

HARDWARE-ACCELERATED MOTION JPEG DECODING ON WINDOWS: DEDICATED JPEG ENGINES VERSUS GPU SHADER OFFLOAD

¹M. Sneha, ²Dr. L. Padma Sree

¹M.Tech Student (Embedded Systems), ²Professor

^{1,2}Department of Electronics and Communication Engineering

^{1,2}VNR Vignana Jyothi Institute of Engineering and Technology, Hyderabad, India

¹SnehaReddym454@gmail.com, ²padmasree_l@vnrvjiet.in

Abstract : Motion JPEG (MJPEG) remains widely used in webcams, IP surveillance and low-latency conferencing because every frame is coded independently, enabling low decode latency, error containment and frame-exact seeking. That same intra-only structure makes MJPEG expensive to decode: no temporal redundancy can be removed, so the inbox decoder remains busy throughout a call. This paper presents a user-mode Media Foundation Transform (MFT) that routes MJPEG decoding to AMD graphics hardware and compares three paths in this order: dedicated JPEG hardware, GPU shader offload and the inbox Microsoft Media Foundation decoder. The component parses baseline-JPEG markers, computes entropy-scan extents that no header field carries, maps quantisation and Huffman tables to semantics now published for DirectX Video Acceleration (DXVA) MJPEG structures, and submits work through a driver interface. Nine sequences spanning 720i and 1080p, 4:2:0 and 4:2:2 chroma, and three encoder-quality tiers were evaluated on two AMD GPUs under 32-bit and 64-bit Windows 10. Dedicated JPEG hardware was the strongest path: relative to the Microsoft decoder it increased arithmetic-mean throughput by 18.93 % on 64-bit Windows and 25.36 % on 32-bit Windows, with a documented per-sequence gain up to 78.66 %, and on 64-bit Windows its cross-sequence coefficient of variation fell from 15.34 % to 1.66 %. The shader path was slower than the Microsoft decoder on all 18 sequence-architecture pairs, by 4.56 % and 6.16 % in aggregate; the observed deficit is consistent with frame-level serialisation at a queue depth of one.

IndexTerms - Motion JPEG, dedicated JPEG hardware, GPU shader offload, Microsoft Media Foundation decoder, DXVA, JPEG marker parsing, host-CPU utilisation.

I. INTRODUCTION

An MJPEG stream is a sequence of independently coded JPEG images [1], [2]. Because no frame references another, a frame can be displayed as soon as it arrives, a lost frame does not propagate error, and every frame is a seek point. These properties keep MJPEG in service in camera capture, IP surveillance, webcams and conferencing, despite a bit rate far above H.264 for equivalent perceptual quality.

The penalty is a high, continuous decode load: every frame must be Huffman-decoded, dequantised and inverse-transformed in full. Microsoft notes that most USB Video Class cameras send MJPEG to fit high resolutions within available bus bandwidth, and that one centralised decode avoids repeated decoding downstream [3]; Chromium likewise implements a hardware MJPEG path for camera capture [4]. On Windows the comparison baseline is the inbox Media Foundation MJPEG decoder [5], obtained by removing the vendor transform. This paper names that baseline the Microsoft decoder and does not claim that Microsoft's implementation is CPU-only or GPU-only.

Modern AMD GPUs offer two distinct mechanisms for relieving the CPU. A **dedicated fixed-function JPEG decode block**, exposed through AMD's Advanced Media Framework [6], performs entropy decoding, dequantisation and the inverse DCT in hardware after host-side marker parsing, so the sequential Huffman stage does not remain on the host. That is its advantage over the **programmable shader array**, which keeps conventional entropy decoding on the CPU and dispatches only the data-parallel reconstruction stages. The shader path therefore still pays the serial part of every frame, plus transfer and launch cost, and need not beat the Microsoft decoder.

AMD's capability query reports hardware, GPU and software acceleration as separate categories [6]. The measured configurations differ in *sign*: dedicated JPEG hardware improved throughput relative to the Microsoft decoder; the unit-depth shader path reduced it. This paper supplies those measurements, describes the interface to the decode block from unmodified applications, and examines the shader-path deficit at a queue depth of one.

II. LITERATURE SURVEY

A. Parallel JPEG decoding on GPUs

Huffman codewords have variable length, so a codeword's start position is unknown until its predecessor is decoded. One family of solutions lets a sequential CPU scan locate coefficient-block boundaries before a parallel GPU stage decodes them; a second

exploits the self-synchronising property of Huffman codes to evaluate codeword positions at arbitrary offsets, enabling wholly GPU-side decompression [7]. Both optimise *within* one image. The shader path measured here is the coarsest form of the first family, which proves decisive in a component whose unit of work is one frame.

B. Dedicated JPEG decode hardware

AMD exposes a dedicated MJPEG hardware decoder through its public Advanced Media Framework on Windows, and reports hardware, GPU and software acceleration as separate categories [6]. That separation is the property this paper relies on: dedicated JPEG hardware is not a mode of the shader array. This work measures application-visible behaviour of the dedicated block through Media Foundation and Direct3D.

C. Energy of hardware decode and of GPU offload

Reviews of other video codecs report substantial energy advantages for fixed-function hardware decoders over optimised software [8]. Conversely, a hybrid HEVC decoder consumed 0.92 J per 4K frame against 0.24 J for its best CPU-only configuration despite higher throughput [9], and a mobile GPU-offload study found setup and tail state exceeding 65 % of one-shot energy [10]. These are analogies rather than MJPEG measurements, but they show why offload is not intrinsically beneficial when little work is amortised per dispatch.

D. Research gap

We found no published study comparing dedicated JPEG hardware, shader offload and the inbox Microsoft decoder through one Windows application stack on a common sequence set. Those energy studies are not MJPEG, and GPU JPEG papers typically report only decode speed, not host-CPU load or picture quality.

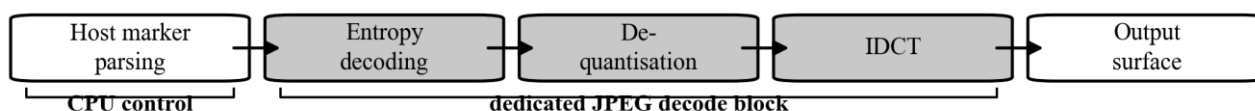
III. TECHNICAL BACKGROUND

A. The baseline decode pipeline

Decoding reverses the encoder's stages on 8×8 blocks. Entropy decoding converts variable-length Huffman codes into quantised coefficients, reconstructs the DC term differentially and undoes the zig-zag order; dequantisation multiplies each coefficient by its quantisation-table entry; the inverse DCT returns the 64 coefficients to spatial samples [1], [2]. Chroma upsampling and YCbCr-to-RGB conversion then yield a displayable surface. Colour conversion is typically performed by the display path and lies outside this decoder comparison.

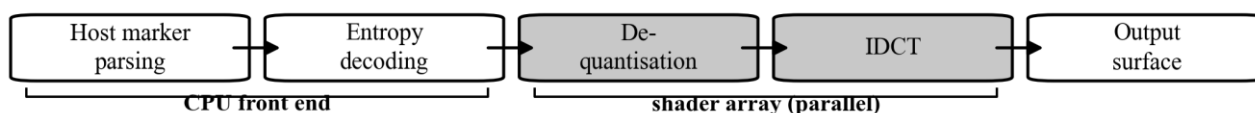
The decisive asymmetry is parallelism. Conventional Huffman decoding carries serial bit-position dependencies, whereas dequantisation and the inverse transform are independent across blocks; specialised parallel entropy algorithms require additional synchronisation or preprocessing [7]. Fig. 1 shows the partition used here: the host parses markers on both paths, the dedicated block performs entropy decoding and reconstruction, and the shader path leaves entropy decoding on the CPU. The standard prescribes no unique inverse DCT [2], permitting fixed-point hardware implementations.

(a) Dedicated-hardware path



The host parses markers and submits parameters; hardware performs entropy decode and reconstruction.

(b) Shader path



The entropy stage remains on the CPU; at unit queue depth, limited overlap can make per-frame cost approach the sum of CPU and GPU stages (Section VI-C).

Fig. 1. Partitioning of the baseline JPEG decode pipeline. Host-side marker parsing is common to both paths. (a) The dedicated JPEG block performs entropy decoding, dequantisation and the inverse DCT. (b) The shader path offloads the data-parallel reconstruction stages while conventional entropy decoding remains on the CPU.

B. Input and output video formats

Decoded surfaces are delivered in YCbCr four-character code (FOURCC) layouts rather than packed RGB. For the common camera cases considered here, 4:4:4 applies no subsampling, 4:2:2 subsamples chroma 2:1 horizontally, and 4:2:0 subsamples it 2:1 in both directions. Table I lists layouts used on the Windows path. The hardware path targets baseline 8-bit 4:2:0 and 4:2:2 [11].

Table I. Common YCbCr FOURCC layouts on the Windows decode path.

Sampling	Bits/pixel	Packed or planar	FOURCC
4:4:4	32	Packed	AYUV
4:2:2	16	Packed	YUY2, UYVY
4:2:0	16	Planar	IMC1, IMC3
4:2:0	12	Semi-planar, planar	NV12, YV12

IMC1 and IMC3 store 16 bits per pixel because of plane pitch, but remain 4:2:0.

C. Bitstream structure and marker parsing

A hardware decoder is configured per frame from parameters carried in the bitstream, so parser accuracy determines whether the block is programmed correctly. All syntax below is public, in Annex B of ITU-T T.81 [2]. A marker is 0xFF followed by a code in 0x01–0xFE, and four byte-level rules must be handled explicitly. *Fill bytes*: any marker may be preceded by arbitrarily many 0xFF bytes, so a run of them must be skipped. *Byte stuffing*: inside entropy-coded data a 0x00 is inserted after every 0xFF the coder emits, so FF 00 denotes the data value 0xFF, not a marker. *Length fields*: most markers carry a big-endian length counting itself but not the marker bytes, making unknown segments skippable; SOI, EOI and restart markers carry none. *Ordering*: table segments may appear both before the frame header and before any scan, so table state must remain mutable.

Table II. Baseline JPEG markers handled by the proposed transform, after Annex B of ITU-T T.81 [2].

Marker	Code	Meaning	Action taken
SOI / EOI	0xFFD8 / 0xFFD9	Start, end of image	Begin frame, reset state; terminate
SOF0	0xFFC0	Baseline frame header	Extract P, Y, X, Nf; per-component Ci, Hi, Vi, Tqi
Other SOFn	0xFFC1–0xFFC3, 0xFFC5–0xFFC7, 0xFFC9–0xFFCB, 0xFFCD–0xFFCF	Extended, progressive or lossless	Reject: outside baseline
DQT	0xFFDB	Quantisation tables	Extract Pq, Tq, 64 elements; install at Tq
DHT	0xFFC4	Huffman tables	Extract Tc, Th, Li, Vi,j; install at Th
DRI / RSTn	0xFFDD / 0xFFD0–0xFFD7	Restart interval, restart	Record Ri; leave RSTn in scan
SOS	0xFFDA	Scan header	Extract Ns, Csj, Tdj, Taj, Ss, Se, Ah, Al; mark scan
APPn / COM	0xFFE0–0xFFEF / 0xFFFE	Application data or comment	Skip using the length field

The per-component sampling factors in SOF0 fix the chroma format: equal factors indicate 4:4:4, twice horizontally 4:2:2 and twice in both dimensions 4:2:0. The DC and AC selectors in SOS must be read rather than assumed, because the standard installs tables at numbered destinations and lets any component reference any of them.

One consequence shapes the design: the length of an entropy-coded segment appears in no header field, because byte stuffing prevents it. Each scan extent must therefore be located by scanning forward under the FF 00 rule, past any restart markers, to the next genuine marker. Some cameras also omit DHT and rely on the example tables of Annex K, so a decoder for real capture streams must substitute them rather than reject the frame. All these rules appear in FFmpeg's MJPEG decoder and parser [12], used here as a cross-check.

D. Media Foundation, DXVA and AMD hardware

The decoder is a Media Foundation Transform, a Component Object Model (COM) object receiving compressed samples through ProcessInput and returning decoded samples through ProcessOutput, with messages for stream start, flush and Direct3D device-manager setup [13]. The input subtype is MFVideoFormat_MJPG, one complete frame is supplied per sample, and no reordering is required; the resulting one-frame queue motivates the analysis in Section VI-C. Media Foundation has supported an MJPEG decoder since Windows 7 [5], and codec merit allows a vendor transform to be preferred without application changes [13].

Beneath the transform, DXVA 2.0 and D3D11 Video manage compressed buffers and output surfaces. Microsoft now documents three MJPEG structures for Windows 11 24H2 and later [11]: picture parameters carrying frame geometry, component count, bit depth, quantisation selectors, restart interval and arrays locating each scan; a 4 × 64 quantisation array in zig-zag order; and four DC and four AC Huffman-table slots. The Windows 10 implementation evaluated here used the vendor interface available at the time, with equivalent picture- and table-buffer semantics; the newer public structures are cited to clarify the mapping, not to imply a later DDI was tested.

IV. PROPOSED DECODER DESIGN

A. Architecture

Control descends from the application through the transform, Direct3D, the proposed user-mode component and the kernel-mode driver to the decode block, and decoded surfaces return by the reverse path (Fig. 2). Per frame, the transform supplies a compressed sample and output surface; the component parses the bitstream and builds picture-parameter, quantisation and Huffman buffers; the driver then submits the prepared work to the dedicated JPEG decode block. This is the measured Windows path.

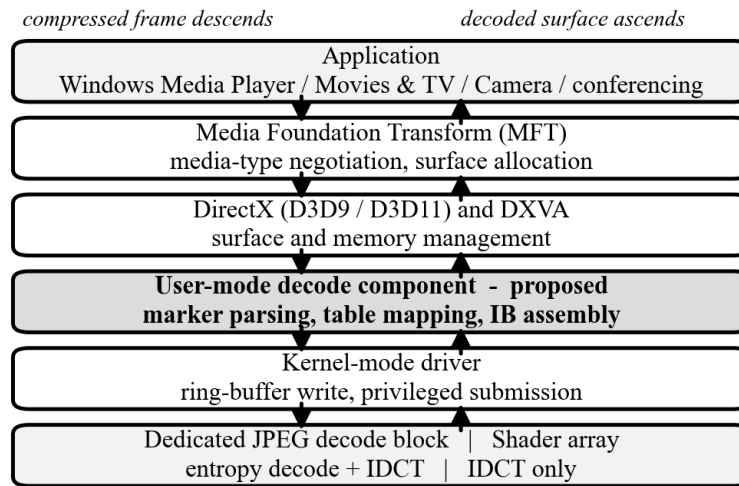


Fig. 2. Layered architecture from the unmodified application to the decode block. The shaded layer is the component contributed by this work.

B. Runtime capability discovery

A single binary must serve a whole product line, so the component probes at instantiation. It first selects the Direct3D 11 or Direct3D 9 surface path according to what the operating system and application support, then tests for a dedicated JPEG component, then for a shader component, and otherwise leaves decoding to the Microsoft decoder — matching the three acceleration types AMD's framework publishes [6]. Probing uses temporary components destroyed immediately, leaving no hardware state behind on paths not taken.

C. Parsing, validation and buffer construction

Mapping a variable-length header onto fixed-length structures imposes validation obligations rather than mere data movement. The component rejects or falls back unless the frame header is SOF0, sample and quantisation precision are both 8 bits, table selectors lie in 0–3, the interface limits of four components and four scans are respected, Huffman symbol counts fit their table slots, the interleaved minimum-coded-unit (MCU) sum $\sum H_i \cdot V_i$ is at most ten, the scan is sequential with $S_s = 0$, $S_e = 63$ and $A_h = A_l = 0$, and the chroma format is supported. Surface allocation respects both the interface's eight-pixel alignment and sampling-specific MCU geometry, with display dimensions carried separately.

One case deserves note. T.81 states that referencing an undefined table destination gives unpredictable results, so submitting uninitialised table memory to a shared decode queue is unsafe; the component substitutes the Annex K defaults instead. Since MJPEG from a fixed encoder commonly repeats identical tables, parsed table buffers are cached under a hash of the DQT and DHT bytes, leaving only the picture parameters to be rebuilt per frame.

Where no dedicated block is present, entropy decoding stays on the CPU while dequantisation and the inverse transform are dispatched to the GPU. Because frames are independent and the transform presents one at a time, the natural implementation processes one frame at a time — the configuration measured below.

V. EXPERIMENTAL METHODOLOGY

A. Platforms

Two AMD parts keep the Microsoft decoder the same but do not isolate the offload mechanism from GPU model or discrete-versus-integrated architecture. The dedicated-hardware platform was a Radeon R9 200 Series discrete GPU exposing a JPEG decode block; the shader platform was Radeon R7 integrated graphics exposing none, so capability discovery selected the shader path. Each ran 64-bit and 32-bit Windows 10, giving four configurations. No application was modified: playback used Windows Media Player and Movies & TV, and capture used the Camera application.

B. Test content

Nine sequences span two resolutions, two chroma formats, both field parities at 720i25, a 720i30 TFF case and a three-step quality ladder, as listed in Table III. Names encode the parameters that determine decoder workload, so a result reads without consulting the table: 720i25tff-422 is 1280 × 720 interlaced at 25 Hz, top field first, 4:2:2, and the lq and vlq suffixes mark coarser quantisation tiers of an otherwise identical clip. The source report does not identify a public download for the exact files; public test media can reproduce the parameter matrix, but not identical content [14]. The quality ladder is deliberate because coarser quantisation changes the entropy-coded workload examined in Section VI-A.

Table III. The nine test sequences. Names encode resolution, scan, rate, chroma and quality tier.

Sequence	Content	Scan	Quality
720i25bff-422	Colour test	Interlaced, BFF	Nominal
720i25tff-422	Colour test	Interlaced, TFF	Nominal
720i30tff-422	High motion	Interlaced, TFF	Nominal
1080p30-420	Performance test	Progressive	Nominal
1080p30-420lq	Performance test	Progressive	Low

1080p30-420v1q	Performance test	Progressive	Very low
1080p30-422	Performance test	Progressive	Nominal
1080p30-422lq	Performance test	Progressive	Low
1080p30-422v1q	Performance test	Progressive	Very low

C. Metric, baseline and path confirmation

The metric is decoded frames per second for the same sequence, application and machine, with only the decoder differing. The baseline is the inbox Microsoft Media Foundation MJPEG decoder [5], obtained by removing the vendor transform and not labelled CPU or GPU. Rates exceed the nominal 25 and 30 Hz because playback is unthrottled, measuring throughput headroom rather than real-time presentation rate. We report arithmetic means, the coefficient of variation across sequences and the geometric mean of per-sequence speedup ratios. The source measurements provide average rates but not repeated-run samples or confidence intervals; therefore, the comparisons below are descriptive and do not establish statistical significance.

The source measurements verified path selection using vendor engine-state indicators. A vendor debug overlay, an orange mark at the frame corner, was also present while the hardware decoder was active; Fig. 3(a) magnifies the fragment of that mark which survives the source screenshot. The overlay corroborates hardware decoding but does not name the engine, so it is a supporting indicator only. For independent replication with public interfaces, MFTEnumEx can confirm that the vendor decoder is registered and categorised as hardware [13], and the framework capability query reports the acceleration type the GPU supports [6]. Windows also exposes per-engine GPU scheduler utilisation by node type [15], so a dedicated path should exercise a video-decode node and a shader path a 3D/compute node; logging these together would exclude silent fallback. Every configuration additionally passed continuous seek-bar motion, which stresses flush and mid-stream table redefinition, plus sustained playback, repeated teardown, and camera capture (Fig. 3).

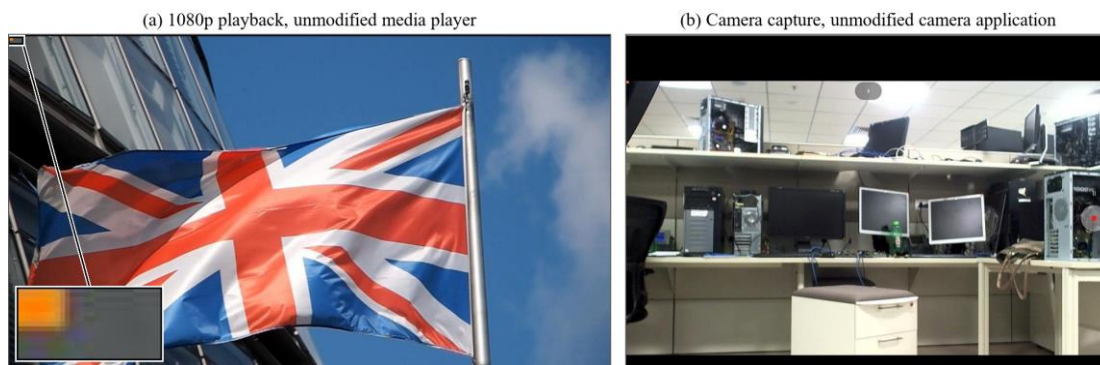


Fig. 3. Decoded output with hardware-accelerated decoding active in unmodified applications: (a) 1080p playback, with an inset magnifying the vendor debug overlay clipped at the frame edge; (b) camera capture. These confirm end-to-end operation, not fidelity (Section VI-E).

VI. RESULTS AND DISCUSSION

Table IV reports per-sequence throughput on the dedicated-hardware platform under 64-bit Windows 10; Table V summarises all four configurations.

Table IV. Per-sequence throughput in frames per second, dedicated-hardware platform, 64-bit Windows 10.

Decoder	720i25bff-422	720i25tff-422	720i30tff-422	1080p30-420	1080p30-420lq	1080p30-420v1q	1080p30-422	1080p30-422lq	1080p30-422v1q	Mean
Microsoft decoder	107.03	107.22	128.12	123.82	134.36	121.08	76.05	103.33	115.73	112.97
Dedicated path	132.35	132.46	130.48	135.03	134.87	135.92	135.87	137.79	134.44	134.36
Change (%)	+23.66	+23.54	+1.85	+9.05	+0.38	+12.25	+78.66	+33.36	+16.16	+18.93

The final Change entry is the relative change between the two arithmetic-mean throughput values, not the arithmetic mean of the nine percentage changes.

Table V. Descriptive aggregate comparison against each GPU's Microsoft decoder. The dedicated and shader paths were measured on different GPU models.

Platform and mechanism	OS	Microsoft decoder (fps)	Vendor path (fps)	Change in means	Geometric-mean ratio	Cross-sequence CV
Radeon R9 200 Series, dedicated JPEG block	x64	112.97	134.36	+18.93 %	1.203×	1.66 %
Radeon R9 200 Series, dedicated JPEG block	x86	101.57	127.33	+25.36 %	1.258×	13.39 %
Radeon R7 Graphics, shader array	x64	86.14	82.21	-4.56 %	0.955×	17.11 %

Radeon R7 Graphics, shader array	x86	69.96	65.65	-6.16 %	0.940×	22.76 %
--	-----	-------	-------	---------	--------	---------

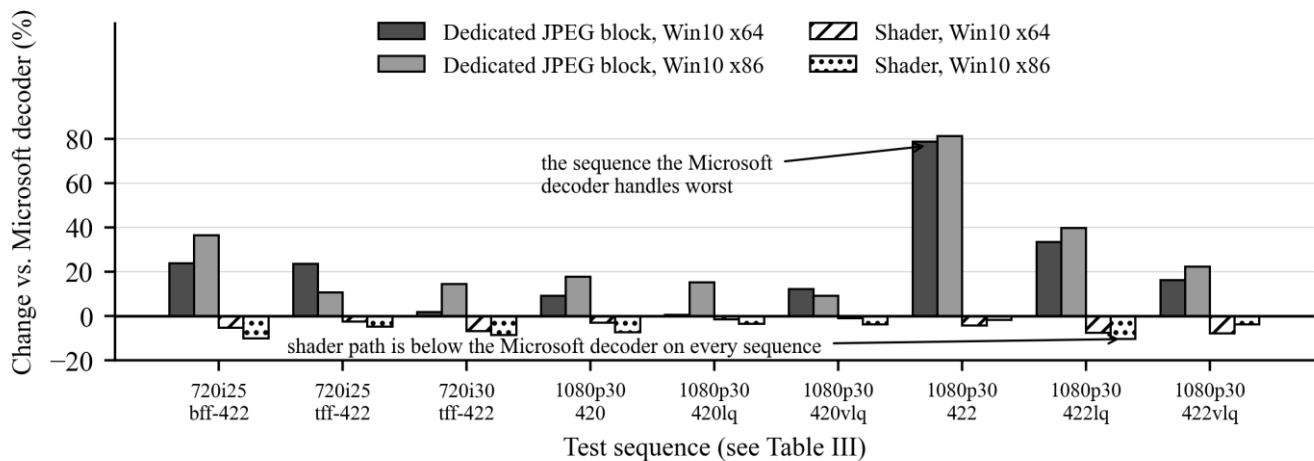


Fig. 4. Per-sequence change relative to the Microsoft decoder on the same GPU. Dedicated and shader paths were measured on different GPU models; the figure compares each path with its local baseline rather than directly comparing the GPUs.

A. Dedicated-hardware path

Geometric-mean ratios of 1.203 and 1.258 confirm that the mean gains in Table V survive multiplicative combination of the per-sequence ratios, over a spread from 0.38 % to 78.66 %.

The gain is largest where the Microsoft decoder is weakest. On 1080p30-422 that decoder manages 76.05 fps, its worst result and 32.7 % below its own mean, while the dedicated path delivers 135.87 fps. Across the 1080p 4:2:2 quality series the gains fall monotonically at 78.66 %, 33.36 % and 16.16 % for nominal, low and very low quality. For otherwise related clips, coarser quantisation commonly shortens the entropy-coded data, so this pattern is consistent with an entropy-decoding bottleneck in the inbox decoder; stage-level profiling would be required to establish causation.

On 64-bit Windows, throughput also becomes less sensitive to the tested content: the dedicated path's standard deviation across the nine sequences is 2.23 fps and its coefficient of variation 1.66 %, against 15.34 % for the Microsoft decoder — a ninefold reduction in relative dispersion, which simplifies worst-case provisioning. The statistic compares average throughput among clips and does not measure frame-time jitter within one.

The 32-bit configuration is the exception: its cross-sequence coefficient of variation is 13.39 %, driven by 720i25tff-422 at 82.82 fps while the other eight clips lie between 127 and 137 fps. Excluding that clip, the coefficient is 2.82 %. The same clip reaches 132.46 fps on 64-bit Windows, so the anomaly is specific to the 32-bit path rather than the underlying coded content. Field-order or surface handling is a candidate explanation, but additional tracing is required.

The 64-bit dedicated results cluster between 130.48 and 137.79 fps despite substantial differences in resolution and coding parameters. This flat range suggests that a shared per-frame cost, such as submission or synchronisation, may dominate after decode offload; profiling is needed to identify the limiting stage.

B. Shader path

The shader path is slower than the Microsoft decoder on all eighteen sequence–architecture pairs, by between 0.91 % and 10.30 %, as Fig. 4 shows. The consistent sign motivates the queue-depth explanation below, although repeated-run variance is unavailable.

C. Interpreting the shader-path deficit

The stages alternate rather than overlap. In the measured one-frame queue, the GPU cannot start reconstruction until CPU entropy decoding has produced coefficients. If the next frame is not admitted concurrently, per-frame time approaches the sum of the CPU stage, transfer and dispatch overhead, and the GPU stage rather than their maximum. MJPEG has no inter-frame dependency to force a deeper queue, so low latency does not by itself supply the pipelining that shader offload needs.

Amdahl's law bounds the gain and overhead can consume it. If entropy decoding is a fraction e of a conventional sequential decode and remains on the CPU in the shader path, offloading every other stage has an ideal speedup ceiling of $1/e$ [16], while each frame still incurs coefficient staging, dispatch, synchronisation and surface-management costs. The aggregate deficits of 4.56 % and 6.16 % are consistent with those costs exceeding the saved reconstruction time, a regime also reported for small GPU-offload workloads [9], [10]. Stage timing was not captured, so this remains a testable hypothesis.

The remedy follows. Because MJPEG frames are independent, k frames may be decoded concurrently, letting the CPU entropy-decode frame $n+1$ while the GPU transforms frame n . This converts per-frame cost from a sum into approximately a maximum and amortises launch overhead over k frames, at the price of up to $k-1$ frame periods of added latency — acceptable for playback, but to be applied cautiously in conferencing.

D. Host-CPU utilisation

Activity sampled at one instant during playback shows the player application's reported CPU utilisation falling from 5.38 % to 3.62 % once the vendor driver and transform are installed and hardware-accelerated decoding is active, a relative reduction of

32.7 %, while reported GPU utilisation rises from zero to 4.79 % (Fig. 5). The source report separates only the driver-absent and driver-present conditions, so the sample is attributed to hardware acceleration generally, not to one engine. Host-side parsing, buffer construction and surface management remain on the CPU, so this is reduced application CPU activity, not CPU elimination.

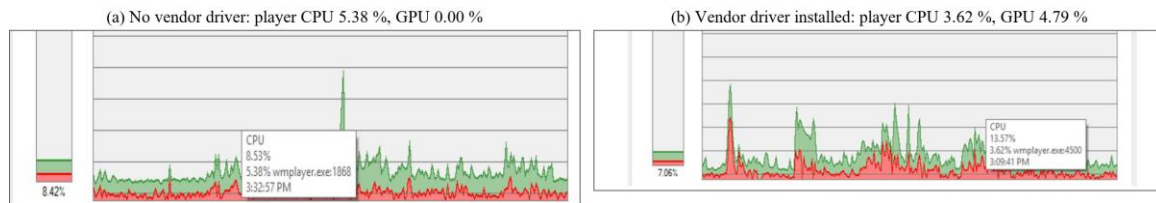


Fig. 5. Host CPU activity during playback of the same clip in the same player: (a) no vendor graphics driver installed, so the Microsoft decoder runs; (b) vendor driver and transform installed, so hardware-accelerated decoding is active. Tooltips give the instantaneous system and per-process values. Total system load is uncontrolled between samples, so only the per-process figures are interpreted.

These are utilisation samples, not power measurements: the source report supplies neither repeated samples nor controlled package-energy readings, and total CPU utilisation also rose from 8.53 % to 13.57 %, so driver overhead and background activity cannot be separated from the decoder change. Utilisation is in any case an unreliable power proxy [17], and activity percentages for a fixed-function block and a CPU core are not commensurable. Hardware decoders show large energy advantages for other codecs [8], but with no same-platform MJPEG energy comparison this study claims CPU-activity reduction only.

E. Fidelity considerations

Conformant JPEG decoders need not produce identical output, because the standard prescribes no single finite-precision IDCT [1], [2]. Entropy parsing, DC reconstruction, run-length expansion, inverse zig-zag ordering and dequantisation are defined integer operations; implementation-dependent variation enters mainly through the IDCT, rounding, chroma upsampling, colour conversion and clamping, and libjpeg-turbo accordingly exposes several IDCT and upsampling choices with differing speed–accuracy trade-offs [18]. ITU-T T.83 | ISO/IEC 10918-2 defines compliance tests through prescribed test data and procedures [19], and as a practical regression criterion Chromium's MJPEG accelerator test accepts a mean absolute YCbCr-sample difference no greater than 1.25 between hardware and software output [4].

A defensible validation must therefore hold chroma layout and colour conversion constant, decode the same conformance and natural-image set through all paths, and report maximum error, mean absolute error and PSNR. The available source data contain no decoded-frame comparisons, so this paper makes no measured fidelity or bit-exactness claim.

VII. CONCLUSION

A user-mode Media Foundation Transform was built to expose a dedicated JPEG decode block to unmodified Windows applications and to compare dedicated JPEG hardware, shader offload and the Microsoft decoder across two AMD GPUs on a common sequence set. Relative to the Microsoft decoder, dedicated-path arithmetic-mean throughput rose by 18.93 % on 64-bit and 25.36 % on 32-bit Windows 10, with a documented per-sequence gain up to 78.66 %, and the 64-bit cross-sequence coefficient of variation fell from 15.34 % to 1.66 %, indicating lower content sensitivity though not measuring frame-time jitter. A single utilisation sample showed 32.7 % lower player CPU activity. The shader path was 4.56 % and 6.16 % slower in aggregate and slower on every sequence–architecture pair; queue serialisation and dispatch overhead are a plausible cause that stage-level profiling would confirm.

The practical lesson is that "hardware acceleration" does not identify one execution model: dedicated offload improved throughput while the unit-depth shader implementation regressed it. A deployment policy should select paths from measured capability and workload data rather than assume any GPU path is faster. For this implementation the supported order is dedicated JPEG hardware, then the Microsoft decoder, with shader offload experimental until batching is evaluated.

VIII. FUTURE SCOPE

Multi-frame batching on the JPEG block. The measured dedicated path still presents one frame at a time, so host-side marker parsing and block execution cannot overlap. A queue of k independent MJPEG frames would let the host parse and build buffers for frame $n+1$ while the JPEG engine decodes frame n , hiding parse and submission latency behind the hardware, at the price of up to $k-1$ extra frame periods.

Cut the fixed per-frame cost. A flat 130.48–137.79 fps across very different clips points to submission and surface management dominating once entropy decoding and IDCT are already on the block. Surface pools, reused picture-parameter and table buffers, asynchronous completion and grouped command submission are the targets.

Widen what the block is asked to decode. Supporting 4:4:4, more than four scans or components, and $\Sigma H_i \cdot V_i$ above ten wherever the JPEG engine allows would remove software rejection of otherwise legal baseline streams. Several camera streams on one block, and concurrent JPEG decode with an active encode engine, should be measured because a single fixed-function unit is then shared.

Bind the Windows 11 public DDI. The evaluated path used the vendor interface available on Windows 10. Mapping the same buffers onto the documented DXVA MJPEG structures [11] would drop the vendor-specific submit path and let the same MFT run on later OS builds without a private decode interface.

REFERENCES

- [1] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Trans. Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.
- [2] ITU-T Recommendation T.81 | ISO/IEC 10918-1, *Digital Compression and Coding of Continuous-Tone Still Images: Requirements and Guidelines*, 1992.
- [3] Microsoft, "MJPEG At Source Autodecode for USB Video Class (UVC)." [Online]. Available: <https://learn.microsoft.com/windows-hardware/drivers/stream/mjpeg-at-source-autodecode>
- [4] The Chromium Authors, "MJPEG decode accelerator interface and unit test," components/chromeos_camera/. [Online]. Available: <https://chromium.googlesource.com/chromium/src>
- [5] Microsoft, "Supported Media Formats in Media Foundation." [Online]. Available: <https://learn.microsoft.com/windows/win32/medfound/supported-media-formats-in-media-foundation>
- [6] AMD, "AMF Video Decode API: decoder component identifiers and accelerator capability headers," Advanced Media Framework SDK. [Online]. Available: <https://github.com/GPUOpen-LibrariesAndSDKs/AMF>
- [7] A. Weißenberger and B. Schmidt, "Accelerating JPEG decompression on GPUs," in *Proc. IEEE Int. Conf. High Performance Computing (HiPC)*, 2021, pp. 121–130.
- [8] M. Kränzler, C. Herglotz and A. Kaup, "A comprehensive review of software and hardware energy efficiency of video decoders," in *Proc. Picture Coding Symposium*, 2024.
- [9] B. Wang, M. Alvarez-Mesa, C. C. Chi and B. Juurlink, "Highly parallel HEVC decoding for heterogeneous systems with CPU and GPU," *Signal Processing: Image Communication*, vol. 62, pp. 93–105, 2018.
- [10] P. Georgiev, N. D. Lane, C. Mascolo and D. Chu, "Accelerating mobile audio sensing algorithms through on-chip GPU offloading," in *Proc. ACM MobiSys*, 2017.
- [11] Microsoft, "DXVA_PicParams_MJPEG, DXVA_QMatrix_MJPEG and DXVA_HuffmanTable_MJPEG," Windows driver documentation, minimum supported client Windows 11 24H2. [Online]. Available: <https://learn.microsoft.com/windows-hardware/drivers/ddi/dxva/>
- [12] FFmpeg Project, "libavcodec/mjpegdec.c and libavcodec/mjpeg_parser.c." [Online]. Available: <https://github.com/FFmpeg/FFmpeg>
- [13] Microsoft, "Media Foundation Transforms, MFTEnumEx, and Codec Merit." [Online]. Available: <https://learn.microsoft.com/windows/win32/medfound/media-foundation-transforms>
- [14] Xiph.Org Foundation, "Video Test Media." [Online]. Available: <https://media.xiph.org>
- [15] Microsoft, "DXGK_ENGINE_TYPE enumeration"; S. Pronovost, "GPUs in the Task Manager," DirectX Developer Blog, 2017.
- [16] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. AFIPS Spring Joint Computer Conf.*, 1967, pp. 483–485.
- [17] S. Rivoire, P. Ranganathan and C. Kozyrakis, "A comparison of high-level full-system power models," in *Proc. USENIX HotPower*, 2008.
- [18] The libjpeg-turbo Project, "DCT/IDCT and decompression options," doc/libjpeg.txt. [Online]. Available: <https://github.com/libjpeg-turbo/libjpeg-turbo>
- [19] ITU-T Recommendation T.83 | ISO/IEC 10918-2, *Digital Compression and Coding of Continuous-Tone Still Images: Compliance Testing*, 1994. [Online]. Available: <https://www.itu.int/rec/T-REC-T.83/en>

Copyright & License: