VDD 20~~3~~4

An Overview of dav1d

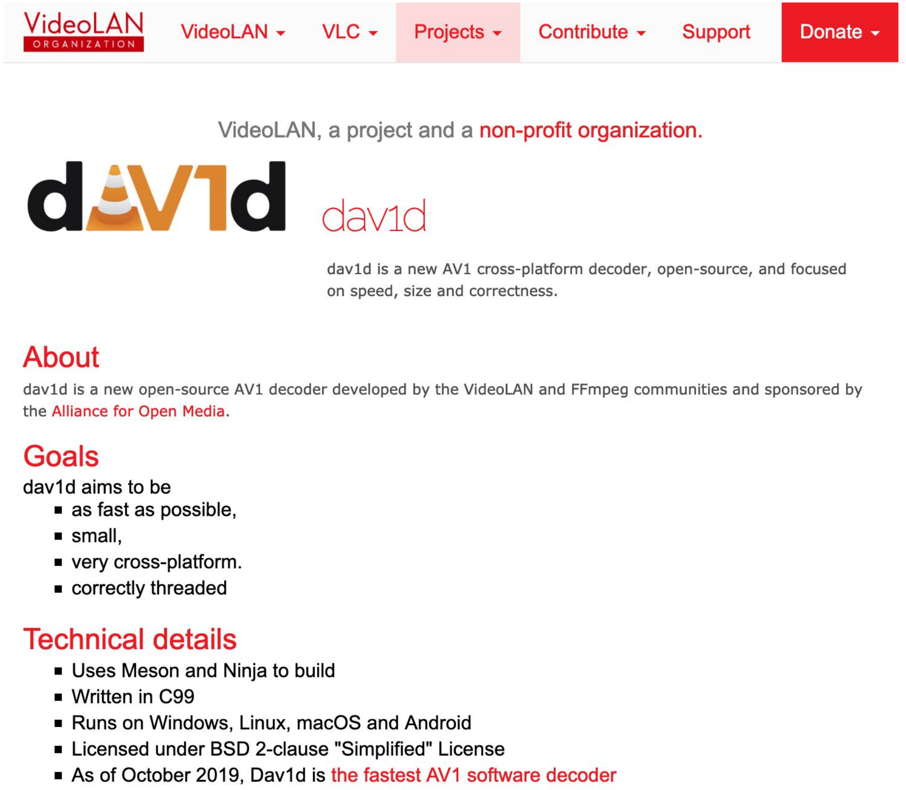
Ronald S. Bultje | Two Orioles

~~September 2023~~November 2024

dav1d: Introduction

Goals of this Presentation:

- Introduction
- Update on our assembly coverage and some recent improvements
- Update on adoption
- A slide that I should have included last year on sponsorship
- Conclusion



VideoLAN, a project and a non-profit organization.

dav1d

dav1d is a new AV1 cross-platform decoder, open-source, and focused on speed, size and correctness.

About

dav1d is a new open-source AV1 decoder developed by the VideoLAN and FFmpeg communities and sponsored by the Alliance for Open Media.

Goals

dav1d aims to be

- as fast as possible,
- small,
- very cross-platform.
- correctly threaded

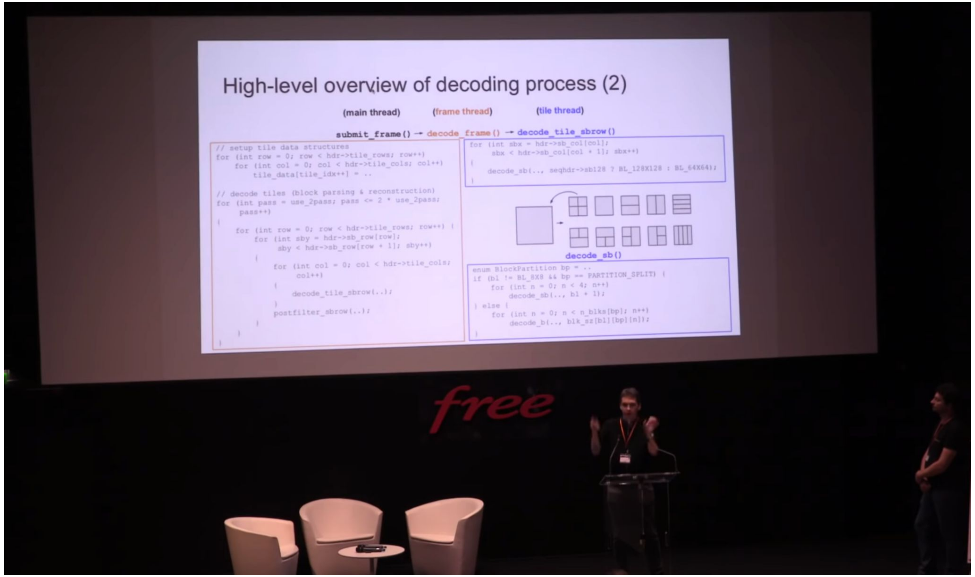
Technical details

- Uses Meson and Ninja to build
- Written in C99
- Runs on Windows, Linux, macOS and Android
- Licensed under BSD 2-clause "Simplified" License
- As of October 2019, Dav1d is the fastest AV1 software decoder

dAV1d: Why do we need a Fast Software Decoder?

Raison d'être

- To show off the awesomeness of AV1, we need an ecosystem that supports it
    - But my phone (or TV, or laptop) doesn't have hardware AV1 decoding yet
    - This slows down early adoption and can cause uptake to stall entirely: content availability and device support are mutually dependent
- We need a software decoder to jumpstart adoption and break the dependency cycle!
    - That is dav1d, introduced here at VDD in 2018
    - Goal: be an awesome software decoder until hardware makes us irrelevant
- So, where are we now?



High-level overview of decoding process (2)

(main thread) (frame thread) (tile thread)

submit_frame() → decode_frame() → decode_tile_sbrow()

```
// setup tile data structures
for (int row = 0; row < hdr->tile_rows; row++)
    for (int col = 0; col < hdr->tile_cols; col++)
        tile_data[tile_idx++] = ..

// decode tiles (block parsing & reconstruction)
for (int pass = use_2pass; pass <= 2 * use_2pass;
     pass++)
{
    for (int row = 0; row < hdr->tile_rows; row++) {
        for (int sby = hdr->sb_row[row];
             sby < hdr->sb_row[row + 1]; sby++)
        {
            for (int col = 0; col < hdr->tile_cols;
                 col++)
            {
                decode_tile_sbrow(..);
            }
            postfilter_sbrow(..);
        }
    }
}
```

```
for (int sbx = hdr->sb_col[col];
     sbx < hdr->sb_col[col + 1]; sbx++)
{
    decode_sb(.., seqhdr->sb128 ? BL_128X128 : BL_64X64);
}
```

decode_sb()

```
enum BlockPartition bp = ..
if (bl != BL_8X8 && bp == PARTITION_SPLIT) {
    for (int n = 0; n < 4; n++)
        decode_sb(.., bl + 1);
} else {
    for (int n = 0; n < n_blks[bp]; n++)
        decode_b(.., blk_sz[bl][bp][n]);
}
```

dAV1d: Where We Are Today (*i.e. last year*)

Current State:

We're pretty much done!

- 80% asm (>150kLOC), 20% C (28kLOC)

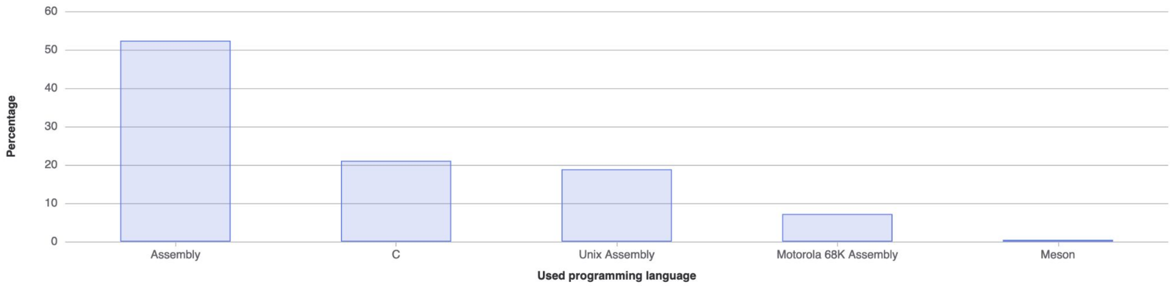
x86:

- 95% complete SSSE3 for 8, 10 and 12bpc content, runs on x86-32 & x86-64
  - 10-bit inverse transforms are SSE4
- 98% complete AVX2 for 8, 10 and 12bpc content, runs on x86-64
- 80% done with AVX512-IceLake for 8, 10 and 12bpc content, runs on x86-64
  - Lacks directional intra prediction

Arm:

- 95% complete Neon for 8, 10 and 12bpc content, runs on arm32 & aarch64

Let's have a look at some of this assembly!



dAV1d: Where We Are Today (*i.e. last year*)

Current State:

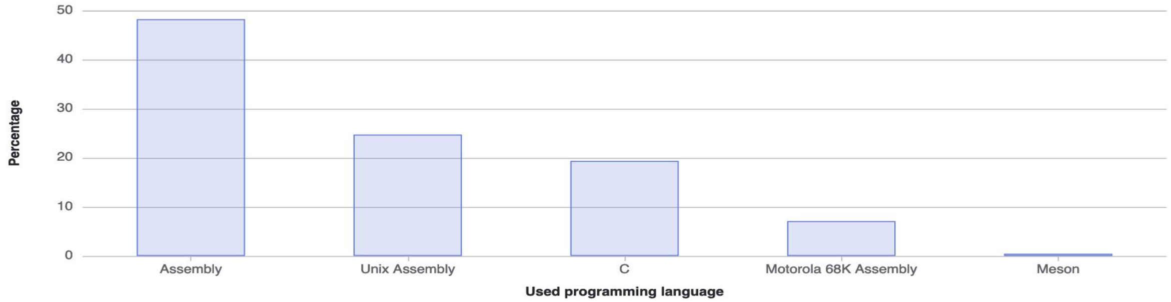
We’re pretty much done!

- 80% asm (>150kLOC), 20% C (28kLOC)

I should not have said that!

Per architecture:

- C: 31kLOC
- x86: 160kLOC (SSSE3, AVX2, AVX512)
- arm: 72kLOC
- risc-V64: 5.1kLOC
- loongaarch: 23kLOC
- ppc: 5.1kLOC



dAV1d: Asm (mostly) Improvements Since 2023

Algorithmic:

arm:

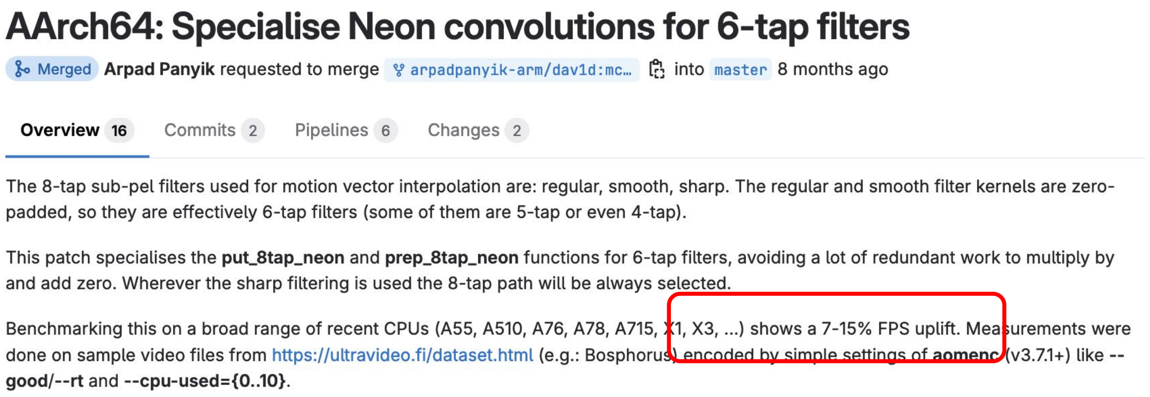
- dotprod, imm8, SVE2 extensions (Arpad)
- use 6-tap (instead of 8-tap) filter for MC when not sharp (Arpad) - 7-15% (!!)
- SGR improvements to calculate (instead of table-look-up) x_by_x (Kyle)

x86:

- use 6-tap (instead of 8-tap) filter for MC when not sharp (Henrik, based on Arpad's arm variant)
- SGR improvements to calculate (instead of table-look-up) x_by_x (Henrik, based on proof-of-concept by Kyle)

risc-v64 (Nathan, Bogdan, Remi), loongaarch (Hecai & all), ppc (Luca):

- Lots of SIMD added.



AArch64: Specialise Neon convolutions for 6-tap filters

Merged **Arpad Panyik** requested to merge `arpadpanyik-arm/dav1d:mc...` into `master` 8 months ago

Overview 16 | Commits 2 | Pipelines 6 | Changes 2

The 8-tap sub-pel filters used for motion vector interpolation are: regular, smooth, sharp. The regular and smooth filter kernels are zero-padded, so they are effectively 6-tap filters (some of them are 5-tap or even 4-tap).

This patch specialises the **put_8tap_neon** and **prep_8tap_neon** functions for 6-tap filters, avoiding a lot of redundant work to multiply by and add zero. Wherever the sharp filtering is used the 8-tap path will be always selected.

Benchmarking this on a broad range of recent CPUs (A55, A510, A76, A78, A715, X1, X3, ...) shows a 7-15% FPS uplift. Measurements were done on sample video files from https://ultravideo.fi/dataset.html (e.g.: Bosphorus) encoded by simple settings of **aomenc** (v3.7.1+) like **--good/--rt** and **--cpu-used={0..10}**.

dav1d: Asm (mostly) Improvements Since 2023

Algorithmic:

arm:

- dotprod, imm8, SVE2 extensions (Arpad)
- use 6-tap (instead of 8-tap) filter for MC when not sharp (Arpad) - 7-15% (!!)
- SGR improvements to calculate (instead of table-look-up) x_by_x (Kyle)

x86:

- use 6-tap (instead of 8-tap) filter for MC when not sharp (Henrik, based on Arpad's arm variant)
- SGR improvements to calculate (instead of table-look-up) x_by_x (Henrik, based on proof-of-concept by Kyle)

risc-v64 (Nathan, Bogdan, Remi), loongaarch (Hecai & all), ppc (Luca):

- Lots of SIMD added.

```
dst[y,x] = c1 * src[y,x-3] +
           c2 * src[y,x-2] +
           c3 * src[y,x-1] +
           c4 * src[y,x+0] +
           c5 * src[y,x+1] +
           c6 * src[y,x+2] +
           c7 * src[y,x+3] +
           c8 * src[y,x+4];
```

```
if c1 == 0 and c8 == 0:

dst[y,x] = c2 * src[y,x-2] +
           c3 * src[y,x-1] +
           c4 * src[y,x+0] +
           c5 * src[y,x+1] +
           c6 * src[y,x+2] +
           c7 * src[y,x+3];
```

dav1d: Asm (mostly) Improvements Since 2023

Algorithmic:

arm:

- dotprod, imm8, SVE2 extensions (Arpad)
- use 6-tap (instead of 8-tap) filter for MC when not sharp (Arpad) - 7-15% (!!)
- SGR improvements to calculate (instead of table-look-up) x_by_x (Kyle)

x86:

- use 6-tap (instead of 8-tap) filter for MC when not sharp (Henrik, based on Arpad's arm variant)
- SGR improvements to calculate (instead of table-look-up) x_by_x (Henrik, based on proof-of-concept by Kyle)

risc-v64 (Nathan, Bogdan, Remi), loongaarch (Hecai & all), ppc (Luca):

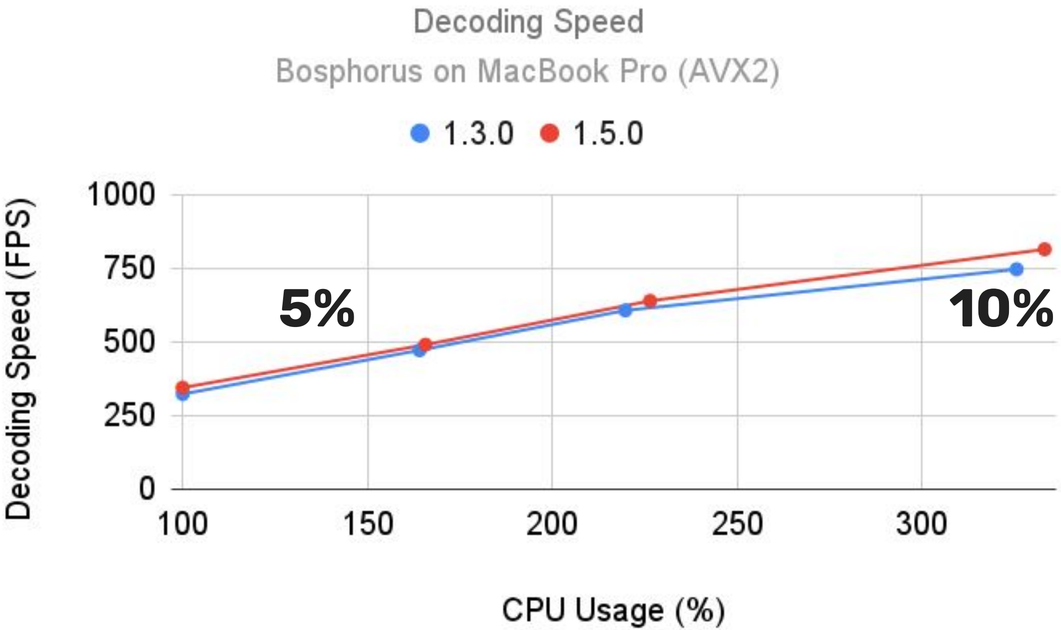
- Lots of SIMD added.

```
x = x_by_x[z];
```

```
x = 256 / (z + 1);

(in float, because there is
no int SIMD div)
```

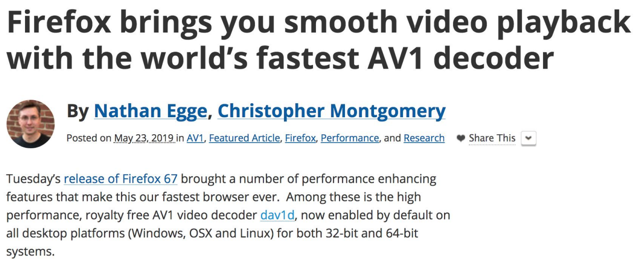
dAV1d: Decoding Performance Analysis



dav1d: concluding remarks

Adoption

- dav1d is a pretty fast AV1 decoder
- It's used in many places:
  - Browsers (Chrome / Firefox / Safari *)
  - Open Source Media Frameworks & applications based on them: VLC, FFmpeg, GStreamer, etc.
  - System Frameworks: AV1 Video Extension available on Microsoft Store, AVIF support using dav1d on MacOS/iPhoneOS, recent integration as software fallback for Android
  - Closed-source mobile applications (Netflix, Instagram Reels)
  - Probably others but we don't really keep track, because…

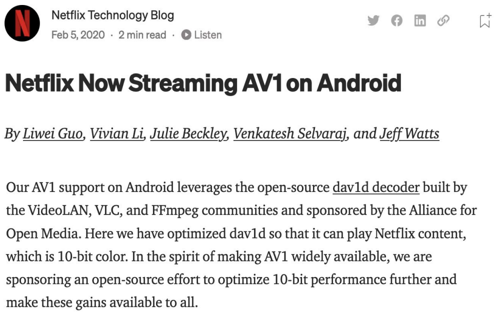


Firefox brings you smooth video playback with the world's fastest AV1 decoder

By Nathan Egge, Christopher Montgomery

Posted on May 23, 2019 in AV1, Featured Article, Firefox, Performance, and Research ❤ Share This

Tuesday's release of Firefox 67 brought a number of performance enhancing features that make this our fastest browser ever. Among these is the high performance, royalty free AV1 video decoder dav1d, now enabled by default on all desktop platforms (Windows, OSX and Linux) for both 32-bit and 64-bit systems.



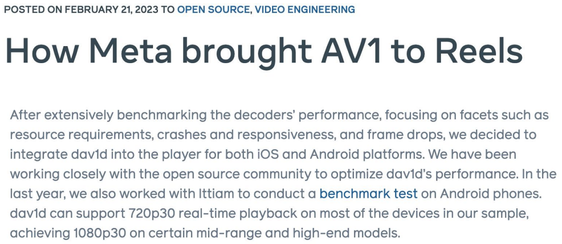
Netflix Technology Blog

Feb 5, 2020 · 2 min read · Listen

Netflix Now Streaming AV1 on Android

By Liwei Guo, Vivian Li, Julie Beckley, Venkatesh Selvaraj, and Jeff Watts

Our AV1 support on Android leverages the open-source dav1d decoder built by the VideoLAN, VLC, and FFmpeg communities and sponsored by the Alliance for Open Media. Here we have optimized dav1d so that it can play Netflix content, which is 10-bit color. In the spirit of making AV1 widely available, we are sponsoring an open-source effort to optimize 10-bit performance further and make these gains available to all.



POSTED ON FEBRUARY 21, 2023 TO OPEN SOURCE, VIDEO ENGINEERING

How Meta brought AV1 to Reels

After extensively benchmarking the decoders' performance, focusing on facets such as resource requirements, crashes and responsiveness, and frame drops, we decided to integrate dav1d into the player for both iOS and Android platforms. We have been working closely with the open source community to optimize dav1d's performance. In the last year, we also worked with Ittiam to conduct a benchmark test on Android phones. dav1d can support 720p30 real-time playback on most of the devices in our sample, achieving 1080p30 on certain mid-range and high-end models.

https://engineering.fb.com/2023/02/21/video-engineering/av1-codec-facebook-instagram-reels/
https://netflixtechblog.com/netflix-now-streaming-av1-on-android-d5264a515202
https://hacks.mozilla.org/2019/05/firefox-brings-you-smooth-video-playback-with-the-worlds-fastest-av1-decoder/

dAV1d: concluding remarks

Future Directions

- CDEF in block stripes (!1458)
- GPU acceleration
- Continue asm work for new instruction sets (e.g. SVE2, RISC-V, PPC, MIPS)
- dAV2d

dav1d: Sponsorship & Thanks (*forgotten last year*)

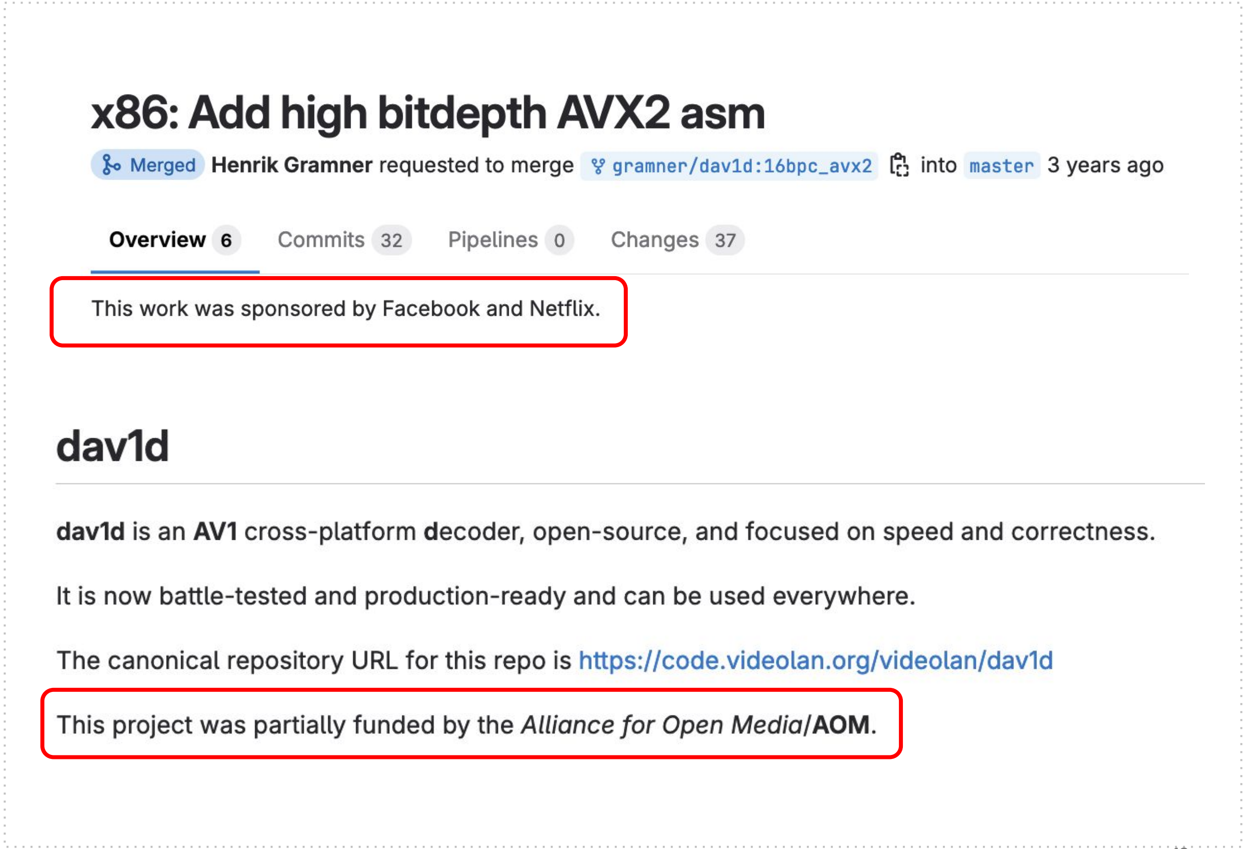
Funding acknowledgements:

Some dav1d contributions were made possible by sponsorship from:

- Alliance of Open Media
- Meta
- Netflix

Some large-scale projects like 10-bit assembly, the task-threading framework or the initial start of the project were made possible by the above parties. *Without them, dav1d would not be where it is today.*

Thank you!



x86: Add high bitdepth AVX2 asm

Merged **Henrik Gramner** requested to merge `gramner/dav1d:16bpc_avx2` into `master` 3 years ago

This work was sponsored by Facebook and Netflix.

dav1d

dav1d is an **AV1** cross-platform **d**ecoder, open-source, and focused on speed and correctness.

It is now battle-tested and production-ready and can be used everywhere.

The canonical repository URL for this repo is https://code.videolan.org/videolan/dav1d

This project was partially funded by the *Alliance for Open Media*/**AOM**.

Thank You

Ronald S. Bultje

rbultje@twoorioles.com

https://code.videolan.org/videolan/dav1d

```
$ git shortlog -sn
Henrik Gramner
Martin Storsjö
Ronald S. Bultje
Janne Grunau
James Almer
Nathan E. Egge
Victorien Le Couviour--Tuffet
Matthias Dressel
Jean-Baptiste Kempf
Marvin Scholz
Luc Trudeau
Arpad Panyik
yuanhecai
Luca Barbato
Niklas Haas
Hugo Beauzée-Luyssen
Konstantin Pavlov
Kyle Siefring
David Michael Barr
Steve Lhomme
Cameron Cawley
Wan-Teh Chang
B Krishnan Iyer
Francois Cartegnie
Liwei Wang
Bogdan Gligorijević
David Conrad
Michael Bradshaw
pengxu
Derek Buitenhuis
Jan Beich
Raphaël Zumer
Xuefeng Jiang
jinbo
Christophe Gisquet
Justin Bull
Boyuan Xiao
Dale Curtis
Emmanuel Gil Peyrot
Raphael Zumer
zhoupeng
Kacper Michajłow
Rupert Swarbrick
Thierry Foucu
Thomas Daede
guxiwei
André Kempe
Colin Lee
Jonathan Wright
Lynne
Michail Alvanos
Nico Weber
Salome Thirot
SmilingWolf
Tristan Laurent
Tristan Matthews
Vittorio Giovara
Yannis Guyon
Andrey Semashev
Anisse Astier
Anton Mitrofanov
Brad Smith
Charlie Hayden
Cosmin Stejerean
Dmitriy Sychov
Ewout ter Hoeven
Fred Barbier
Hao Chen
Jean-Yves Avenard
Joe Drago
MARBEAN
Mark Shuttleworth
Matthieu Bouron
Mehdi Sabwat
Nicolas Frattaroli
Pablo Stebler
Peter Collingbourne
Rostislav Pehlivanov
Sebastian Dröge
Shiz
Steinar Midtskogen
Sylvain BERTRAND
Sylvestre Ledru
Timo Gurr
Vibhoothi
Vignesh Venkatasubramanian
Xavier Claessens
Xu Guangxin
kossh1
skal
```

Two Orioles