Memory access patterns in Web Codecs

Current state and future developments

Paul Adenot

moz://a

W3C/SMPTE Joint Workshop on Professional Media Production on the Web

memcpy is murder

Some numbers for a frame

- YUV420 1080p video frame SDR:
  1920 * 1080 * 2 ≈ **4MB**
- YUV420 4k video frame SDR:
  3840 * 2160 * 2 ≈ **16MB**
- P010 (≈YUV420 10-bits) 4k video frame HDR:
  3840 * 2160 * 4 ≈ **32MB**

Time for a copy on really fast desktop workstation

AVX512 memcpy + DDR4 + optimized C++ = close to best case

- Hot caches
  - YUV420 1080p video frame SDR ≈ 1.5ms
  - YUV420 4k video frame SDR ≈ 6.6ms
  - P010 4k 10-bits video frame HDR ≈ 15ms
- Cold caches
  - YUV420 1080p video frame SDR ≈ 4.5ms
  - YUV420 4k video frame SDR ≈ 17ms
  - P010 4k 10-bits video frame HDR ≈ 33ms

GPU to CPU readback and upload

- Hardware decoded frames in GPU memory sometimes need to be copied to regular memory, this is very expensive
- It's always better to keep the VideoFrames on the GPU *if possible*

Why copy

- Custom post-processing in JavaScript or WASM
- Necessary to move data over to the WASM heap
- Sometimes necessary to work with other Web APIs

WebCodecs tries very hard to minimize copies

- Memory not explicitely visible: optimizations happen under the hood (e.g. copy on write), GPU surfaces are efficiently referenced
- Explicit copyTo methods to make it extra clear
- clone() method does *not* do a deep-copy

Necessary copies part 1 - easy fixes

- decode input: data is copied (WebCodecs issue #104)
- VideoFrame and AudioData copyTo: no way to "steal" the underlying memory yet (WebCodecs issue #287)
- memory cycling / allocator pressure (WebCodecs issue #212)

Buffer stealing

```
partial interface VideoFrame {
  // closes the VideoFrame and transfer memory
  Promise<ArrayBuffer> detach();
};
```

(similar for AudioData).

Limit native allocator pressure

```
partial interface AudioDecoder {
  // Detaches destination (need to be big enough)
  // and write into it
  // Detaches the memory in EncodedAudioChunk
  undefined decode(EncodedAudioChunk chunk,
                   ArrayBuffer destination);
};

// `input` is the memory that was owned by `chunk`
callback AudioDataOutputCallback =
    undefined(AudioData output, ArrayBuffer input);
```

Necessary copies part 2 - harder problems

- Necessity to copy from/to the WASM heap
- Danger of SharedArrayBuffer vs. non-auditable codecs
- No read-only memory ranges
- No read-only memory: can't use memory ranges in encoder/decoders (BYOB)

Summary and positions at https://github.com/WICG/reducing-memory-copies, WebCodecs positions issue #1.

Conclusion

There are problems, but there are also solutions in the works.

Generally, lots of common scenario work really well, but advance use-cases can be improved.