AV1 Decoder Performance Optimization and Device Test

23-AUG-2022

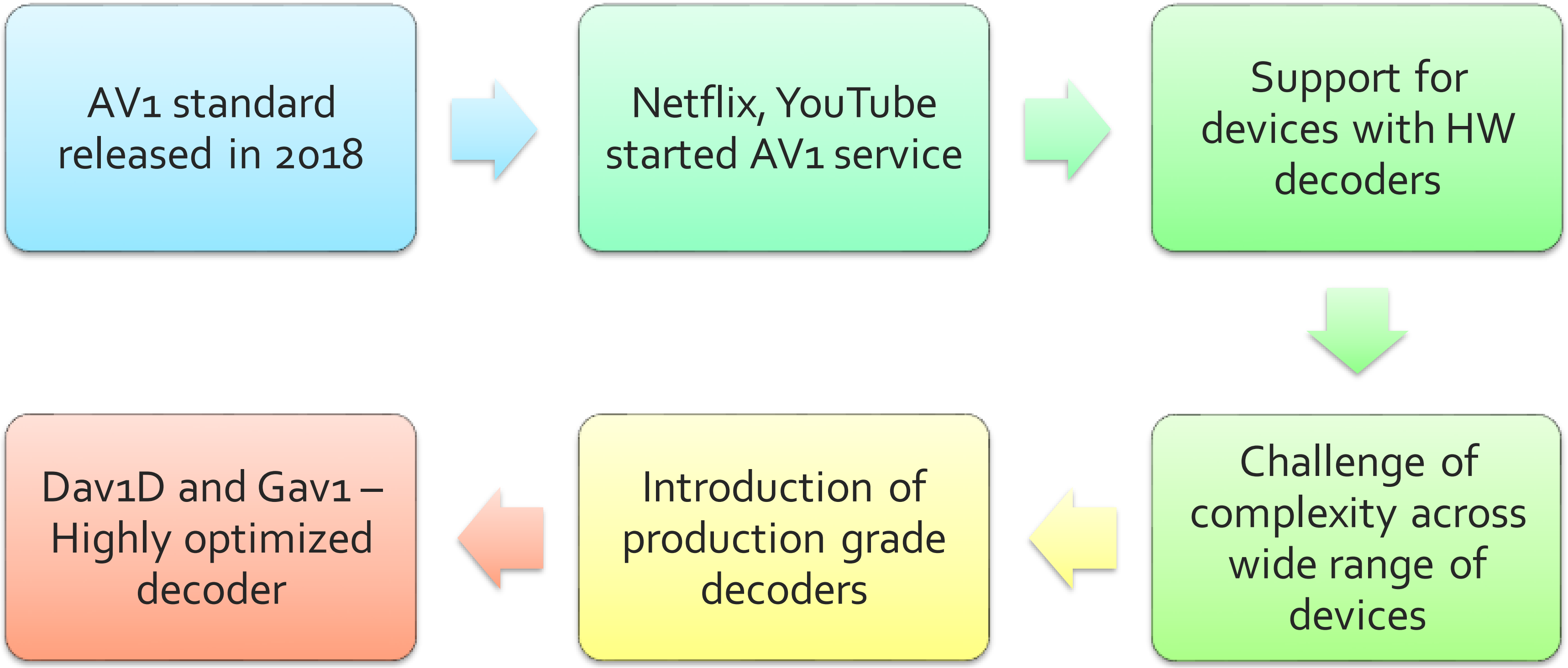
Presenter:
Ronald Bultje – Two Orioles
Kaustubh Patankar - ITTIAM

Outline

- Introduction
- AV1 Decoder Optimization – dAV1d
- Benchmarking and Analysis
- Conclusion

Introduction

AV1 – Quick recap



AV1 Decoder Optimization dAV1d

Magic in dav1d's Inverse Transforms

INVERSE TRANSFORMS:

GAV1 IMPLEMENTATION

Gav1 has 2 versions of each inverse 1D transform with integrated load, transpose, store and residual_add.

OVERHEAD

- Load/store overhead in 1D transforms
- ~~Load/store overhead in transpose~~
- Load/store overhead in `RowShift()`
- Load overhead in `residual_add()`
- Some overhead can be reduced by inlining, but that increases binary size

```
INLINE Dct8_NEON(Residual *buffer, step, bool transpose);

Dct8TransformLoopRow_neon(Array2dView<Pixel> *frame,
                          Residual *buffer, tx_size, tx_type)
{
    for (int y = 0; y < col_1d_size; y += vector_size)
        Dct8_NEON(buffer, vector_size, /*transpose=*/true);
    RowShift(buffer);
}

Dct8TransformLoopColumn_neon(Array2dView<Pixel> *frame,
                             Residual *buffer, tx_size, tx_type)
{
    for (int x = 0; x < row_1d_size; x += vector_size)
        Dct8_NEON(coefficients[x], row_1d_size,
                  /*transpose=*/false);

    for (y = 0; y < row_1d_size; y++)
        residual_add(&pixels[y][x],
                     coefficients[x + y * row_1d_size]);
}
```

Magic in dav1d's Inverse Transforms

INVERSE TRANSFORMS:

GAV1 IMPLEMENTATION

Gav1 has 2 versions of each inverse 1D transform with integrated load, transpose, store and residual_add.

OVERHEAD

- Load/store overhead in 1D transforms
- ~~Load/store overhead in transpose~~
- Load/store overhead in `RowShift()`
- Load overhead in `residual_add()`
- Some overhead can be reduced by inlining, but that increases binary size

```
INLINE Dct8_NEON(Residual *buffer, step, bool transpose)
{
     [load]
     if (transpose) {
          [transpose]
     }
     [ actual 1d inverse transform ]
     if (transpose) {
          [transpose]
     }
     [store]
}

Dct8TransformLoopRow_neon(Array2dView<Pixel> *frame,
                          Residual *buffer, tx_size, tx_type);

Dct8TransformLoopColumn_neon(Array2dView<Pixel> *frame,
                             Residual *buffer, tx_size, tx_type);
```

Magic in dav1d's Inverse Transforms

INVERSE TRANSFORMS:

DAV1D IMPLEMENTATION

Dav1d uses transform implementations with a custom calling ABI which retains the vector registers as part of the function call interface. This is only possible because it's hand-written assembly (not intrinsics).

OVERHEAD

- ~~Load/store overhead in 1D transforms~~
- ~~Load/store overhead in transpose~~
- ~~Load overhead in `residual_add()`~~
- ~~Load/store overhead in downshifts~~
- Custom calling ABI eliminates overhead without increasing binary size
- dav1d also eliminates the first transpose by integrating it with the scan table

```
inv_txfm_add_dct_dct_8x8_8bpc_neon:
        mov             x15, x30
        [ dc-only checks ]
        adr             x5,  inv_dct_8h_x8_neon
        adr             x4,  inv_dct_8h_x8_neon
        b               inv_txfm_add_8x8_neon

inv_txfm_add_8x8_neon:
        [ prologue ]
        [ load coefficients ]
        blr             x4
        [ 8x right-shift-by-one ]
        [ transpose ]
        blr             x5
        [ write out ]
        [ epilogue ]
        br              x15

inv_dct_8h_x8_neon:
        [ actual 1d transform ]
        ret
```

Magic in dav1d's Inverse Transforms

INVERSE TRANSFORMS:

DAV1D IMPLEMENTATION

Dav1d uses transform implementations with a custom calling ABI which retains the vector registers as part of the function call interface. This is only possible because it's hand-written assembly (not intrinsics).

OVERHEAD

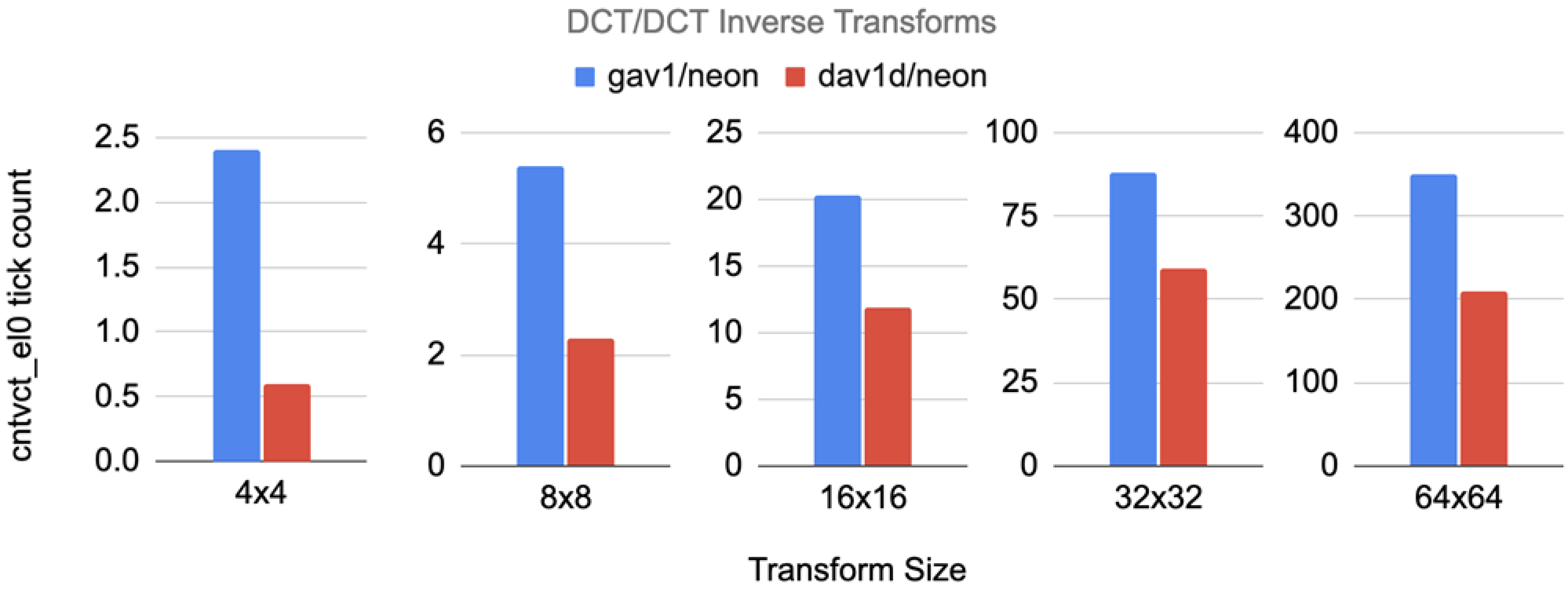
- Special identity implementation merges first 1D identity (which upshifts by 1) and downshifts (which downshift by 1)

```
inv_txfm_add_identity_dct_8x8_8bpc_neon:
        mov             x15, x30
        adr             x5,  inv_dct_8h_x8_neon
        b               inv_txfm_identity_add_8x8_neon

inv_txfm_identity_add_8x8_neon:
        [ prologue ]
        [ load coefficients ]
        [ transpose ]
        blr             x5
        [ write out ]
        [ epilogue ]
        br              x15

inv_dct_8h_x8_neon:
        [ actual 1d transform ]
        ret
```

Magic in dav1d's Inverse Transforms



other coding tool implementations

CODING TOOLS:

DAV1D vs. GAV1

- Loop Restoration: gav1's arm implementation of the selfguided filter is superior (because better cache efficiency)
- CDEF: both implementations have special-case implementations for single filters (when the primary or secondary CDEF filter strength is zero), but gav1's secondary-only filter implementation is (unexpectedly) slower than the dual-filter implementation
- Motion Vector Referencing SIMD: both decoders implement SIMD for different parts of the code
- Arithmetic coding: dav1d has hand-written assembly (including SIMD for CDF updates), which gav1 does not have
- Film Grain: both decoders have SIMD, but the Neon film grain is disabled in gav1
- Directional Intra Prediction (z1-3): gav1 has implementations for arm & x86, whereas dav1d only has SIMD implementations for x86

Conclusion: both decoders have some unique implementation ideas, and both can be improved further.

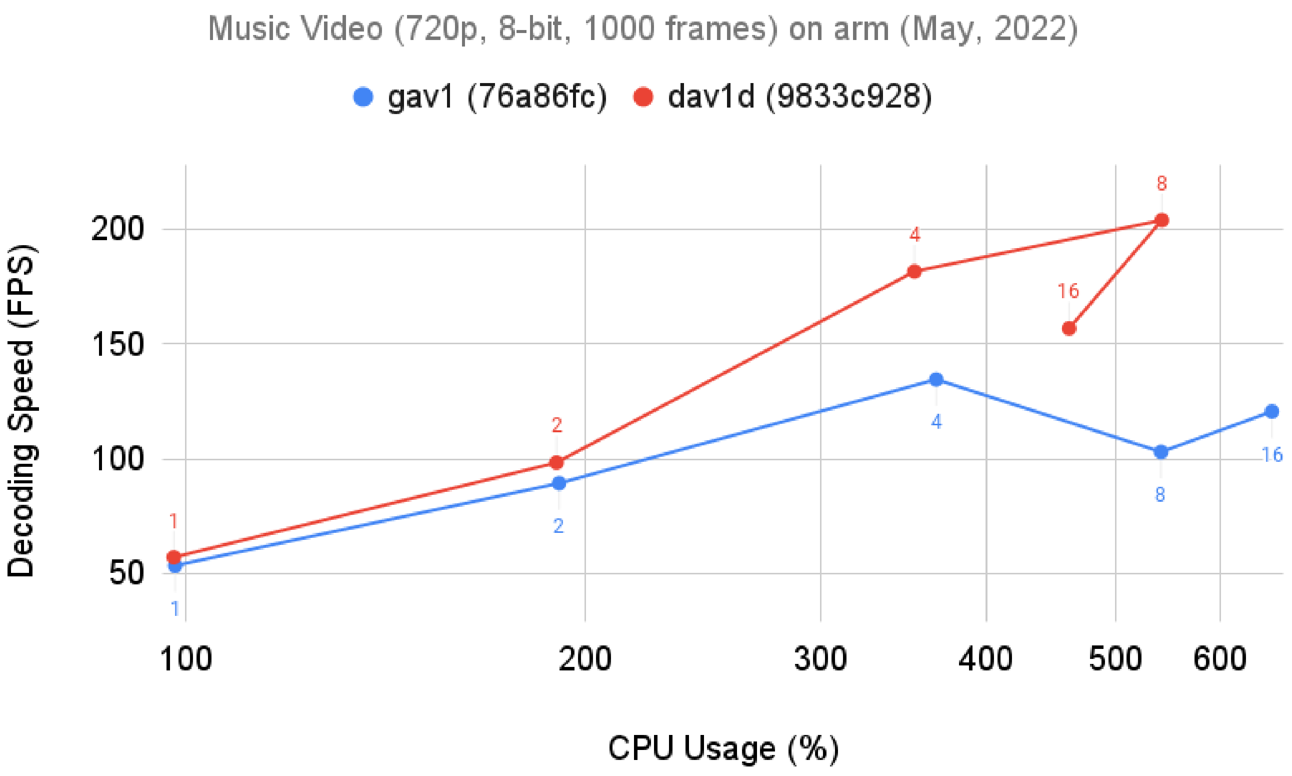
Multi threading

MULTI-THREADING:

dav1d uses a task-queue design, where each component in the decoding loop runs as a generic task with a simple dependency resolution mechanism:

- entropy tile-sbrow reading
- tile-sbrow reconstruction
- deblock, cdef & loop restoration sbrow in-loop post-filtering
- film grain sbrow out-loop post-filtering
- multiple frames in parallel

Together, this keeps low number of worker threads sufficiently busy without relying on particular bitstream features. Also, it does not require task-specific worker threads, and is therefore resource-friendly.



Benchmarking and Analysis

Purpose of AV1 Benchmarking

ON ANDROID ECOSYSTEM

- AV1 is the first generation of royalty-free video coding format developed by AOM
  - Helps to improve the quality of video delivery
- There are many service providers in the market already using AV1 for video streaming
- AV1 real-time playback capability on clients is critical
  - Already there is HW decoding support for AV1 on smart TV and set-top boxes
- Adoption of AV1 into android ecosystem is important and support for HW decoding is still not widely available and its penetration may take a couple of more years.
- Wide range of android devices with different chipsets are existing in the market and their processing capability varies
- This study helps to know the AV1 SW decoding capability on a sample of devices across the range and enables the market for wider adoption of AV1 delivery

Benchmarking Methodology

SETUP AND CONFIGURATIONS

Decoders

- Dav1d
- Gav1

APP

- Integrated to VLC Player on Android
- aarch32 and aarch64

Metrics

- Real time performance
- CPU load

Benchmarking Methodology

MEASUREMENT METHODOLOGY

Performance

- Using VLC Benchmarking APP
- Reports number of frames dropped
- Frame drop > 2% is non real time
- ST and MT

CPU Load

- Using "adb top" command
- Validation of SW decoders
- Sampling at 1 sec interval

Benchmarking Methodology

DEVICE SELECTION

Category	Cortex Version	ARM Architecture	Clock Frequency
High-end	4xA7x + 4xA5x	v8	Any
	2xA7x + 6xA55	v8	Any
Mid-range	8xA53	v8	>= 1.8GHz
	2xA7x + 6xA53	v8	Any
Low-end	8xA53	v8	< 1.8 GHz
	Any	v7	Any

- Total devices – Android – 61
- Covering 15 global brands
- Release year from 2018 to 2020
- Classification as High, Medium and Low complexity
    - Based on CPU capability

Benchmarking Methodology

TEST CLIP, CONFIGURATIONS,

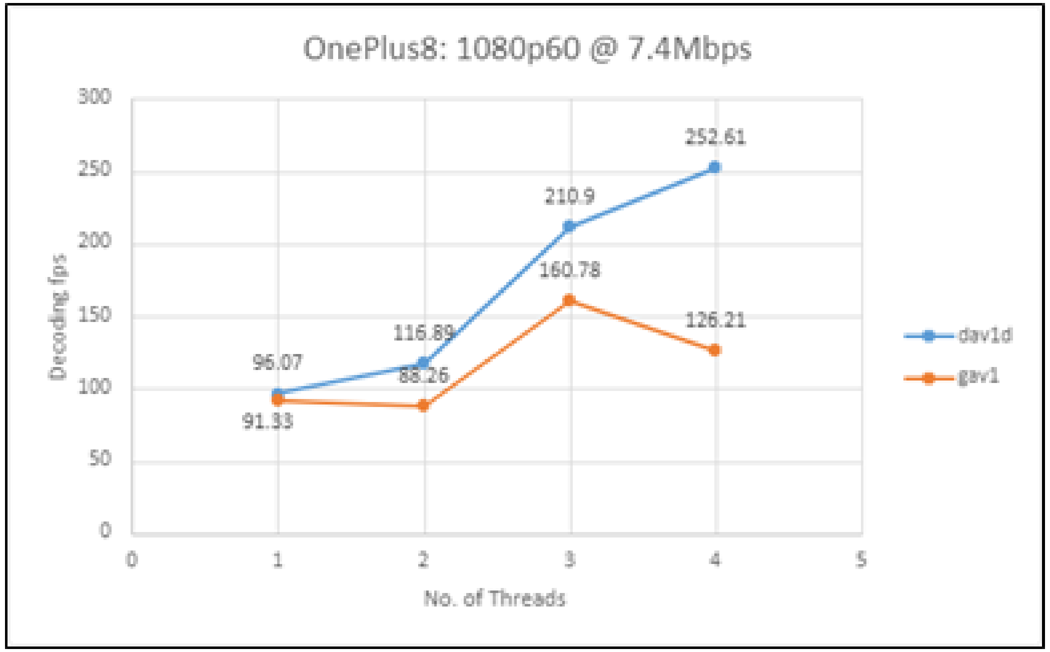
- TOOLS ON – [T-ON]: all AV1 supported coding tools are used while encoding.

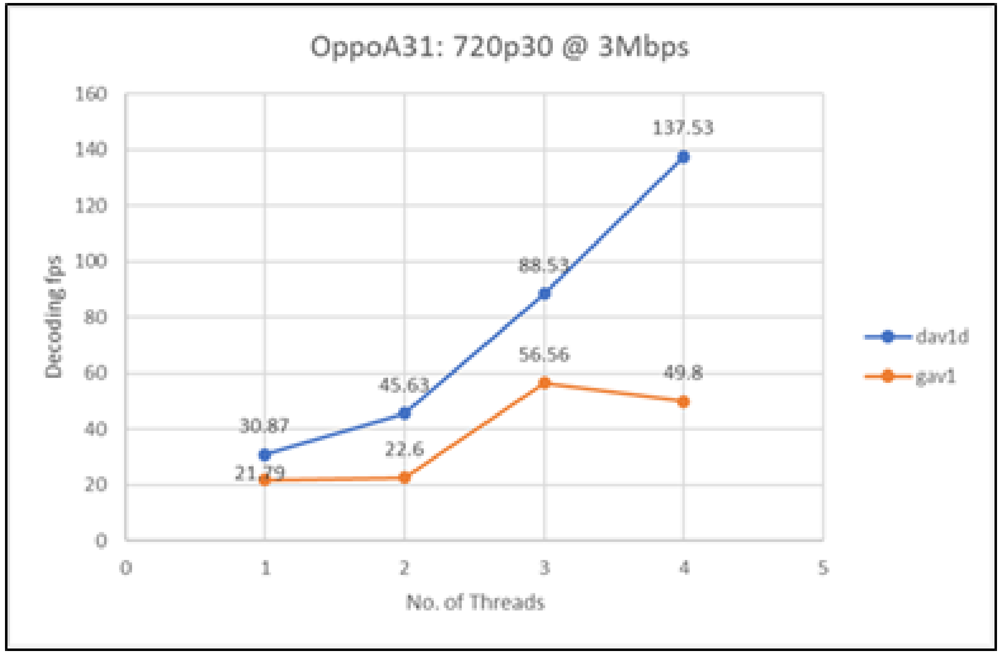
- TOOLS OFF[8] – [T-OFF]: AV1 coding tools with a poor trade-off between decoding complexity and compression efficiency are disabled intelligently (fast-decode option in SVT-AV1) as described below.
  - Enable Motion Field MV prediction only for a well-predictable motion field
  - Enabled tools
    - OBMC
    - Warp
    - Partition depth
    - Reduced complexity approximations on Self-Guided filter, CDEF
  - Intelligent application of CDEF and Deblocking at super block level

- Test Clip – AOM – CTC
  - Original resolution of 1920x1080
  - Down scaled to 5 resolutions – 270p, 360p, 540p, 720p, 1080p
  - FPS – 30 and 60
  - QP Points – 9

QP	Average Bitrate					
	Tools-On			Tools-Off		
	540p	720p	1080p	540p	720p	1080p
32	2.08	2.99	5.12	2.06	3.02	5.14
43	1.05	1.46	2.47	1.04	1.46	2.46
55	0.45	0.61	1.06	0.45	0.61	1.04
63	0.12	0.17	0.3	0.12	0.16	0.3

Standalone Decoder Performance



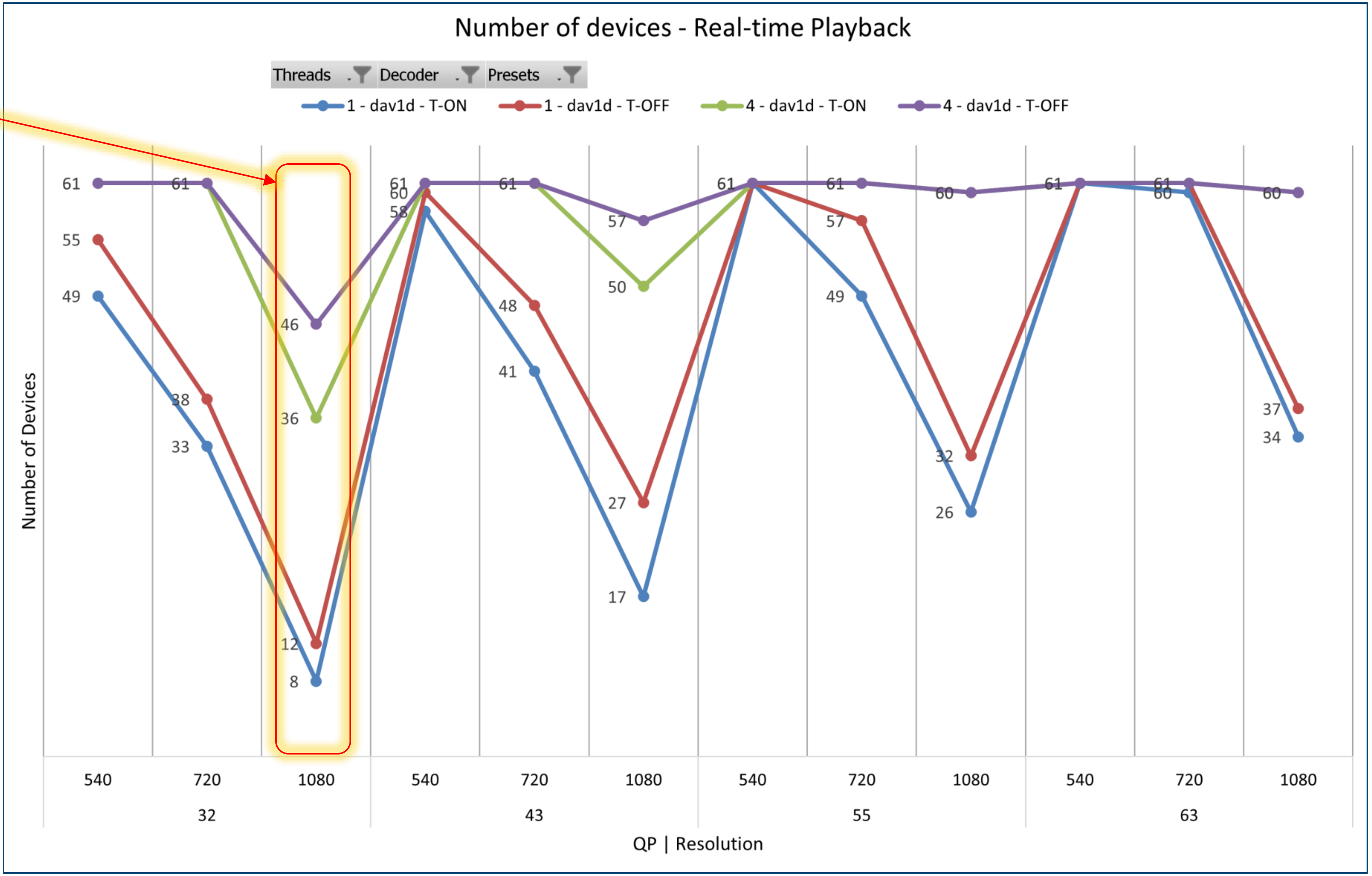


Tools ON vs Tools OFF

DAV1D WITH SINGLE AND 4 THREADS

For **1080p @ QP 32**, Number of devices capable of AV1 real-time playback for **dav1d**

1. **8** devices for **single** thread **Tools- ON**
2. **12** devices for **single** thread **Tools-OFF**
3. **36** devices for **4** threads **Tools-ON**
4. **46** devices for **4** threads **Tools-OFF**

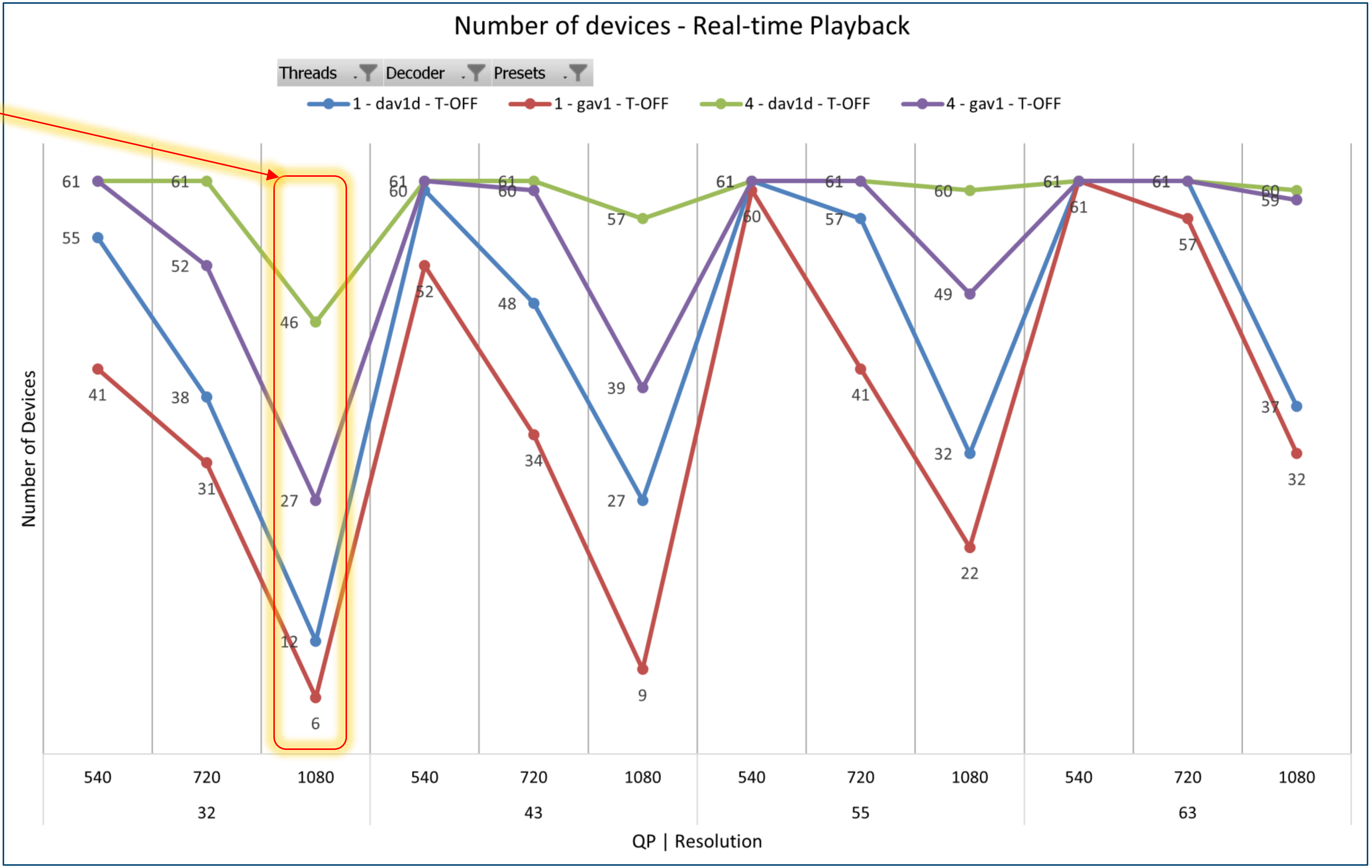


Performance of Dav1d and Gav1

SINGLE AND 4 THREADS – TOOLS OFF CONFIGURATION

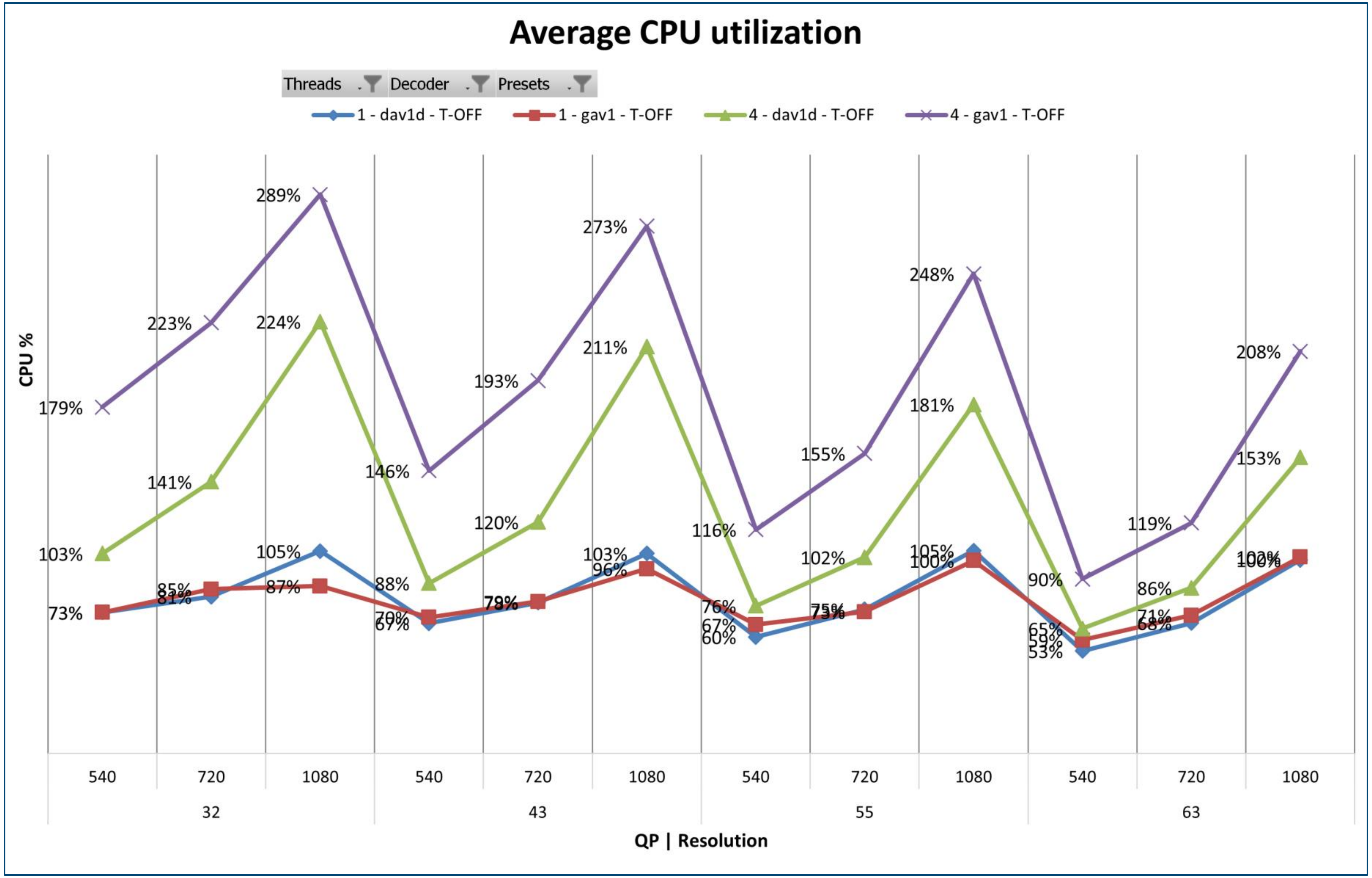
For **1080p @ QP 32**, Number of devices capable of AV1 real-time playback for **Tools OFF**

1. **6** devices for **single** thread **gav1**
2. **12** devices for **single** thread **Dav1d**
3. **27** devices for **4** threads **gav1**
4. **46** devices for **4** threads **Dav1d**



Average CPU Utilization

DAV1D AND GAV1 – SINGLE AND FOUR THREADS



Conclusion

Summary

HW AV1 decoders penetration into market may take a couple of years

Well optimized SW decoders are necessary for AV1 adoption into android ecosystem to improve quality of video delivery

With Tools Off Configuration, 10-30% of decoder runtime reduction can be achieved than default (Tools On Configuration) with very small quality impact.

It is possible to achieve 720p30 real time playback on most of the Android devices

For mid-range to high end Android devices, it is possible to achieve 1080p30 real time playback

AV1 is ready to be deployed widely on Android devices with software decoders!