Fast &

High Quality &

AV1

By Ronald S. Bultje <rbultje@twoorioles.com>

BigApple.Video 2019
Goal of this talk

Our contributions to the AV1 platform:

- overview of dav1d (software decoder)
  - performance comparison vs. other AV1 (libaom) and non-AV1 (FFmpeg’s built-in VP9/H264/HEVC) software decoders

- overview of Eve-AV1 (software encoder)
  - performance comparison vs. other publicly-available AV1 (libaom, rav1e, svt-av1) and non-AV1 (x264, x265, vpxenc) software encoders

Definition of “overview”:

- speed
- quality
- multi-threading
  - speedup
  - memory usage
  - system load
- portability
- adoption
dav1d is an AV1 decoder
What is dav1d?

- an AV1 decoder, written from scratch
- released at VDD’18 (Sept) in Paris [1]
  - part of the Videolan project
- BSD-licensed with open community
  - https://code.videolan.org/videolan/dav1d
  - irc.freenode.net #dav1d
- sponsored by the Alliance for Open Media
  - collaboration of Two Orioles, VideoLabs and MultiCoreWare

and why?

- we (all of us) need a production-grade AV1 decoder to jumpstart AV1 platform support while we wait for initial hardware implementations
- libaom is not good enough
- we know how to do this, because we’ve done it before [2] [3]

dav1d - decoding speed vs. libaom

Experiment:
- encode some standard 1080p clips [1] at standard CRFs using aomenc
- decode files using FFmpeg’s libaom or dav1d wrappers (single-threaded)
- on a MacBook Pro 2015, x86-64 with AVX2 (Haswell)

Result:
- dav1d is twice as fast as libaom

dav1d - multi-threading vs. libaom

Experiment:

- repeat previous experiments w/ threading
  - tile threading
  - frame threading

Results:

- dav1d is 3x as fast as libaom
  - faster baseline
  - scales further
3x as fast!
dav1d - memory usage vs. libaom

Experiment:
- repeat previous experiments, but measure memory usage instead of runtime

Results:
- in the optimal configuration (4 frame threads x 4 tile threads), dav1d uses the same amount of memory as libaom
  - A patch currently under review reduces memory usage by up to 70%!
dav1d vs. libaom - code size and binary size

- **dav1d contains:**
  - 32.1 kLOC C;
  - 37.5 kLOC AVX2/SSSE3 (x86); and
  - 10.0 kLOC Neon (arm) assembly.

- **libaom (decoder only) contains:**
  - 73.9 kLOC C;
  - 30.1 kLOC SSE2/SSSE3/SSE4/AVX2 (x86); and
  - 12.6 kLOC Neon (arm) assembly;
  - 0.1 kLOC PPC assembly.

- **libdav1d.2.dylib is 0.91MB; libaom.a (decoder only) is 2.91MB**
How is dav1d so small and yet so fast?

Example: inverse (2D) transforms

- forward transform: reduce signal in spatially correlated values
  - reduces quantization quality loss
- inverse transform: do the opposite of whatever the forward transform did
How is dav1d so small and yet so fast?

Example: inverse (2D) transforms

- 1D transform sizes: 4, 8, 16, 32 or 64
  - 1:4, 1:2, 1:1, 2:1 and 4:1 ratio
- 1D transform types:
  - 4, 8, 16: DCT, ADST, flipADST and identity
  - 32: DCT and identity
  - 64: DCT
  - 16x16 has no identity/(flip)ADST combinations
  - 4x4 can also be lossless (WHT/WHT)
- 155 unique 2D transforms!
  - How to keep binary size low?
How is dav1d so small and yet so fast?

Example: inverse (2D) transforms

- 1D transform sizes: 4, 8, 16, 32 or 64
  - 1:4, 1:2, 1:1, 2:1 and 4:1 ratio

- 1D transform types:
  - 4, 8, 16: DCT, ADST, flipADST and identity
  - 32: DCT and identity
  - 64: DCT
  - 16x16 has no identity/(flip)ADST combinations
  - 4x4 can also be lossless (WHT/WHT)

- 155 unique 2D transforms!
  - How to keep binary size low?
  - Write it out as 1D transforms!
    - Now, you only need 12 (+1 for lossless)
How is dav1d so small and yet so fast?

Example: inverse (2D) transforms

- **1D transform sizes:** 4, 8, 16, 32 or 64
  - 1:4, 1:2, 1:1, 2:1 and 4:1 ratio
- **1D transform types:**
  - 4, 8, 16: DCT, ADST, flipADST and identity
  - 32: DCT and identity
  - 64: DCT
  - 16x16 has no identity/(flip)ADST combinations
  - 4x4 can also be lossless (WHT/WHT)
- **155 unique 2D transforms!**
  - How to keep binary size low?
- **Write it out as 1D transforms!**
  - Now, you only need 12 (+1 for lossless)
How is dav1d so small and yet so fast?

Example: inverse (2D) transforms

- 1D transform sizes: 4, 8, 16, 32 or 64
  - 1:4, 1:2, 1:1, 2:1 and 4:1 ratio
- 1D transform types:
  - 4, 8, 16: DCT, ADST, flipADST and identity
  - 32: DCT and identity
  - 64: DCT
  - 16x16 has no identity/(flip)ADST combinations
  - 4x4 can also be lossless (WHT/WHT)
- 155 unique 2D transforms!
  - How to keep binary size low?
  - Write it out as 1D transforms!
  - Now, you only need 12 (+1 for lossless)
Down the rabbit hole: the AVX2 16x16 inverse DCT

```c
inv_transform_2d() {
    transpose()
    row_inv_transform_1d()
    transpose()
    col_inv_transform_1d()
    residual_add()
}
```

```c
transpose_16bit_16x16_avx2() {
    for (int n = 0; n < 16; n++)
        register_$n = in[n][0-15]
    [ rotate registers ]
    for (int n = 0; n < 16; n++)
        out[n][0-15] = register_$n
}
```
Down the rabbit hole: the AVX2 16x16 inverse DCT

```c
inv_transform_2d() {
    transpose()
    row_inv_transform_1d()
    transpose()
    col_inv_transform_1d()
    residual_add()
}
```

```c
idct16_new_avx2() {
    coef/round/shift_register = some constant
    for (int n = 0; n < 16; n++)
        register_$n = in[n][0-15];
        [ multiplies, additions and shifts ]
    for (int n = 0; n < 16; n++)
        out[n][0-15] = register_$n
}
```
Down the rabbit hole: the AVX2 16x16 inverse DCT

```c
inv_transform_2d() {
    transpose()
    row_inv_transform_1d()
    transpose()
    col_inv_transform_1d()
    residual_add()
}

lowbd_write_buffer_16xn_avx2() {
    for (int n = 0; n < 16; n++) {
        register = in[n][0-15]
        register += predictor[n][0-15]
        reconstruction[n][0-15] = register
    }
}
```
Down the rabbit hole: the AVX2 16x16 inverse DCT

```c
inv_transform_2d() {
    transpose()
    row_inv_transform_1d()
    transpose()
    col_inv_transform_1d()
    residual_add()
}
```

- 464 instructions or 2165 bytes for idct16_new_avx2()
  - Prologue: 5 instructions
  - Loading constants: 123 instructions
  - Loading input data: 16 instructions
  - Stack scratch load/store: 43 instructions
  - Storing output data: 16 instructions
  - Actual transform:
    - 13x16 = 208 instructions
    - 24x2 = 48 instructions
  - Epilogue: 5 instructions
Down the rabbit hole: the AVX2 16x16 inverse DCT

- 242 instructions or 1144 bytes for `dav1d_idct_16x16_internal_avx2.main`
  - **Prologue**
  - Loading constants: 21 instructions
  - Loading input data
  - Stack scratch load/store: 12 instructions
  - Storing output data
  - Actual transform:
    - 9x16 = 144 instructions
    - 4x4 = 16 instructions (simplify!)
    - 24x2 = 48 instructions
  - Epilogue: 1 instruction (just ret!)

- 464 instructions or 2165 bytes for `idct16_new_avx2()`
  - **Prologue**: 5 instructions
  - Loading constants: 123 instructions
  - Loading input data: 16 instructions
  - Stack scratch load/store: 43 instructions
  - Storing output data: 16 instructions
  - Actual transform:
    - 13x16 = 208 instructions
    - 24x2 = 48 instructions
  - Epilogue: 5 instructions
Down the rabbit hole: the AVX2 16x16 inverse DCT

- 242 instructions or 1144 bytes for
dav1d_idct_16x16_internal_avx2.main
  - Prologue
  - Loading constants: 21 instructions
  - Loading input data
  - Stack scratch load/store: 12 instructions
  - Storing output data
  - Actual transform:
    - 9x16 = 144 instructions
    - 4x4 = 16 instructions (simpler!)
    - 24x2 = 48 instructions
  - Epilogue: 1 instruction (just ret!)
How is dav1d so small and yet so fast?

- Better and more assembly
  - Highly customized & hand-written
  - Includes assembly for multi-symbol arithmetic coder
- Dynamic research assumptions are replaced by static, built-in code assumptions
- Combination of two threading types (frame + tile) that make up for each other’s weaknesses
  - See talk at VDD last year for details [1]

How does dav1d compare to other codecs?

With industry experts estimating that AV1 decoding is an order of magnitude more complex than previous formats such as HEVC and VP9, SoC developers...
dav1d - decoding speed vs. VP9 / HEVC / H264

Experiment:

- encode clips using vpxenc, x265 or x264 at CRFs to match aomenc quality
- decode (no threading) using FFmpeg's built-in VP9/HEVC/H264 decoders

Result:

- dav1d is a bit (9%) slower than ffhevc
- dav1d is a lot (34% and 43%) slower than ffh264 and ffvp9

<table>
<thead>
<tr>
<th>Clip</th>
<th>Encoder</th>
<th>CRF</th>
<th>Bitrate (mbps)</th>
<th>PSNR (dB)</th>
<th>SSIM (dB)</th>
<th>VMAF (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarScene</td>
<td>aomenc</td>
<td>44</td>
<td>0.71</td>
<td>42.94</td>
<td>14.71</td>
<td>93.66</td>
</tr>
<tr>
<td></td>
<td>vpxenc</td>
<td>37</td>
<td>1.12</td>
<td>42.97</td>
<td>14.74</td>
<td>94.33</td>
</tr>
<tr>
<td></td>
<td>x265</td>
<td>26</td>
<td>1.02</td>
<td>42.47</td>
<td>14.34</td>
<td>93.74</td>
</tr>
<tr>
<td></td>
<td>x264</td>
<td>24</td>
<td>1.68</td>
<td>42.08</td>
<td>14.19</td>
<td>93.55</td>
</tr>
<tr>
<td>DrivingPOV</td>
<td>aomenc</td>
<td>44</td>
<td>3.70</td>
<td>38.11</td>
<td>12.76</td>
<td>91.88</td>
</tr>
<tr>
<td></td>
<td>vpxenc</td>
<td>41</td>
<td>5.03</td>
<td>38.19</td>
<td>12.81</td>
<td>91.60</td>
</tr>
<tr>
<td></td>
<td>x265</td>
<td>28</td>
<td>5.05</td>
<td>38.01</td>
<td>12.58</td>
<td>92.18</td>
</tr>
<tr>
<td></td>
<td>x264</td>
<td>25</td>
<td>6.52</td>
<td>37.74</td>
<td>12.73</td>
<td>92.05</td>
</tr>
<tr>
<td>ToddlerFountain</td>
<td>aomenc</td>
<td>44</td>
<td>8.64</td>
<td>36.06</td>
<td>11.14</td>
<td>76.84</td>
</tr>
<tr>
<td></td>
<td>vpxenc</td>
<td>41</td>
<td>10.77</td>
<td>36.19</td>
<td>11.23</td>
<td>77.41</td>
</tr>
<tr>
<td></td>
<td>x265</td>
<td>29</td>
<td>10.05</td>
<td>35.85</td>
<td>11.15</td>
<td>79.37</td>
</tr>
<tr>
<td></td>
<td>x264</td>
<td>27</td>
<td>13.38</td>
<td>35.51</td>
<td>11.21</td>
<td>79.93</td>
</tr>
</tbody>
</table>
dav1d - decoding speed vs. VP9 / HEVC / H264

Experiment:

- encode clips using vpxenc, x265 or x264 at CRFs to match aomenc quality
- decode (no threading) using FFmpeg's built-in VP9/HEVC/H264 decoders

Result:

- dav1d is a bit (9%) slower than ffhevc
- dav1d is a lot (34% and 43%) slower than ffh264 and ffvp9
dav1d - multi-threading vs. VP9 / HEVC / H264

Experiment:
- repeat previous experiments w/ threading

Results:
- dav1d is a bit (11%) slower than ffhevc
  - Compared to 9% single-threaded
- dav1d is “only” 21% slower than ffvp9
  - Compared to 43% single-threaded
  - this is primarily because ffvp9 fails to scale at higher thread counts
- dav1d is 38% slower than ffh264
  - Compared to 34% single-threaded
dav1d - memory usage vs. VP9 / HEVC / H264

Experiment:
- repeat previous experiments, but measure memory usage instead of runtime

Results:
- in the optimal configuration (4 frame threads x 4 tile threads), dav1d uses:
  - the same amount of memory as FFmpeg’s built-in decoders at 16 frame threads
  - almost twice as much memory as FFmpeg’s built-in decoders at 8 frame threads
Portability

Initial x86 efforts focused on 64-bit systems with AVX2 support (Haswell or later), which is 50% of the desktop/laptop user share.

For the remaining 50%, dav1d includes nearly complete SSSE3 optimizations, which are compatible with 32-bit systems as well, and cover >95% of the desktop/laptop user share.

On these systems, as with AVX2, dav1d is significantly faster than libaom.

Progress is monitored here:

https://code.videolan.org/videolan/dav1d/issues/216

dav1d also includes Neon optimizations for Arm as well as Aarch64. Assembly coverage is over 50% so far, and will soon hit 75%.

Progress is monitored here:

https://code.videolan.org/videolan/dav1d/issues/215

As of a few weeks ago, dav1d includes support for VSX extensions on PPC as well.

Progress is monitored here:

https://code.videolan.org/videolan/dav1d/issues/281
Adoption

dav1d is now included in:

- Chrome M74
- Firefox 67 (with 11.8% of all video playback being AV1 today!)
- VLC 3.0.5
- FFmpeg 4.2
- You?

Please send donations to VideoLAN:

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

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, OS X and Linux) for both 32-bit and 64-bit systems.


http://videolan.org/
dav1d: the world’s fastest AV1 decoder
What is Eve-AV1?

- Eve is an encoder framework from scratch
  - initially written for Eve-VP9
  - and now also used for AV1 encoding
- Eve-VP9 was released in 2016
- Eve-AV1 was released in 2019
- Design goals:
  - intended for VoD use cases
  - optimized for visual quality
  - production-grade rate control
  - commercial license to guarantee long-term development & support

and why?

- libaom is not good enough
  - we’ve seen enough encoder talks today
  - “Visual results: Unfortunately, since the current VP8 encoder optimizes entirely for PSNR, the visual results are less than impressive.” (2010) [1]
  - people have been repeating this ever since
  - “Both VP9 and AV1 are heavily optimized for specific metrics.” (2019) [2]
- we know how to do this, because we’ve done it before [3]

Eve-AV1 encoding performance (best quality)

Experiment:

- encode some standard 1080p clips [1]
  - 1-pass CRF (except libaom/vpx, which requires 2-pass CRF)
  - 4-second max. keyframe interval
  - (nearly) best quality, slowest encode
- measure bitrate and objective quality
  - metric: VMAF

Eve-AV1 encoding performance (best quality)

Experiment:

- encode some standard 1080p clips [1]
  - 1-pass CRF *(except libaom/vpx, which requires 2-pass CRF)*
  - 4-second max. keyframe interval
  - (nearly) best quality, slowest encode
- measure bitrate and objective quality
  - metric: VMAF

Results:

- Eve-Av1 >>
  - libaom / Eve-VP9 >>
  - libvpx / x265 / SVT-AV1 >>
  - rav1e / x264

Eve-AV1 encoding performance (best quality)

Experiment:
- encode some standard 1080p clips [1]
  - 1-pass CRF (except libaom/vpx, which requires 2-pass CRF)
  - 4-second max. keyframe interval
  - (nearly) best quality, slowest encode
- measure bitrate and objective quality
  - metric: VMAF

Results:
- Eve-AV1 >>
  - libaom / Eve-VP9 >>
  - libvpx / x265 / SVT-AV1 >>
  - rav1e / x264 ???

<table>
<thead>
<tr>
<th>1080p clips</th>
<th>% Bitrate reduction</th>
<th>Runtime (sec/frame)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eve-AV1 1.3.5</td>
<td>0.00%</td>
<td>135.57</td>
</tr>
<tr>
<td>libaom a385cc44e</td>
<td>-20.95%</td>
<td>86.13</td>
</tr>
<tr>
<td>rav1e c68d68c</td>
<td>-50.88%</td>
<td>41.01</td>
</tr>
<tr>
<td>SVT-AV1 6fd5646</td>
<td>-33.88%</td>
<td>109.29</td>
</tr>
<tr>
<td>Eve-VP9 1.2.6</td>
<td>-26.32%</td>
<td>9.49</td>
</tr>
<tr>
<td>libvpx f836d8ba8</td>
<td>-36.79%</td>
<td>5.90</td>
</tr>
<tr>
<td>x264 5493be84</td>
<td>-54.12%</td>
<td>1.55</td>
</tr>
<tr>
<td>x265 12522</td>
<td>-38.84%</td>
<td>9.47</td>
</tr>
</tbody>
</table>

Eve-AV1 encoding performance (single-threaded)

Experiment:

- encode some standard 1080p clips [1]
  - 1-pass CRF (except libaom/vpx, which requires 2-pass CRF)
  - 4-second max. keyframe interval
  - encode at all speed presets
- measure bitrate vs. objective quality and average runtime
  - metric: VMAF

Eve-AV1 encoding performance (single-threaded)

Experiment:

- encode some standard 1080p clips [1]
  - 1-pass CRF (except libaom/vpx, which requires 2-pass CRF)
  - 4-second max. keyframe interval
  - encode at all speed presets
- measure bitrate vs. objective quality and average runtime
  - metric: VMAF

Result:

- Eve-AV1 provides best quality
- libaom provides good quality, but does not scale well at higher speeds
- SVT-AV1 provides quality slightly beyond x265/libvpx, and (like x265) scales particularly well at higher-speed presets
- rav1e appears a little less mature, but beats x264 (although at slower speed)

What about multi-threading?

1080p single-threaded

VMAF bitrate reduction (%)

Runtime (seconds/frame)

- Eve-AV1 1.3.5
- libaom a385cc44e
- SVT-AV1 6fd5646
- x265 12522
- x264 5493be84
- rav1e c68d68c
- libvpx f836d8ba8
- Eve-VP9 1.2.6
Eve-AV1 encoding performance (multi-threaded)

Experiment:

- encode some standard 1080p clips [1]
  - 1-pass CRF (except libaom, which requires 2-pass CRF)
  - 4-second max. keyframe interval
  - encode at speed=2 in various multi-threading configurations
- measure bitrate vs. objective quality and average runtime
  - metric: VMAF

Eve-AV1 encoding performance (multi-threaded)

Experiment:

- encode some standard 1080p clips [1]
  - 1-pass CRF (except libaom, which requires 2-pass CRF)
  - 4-second max. keyframe interval
  - encode at speed=2 in various multi-threading configurations
- measure bitrate vs. objective quality and average runtime
  - metric: VMAF

Eve-AV1 encoding performance (multi-threaded)

Experiment:

- encode some standard 1080p clips [1]
  - 1-pass CRF (except libaom, which requires 2-pass CRF)
  - 4-second max. keyframe interval
  - encode at speed=2 in various multi-threading configurations
- measure bitrate vs. objective quality and average runtime
  - metric: VMAF

Eve-AV1 encoding performance (multi-threaded)

Experiment:
- encode some standard 1080p clips [1]
  - 1-pass CRF (except libaom/vpx, which requires 2-pass CRF)
  - 4-second max. keyframe interval
  - encode at speed=2 in various multi-threading configurations
- measure bitrate vs. objective quality and average runtime
  - metric: VMAF

Result:
- Eve-AV1 (still) provides best quality
- libaom (still) provides good quality, but does not scale well at higher speeds
- SVT-AV1 has worse baseline quality than libaom or Eve-AV1, but it's multi-threading scalability is the best
- rav1e scales better (in terms of speed) than libaom (even though the threading mechanism is identical), but suffers from the quality loss being the worst

AV1 encoders - take-home lesson

From an end-user point-of-view:

- Eve-AV1 provides better quality vs. bitrate trade-off than libaom
- Eve-AV1 and SVT-AV1 each provide better quality vs. speed trade-off than libaom
- Eve-AV1, SVT-AV1 and rav1e all provide better multi-threading scalability than libaom

Conclusion: reference software (libaom) is not the theoretical limit, but rather the baseline.

Should & can you use AV1? And how?

- It ultimately depends on your cost function
- If you need x264 --preset=medium or faster, then no
- In most other cases, AV1 encoding is possible and does provide good quality gains at reasonable encoding costs and speeds; the trick is to find the right speed setting
Questions?

Eve-AV1