Also, each row or column of LPa or LPb respectively is not dependent on calculations of the rows or columns above or to the left respectively. Therefore, all rows and columns of LP coefficients can be calculated in parallel. For a tile with size 1024×1024 samples, this means a total number 512 CUDA threads can be initialized, calculating 256 LP rows and 256 LP columns in parallel. Each row or column will be calculated by one SM. Within the SM, all SP will calculate the prediction by using local synchronization.

HP coefficients can be calculated in parallel very easily. Unlike DC and LP prediction, HP prediction uses no other

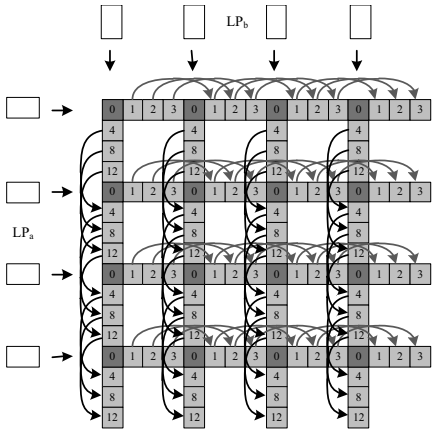

Fig. 4. Prediction of LP coefficients.

previously-decoded HP predictions, but only resides on surrounding previously-decoded LP predictions. As a result, each MB provides a total number of 144 jobs, well sufficient for massively-parallel hardware. Here, we propose a one-on-one mapping of CUDA thread and HP coefficient.

3.3. Inverse Transformation, Quantization, and Colorspace conversion

The Photo Core Transform (PCT) can be well executed in parallel. First, on each 4×4 HP block, four Hadamard transforms are applied. From these blocks, LP coefficients are gathered and are again transformed. Here, each CUDA SP will transform a 4×4 HP block allowing up to 65,536 threads for a 1024×1024 image. Next, a new kernel will use each CUDA thread to transform a 4×4 LP block allowing up to 24,576 threads for the same image. The POT (Photo Overlap Transform) occurs over tile boundaries to counteract block artifacts introduced by the PCT transform. This transform is not discussed in this paper. Finally, a kernel executes inverse quantization and converts the output from the YCoCg to the RGB color space. We refer to previous work in [7] for the most efficient way to do this transformation. The RGB data resident in GPU memory can then be visualized on screen.

3.4. CPU and GPU Pipelined Architecture

Because of the limited opportunities for parallel processing, entropy decoding and coefficient scanning is done on the CPU in system memory. All decoded information is then uploaded to GPU memory and asynchronous decoding on the GPU is commenced. As the decoded output is visualized using the GPU, bus communication is limited to one way. Note that the GPU's pci-express bus interface bandwidth currently limits the maximum throughput in our pipelined scenario. Indeed, in this scenario, throughput is limited to roughly 4:2:0 UHD resolution with 32 bit channels at thirty hertz. Hence, in the next section we focus on the GPU processing speed.

Table 1. Performance results for a NVIDIA GeForce GTX480.

Image Size	Kernel Time (ms)	mp/s	f/s
352x288	0.27	370	3652
720x480	0.48	714	2067
1280x720	0.82	1124	1219
1920x1080	1.41	1475	711
4096x2160	5.21	1697	192
7680x4320	21.63	1534	46

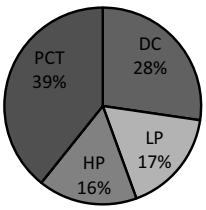

Fig. 5. Workload distribution of the GPU kernels for 2160p images.

4. EXPERIMENTAL RESULTS

Table 1 shows the performance results for JPEG XR-coded images with various sizes using a test system with an Intel i7 950 CPU and an NVIDIA GeForce GTX480 GPU. Each image contains exactly one slice to simulate a worst-case scenario. Images are coded using 32-bit luma and chroma components with 4:2:0 chroma sub sampling. The table shows the GPU time to execute coefficient prediction, transformation, and inverse quantization, the amount of megapixels processed per second (mp/s), as well as the amount of frames processed per second (f/s).

The results show the amount of megapixels processed per second to increase with larger images or tiles as the occupancy of the GPU streaming processors increases. As such, large tiles have become more beneficial to processes in a data-parallel fashion. To predict and transform one 4320p video picture, it takes 21.63 milliseconds. This enables real-time prediction of 4320p at thirty hertz in a Motion JPEG XR scenario. Next, Fig. 5 shows the workload distribution of the different CUDA kernels for a 2160p image. The figure shows how DC prediction and PCT take the most time as they provide respectively the least parallel processing opportunity and process the most data. Notice how the mp/s in Table 1 starts to lower for 4320p as the weight of the prediction of DC coefficients increases. Fortunately, the hierarchical prediction scheme of JPEG XR limits the size of the DC band.

5. CONCLUSIONS AND FUTURE WORK

In this paper we analyzed the parallel decoding capabilities of the Motion JPEG XR video coding standard. We proposed a parallel decoding scheme suited for the massively-parallel architecture of the GPU using the NVIDIA CUDA platform. Using this scheme, we implemented a GPU decoder and showed how JPEG XR images can be decoded on the GPU in a highly-efficient manner. A decoding speed of 1697 megapixels per second was achieved enabling the use of Motion JPEG XR for Ultra High Definition (4320p) at 46 frames per second. Future work will target further optimizations and parallelization of the entropy decoding phase for one or multiple tiles to limit bus communication to the GPU.

6. ACKNOWLEDGEMENTS

The research as described in this paper was funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research-Flanders (FWO-Flanders), and the European Union.

7. REFERENCES

[1] ITU-T Rec. T.832 – ISO/IEC 29199-2, "Information technology – JPEG XR image coding system – Image coding specification." [Online]. Available: http://www.itu.int/rec/T-REC-T.832

[2] S. Srinivasan, C. Tu, S. L. Regunathan, and G. J. Sullivan, "HD Photo: A New Image Coding Technology for Digital Photography," in *Proc. of SPIE: Applications of Digital Image Processing XXX*, vol. 6696, September 2007.

[3] W. chun Feng and S. Xiao, "To gpu synchronize or not gpu synchronize," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, June 2010, pp. 3801–3804.

[4] Z. Zhao and P. Liang, "Data partition for wavefront parallelization of h.264 video encoder," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, May 2006, p. pp. 2672.

[5] B. Pieters, C.-F. Hollemeersch, J. De Cock, W. De Neve, P. Lambert, and R. Van de Walle, "Parallel Deblocking Filtering in MPEG-4 AVC/H.264 on Massively Parallel Architectures," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 1, pp. 96 –100, jan. 2011.

[6] J. C. A. Baeza, W. Chen, E. Christoffersen, D. Dinu, and B. Friemel, "Real-Time High Definition H.264 Video Decode Using the Xbox 360 GPU," in *Proc. of SPIE: Applications of Digital Image Processing XXX*, no. 1, 2007.

[7] W. De Neve, D. Van Rijsselbergen, C. Hollemeersch, J. De Cock, S. Notebaert, and R. Van de Walle, "GPU-assisted decoding of video samples represented in the YCoCg-R color space," in *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*. New York, USA: ACM, 2005, pp. 447–450.