rav1e 0.3.0

Luca Barbato

Who am I?

- Luca Barbato
  - **rav1e** and **dav1d** contributor among many other open source software.
  - Contacts
    - lu_zero@gentoo.org / lu_zero@videolan.org
    - https://twitter.com/lu_zero_
    - https://github.com/lu-zero

- Who are you?
  - I will talk a little about AV1

- Who are you?
  - I will talk a little about AV1
  - I will talk a lot about rav1e

- Who are you?
  - I will talk a little about AV1
  - I will talk a lot about rav1e
    - I will mention Rust a couple of times, without being too preachy

- Who are you?
  - I will talk a little about AV1
  - I will talk a lot about rav1e
    - I will mention Rust a couple of times, without being too preachy
    - I will talk about Memory and Performance Profiling on Linux

- Who are you?
  - I will talk a little about AV1
  - I will talk a lot about rav1e
    - I will mention Rust a couple of times, without being too preachy
    - I will talk about Memory and Performance Profiling on Linux
    - I will **not** dive too deep in details about encoding features.

- Who are you?
  - I will talk a little about AV1
  - I will talk a lot about rav1e
    - I will mention Rust a couple of times, without being too preachy
    - I will talk about Memory and Performance Profiling on Linux
    - I will **not** dive too deep in details about encoding features.
    - I'll give you the roadmap for the next months

- Who are you?
  - I will talk a little about AV1
  - I will talk a lot about rav1e
    - I will mention Rust a couple of times, without being too preachy
    - I will talk about Memory and Performance Profiling on Linux
    - I will **not** dive too deep in details about encoding features.
    - I'll give you the roadmap for the next months
  - You are welcome to interrupt me and ask questions
    - I will try to answer to them immediately

- Who are you?
  - I will talk a little about AV1
  - I will talk a lot about rav1e
    - I will mention Rust a couple of times, without being too preachy
    - I will talk about Memory and Performance Profiling on Linux
    - I will **not** dive too deep in details about encoding features.
    - I'll give you the roadmap for the next months
  - You are welcome to interrupt me and ask questions
    - I will try to answer to them immediately

I hope you'll have fun (or at least find the whole thing bearable).

rav1e is an AV1 encoder

- rav1e is written in Rust
  - With a fair amount of arch-specific assembly for x86_64 (and aarch64)
- rav1e can be used as a command line tool
  - **cargo install rav1e**
- Or as a normal library using the common open source frameworks
  - GStreamer
  - FFmpeg
  - Anything else that can consume a C or a Rust API.
- rav1e aims to be fast, featureful and safe.
  - Today we'll see how far are we from this.

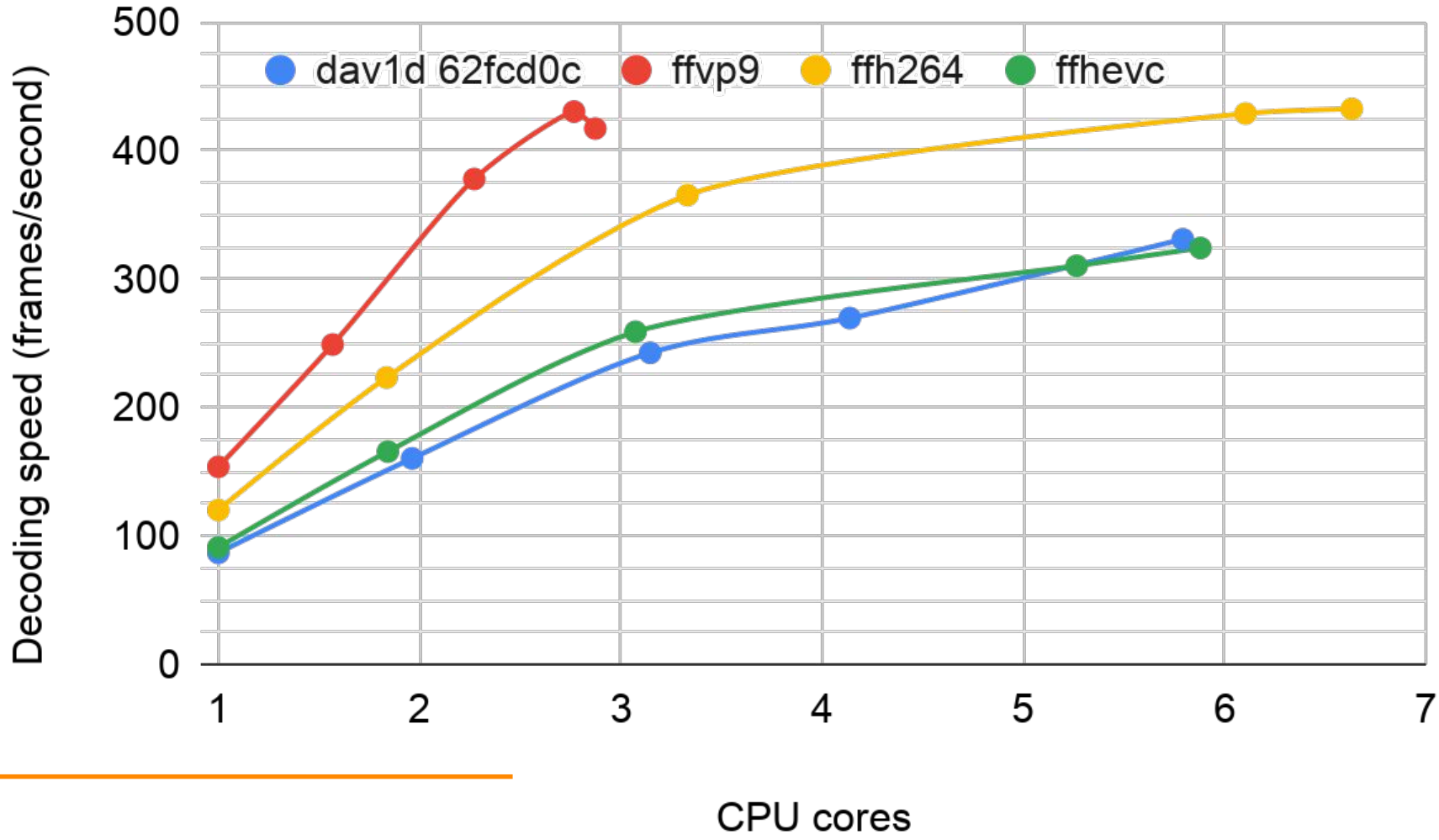
AV1 is a next-generation video codec standard published by AOMedia

- Offers best compression out of all available formats
  - Designed to be used in a broad set of use cases
- Can be used royalty-free by anyone for any purpose
  - Has the support by many of the largest technology companies in the world
  - Alibaba, Amazon, Apple, ARM, Cisco, Facebook, Google, iQiyi, Intel, Microsoft, Netflix, NVIDIA, Samsung, Vimeo, Tencent and more than 30 others are part of the Alliance
  - All of whom have pledged their patents toward AV1
- It is already fairly ubiquitous, at least if we think about **decoding**

AV1 decoding capability is nearly ubiquitous already

- AV1 decoding capability is built-in Android Q
- Microsoft has a media extension for it.
- Linux & macOS
  - VLC supports AV1 (using dav1d)
- Fast AV1 decoding in most browsers
  - Firefox 69
  - Chrome 77
  - Edge Insider (dev)
    - Native AV1 decode with dav1d
  - Safari 12.1.12
    - WASM-compiled dav1d

Software decoding AV1 is already VERY fast



Hardware decoding AV1 coming VERY soon

Many hardware companies involved in AV1 from the beginning

- Broadcom demo BCM7218 set top box decoder at IBC 2019
- MediaTek Dimensity 1000 **mobile SoC** (expect at CES or MWC)
- Amlogic S908X, S905X FHD/4k/8k media SoC (coming in 2020)
- Realtek RTD1319 / RTD1311 SoC demo 4k@60 YouTube playback

Encoding is always harder

- **x264** took about 7 years to become great
- **x265** needed nearly the same time to become a good competitor
  - It managed to leverage a good deal of experience
  - But making effective use of the HEVC features takes more effort.

Encoding Av1 is **not** different

- **libaom** and **SVT-Av1** are built upon years of past experience
  - **libvpx** and the whole **SVT** family of encoders
- Lots of effort is being spent in using well what makes Av1 amazing.

Encoding Av1 since the beginning had been excruciating **slow**

- **libaom** managed to prove you can have amazing quality
  - But at the cost of spending a huge amount of time
  - Reference SW: codebase is historically large and hard to improve
- **SVT-Av1** can be blazing fast
  - As long you have enough hardware
  - And you are satisfied with the current quality & trade-offs made
- **rav1e** aims to be **leaner** and more frugal on resources
  - It is written from scratch in Rust
  - It tries to cover more use cases
    - Focuses on explore different solutions and algorithms
    - Leverage what we learned from daala to improve perceptual quality over PSNR
  - Idea: start with a fast encoder and stay fast

rav1e started with the following goals

- Having a lean implementation
  - The code should stay readable
- Be fast
  - Within a bounded amount of resources required
- Be compact
  - Make so you can run multiple encodings on normal hardware
- Be good for as many purposes as possible
  - Real-time encoding
  - Batch VOD encoding
  - Everything that could be in-between the two above

Every line of code is something you have to read and understand

- **libaom** is about 500k lines of code spread across C, C++ and some assembly
- **rav1e** is about 122k lines of code, 55k rust and the rest mainly assembly.
  - We have more arch-specific assembly than actual arch-independent rust code nowadays.
- The aggregate codebase of **rav1e + dav1d** is still half the size of **libaom**
  - You can read how Av1 encoding and decoding works in about 100k lines of code in total.

In order to be fast you have the following choices

- Use less resources
  - By improving the algorithm in use
  - By avoiding unneeded computation
- Use the same resources but in better ways
  - Leverage the SIMD extensions available
  - Cache locality optimization
- Use more resources
  - Multithread processing

Av1 post-filters are particularly computationally intensive so we focused on improving their efficiency.

- Using Integral Images managed to speed up the Loop Restoration Filter passes.

The Rate Distortion Optimization is another onerous part of the code

- Adding proper early exit conditions in the inner loops avoided a large number of redundant computations without impacting the quality.

More work is ongoing in integrating the filters within the rate distortion optimizer and factorize further commonalities.

A good deal of code is inherently parallel.

- The **rav1e** works together with the **dav1d** in sharing the SIMD assembly optimized routines that are common between encoders and decoders.
- Encoder-specific codepaths are usually optimized using the rust **arch-specific** intrinsics.
- Since the Rust language provides more chances for the compiler to unroll and auto-vectorize a good part of the codebase it is compiled to **SSE** instructions on x86_64 and **NEON** instructions on AArch64.
  - Using **-C target-features=+avx2,+fma** produce an even **faster** binary, with the shortcoming of working only on recent CPUs.

- Writing multithreaded code is usually cumbersome and error prone.
- Rust has two features that combined make exploring parallel processing incredibly easy
  - Fearless concurrency
    - In safe Rust use-after-free and race-conditions are **impossible**.
    - This makes fairly easy write complex multithreaded code
      - Either it works or it would not compile (up to a point)
  - Zero-Cost-Abstraction Iterators
    - Iterators are normally optimized a LOT by the compiler
- There is a crate called rayon that converts normal Iterators in parallel iterators.

This is our main encoding loop

```
let (raw_tiles, tile_states): (Vec<_>, Vec<_>) = ti
  .tile_iter_mut(fs, &mut blocks)
  .zip(cdfs.iter_mut())
  .collect::<Vec<_>>()
  .map(|(mut ctx, cdf)| {
    let raw = encode_tile(fi, &mut ctx.ts, cdf, &mut ctx.tb, inter_cfg);
    (raw, ctx.ts)
  })
  .unzip();
```

This is our main encoding loop, multithreaded

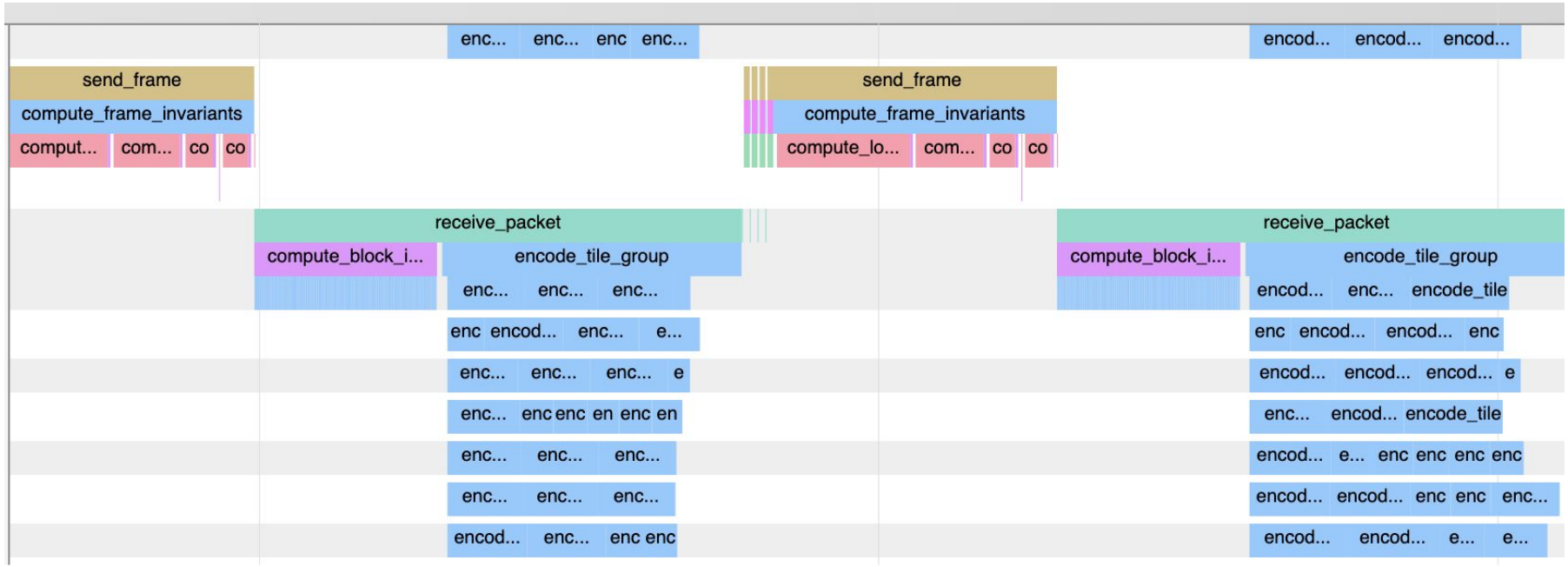
```
let (raw_tiles, tile_states): (Vec<_>, Vec<_>) = ti
  .tile_iter_mut(fs, &mut blocks)
  .zip(cdfs.iter_mut())
  .collect::<Vec<_>>()
  .into_par_iter()
  .map(|(mut ctx, cdf)| {
    let raw = encode_tile(fi, &mut ctx.ts, cdf, &mut ctx.tb, inter_cfg);
    (raw, ctx.ts)
  })
  .unzip();
```

Adding **par_iter()** requires that the Iterator obeys certain constraints

- It is working on **Send** data types
- It is not mutating variables captured by the closure

That may require some initial refactor but it usually pays off well.
Currently we are working on using crossbeam channels to experiment with additional levels of parallelism and provide the users alternative APIs.

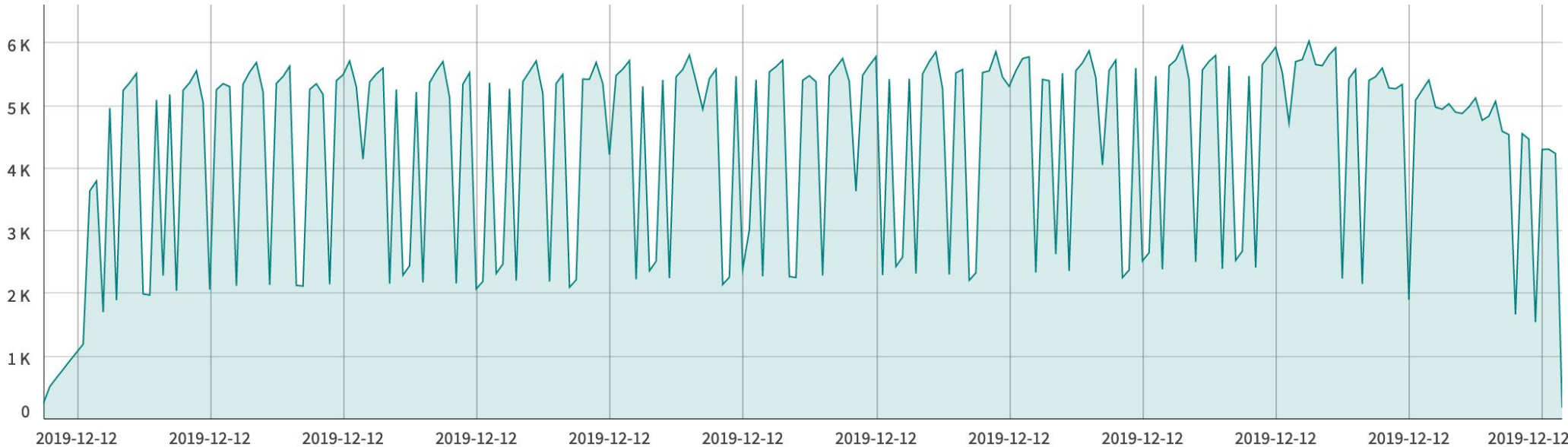
It will be possible to run **send_frame** and **receive_packet** in parallel soon.



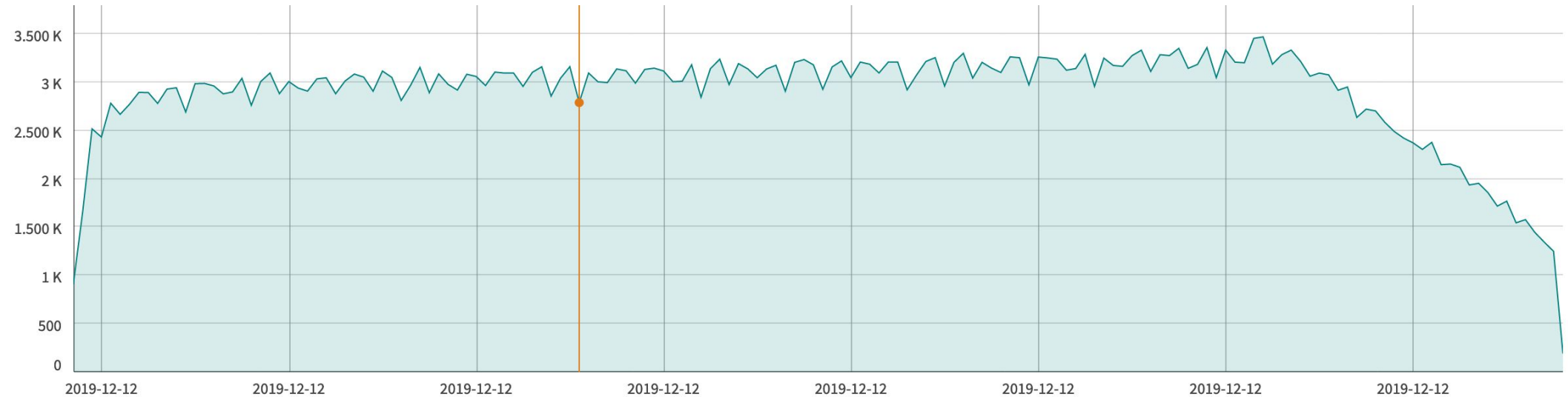
During development we use a number of tools make sure we keep the resource usage under control

- memory-profiler and malt to keep track of the memory allocations.
- perf and cargo-instruments to measure the cpu usage and find hot spots to optimize
- rust_hawktracer to visualize the critical path and decide which are the hotspots that should have priority.
- time as a simple way to quickly keep track of the resident set usage and actual time spent.

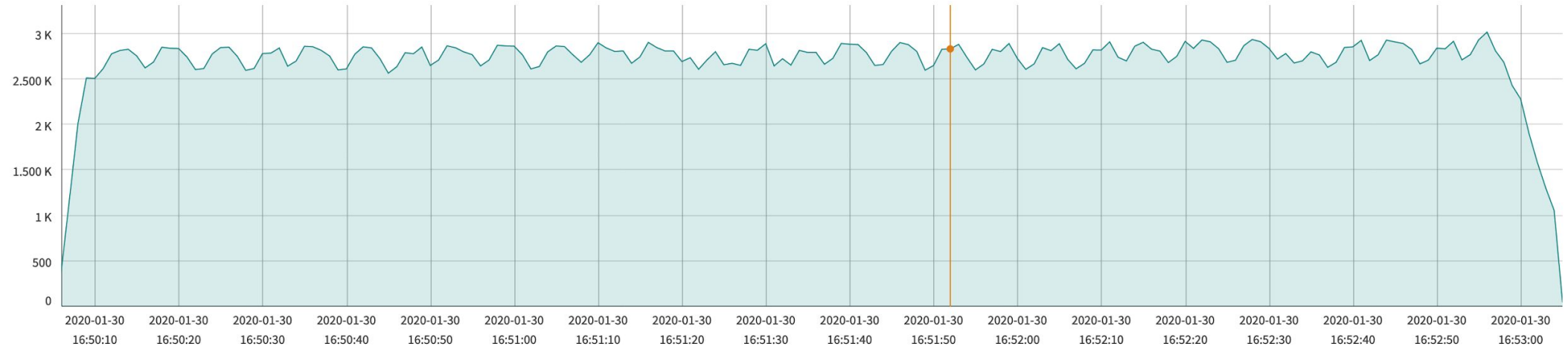
Live allocations for rav1e 0.1.0: **6039 peak**



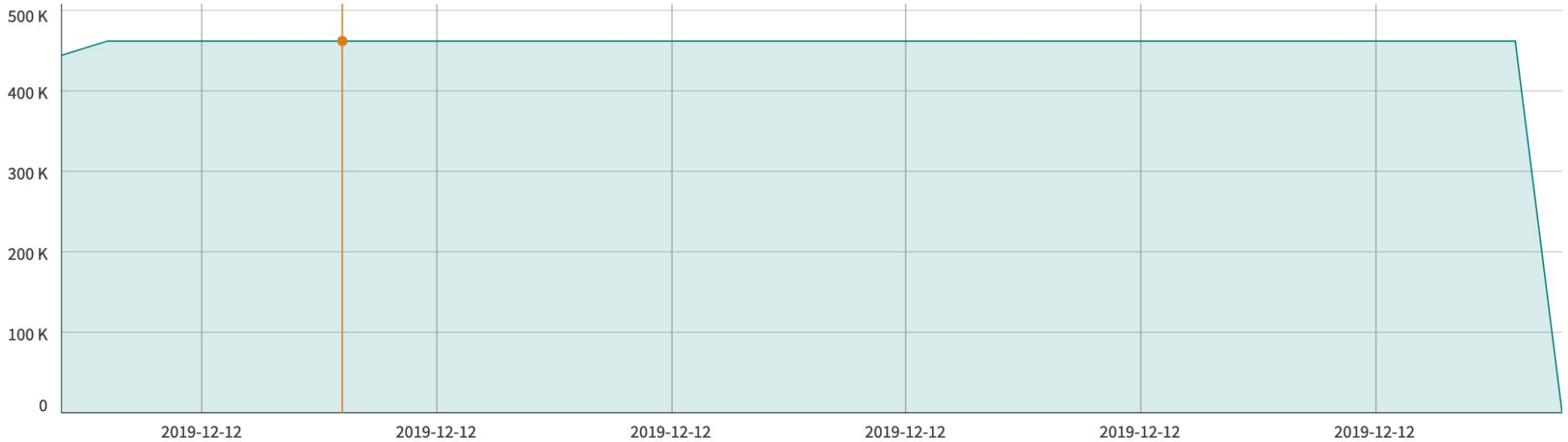
Live allocations for rav1e 0.2.0-p20191201: **3500 peak**



Live allocations for rav1e current (da62d7a46): **3000 peak**

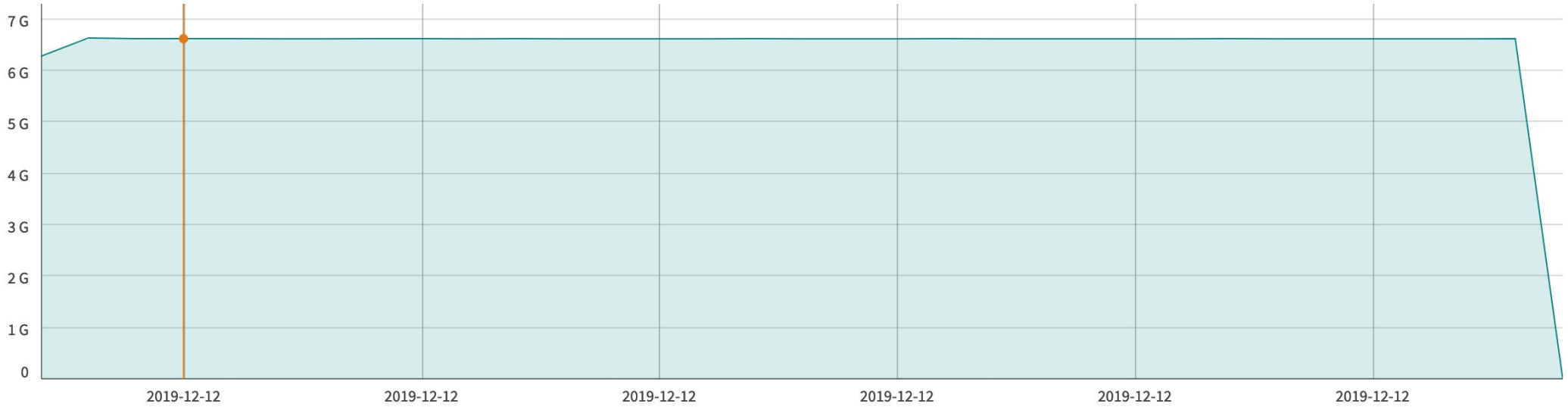


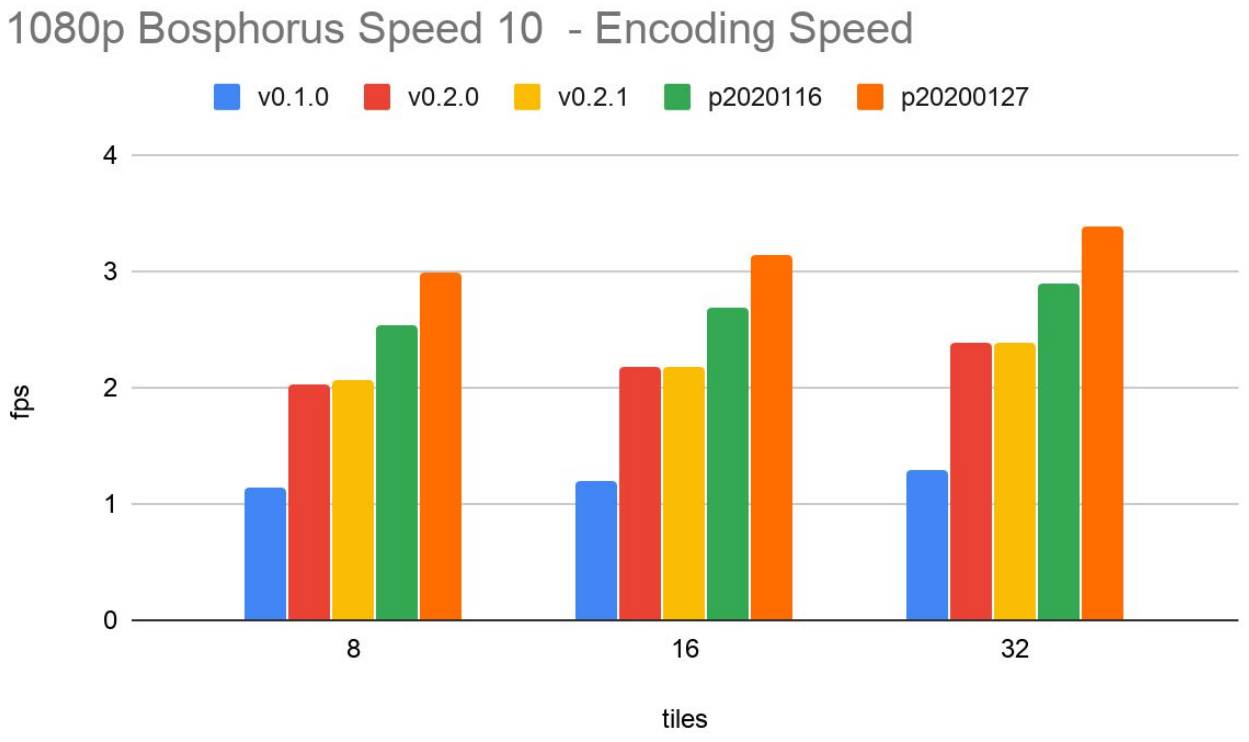
Live allocations for svt-av1 : **462154 peak**



Memory usage for rav1e 0.2.0-p20191201: **1.030 GBytes**

Memory usage for svt-av1: **6.624 GBytes**




1080p Bosphorus Speed 10 - Encoding Speed
v0.1.0
v0.2.0
v0.2.1
p2020116
p20200127
4
3
2
1
0
fps
8
16
32
tiles

Beside work on performance a number of features is going to be tuned in the next releases:

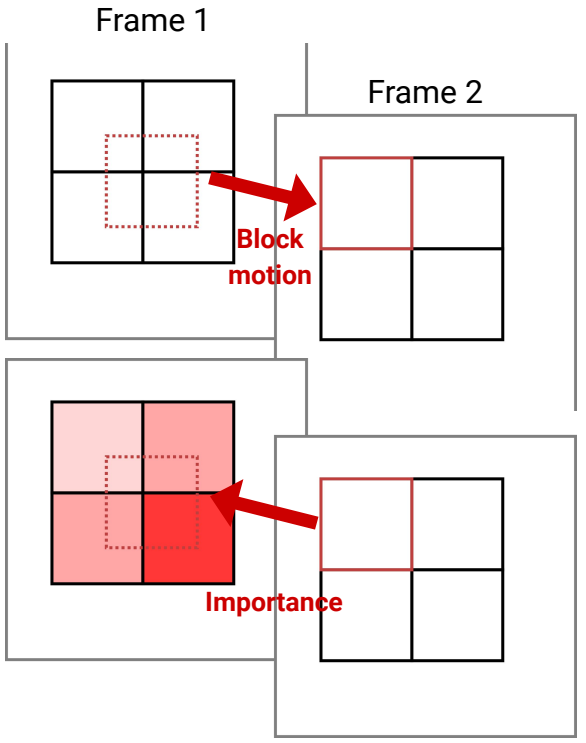
- **RDO biasing**
  - bias the bits allocation so blocks known to stay well predicted in the future frames will have a larger bits budget
  - bias the bits allocation so high activity blocks have their budget decreased
- **chroma-luma balance:** the general rule that the human eye is more sensible to luma differences compared to chroma differences it is not always valid.
- **segmentation:** per-frame quantizer deltas that can be used independently by each block in the frame.

Spend more bits on objects which are well-predicted in future frames

Images from park_joy and Netflix-FoodMarket

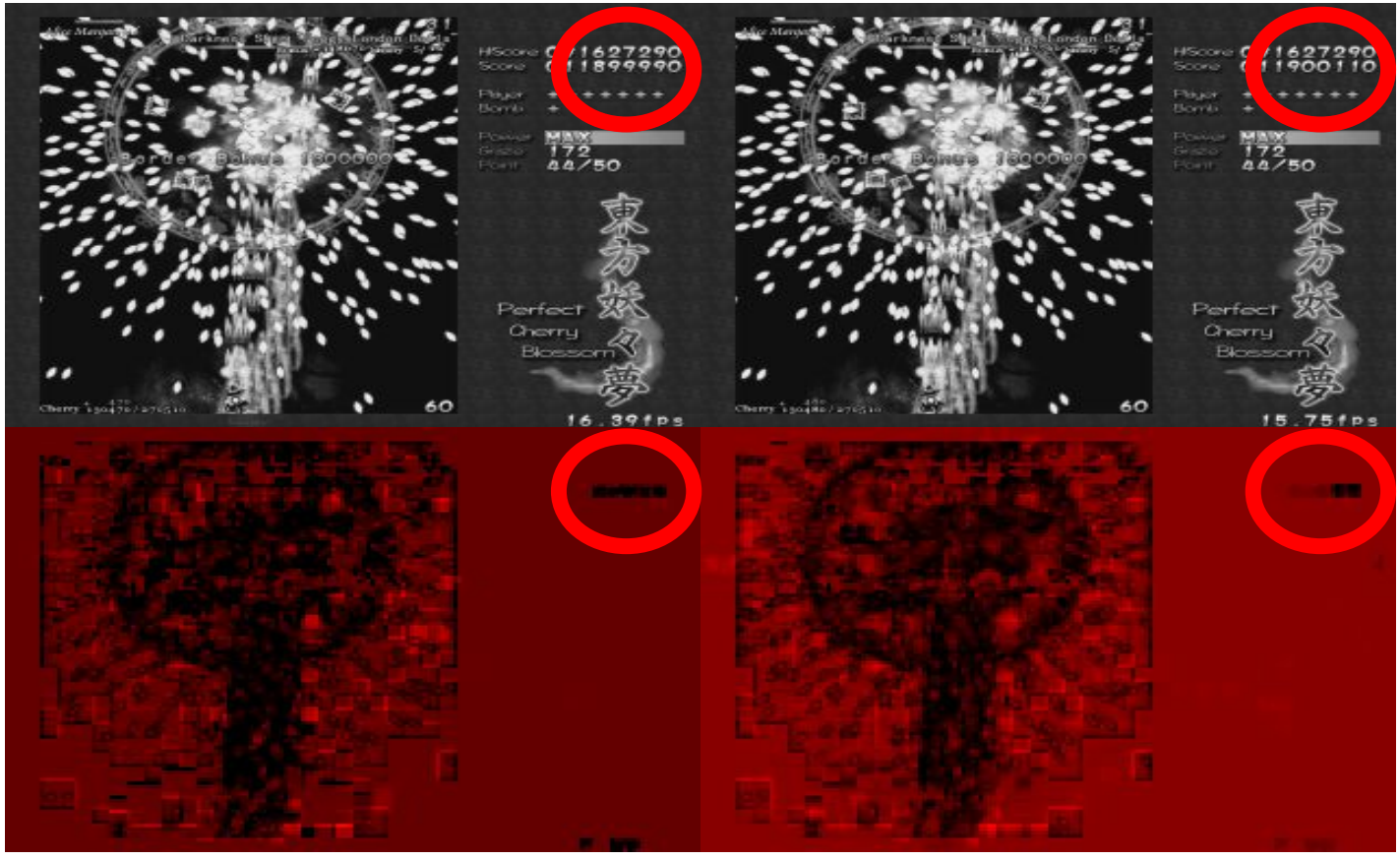
Block Importance

- Future-importance is computed on a 4×4 block basis via propagation using motion vectors
- The propagated value depends on the coding cost of inter- vs. intra-prediction
- Make reference blocks more important when it's cheaper to code the current block using inter-prediction



Block Importance

Block Importance



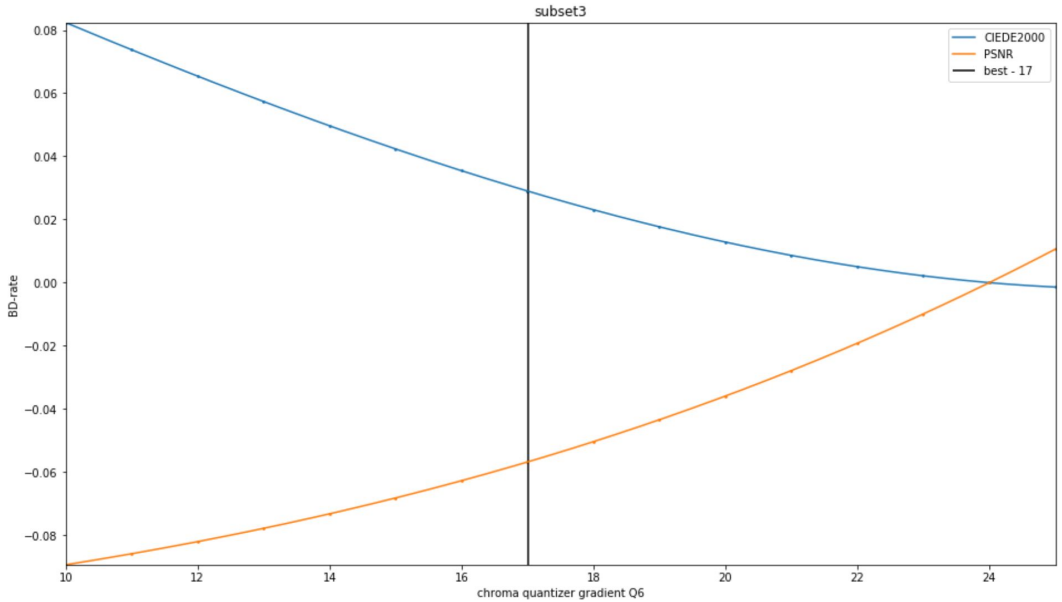
- The computed block importance values are used during the RDO process to bias the distortion:

$$\text{Cost} = \alpha \times \text{Distortion} + \lambda \times \text{Rate}$$

- For areas of low importance, the distortion is decreased ($\alpha < 1$), leading to less bits spent

- For areas of high importance, the distortion is increased ($\alpha > 1$), leading to higher

- AV1 allows **different** quantizers for chroma and luma
- It is possible to perceive the chroma differences more than the luma difference for a specific range of respective quantizations
  - Investing more bits on **chroma** can lead to a perception boost.



- Both **RDO biasing** and **chroma-luma balancing** involve fine **tuning** the quantizers per-blocks
- Av1 allows to store a number of quantizer deltas per-frame and use them independently for each block.
  - Selecting the optimal set of quantizer improves largely the efficiency
  - Discovering the optimal set by brute force increase a lot the overall RDO complexity.
- There is a lot of ongoing work to tweak and improve both the selection of parameters and speed up the overall process.

rav1e aims at being useful for a wider audience.

- Right now it works well for you enough if
  - You need to encode batches of videos at the same time
    - Its small-ish cpu and memory footprint helps fitting multiple processes in the same system.
  - You care about the rate control and you can afford doing a 2-passes encoding
- The ongoing work for the next feature releases will focus on
  - Improving the speed while keeping the resource usage at bay (Started with **0.2.0**)
  - Improve the quality while not decreasing the speed (from **0.3.0**)

rav1e 0.2.0

- Depending on the workload it is up to twice as fast as **0.1.0**
  - This alone does not enable the real-time encoding use case
    - Work on it will happen later, targeting **0.5.0**
  - The work done on supporting **AArch64** makes rav1e much more usable on that architecture.
- It does largely reduce amount of allocations
  - Less memory fragmentation chances

rav1e 0.2.1

- About 30% smaller binaries
  - About 14% faster build time
- Better quality
  - About 0.5-1.5% slower for about 1% better on x86_64

rav1e 0.3.0 - (coming soon)

- Is faster than **0.2.0** at speeds > 5
  - Multi-threaded deblocking filter
  - Additional SIMD code
  - More auto-vectorizable codepaths and bounds check elisions
  - ⅛ less memory allocations
- A full overhaul on the RDO **biasing** features and their selection logic
  - This causes some speed regression on speeds < 5
- Additional encoding tools
  - Fine directional intra prediction
  - Intra edge filter
- Additional API surface and features
  - Switch frame support
  - Still Picture support (AVIF)
- Webassembly target (rayon-less builds)

rav1e 0.4.0 - (expected in late March)

- Additional channel-base API
  - it will provide a simpler usage mode
  - better threads/core usage.
- Better rate-control
  - The rate-control will support two-pass chunked encoding
  - Fast first-pass mode
- The rate-control API will be expanded
  - Aggregate the per-chunk rate-control information to produce a per-sequence summary.

Av1 is amazing and you have fewer and fewer reasons not to use it.

- There are good software decoders for all the widespread platforms
- The hardware support is coming along nicely
- You might not have all the hardware other encoders require or your use-case is not covered yet, but **rav1e** is going to address both soon.

Thank You