Highly parallel GPU accelerator for HEVC transform and quantization

Čobrnić, Mate; Duspara, Alen; Dragić, Leon; Piljić, Igor; Kovač, Mario

Highly parallel GPU accelerator for HEVC transform and quantization

Mate Cobrnic*[a], Alen Duspara[a], Leon Dragic[a], Igor Piljic[a], Mario Kovac[a]
[a]University of Zagreb, Faculty of electrical engineering and computing, Unska 3, 10000 Zagreb, Croatia; mate.cobrnic@fer.hr

ABSTRACT

When analysing Internet traffic today it can be found that digital video content prevails. Its domination will continue to grow in the upcoming years and reach 82% of all traffic by 2021. If converted to Internet video minutes per second, this equals about one million video minutes per second. Providing and supporting improved compression capability is therefore expected from video processing devices. This will relieve the pressure on storage systems and communication networks while creating preconditions for further development of video services. Transform and quantization is one of the most compute-intensive parts of modern hybrid video coding systems where coding algorithm itself is commonly standardized. High Efficiency Video Coding (HEVC) is state-of-the-art video coding standard which achieves high compression efficiency at the cost of high computational complexity. In this paper we present highly parallel GPU accelerator for HEVC transform and quantization which targets most common heterogeneous computing CPU+GPU system. The accelerator is implemented using CUDA programming model. All the relevant state-of-the-art techniques related to kernel vectorization, shared memory optimization and overlapping data transfers with computation were investigated, customized and carefully combined to obtain a performance efficient solution across all applicable transform sizes. The proposed solution is compared against reference implementation which uses NVIDIA cuBLAS library to perform the same work. Obtained speedup factors for DCI 4K frame are 2.46 times for largest transform size and 130.17 times for smallest transform size what revealed substantial performance gap of this library when targeting GPU of the Kepler architecture. Achieved processing time of frame transform and quantization are up to 4.82 ms.

Keywords: Integer discrete cosine transform (DCT), high efficiency video coding (HEVC), graphics processor unit (GPU), matrix multiplication, compute unified device architecture (CUDA), high performance computing (HPC)

1. INTRODUCTION

Today, more than ever before, digital video dominates many industries and services. IP video traffic statistics and forecast for the time period from 2017 to 2022 show that it is in the range of 75 to 82 percent of total IP traffic[1]. Aside from high customer demand for video content, this high share is also caused by introduction of the Ultra-High-Definition (UHD) or 4K video streaming. It is estimated that by 2022 the UHD IP video will account for 22 percent of the global IP video traffic. This emerging technology reflects customer requests for high resolution and high-quality video content. There exist different forms of IP video content including Internet video, Video on Demand (VoD), video conferencing, video-streamed gaming and video files exchanged through file sharing. The very broad range of the forms and applications characterized by different specifications and constraints have one thing in common, video (de)compression. Compression is done by content providers and decompression is carried out at the consumer device. Video compression is a necessary processing step for affordable and efficient storage and transmission. Without compression, it would not be feasible to store and transmit digital video content because of the required huge storage capacity and a very large bandwidth.

The new generation of video compression standards provides higher quality at lower bit rates. This accomplishment was made at the cost of a big increase in the computational complexity of video encoding and decoding. One of the major state-of-the-art video coding standards is HEVC[2]. It doubled compression efficiency compared to its predecessor Advanced Video Coding (AVC).

*mate.cobrnic@fer.hr; www.fer.hr

2020 International Conference on Image, Video Processing and Artificial Intelligence, edited by Ruidan Su, Dora Zhang, Proceedings of SPIE Vol. 11584, 115840V
 · doi: 10.1117/12.2581228

Transform and quantization (TQ) is very important computation block not only in video coding systems but in multimedia compression systems in general. Mapping signals from spatial into the transform domain provides energy compaction and decorrelation. Once this is obtained, spectral components contributing less to video quality are removed during quantization. Discrete cosine transform (DCT) is the most widely used computational kernel in lossy compression due to its properties, which contribute to both compression efficiency and efficient implementation. Driven by further increase in implementation efficiency and interoperability, modern compression standards adopt integer approximation of the DCT like the one specified in the HEVC[3].

Video coding of high-resolution and high frame rate videos using new generation video codecs demands a high processing speed and compute capability. Computing systems based on the general Central Processing Unit (CPU) have reached the upper boundary in computation speed per Watt and cannot be efficiently utilized for video coding applications. The emerging heterogenous multiprocessor high performance computers can be used as a solution for those challenges. As one of the most processing and data demanding part of the HEVC compression algorithm, TQ can be implemented on CPU+GPU system, which is the most common heterogeneous computing system architeture[4] nowadays. Optimized implementation is a key to achieve high performance video encoding.

F. de Souza et al[5]. proposed highly parallel HEVC decoder in which all functional blocks except entropy decoder were transferred to GPU. It is characterized by frame-level parallel processing, unified decoder design to exploit GPU memory hierarchy and optimization of all decoding steps to achieve high parallelism level, efficient memory access and high instruction throughput. Igarashi et al.[6] tackled two processing overheads in TQ, gathering different size Transform Units (TU) and excessive data transfers, which appeared in proposal from Chen et al.[7]. The gathering process was removed through irregular load/store of TU data in separated memory addresses. Parallel packing of transform coefficients reduced the CPU-GPU data transfer. Less efficient parallelization on Coding Tree Unit (CTU) level was carried out from He et al[8]. Their TQ accelerator on GPU combined two control tables, one for TU partitioning and the other for storing of quantization parameter (QP) value. All these works either don't discuss TQ kernel design and low-level optimizations in detail or parallelization is not carried out on frame-level to exploit GPU parallelism.

In this paper we present highly parallel GPU accelerator for HEVC TQ with parallelization carried out on the frame-level with previously grouped transform blocks (TB) at the CPU side. Efficient vectorized memory access, data transfer minimization, CUDA kernel flexibility and overlapping data transfers with computation were investigated mutually to obtain a performance efficient solution across all applicable transform sizes.

The rest of the paper is organized as follows: In section 2 proposed GPU parallelization of TQ functional block is presented. In section 3 implementation is described and obtained results are evaluated. Section 4 gives the conclusions.

2. PROPOSED GPU PARALLELIZATION OF TQ

Compared to HEVC, standard TQ functional block will be slightly reduced during the design of the proposed accelerator. Processing the transform flags, i.e. Coded Block Flag, Transform Skip Flag and Transquant Bypass Flag are not considered in this work to simplify the parallelization analysis. Nevertheless, the evaluation results are applicable to the fully functional HEVC TQ accelerator since most of the workload relates mainly to matrix multiplications, and scalar operations, which are both covered here.

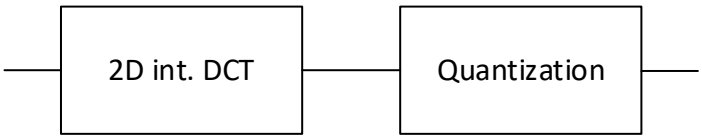

Figure 1. Block diagram of the HEVC transform and quantization.

TQ processes TBs grouped at the CPU side according to their size and sent in groups to GPU for processing. The accelerator outputs blocks of quantized transform coefficients. Block diagram of the accelerator is shown in Figure 1. 2D integer DCT is specified in the standard for four different TB sizes (4×4, 8×8, 16×16 and 32×32) and is given by

$$Y = D \times X \times D^T \tag{1}$$

where D is the HEVC transform matrix with known values, X is the prediction error matrix and D^T is a transpose of the transform matrix. Both two N-point 1D transforms are followed by scaling. Quantization is a scalar operation as well.

The heterogeneous processing system used in this work contains the CPU and its memory, referred to as the host, and GPU and its memory referred to as the device.

Compared to homogenous computing using CPU's only, there are two major overheads in systems with GPU accelerators : initiating a task on a device and transferring data from host to device and back. To save processing time, an unique kernel, which includes all operations made with TBs in a frame, is designed. Data transfers are realized directly over host's page-locked memory. Transform matrices for every transform size is written to the device global memory in advance and used for all frames that are processed.

All data needed for processing reside in global memory and are transferred from there to streaming multiprocessors (SM). Data are modeled using thread-blocks (ThB) and SM executes threads in warps, groups of 32 threads. Prediction error data, transform matrices and intermediate data, are stored and retrieved from the on-chip shared memory during TQ process. If the latencies for these two types of memory are compared, then the shared memory latency is approximately 100 times lower. To maximize parallelization and achieve higher performance ThB have to be highly utilized. It has been shown that highest performance is obtained when ThB are configured with quarter of the maximum number of the threads per block. In case of higher utilization fewer data requests can be handled from the L2 cache load, global memory has to be involved and its throughput decreases.

Performance of the accelerator is primarily affected by the access efficiency to the shared memory. When multiplying two matrices, scalar product has to be calculated for every element of result matrix. Multiply-add operation requires that N pairs of elements are retrieved from the memory and intermediate results stored there. The shared memory is organized in 32 memory banks where each bank has bandwidth of 64 bits per clock cycle. Thus, it was intuitive to group 4 byte-aligned 16-bit prediction error values into a data vector and configure the memory for 64-bit addressing mode. Next task was to design access pattern for all transform sizes which will minimize bank conflicts that occur when multiple threads' requested addresses within a warp map to the same memory bank. This is accomplished by padding the 2D arrays related to both operands with an additional column. This padding prevents bank conflicts when executing top-left thread in a ThB that computes four top-left elements of a resulting matrix product. The example of this for the largest size transform can be seen in Figure 2. Without padding, bank conflicts would occur when retrieving four leftmost elements in every fourth row since they are in the same bank. It has been shown that memory padding method is not efficient for 4- and 8-point transforms as more bank conflicts occur due to added column.

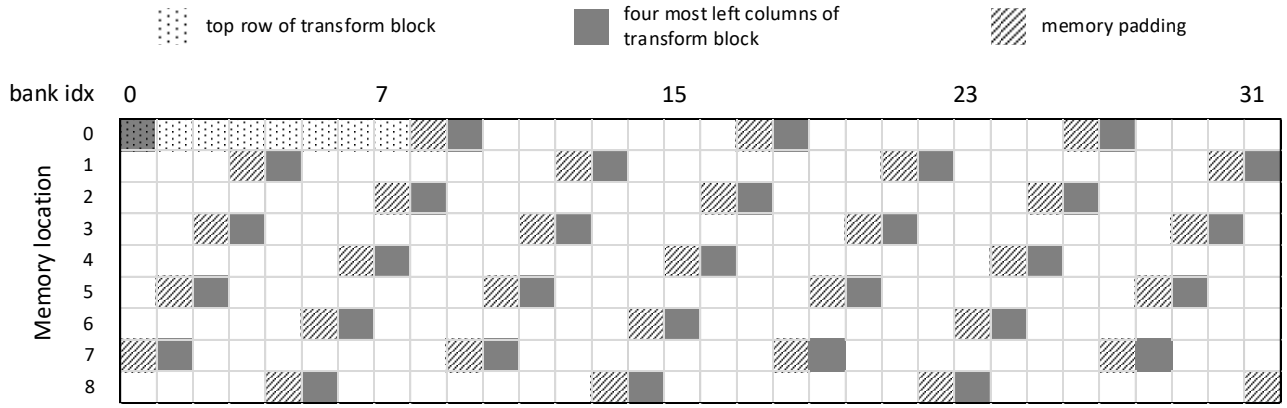

Figure 2. Distribution of data structured in shared memory using short4 data type and processed by the top left thread in a ThB for 32 × 32 TB

Considering a four-element vectorization and optimal configuration of ThB, size mapping ratio for prediction error blocks can be calculated as $1024/N^2{:}1$. Another operand in matrix multiplication, HEVC transform matrix, is loaded in the shared memory once per ThB. Reading transform matrix coefficients from the single instance results in performance efficient conflict-free broadcast access.

3. IMPLEMENTATION AND EVALUATION

The development environment set up to perform planned evaluations, consists of Intel Core i5-3570K@3.40GHz and NVIDIA Tesla K40c graphic card. Both host and device exchange data over the PCI Express x16 Gen3 high-speed bus (PCI-e). The Compute Unified Device Architecture[9] (CUDA) programming model v10.1 integrated into the Microsoft Visual Studio Community 2017 v15.9.7 was used to support heterogeneous computation with GPU accelerator for HEVC TQ. Three different implementations were supported. Referent CPU implementation computed quantized transform coefficients of a DCI 4K prediction error frame for given distribution of TBs. Its results were compared against proposed GPU implementation, which followed design principles described in previous section, and cuBLAS implementation that targeted GPU as well but employed NVIDIA library functions cuBLAS[10] to compute HEVC TQ of batch of TBs.

As cuBLAS doesn't support integer data sizes larger than 8-bit, the data type for all blocks and intermediate values is changed to a 32-bit floating-point. The 16-bit floating-point data are not used to avoid a compromise on accuracy. The data type change made scaling based on a right-shift obsolete. Scaling is therefore implemented as a scalar multiplication. The results obtained from the cublasSgemmStridedBatched(…) API in cuBLAS implementation are rounded in a separate kernel by using the floor command to obtain HEVC compliant results.

Performance measurements for two GPU implementations included overall and kernel processing time per frame. TQ functions that will be executed on a GPU were invoked ten times and the final measurement value was obtained as an average of all processing times. Two TB distributions, an edge cases considering computational complexity were evaluated, frame split to all 4 × 4 TBs and 32 × 32 TBs. Measurement results are shown in Table 1.

Table 1. Processing time comparison of the CPU and GPU implementations of transform and quantization for two transform block distributions.

TB distribution	Frame processing time [ms]				
	CPU	GPU overall	GPU kernel	CUBLAS overall	CUBLAS kernel
32 × 32	705.95	8.77	3.65	19.25	8.97
4 × 4	234.59	5.9	0.78	117.24	101.75

Results show that GPU implementation outperforms other two implementations. In case of the distribution with largest TB size speed-up over CPU and cuBLAS implementation considering kernel processing time is 80.53 times and 2.46 times respectively. For distribution based on smallest size TB these values are 39.73 and 130.17 times respectively. cuBLAS implementation exhibits significantly lower performance here. Analysis using a profiling tool showed that 1D integer transform kernels were invoked thirteen times altogether what involves significant overhead in task initiation at the device. Furthermore, ThB are configured with 128 threads from which only 16 are utilized, global load and store efficiency are very low, below 50% and shared efficiency is less than 30%.

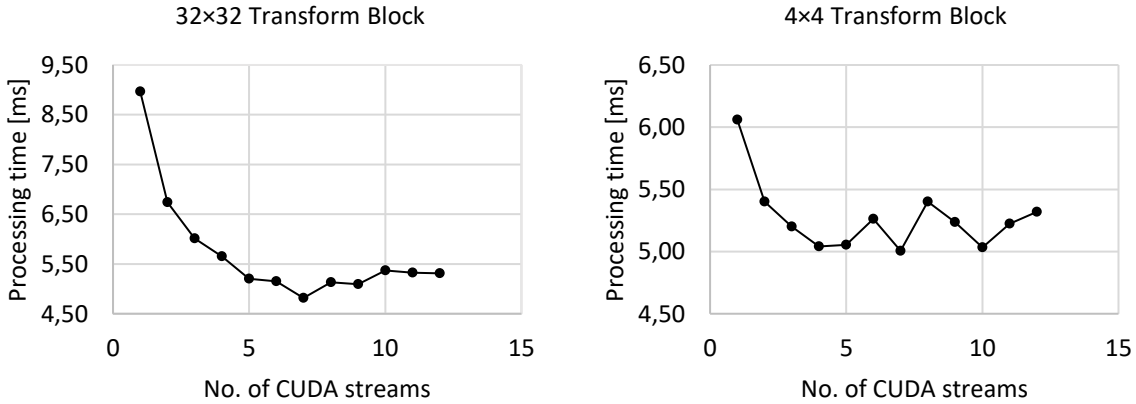

Figure 3. Impact of the number of employed CUDA streams on performance when processing a DCI 4K video frame

For an even more efficient GPU implementation, overlapping kernel execution and data transfer was exercised. Only the GPU implementation is in the scope of this evaluation. Prediction error frames are broken up into chunks, which can be interpreted as horizontal frame segments, and each chunk is assigned to its own CUDA stream. The objective was to find the frame segmentation for which the highest performance in terms of frame processing time is achieved. The results are shown in Figure 3. Lowest processing times 4.82 ms and 5.01 ms were achieved when seven streams are used. Speed-up between adjacent implementations decreases as number of streams increases. When moving to implementation with two stream, highest adjacent speed-up is obtained as data transfer and kernels are executed concurrently now compared to implementation with single (default) stream without any concurrency. As number of streams further increases, so increases the number of sequentially executed kernel chunks, and data transfer chunks but overlapping capability decreases. The overall performance gain using the overlapping is highest for the case when the data transfer and kernel periods are the same. Frame split to 32 × 32 TBs is closer to that optimal initial state with 58% of data transfer share in overall processing time (87% for 4 × 4 TBs). Therefore, its TQ takes less time.

4. CONCLUSION

This paper presents highly parallel GPU accelerator for HEVC TQ. During its design different optimizations techniques like vectorized memory access, page-locked data transfer, overlapping data transfers and kernel executions were adopted and adjusted to standardized TQ process and data types. Data transfers between host and device and within the device were designed to make full use of memory hierarchy and access pattern were created to maximize memory throughput. In experimental analysis using heterogeneous computing system with high-end GPU, TQ process was applied on production error signal in DCI 4K video frame. The proposed GPU implementation outperformed CPU and cuBLAS implementation 39.73 and 2.46 times respectively and revealed significant performance degradation when the latter is used for smallest size TBs. With overlapping data transfers and kernel execution frame processing time was as low as 4.82 ms.

ACKNOWLEDGMENTS

The work in this paper was partially financed by the MEEP project that has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 946002 and by the TETRAMAX project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 761349.

REFERENCES

[1] Cisco, "Cisco visual networking index: Forecast and trends," November 2018, <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html> (18 December 2019).
[2] Sullivan, G. J., Ohm J. R., Han, W. J. and Wiegand, T. "Overview of the High Efficiency Video Coding (HEVC) standard," IEEE Trans. Circuits Syst. Video Technol. 22(12), 1649-1668 (2012).
[3] Budagavi, M., Fuldseth, A., Bjøntegaard, G., Sze, V. and Sadafale, M., "Core transform design in the High Efficiency Video Coding (HEVC) standard," IEEE J. of Sel. Topics Signal Process. 7(6), 1029-1041 (2013).
[4] Liu, F., Liang, Y. and Wang, L. "A Survey of the Heterogeneous Computing Platform and Related Technologies," Proc. DEStech Trans. Eng. Technol. Research IMEIA, 6-12 (2016).
[5] De Souza, D. F., Ilic, A., Roma, N. and Sousa, L., "GHEVC: An efficient HEVC decoder for Graphics Processing Units," IEEE Trans. Multimedia 19(3), 459-474 (2017).
[6] Igarashi, H., Takano, F. and Moriyoshi, T., "Highly parallel transformation and quantization for HEVC encoder on GPUs," Proc. IEEE CFP1681P-POD, 381-384 (2016).
[7] Chen, Q., Wang, H., Zhuang, S. and Liu, B., "Parallel algorithm of IDCT with GPUs and CUDA for large-scale video quality of 3G," J. Comput. 7(8), 1880-1886 (2012).
[8] He, L. and Goto, S., "A high parallel way for processing IQ/IT part of HEVC decoder based on GPU," Proc. IEEE CFP14580-POD, 211-215 (2014).

[9] NVIDIA Corp., “CUDA C++ programming guide,” Version 10.1.243, <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html> (19 August 2019).
[10] NVIDIA Corp., “cuBLAS,” Version 10.2.89, <https://developer.nvidia.com/cublas> (19 August 2019).