A GPU vs CPU performance evaluation of an experimental video compression algorithm

Stamos Katsigiannis, Vasilis Dimitsas, Dimitris Maroulis
Department of Informatics and Telecommunications
National and Kapodistrian University of Athens
Athens, Greece
{stamos, vdimi, dmaroulis}@di.uoa.gr

***Abstract*—Modern video compression algorithms put significant strain on a system's CPU, especially for video encoding. The ever increasing demands for using video compression algorithms in a wide range of applications necessitate the use of processing components that boost the speed and quality of the video compression algorithm's execution. The vast parallel computational capabilities of modern graphics processing units (GPUs) that usually remain underutilized makes them suitable for handling the processing load for video coding. This paper examines and evaluates the performance benefits of using the GPU over the CPU for an experimental video compression algorithm. An NVIDIA CUDA GPU implementation is evaluated against a traditional multithreaded CPU implementation. Experimental results show that at the highest resolution examined, the GPU approach achieved an impressive speedup ratio of 21.303x against the CPU for the decoding process, while for encoding, the speedup ratio reached 11.048x. Overall results indicate the prevalence of the GPU over the CPU, justified reasonably by the massive parallelism offered by the GPGPU computing paradigm, showing that the GPU should be the architecture of choice for high definition video coding.**

Keywords—video compression; GPGPU; CUDA; contourlet transform;

I. Introduction

The evolution of computer networks and the increase in computing capabilities of modern computers has led to a major shift in the everyday usage of computers and handheld devices capable of general-purpose computations. Applications related to digital video, like video streaming and video conferencing, consume a large percentage of the available network bandwidth while also pushing the available hardware to its limits. Especially in the case of video encoding, even modern powerful CPUs are not capable of providing satisfactory performance in terms of speed for high quality and high resolution digital video.

Modern video compression algorithms (e.g. H264 AVC [1], Dirac [2], VP8 [3], HEVC [4]) are able to provide high visual quality at high levels of compression. Since such algorithms typically exhibit lower computational complexity for the decoding process, the increased computational complexity for the encoding process may not be a problem in the case of stored video content. For such a case, the digital video is encoded once and is then available for decoding either through physical media (e.g. DVDs, Blue-rays) or through video streaming services. Nevertheless, in the case of real-time video encoding (e.g. video conferencing applications), CPUs fail to provide both high visual quality and high compression levels, while achieving real-time performance. Moreover, even when newer CPUs are able to provide high quality real-time video encoding, the increased utilization of the CPU reduces the multitasking capabilities of the system used.

The Graphics Processing Unit (GPU) that exists in modern computer systems and usually remains underutilized seems like an ideal candidate for the solution to this problem. The advent of GPUs along with user-friendly APIs, such as the NVIDIA CUDA [5] and OpenCL [6], expanded their employment in general-purpose computing. As a consequence, inherently parallel algorithms can be considerably accelerated if they are computed on the GPU.

Considering the computational capabilities of modern GPUs and their availability on everyday computer systems, recently proposed video compression algorithms are designed to be able to take advantage of the GPU computing power. The High Efficiency Video Coding (HEVC) [4] algorithm of the MPEG & VCEG Joint Collaborative Team on Video Coding (JCT-VC), that has been designed to replace the H264/AVC [1] standard, is the most notable example. This trend indicates that video coding on GPUs will become more common in the future and will offload the CPUs from video-related heavy computations.

In this work, the authors examine and evaluate the performance benefits of video encoding and decoding on the GPU compared to the CPU for an experimental video compression algorithm that is aimed on providing real-time scalable video coding for applications like video conferencing and has been designed for computation on the GPU. Two optimized implementations of the aforementioned algorithm were created for the performance evaluation: a) one employing the NVIDIA CUDA architecture and b) a traditional multithreaded CPU implementation.

The proposed implementations are evaluated on a system equipped with a CPU and an NVIDIA GPU that had very close release dates, in order to provide a "fair" performance comparison. Performance was measured in terms of execution times and is presented both in absolute execution times and as speedup ratios of the GPU over the CPU.

The rest of this paper is organized in three sections. Section II provides a brief description of the video compression algorithm utilized in this work along with descriptions of the GPU and CPU implementations. The performance evaluation of the proposed implementations is presented in Section III, whereas conclusions are drawn in Section IV.

II. THE CONTOURLET VIDEO COMPRESSION ALGORITHM

A. Algorithm overview

The contourlet video compression algorithm examined in this paper is an upgraded version of the proof-of-concept scalable video compression algorithm proposed by Katsigiannis *et al.* in [7] and [8]. It supports scalable video streams that can include multiple resolutions of the video, while offering enhanced visual quality and more eye-friendly distortion at very high compression levels. The algorithm was designed for computations on the GPU and specifically for the NVIDIA CUDA architecture and aimed to be a low complexity alternative for applications that demand real-time video encoding and decoding, e.g. video conferencing. Experimental evaluation provided promising results, indicating that a GPU-based video compression algorithm can provide significant performance advantages for everyday computer systems that underutilize the available computational power of the GPU, while maintaining satisfactory visual quality and compression levels.

The examined contourlet video compression algorithm can be summarized as follows: Considering that the raw video frames are coded in the RGB color space with 8 bits depth per channel, the first step of the algorithm is the transformation from the RGB to the YCoCg [9] color space. The YCoCg is a color space similar to the widely used YCbCr that offers some advantageous characteristics and has been already successfully used for compression applications, e.g. [10], [11]. Then, chroma subsampling is applied on the chrominance channels (Co, Cg) and motion estimation is applied on the luminance channel (Y) by utilizing a full-search block matching algorithm in order to calculate the motion vectors between the current and the previous frame.

The next step is the decomposition using the contourlet transform (CT) [12]. The CT is a reversible transform that offers multiscale and directional decomposition. At each scale an image is decomposed into a downsampled lowpass version of the original image and the directional subbands that contain the supplementary high frequencies. The final result of CT decomposition is a lowpass image and the directional subbands of each scale. The CT is utilized in this compression algorithm in order to provide scalable video streams, i.e. streams containing more than one resolution that can be decoded separately from each other. Moreover, the sparseness of the subbands makes them ideal candidates for compression after further manipulation.

At the next step, the number of non-zero valued elements of the CT directional subbands is reduced by means of quantization, and are then rounded to the integer. This procedure effectively reduces the number of non-zero-valued CT coefficients leading to more compressible data. Then, if the frame is a predicted frame (p-frame), all its components are computed as the difference between the motion-compensated prediction of the components of the current frame and the respective components of the previous frame, and the zero-valued elements of the lowpass CT component are run-length encoded. Then, the zero-valued elements of the directional subbands of both keyframes and p-frames are run-length encoded and each CT component is finally encoded separately using the DEFLATE [13] algorithm.

The reverse procedure is followed for decoding a video frame. The chrominance samples dropped by subsampling are computed by using bilinear interpolation from the four neighboring pixels. The INFLATE [13] algorithm is used for decoding the DEFLATE encoded data, while the reverse procedures for the other steps of the algorithm are evident.

B. GPU implementation

The NVIDIA CUDA architecture was selected for the GPU implementation due to its simplicity and the optimized API it offers. Specifically, the C for CUDA programming language was used for the implementation, which is an extension to the C programming language. Resource allocation is done according to the CUDA guidelines in order to exploit the advantages of the CUDA scalable programming model [14], i.e. code is expected to run faster if a GPU with more computing cores is utilized.

In order for a frame to be encoded, the frame is first transferred from the main memory of the system to the GPU device memory. Computations are then all performed on the GPU until before the run-length encoding (RLE) step where data is transferred back to the main memory and the last two steps of the algorithm are computed on the CPU. Run-length encoding and DEFLATE are inherently serial algorithms that cannot be efficiently parallelized. Computing them on the GPU would lead to reduced performance. It is possible to compute the run-length encoding step on the GPU by diving data into blocks and by encoding each block in parallel. Nevertheless, this would lead to sub-optimal compression and would effectively increase the bitrate of the encoded video and is thus avoided. Fig. 1 depicts the block diagram for encoding a video frame using the proposed implementation.

In the CUDA architecture, GPU threads are organized into a grid of blocks of threads. Blocks of 32x32 threads were used in the proposed implementation, whereas the size of the grid depends on the actual size of the input image and the nature of the computations performed at each step. A CUDA thread was assigned for computing the result for each pixel of the input frame or for each element of each CT component at the following steps of the video compression algorithm: a) RGB to YCoCg transform, b) chroma subsampling, c) quantization, d) rounding, f) computing the components difference for the p-frames, as well as for the inverse operations at the decoding stage.

The full search motion estimation algorithm employed in this algorithm is an exhaustive search algorithm that significantly increases the computational complexity. Nevertheless, it can be efficiently computed in parallel using the capabilities of the GPU. For the motion estimation procedure, the input frame is divided into blocks of pixels. As a consequence, a CUDA thread is assigned to examine each possible motion vector for each block of the input image. For computing the motion compensated (MC) prediction, one CUDA thread is assigned to each block of each CT component of the frame.

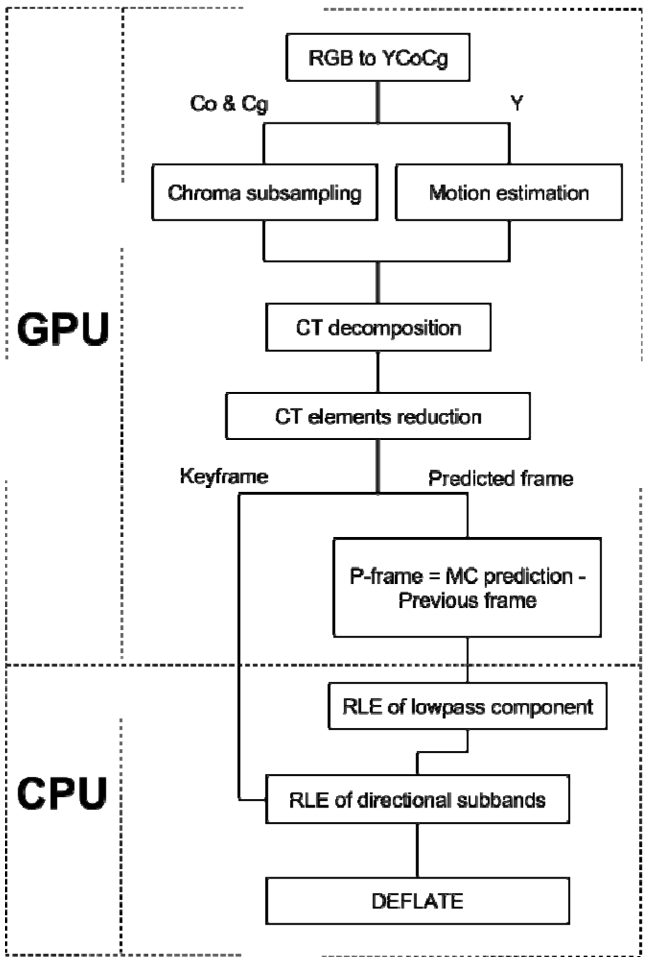


Fig. 1. Block diagram for the encoding process of one frame using the proposed GPU implementation. CT, RLE and MC stand for "contourlet transform", "run-length encoding" and "motion compensation" respectively.

The most computationally intensive part of the CT is the 2D convolutions needed for applying the filters used for decomposition and reconstruction. 2D convolutions can be efficiently computed in parallel by means of the Fast Fourier Transform (FFT), as known from the related signal processing theory. The CUDA API offers "cuFFT", a very fast and efficient library for parallel FFT computation on the GPU. The proposed implementation utilizes the "cuFFT" library for both CT decomposition and reconstruction, and the implementation and resource allocation follow the reference implementation provided by the CUDA API [15].

Finally, for the run-length encoding and decoding, as well as for the DEFLATE and INFLATE steps, one CPU thread is assigned for the encoding and decoding of each CT component, with the number of concurrent threads executed set as the number of CPU processing cores.

C. *CPU implementation*

A CPU implementation for multi-core CPUs was also developed, using the C programming language, in order to provide an objective performance comparison. A series of for-loops that can be effectively parallelized were computed by assigning a number of CPU threads for the parallel computation of the iterations of the loop. The proposed implementation follows the structure of the aforementioned GPU implementation for partitioning tasks for parallel computations. The only difference is the number of threads assigned for each task which is limited by the number of processing cores available to the system's CPU.

III. EXPERIMENTAL EVALUATION

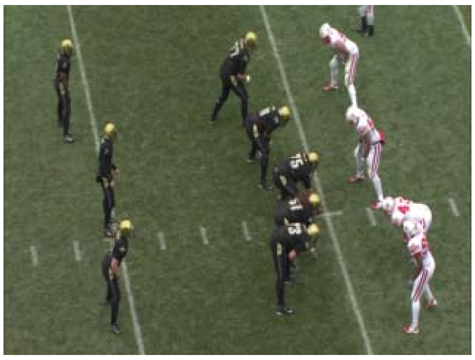
Three publicly available video sequences [16], each exhibiting different levels of motion, were used for the performance evaluation of the contourlet video compression algorithm. The three video sequences (namely "SpeedBag", "TouchdownPass" and "RushFieldCuts") were each resized from their original resolution of 1920x1080 pixels to three lower resolutions in order to create a dataset of twelve video sequences (including the originals). The resolutions included in the dataset were the 320x240 (QVGA), 640x480 (VGA), 1280x720 (HD 720p) and 1920x1080 (HD 1080p) and were selected by taking into consideration the most usual everyday applications of video encoding and decoding, i.e. video conferencing, video playback on portable devices, high definition video playback, etc. Sample frames from the aforementioned video sequences are shown on Fig. 2.

The performance of the video compression algorithm was evaluated both for encoding and decoding. The encoding and decoding parameters for the evaluation were set as follows: a) the video stream contained the original as well as half that resolution, b) the chromatic channels were stored at a quarter of their resolution, c) the interval between keyframes (intra-coded frames) was set to 10 frames, d) the search distance for motion estimation was set to 8 pixels, and e) the quantization parameter for reducing the non-zero valued CT directional subbands elements was set to 10. Using this parameters, the algorithm achieved an average Y-PSNR of 41.57 dB for the three video sequences used.

A system equipped with a CPU and a GPU released on the same period was utilized for the performance evaluation of the video compression algorithm's implementations. Both devices are considered as high-performance equipment. The details of the hardware specifications for the system are summarized in Table I.

The execution times reported in this work are the averages of ten (10) runs for each video sequence. The execution times for each video sequence slightly differ due to the different levels of motion in each video that affect the performance of the motion estimation step of the video compression algorithm. As a result, the average times between all the video sequences are reported in order to offer a more balanced and reliable performance evaluation of the algorithm. It must be noted that the execution times reported do not include the time needed for loading the video frames from the hard disk to the main memory. Nevertheless, the data transfers between the main memory and the GPU device memory have been taken into account when measuring the execution time. Moreover, since the last two steps of the video compression algorithm are always performed on the CPU, as explained in Section II.B, the execution times referring to the GPU approach also include the computations performed on the CPU for those steps. Random computations were also performed before the measurement of the execution times in order to allow the CPU and the GPU to "warm up".

(a)



(b)

(c)

Fig. 2. Sample frames from the video sequences used for the evaluation. (a) Speedbag, (b) TouchdownPass, and (c) RushFieldCuts.

TABLE I. HARDWARE SPECIFICATION OF THE EVALUATION SYSTEM

CPU	GPU
AMD Phenom^{TM} II X4 965 4 cores, @ 3.4 GHz 8 GB RAM	NVIDIA Tesla C2070 (Fermi architecture) 6 GB GDDR5 RAM 448 CUDA Cores

The GPU and CPU execution results are summarized in Table II in the form of speedup ratios, i.e. the ratio of the CPU execution time to the GPU execution time, which is a measure that actually shows how much faster GPU performs over CPU for the specific algorithm's implementation. Furthermore, these figures are depicted in Fig. 3 in graphical form. The absolute execution times are presented in Tables III and IV for the encoding and decoding processes respectively.

TABLE II. THE AVERAGE SPEEDUPS PER FRAME OF THE GPU APPROACH AGAINST THE CPU APPROACH FOR THE ENCODING AND DECODING PROCESSES

	320x240	640x480	1280x720	1920x1080
Encoding	2.808	6.230	9.493	11.048
Decoding	3.330	8.394	14.933	21.303

As Tables II, III and IV show, the GPU outperforms the CPU in terms of performance. Speedup ratios range from 2.808x and 3.330x for the encoding and decoding processes respectively for low resolution videos (QVGA), reaching 11.048x and 21.303x respectively for the highest resolution (HD 1080p). The big difference in speedup ratio between the encoding and decoding processes, especially for higher resolutions, is attributed to the fact that the complexities of the motion compensation, run-length decoding and INFLATE steps at the decoding stage are significantly lower than the corresponding steps at the encoding stage, i.e. motion estimation, run-length encoding and DEFLATE. Since the complexity of the other steps is similar for encoding and decoding, GPU decoding performance benefits not only from the faster computation of the motion compensation step on the GPU, but also due to the reduced delay from computing the run-length decoding and INFLATE steps on the CPU. Consequently, the decoding process achieves a more efficient workload allocation between the GPU and the CPU compared to encoding.

Another important observation is that the speedup ratio of the GPU over the CPU increases with the increase of video resolution. The almost exponential increase of the workload for encoding and decoding higher resolution video favors the throughput oriented highly parallel GPU architecture. Furthermore, the continuous trend of demanding higher resolution video for everyday applications makes modern GPU architectures suitable candidates for handling this intense computational load.

TABLE III. AVERAGE EXECUTION TIMES PER FRAME (IN MS) ACHIEVED BY EACH ARCHITECTURE FOR THE ENCODING PROCESS

	320x240	**640x480**	**1280x720**	**1920x1080**
CPU	159.100	612.833	1866.233	4239.433
GPU	56.667	98.367	196.600	383.733

TABLE IV. AVERAGE EXECUTION TIMES PER FRAME (IN MS) ACHIEVED BY EACH ARCHITECTURE FOR THE DECODING PROCESS

	320x240	**640x480**	**1280x720**	**1920x1080**
CPU	149.200	594.600	1828.333	4202.433
GPU	44.800	70.833	122.433	197.267

The experimental evaluation of the CPU and GPU implementations of the contourlet video compression algorithm revealed the following:

- The GPU approach provides significantly faster performance than the CPU one for all the video resolutions examined. Speedup reaches an impressive 11.048x and 21.303x for the encoding and decoding stages respectively for high definition videos with a resolution of 1920x1080 pixels.
- The speedup achieved by the GPU approach significantly increases with the increase of video resolution. The performance of the CPU approach is limited for higher resolutions, indicating that GPUs should be the architecture of choice for high definition videos.
- The benefits of the GPU approach are more evident at the decoding stage of the algorithm where the speedup achieved was almost double the one achieved for encoding at the highest resolution. This fact indicates that the motion estimation, run-length encoding, and DEFLATE steps of the encoding algorithm significantly affect the performance of the encoder.

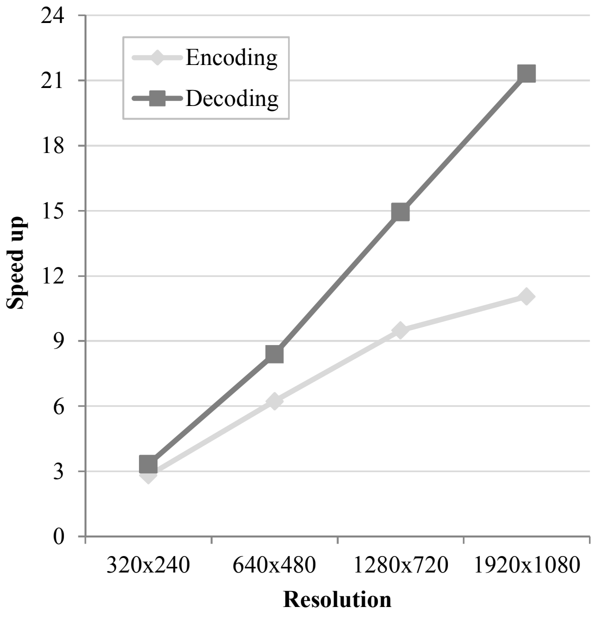


Fig. 3. Average GPU speedup per frame diagram for the encoding and decoding processes.

IV. CONCLUSIONS

In this work, the authors examined and evaluated the performance of the experimental contourlet video compression algorithm in terms of execution speed on a multicore CPU and manycore GPU architecture. Experimental results showed that the GPU approach provides significantly faster performance than the CPU one, especially at the decoding stage of the algorithm. When compared to the CPU for video resolutions spanning from 320x240 to 1920x1080 pixels, the GPU approach achieved speedup ratios from 2.808x to 11.048x for video encoding, and 3.330x to 21.303x for video decoding. Moreover, the performance analysis of the two implementations showed that the motion estimation, run-length encoding and DEFLATE steps of the compression algorithm should be examined for further optimization of their implementations. Results indicate the superiority of the GPU architecture for computations that are highly parallel, as in the case of the examined video compression algorithm, and support the trend of designing computationally intensive algorithms that can exploit the massive parallel computing capabilities of modern manycore architectures. As a result, one can argue that the GPU could become the architecture of choice for high definition video coding.

ACKNOWLEDGMENT

This work was partially supported by the National and Kapodistrian University of Athens Special Account of Research Grants.

References

[1] ITU-T Recommendation H.264: Advanced video coding for generic audiovisual services, 2012.

[2] Dirac specification, version 2.2.3, September 23, 2008. Available: http://diracvideo.org/download/specification/dirac-spec-latest.pdf

[3] RFC 6386: VP8 Data format and decoding guide, 2011.

[4] G.J. Sullivan, J. Ohm, W.-J. Han, T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) standard," IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1649-1668, 2012.

[5] NVIDIA CUDA architecture, http://www.nvidia.com/object/cuda_home_new.html

[6] OpenCL Standard for Parallel Programming of Heterogeneous Systems, https://www.khronos.org/opencl/.

[7] S. Katsigiannis, G. Papaioannou, D. Maroulis, "A contourlet transform based algorithm for real-time video encoding," in Proc. SPIE 8437, Real-Time Image and Video Processing 2012, 843704, June 1, 2012.

[8] S. Katsigiannis, G. Papaioannou, D. Maroulis, "A GPU based real-time video compression method for video conferencing," in. Proc. 18th International Conference on Digital Signal Processing 2013, 1-3 July 2013.

[9] H. Malvar, G. Sullivan, "YCoCg-R: A Color space with RGB reversibility and low dynamic range," Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Document No. JVTI014r3. 2003.

[10] P. Mavridis, G. Papaioannou, "The compact YCoCg frame buffer," J Computer Graphics Techniques, vol. 1, no. 1, pp. 19-35, 2012.

[11] J.M.P. van Waveren, I. Castano, "Real-time YCoCg-DXT compression," NVIDIA developer site. Available: http://www.nvidia.com/object/real-timeycocg-dxt-compression.html. Accessed 27 February 2015.

[12] M.N. Do, M. Vetterli, "The contourlet transform: an efficient directional multiresolution image representation," IEEE Trans. Image Process, vol. 14, no. 12, pp. 2091-2106, 2005

[13] RFC 1951: DEFLATE Compressed Data Format Specification, 1996.

[14] NVIDIA Corporation, "NVIDIA CUDA C programming guide version 6.5," August 2014.

[15] NVIDIA Corporation, "CUFFT library user's guide DU-06707-001_v5.5.," 2013.

[16] NTIA/ITS video sequences, Project Number 3141012-300, Video Quality Research, 2008. Available: ftp://vqeg.its.bldrdoc.gov/HDTV/NTIA_source/ .