GPGPU IMPLEMENTATION OF VP9 IN-LOOP DEBLOCKING FILTER AND IMPROVEMENTS FOR AV1 CODEC

Zhijun Lei[1], ryan.z.lei@intel.com
Srinath Reddy[2], srinath.reddy@microsoft.com
Victor Cherepanov[2], victor.cherepanov@microsoft.com
Zhipin Deng[1], zhipin.deng@intel.com

[1]Intel Corporation, [2]Microsoft Corporation

ABSTRACT

This paper describes the algorithm and processing flow of the in-loop deblocking filter in the VP9 coding standard, one of the most computationally intensive toolsets of the VP9 codec. Due to its inherent data dependency, it is a great challenge to efficiently implement the algorithm on massively parallel computing architectures, such as a General Purpose Graphical Processing Unit (GPGPU). In this paper, we describe the challenges involved in a GPGPU implementation of the VP9 Deblocking filter and introduce an innovative thread dispatching approach to address the parallelization challenges. This approach has been successfully implemented and productized in the VP9 decoder and encoder solutions on Intel GPUs. In order to further improve the parallelism of the deblocking algorithm itself, an improved in-loop deblocking algorithm and process flow is jointly proposed by Intel and Microsoft for the upcoming AV1 codec standard, developed by the Alliance for Open Media (AOM). A description of the algorithm and evaluation of the quality impact of this algorithm is presented with respect to the current state of the art AV1 reference codec.

Index Terms— deblocking, VP9, AV1, in-loop filtering

1. INTRODUCTION

AV1 is a new royalty free video coding standard being developed by the Alliance for Open Media (AOM) [1], which consists of major hardware and software codec vendors in the industry, such as Intel, Google, Microsoft, Cisco, etc. It is inherited from the VP9 coding standard previously developed by Google and incorporates many new coding tools to improve the coding efficiency beyond current state-of-art codecs like VP9 and HEVC. Similar to previous video coding standards, such as H.264 and HEVC, VP9 and AV1 are based on hybrid coding scheme using block-based prediction and transform coding. Please refer to [2] for detailed description of VP9 coding standard and process flow.

Block-based prediction and transform coding techniques cause non-smooth transitions at the block boundaries of reconstructed video frames. An in-loop deblocking filter is typically used in current video codecs to reduce blocking artifacts [3,4]. An overview of the VP9 in-loop deblocking filter is presented in Section 2. The challenges for a parallel implementation of the deblocking filter using GPGPUs and an innovative parallel implementation scheme are presented in Section 3. In Section 4, the proposed improvements for AV1 deblocking filter and quality impact data are provided.

2. VP9 DEBLOCKING FILTER ALGORITHM

Similar to other video coding standards, the VP9 deblocking filter was designed to improve subjective quality. It is performed as the last stage of the encoding or decoding operation and applied only to transform block edges. Within a frame, every Super Block (64×64 pixel block) is deblocked following raster scan order. Within a Super Block, all vertical edges along the transform block boundaries are filtered first, followed by all horizontal edges along the transform block boundaries. In VP9, filtering decisions are made separately for each transform block boundary, which range from 4×4 block to 32×32 block.

2.1. Filter mask generation for transform blocks

For every Super Block and all its internal transform block boundaries, the filtering decision is based on the following rules:

- Picture boundaries shall not be filtered.
- Filtering is applied on a transform block boundary only if the transform block has at least one non-zero coefficient, or the block is encoded with intra prediction mode.
- Super block boundary is always filtered.
- Filter taps are determined by the transform block size. For 32×32 and 16×16 transformed block boundaries, a filter up to 15 taps (filter16) is used. For 8x8 transformed block boundaries, a filter up to 7 taps (filter8) is used. For 4×4 transformed block boundaries, a 4-tap filter (filter4) is used.
- If the boundary of a 4×4 transformed block is also a boundary of a 32×32 prediction block, then a filter up to 8 taps is used.

Detailed algorithm for filter tap determination is described in the following sections.

2.2. Filter level and strength determination

In the VP9 frame header, two fields, *loop_filter_level* and *loop_filter_sharpness*, are used to calculate the thresholds and limits used in the deblocking filtering process. In addition, the filter strength level can also be adjusted for any 8×8 sub-block based on segmentation setting, i.e, any 8×8 block can get its *loop_filter_level* from the segmentation header that it belongs to. *loop_filter_level* can also be adjusted based on the mode and reference frame used to predicted a block. The final value of these two fields, *lvl* and *sharpness*, together determine whether a block edge is filtered and by how much the filtering may change the sample values. Based on the final *lvl* and *sharpness*, a few variables, *limit*, *blimit* and *thresh* are computed:

$$shift = \begin{cases} 1, 0 < sharpness \le 4 \\ 2, sharpness > 4 \\ 0, Otherwise \end{cases}$$

$$clip3(x,y,z) = \begin{cases} x; z < x \\ y; z > y \\ z; otherwise \end{cases}$$

$$limit = \begin{cases} Clip3\left(1, 9 - sharpness, \frac{lvl}{2^{shift}}\right), \\ sharpness > 0; \\ max\left(1, \frac{lvl}{2^{shift}}\right), Otherwise \end{cases}$$

$$blimit = 2 \times (lvl + 2) + limit$$

$$thresh = \frac{lvl}{16}$$

It should be noted that all computations in this paper assume video sample color depth is 8 bit. For higher bit depth, corresponding calculation can be adjusted accordingly. In VP9, for each pixel on the transform block boundary that potentially needs to be filtered, a filtering decision is performed based on the following ordered steps:

2.2.1. block detection

Block detection evaluates 4 pixels on each side of a block boundary as illustrated in Figure 1. The value of *filter_mask* indicates whether adjacent pixels close to the edge vary by less than the limits given by *limit* and *blimit*. It is used to determine if any filtering should occur and is calculated as in the following steps:

$$|p_i - p_{i-1}| > limit, i = 1..3 \quad (1)$$

$$|q_i - q_{i-1}| > limit, i = 1..3 \quad (2)$$

$$\left(|p_0 - q_0| \times 2 + \frac{|p_1 - q_1|}{2}\right) > blimit \quad (3)$$

filter_mask is equal to 1 only when equation (1), (2), (3) are all false for *i* from 1 to 3. Equation (1), (2), (3) attempts to identify whether the difference between two adjacent pixels is greater than the threshold and the edge is likely to be a real edge in the original video itself, rather than a blocking artifact introduced in the encoding process. If so, the filtering operation for the particular pixel is not needed and the following operations can be skipped.

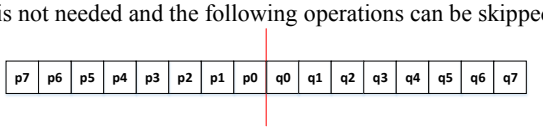

Figure 1. Pixels used for deblocking of a vertical edge

2.2.2. high edge variance detection

High edge variance detection evaluates 2 pixels on each side of a block boundary. The value of *hev_mask* indicates whether the sample has high edge variance. It is computed as below:

$$|p_1 - p_0| > thresh \quad (4)$$

$$|q_1 - q_0| > thresh \quad (5)$$

hev_mask is equal to 1 when either (4) or (5) are true.

2.2.3. flat region detection

Flat region detection is used to determine whether pixels from each side of the specified boundary are in a flat region. It is only required for transform block size greater than 8×8, i.e, filter8 or filter16. For filter8, *flat_mask_narrow* calculation checks 4 pixels from each side of the specified boundary as in below steps:

$$|p_i - p_0| > 1, i = 1..3 \quad (6)$$

$$|q_i - q_0| > 1, i = 1..3 \quad (7)$$

flat_mask_narrow is true only when both equation (6) and (7) are false for all *i* from 1 to 3. For filter16, *flat_mask_wide* calculation checks 8 pixels from each side of the specified boundary as in below steps:

$$|p_i - p_0| > 1, i = 1..7 \quad (8)$$

$$|q_i - q_0| > 1, i = 1..7 \quad (9)$$

flat_mask_wide is true only when both equation (8) and (9) are false for all *i* from 1 to 7.

2.3. Filtering operation

After calculating *filter_mask*, *hev_mask*, *flat_mask_narrow* and *flat_mask_wide*, pixels along the transform block boundary are filtered. 4-tap filter (filter4) is used for left and above boundaries of a 4×4 transformed block. filter4 checks and modifies up to 2 pixels on each side of the edge. It uses a threshold-limited blur to modify one pixel (p_0 and q_0) on each side of the boundary. When the variance of the four input pixel value is low, the two outer pixels (p_1 and q_1) are also modified to smooth the pixel value transitions. The detailed filtering operations are as below:

$$round2(x,n) = \left(x + \left(1 \ll (n-1)\right)\right) \gg n \quad (10)$$

$$clamp(value) = clip3(-128,128,value)) \quad (11)$$

$$ps_1 = p_1 - 128, ps_0 = p_0 - 128 \quad (12)$$

$$qs_0 = q_0 - 128, qs_1 = q_1 - 128 \quad (13)$$

$$\Delta = \begin{cases} clamp(ps_1 - qs_1), hev\_mask = 1 \\ 0, hev\_mask = 0 \end{cases} \quad (14)$$

$$\Delta_0 = clamp(\Delta + 3 \times (qs_0 - ps_0)) \quad (15)$$

$$\Delta_1 = \frac{clamp(\Delta_0 + 4)}{8} \quad (16)$$

$$\Delta_2 = \frac{clamp(\Delta_0 + 3)}{8} \quad (17)$$

$$\Delta_3 = round2(\Delta_1, 1) \quad (18)$$

$$q'_0 = clamp(qs_0 - \Delta_1) + 128 \quad (19)$$

$$p'_0 = clamp(ps_0 + \Delta_2) + 128 \quad (20)$$

$$q'_1 = \begin{cases} clamp(qs_1 - \Delta_3) + 128, hev\_mask = 0 \\ q_1, hev\_mask = 1 \end{cases} \quad (21)$$

$$p'_1 = \begin{cases} clamp(ps_1 + \Delta_3) + 128, hev\_mask = 0 \\ p_1, hev\_mask = 1 \end{cases} \quad (22)$$

where q'_0, p'_0, q'_1, p'_1 are modified output pixels at position q_0, p_0, q_1, p_1.

filter8 is used for edges of 8×8 transformed block. If the pixels are not in a flat region, the same filter4 is used to filter the pixel. If the pixels are identified as a flat area (*flat_mask_narrow* = 1), a simple 7-tap filter is used to filter the pixel, which checks 4 pixels on each side of the edge and can modify up to 3 pixels on each side of the edge. The filter taps are [1, 1, 1, 2, 1, 1, 1]. Detailed calculation is described as in below steps:

$$p'_2 = round2(p_3 + p_3 + p_3 + 2 \times p_2 + p_1 + p_0 + q_0, 3) \quad (23)$$

$$p'_1 = round2(p_3 + p_3 + p_2 + 2 \times p_1 + p_0 + q_0 + q_1, 3) \quad (24)$$

$$p'_0 = round2(p_3 + p_2 + p_1 + 2 \times p_0 + q_0 + q_1 + q_2, 3) \quad (25)$$

$$q'_0 = round2(p_2 + p_1 + p_0 + 2 \times q_0 + q_1 + q_2 + q_3, 3) \quad (26)$$

$$q'_1 = round2(p_1 + p_0 + q_0 + 2 \times q_1 + q_2 + q_3 + q_3, 3) \quad (27)$$

$$q'_2 = round2(p_0 + q_0 + q_1 + 2 \times q_2 + q_3 + q_3 + q_3, 3) \quad (28)$$

filter16 is used for edges of 16×16 or 32×32 transformed blocks. If the pixels are not in narrow flat region (*flat_mask_narrow* = 0) or wide flat region (*flat_mask_wide* = 0), filter4 is used. Otherwise, if the pixels are in narrow flat region (*flat_mask_narrow* = 1) but not in wide flat region (*flat_mask_wide* = 0), the above mentioned 7-tap filters are used. At last, if the pixels are in wide flat region (*flat_mask_wide* = 1), a simple 15-tap filter is used to filter the pixel, which checks 8 pixels on each side of the edge and can modify up to 7 pixels on each side. The filter taps are [1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1], and the computation equations are similar as equations (23) to (28).

3. GPGPU IMPLEMENTATION OF THE VP9 DEBLOCKING FILTER

In VP9, Super Blocks are loop-filtered following raster scan order, and within a Super Block, first all the vertical edges are filtered from left to right based on filter mask. Then horizontal edges are filtered from top to bottom based on filter mask. Due to this overlapping in filtering operation and filter length, the filter operations for any edge in the current Super Block may depend on the filtered output from its left, above, above-left and above-right Super Blocks. As an example in Figure 2, for the bottom block coded using 4×4 transform in SB1, the vertical edge V1 is filtered first, followed by the horizontal edge H1. When filtering the vertical edge V2 on the bottom left block in SB2 (coded using 16x16 transform), the filtering operation uses a 15-tap filter, and hence depends on, and may modify the previously filtered pixels from filtering operations for V1 and H2. Thus, SB2 has a data dependency on SB1. Similar dependency exists on its above-left, above and above-right SBs.

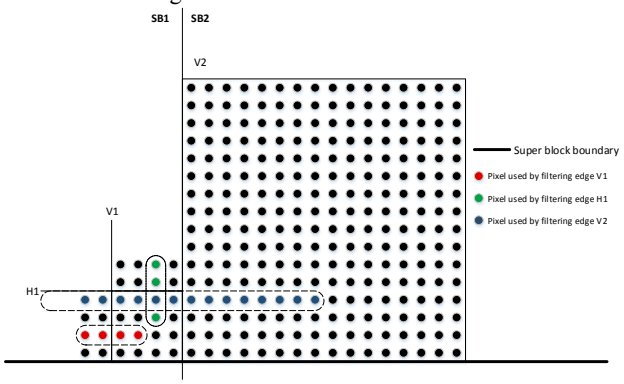

Figure 2. Example loop filtering dependencies.

Due to this dependency, in VP9, it is not easy to fully parallelize the vertical and horizontal edge filtering across the frame in a GPGPU implementation (as is possible in HEVC [2]). One approach to parallelize the deblocking operation in a GPGPU implementation is a wavefront pattern as illustrated in Figure 3.

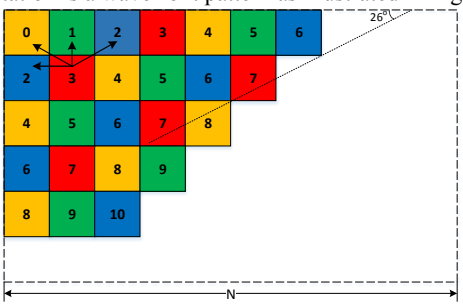

Figure 3. Wavefront dependency

In this diagram, each block represents a 64×64 Super Block. To deblock the current Super Block, the left, above, above-left, above-right Super Blocks are already filtered. Analyzing this dependency pattern, it is seen that Super Blocks with the same number (wavefront index) and color along the 26 degree wavefront can be processed concurrently by independent GPGPU threads. However, since the Super Block size is 64×64, there are only 30 Super Blocks in a Super Block row for a 1920×1080 HD frame. Hence only up to 15 Super Blocks along the waterfront can be processed in parallel. Considering the ramp up and ramp down, thread utilization is very low, especially for modern GPUs which have hundreds or thousands of threads.

A hybrid VP9 decoder was implemented on Intel Broadwell generation of GPGPUs as a time to market solution before a fully ASIC based solution was available. In this hybrid implementation, entropy decoding was implemented on the CPU to parse header syntax and decode transform coefficients. Inverse quantization, inverse transform, intra prediction, motion compensation and deblocking stages were all implemented on GPGPU. The deblocking block level boundary mask generation and threshold calculation was also performed on the CPU.

The basic building block of the GPGPU engine in Intel Broadwell generation of GPU [6] is the Execution Unit (EU), which is a highly multi-threaded processor with 7 hardware threads per EU. At the instruction level, SIMD instructions are available to process multiple data units with a single instruction. SIMD width (number of data units can be processed with a single instruction) is determined by the data type and can be as high as 32. Eight EUs are grouped together to form a slice, and multiple slices can be grouped together to scale the performance on different SKUs. For example, in a typical Broadwell GT2 SKUs, there are 3 slices with 24 EUs each and total number of hardware threads is 168. Intel GPGPU is the perfect architecture for parallel workload, such as image and video processing, where each hardware thread can independently process multiple blocks of image or video and pixel level parallelism can be achieved by the SIMD instructions. Within the EU, there are also some dedicated hardware units to manage thread dispatching and dependencies among different threads to reduce overhead of thread creation and context switching.

As explained in the previous section, in a straightforward parallel implementation, each hardware thread processes one Super Block and the dependencies between Super Block threads follow the wave front pattern. However, thread utilization is low because only 15 Super Blocks are processed in parallel at peak state. To resolve this limitation, a two level thread dispatching approach was utilized in the GPGPU implementation of the VP9 decoder. First, at the frame level, a software thread, *SB_thread*, is created to process each Super Block and its dependency follows the wavefront dependency pattern. Secondly, within each Super Block, a 64×64 block is divided into 8×8 blocks. For each 8×8 block, two software threads are spawned by its corresponding *SB_thread*, generating a total of 128 threads per Super Block. The first thread, *v_thread*, only deblocks vertical edges within the 8×8 block. The second thread, *h_thread*, only deblocks horizontal edges within this 8×8 block. As shown in Figure 4, within a Super Block, each *v_thread* only depends on the *v_thread* for the 8×8 block on its left. Each *h_thread* depends on the *h_thread* for the 8×8 block on its top and *v_thread* for itself and two 8×8 blocks on its left and two 8×8 blocks on its right. For example, in Figure 4, *v_thread*[i, j] depends on *v_thread*[i, j-1] and *h_thread*[i, j] depends on *h_thread*[i-1, j], *v_thread* [i, j], *v_thread*[i, j-1], *v_thread*[i-1, j-1], *v_thread*[i, j+1], *v_thread*[i-1, j+1].

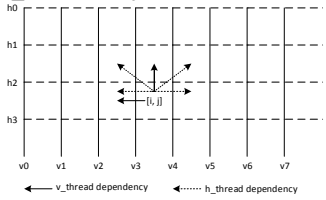

Figure 4. Dependency of v_thread and h_thread

In Intel GPGPU, a dedicated hardware unit called Scoreboard is used to manage thread dependency. A software thread is mapped to an entry in the scoreboard and up to 8 dependent threads can be specified. A software thread can start execution immediately without any synchronization overhead when all its dependent threads have finished execution. Besides the two-level thread dispatching, software threads that have more threads dependent on them are dispatched first, so dependencies can be cleared quickly. With this approach, the thread utilization for VP9 deblocking stage reaches 98% and deblocking a 1080p frame takes close to 1ms.

Detailed performance for the decoding stage is presented in Table 1. Performance was evaluated when decoding 100 frames from an HD (1920×1080) and an 4K (3840×2160) VP9 bitstream on the Broadwell GT2 with the GPU running at 900 MHz. Average time per frame for each stage implemented on GPU side is summarized. As can be seen, 60 fps for 1080p and 30 fps for 4K decoding are easily achieved. It is also observed that, even with the innovative thread dispatching approach, the deblocking stage remains the most time consuming stage of the decoder, and takes more than 30% of the decoding time.

	1920x1080		3840x2160	
	Time (ms)	Percentage	Time (ms)	Percentage
IQ/IT	0.847	23.77%	4.634	26.65%
Intra Prediction	1.083	30.39%	4.759	27.36%
Motion Compensation	0.547	15.35%	2.186	12.57%
Deblocking	1.087	30.50%	5.812	33.42%
Total	3.564		17.391	

Table 1. VP9 decoder performance breakdown

4. IMPROVEMENTS IN AV1 DEBLOCKING FILTER

The complexity of the VP9 deblocking filter is quite high when compared to deblocking in H.264 and HEVC. The 7-tap and 15-tap filters used may cause higher bandwidth requirement for software or hardware implementations. In addition, long filter length and overlap between horizontal edge and vertical edge filtering introduces data dependency between different blocks. To reduce the overall complexity and dependency of the deblocking stage, a few simplifications and improvement are proposed for the AV1 coding standard:

- The filtering order is modified to filter all vertical edges of the entire frame followed by all horizontal edges.
- 15-tap filter (filter16) and *flat_mask_wide* computation is removed. 8×8, 16×16 and 32×32 transformed blocks are all filtered with filter8.
- If any of the transform blocks on either side of a horizontal or either side of a vertical edge has non-zero coefficients, the corresponding edge is filtered. In VP9, for a transform block with no non-zero coefficients, the left and above boundaries are not filtered, which may cause non-smoothing transition if its left or above neighboring blocks have non-zero coefficients. In this proposal for AV1, blocks on both sides of the boundary are considered. Only when both have no non-zero coefficients, filtering of the edge is skipped.
- The 4-tap filter (filter4) only utilizes two pixels on each side of the edge to detect an edge and calculate *filter_mask*. In VP9, *filter_mask* is calculated using 4 pixels on each side of the edge. This requirement causes serialization and propagation of filtered pixels from the current transform block to neighboring blocks.
- De-blocking filter length is determined based on the transform block size from both sides of the edge. Size of the smaller transform block is used to determine the filter taps. For example, if either side of the edge uses 4×4 transform, then filter4 is used. Otherwise, filter8 is used.
- In VP9, there is a special rule which applies an 8-tap filter (filter8) on a 4×4 transform block edge if the edge is an outer edge of a Super Block. This requirement is removed.

With these improvements, the complexity of the filtering process and dependency are significantly reduced. It is now possible to divide a frame into independent non-overlapped blocks of reasonable size and process such blocks by multiple threads in parallel. There are the following parallelization options:

- Similar as the approach described in [4,5], a picture can be divided into non-overlapping blocks of 4x4 samples. These 4x4 blocks are shifted by 2 pixels both horizontally and vertically so they do not overlap with any transform block boundaries. Within each 4x4 block, vertical edges are deblocked first followed by horizontal edges. All 4x4 blocks are processed fully in parallel without any dependency.
- Deblocking can be divided into two stages. The first stage processes all vertical edges in a frame and the second stage processes all horizontal edges. In each stage, a picture can be divided into non-overlapping blocks of any size, and all blocks can be processed fully in parallel. With additional synchronization, the second stage can begin as soon as some blocks in the first stage have been filtered.
- Deblocking is performed using the wavefront approach described above, but with a smaller block granularity, so that more blocks are processed in parallel. For each block, all vertical edges within this block are first filtered, then all horizontal edges of its left neighboring block are filtered. With this approach, a significantly higher parallelism can be achieved while each block has enough workload to benefit from the instruction level parallelism using SIMD.

The proposal has been implemented in the AV1 reference software[7]. Extensive quality tests, both objective and subjective, have been carried out to evaluate the quality impact. The objective quality test result from AWCY[8] in BD-RATE is summarized in Table 2. A negative BDRATE indicates better quality than VP9 deblocking filter. As can be seen, objective quality impact of proposed deblocking filter is negligible when comparing to VP9 deblocking filter in both Low Delay and High Delay encoding configuration and different QP range. Subjective quality inspection also confirmed that no visible quality differences are observed.

	Low Delay	High Delay
Low QP [17,22,27,32]	-0.06%	-0.26%
Medium QP [32, 37,42,47]	0.01%	-0.15%
High QP [47, 52, 57, 62]	0.12%	0.00%

Table 2. Objective quality test result of the proposal

5. SUMMARY

The proposed deblocking filter for the upcoming AV1 standard significantly reduces the complexity and dependency of the deblocking filter in previous VP9 standard. Quality impact, both objective and subjective, is negligible. The proposed AV1 deblocking filter can be easily performed in parallel, which is important for utilizing modern GPGPU to accelerate the encoding and decoding process.

6. REFERENCES

[1] Alliance for Open Media (http://aomedia.org)

[2] Adrian Grange, Peter de Rivaz, Jonathan Hunt, "VP9 Bitstream & Decoding Process Specification, version 0.6," https://storage.googleapis.com/downloads.webmproject.org/docs/vp9/vp9-bitstream-specification-v0.6-20160331-draft.pdf

[3] Peter List, Anthony Joch, Jani Lainema, et. al, "Adaptive Deblocking Filter", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, July 2003.

[4] Andrey Norkin, Gisle Bjøntegaard, Arild Fuldseth, et.al, "HEVC Deblocking Filter," IEEE Transactions on Circuit and Systems for Video Technology, Vol. 22, No. 12, December 2012.

[5] Frank Bossen, Member, IEEE, Benjamin Bross, Student Member, IEEE, Karsten Suhring, and David Flynn, "HEVC Complexity and Implementation Analysis," in IEEE Transactions on Circuits and Systems for Video Technology, Vol. 22, No. 12, December 2012.

[6] Stephen Junkins, "The Compute Architecture of Intel Processor Graphics GEN8," Intel Developer Forum 2015. https://software.intel.com/sites/default/files/Compute%20Architecture%20of%20Intel%20Processor%20Graphics%20Gen8.pdf

[7] https://aomedia.googlesource.com/aom. AOM AV1 Reference Software.

[8] https://arewecompressedyet.com/. AWCY, video codec quality benchmarking system.