



THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

# Optimizing zlib on Arm: The power of NEON

Adenilson Cavalcanti  
ARM - San Jose (California)

@adenilsonc



# Why zlib?

## zlib

Used everywhere (libpng, Skia, freetype, **cronet**, Firefox, Chrome, Linux kernel, android, iOS, JDK, git, etc).

Old code base released in 1995.

Written in K&R C style.

## Context

Lacks any optimizations for Arm CPUs.

## Problem statement

Identify potential optimization candidates and verify positive effects in Chromium.

# Previous SIMD art

- Cloudflare
- Intel
- zlib-ng

# Before deepening the fork...

- Performed some benchmarking.
- Contacted each project.
- Mixed results (1 project never replied back).
- Also contacted canonical zlib.

# Before forking...

- Performed some benchmarking.
- Contacted each project.
- Mixed results (1 project never replied back).

None focused on **decompression**\* or had Arm specific optimizations.

\*Important for a Web Browser.

# Meet Mr. Parrot

PNGs rely on zlib

- Transparent.
- Pre-filters.
- High-res.



Source: <https://upload.wikimedia.org/wikipedia/commons/3/3f/ZebraHighRes.png>



# Parrots are not created equal



Original: 2.7MB



Palette: 0.8MB



Zopfli: 2.6MB

# Perf to the rescue

== Image has pre-compression filters (2.7MB) ==

Lib	Command	SharedObj	method	CPU (%)
zlib	TileWorker	liblink	inflate_fast .....	1.96
zlib	TileWorker	libblnk	adler32 .....	0.88
blink	TileWorker	liblink	ImageFrame::setRGBAPremultiply ..	0.45
blink	TileWorker	liblink	png_read_filter_row_up.....	0.03*

== Image was optimized using zopfli (2.6MB) ==

Lib	Command	SharedObj	method	CPU (%)
zlib	TileWorker	liblink	inflate_fast .....	3.06
zlib	TileWorker	libblnk	adler32 .....	1.36
blink	TileWorker	liblink	ImageFrame::setRGBAPremultiply ..	0.70
blink	TileWorker	liblink	png_read_filter_row_up.....	0.48*

== Image has no pre-compression filters (0.9MB) ==

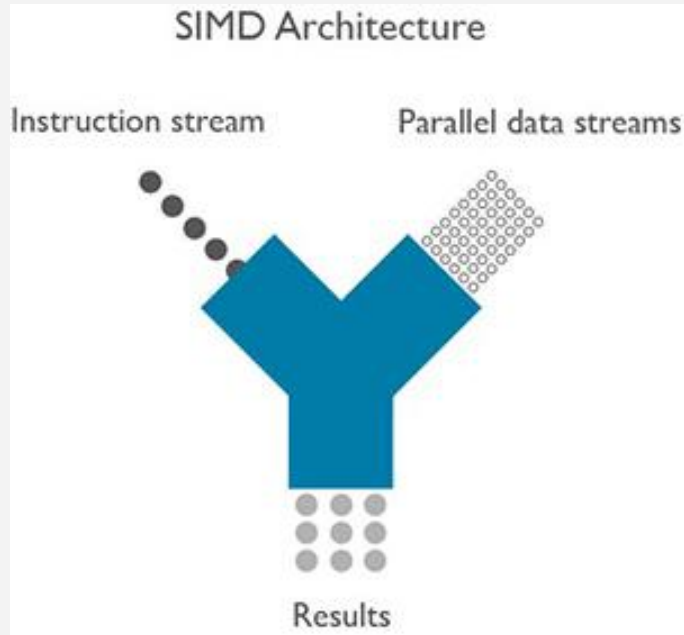
Lib	Command	SharedObj	method	CPU (%)
libpng	TileWorker	liblink	cr_png_do_expand_palette .....	0.88
zlib	TileWorker	liblink	inflate_fast .....	0.62
blink	TileWorker	liblink	ImageFrame::setRGBAPremultiply ..	0.49
zlib	TileWorker	libblnk	adler32 .....	0.31



# NEON: Advanced SIMD

(Single Instruction Multiple Data)

# NEON



- Optional on ARMv7.
- Mandatory on ARMv8.

# Registers

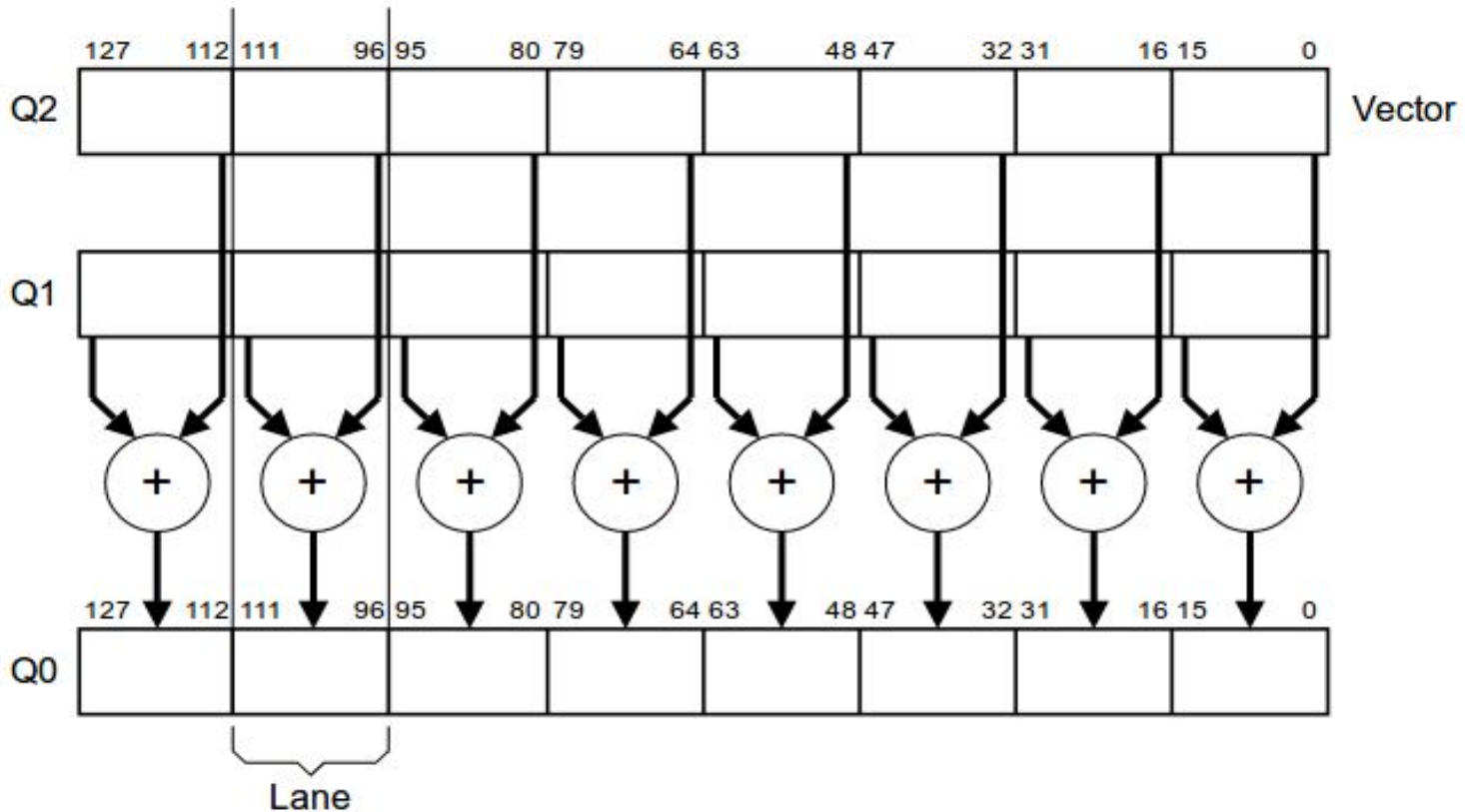
## Armv7

- 16 registers@128 bits: Q0 - Q15.
- 32 registers@64bits: D0 - D31.
- Varied set of instructions: load, store, add, mul, etc.

## Armv8

- 32 registers@128 bits: Q0 - Q31.
- 32 registers@64bits: D0 - D31.
- 32 registers@32bits: S0 - S31.
- 32 registers@8bits: H0 - H31.
- Varied set of instructions: load, store, add, mul, etc.

# An example: VADD.I16 Q0, Q1, Q2



# Entropy & Compression



# Entertaining definition



Silicon Valley, season 1.

# Formal definition

## Shannon Entropy

$$H = - \sum_i p_i \log_b p_i$$

Where:

$p_i$ : probability of character  $i$  appearing in the stream of characters.

# Practical explanation

a) HTML

```
Mon, 08 Nov 2004 17:22:09 GMT\r\nContent-Type: text/html\r\nConnection: close\r\n\r\n-Powered-By: PHP/4.3.8\r\nServer: Apache/1.3.31 (Unix) mod_gzip/1.3.19.1a PHP/4.3.8\r\n\r\nDate: Mon, 08 Nov 2004 17:19:07 GMT\r\n\r\n<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">\r\n<html>\r\n<head>\r\n<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">\r\n<BASE target="_top">\r\n<TITLE>Micro Achat : Ordinateurs, PDA - Toute l'informatique avec 01Informatique, L'Ordinateur Individuel, Micro Hebdo, D\351cision Informatique et 01R\351seaux</TITLE>\r\n<META NAME="Description" CONTENT="Micro Achat : Ordinateurs, PDA - Toute l'informatique avec 01Informatique, L'Ordinateur Individuel, Micro Hebdo, D\351cision Informatique et 01R\351seaux">\r\n<LINK REL="STYLESHEET" TYPE="text/css" HREF="http://www.01net.com/styles/01net.css">\r\n<LINK REL="STYLESHEET" TYPE="text/css" HREF="http://www.01net.com/styles/tableau_autre.css">\r\n<STYLE type="text/css">\r\nA{TEXT-DECORATION: none;color:#000000;}\r\nA:visited{TEXT-DECORATION: none;color:#000000;}\r\n</STYLE>\r\n<SCRIPT LANGUAGE="JavaScript1.1">\r\nvar sameAsBigDay = new Date();\r\nvar ord = (sameAsBigDay.getTime());\r\nvar pubsMotsCles = "\\";\r\nvar exc;\r\nvar lienhautdepage = "\\";\r\nvar etatmessage = false;\r\nvar enchainement = false;\r\nvar sas_tmstp=Math.round(Math.random()*10000000000);\r\nfunction SmartAdServer(sas_pageid,sas_formatid,sas_master,sas_target) {\r\nif ((sas_master!="M")&&(sas_master!="S")) {sas_master="S";}\r\n\r\ndocument.write("<SCR"+"<IPT SRC="+"http://www.smartadserver.com/call/pubj/"+ sas_pageid + "/" + sas_formatid + "/" +sas_master + "/" + sas_tmstp + "/" + escape(sas_target) + "'>"</SCR"+"<IPT>");\r\n\r\n}</SCRIPT>\r\n<SCRIPT LANGUAGE="JAVASCRIPT" SRC="http://www.01net.com/js/exc.js"></SCRIPT>\r\n<script language="javascript" src="http://telecharger.01net.com/shopping/js/headservicesV4.js"></script>\r\n<noscript>\r\n<a href="http://cinema.01net.com">T\351l\351charger des bandes-annonces, des teasers, des extraits et des making of de vos films c
```

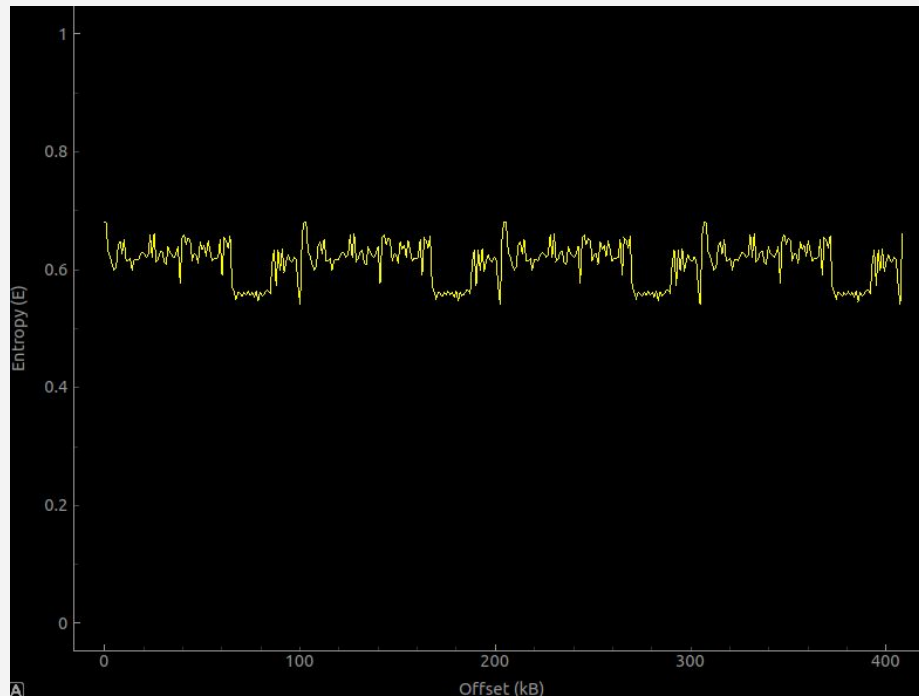
b) JPEG



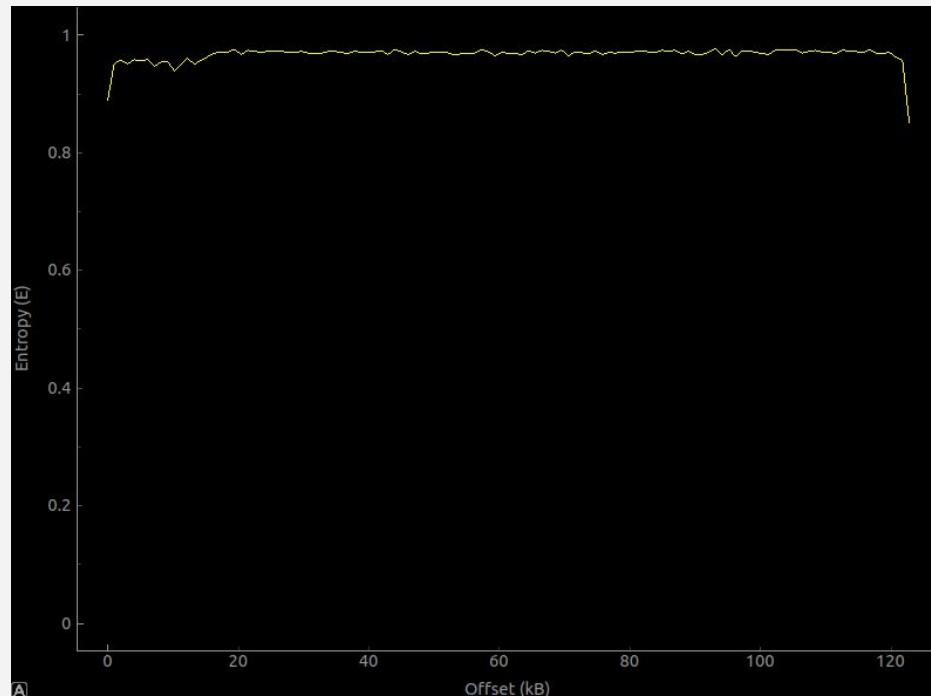
# Practical visualization

./binwalk -E file

a) HTML: 0.68



b) JPEG: 0.95



# Decompression optimizations



# Adler-32 checksum

$$A = 1 + D_1 + D_2 + \dots + D_n \pmod{65521}$$

$$B = (1 + D_1) + (1 + D_1 + D_2) + \dots + (1 + D_1 + D_2 + \dots + D_n) \pmod{65521}$$

$$= n \times D_1 + (n-1) \times D_2 + (n-2) \times D_3 + \dots + D_n + n \pmod{65521}$$

$$\text{Adler-32}(D) = B \times 65536 + A$$

# Adler-32 simplistic implementation

```
// From: https://en.wikipedia.org/wiki/Adler-32
const int MOD_ADLER = 65521;
unsigned long naive_adler32(unsigned char *data,
                           unsigned long len)
{
    uint32_t a = 1, b = 0;
    unsigned long index;

    for (index = 0; index < len; ++index) {
        a = (a + data[index]) % MOD_ADLER;
        b = (b + a) % MOD_ADLER;
    }

    return (b << 16) | a;
}
```

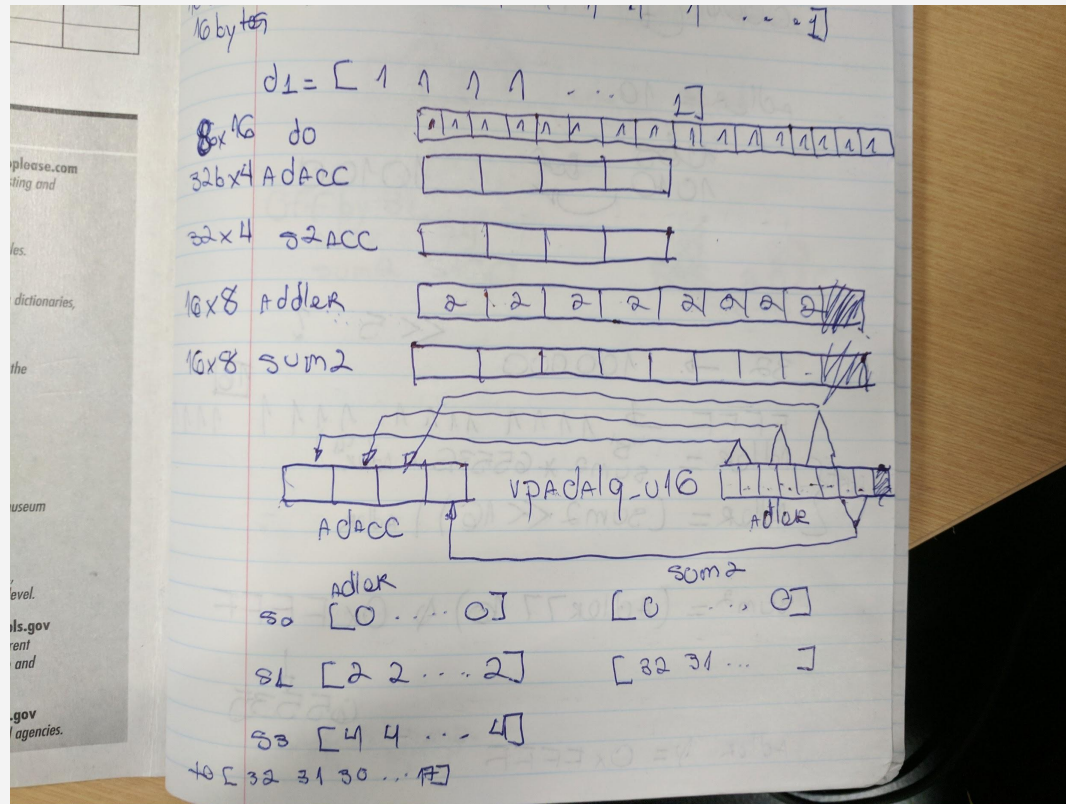
<https://en.wikipedia.org/wiki/Adler-32>

# Adler-32: problems

- zlib's Adler-32 was more than **7x faster** than naive implementation.
- It is hard to vectorize the following computation:

```
void accum(uint32_t *pair, const unsigned char *buf,
           unsigned int len)
{
    unsigned int i;
    for (i = 0; i < len; ++i) {
        pair[0] += buf[i];
        pair[1] += pair[0];
    }
}
```

# Adler-32: technical drawing (Jan 2017)





# Adler-32

## 'Taps' to the rescue

Assembly:

<https://godbolt.org/g/KMeBAJ>

```
static void NEON_accum32(uint32_t *s, const unsigned char *buf,
                        unsigned int len)
{
    static const uint8_t taps[32] = {
        32, 31, 30, 29, 28, 27, 26, 25,
        24, 23, 22, 21, 20, 19, 18, 17,
        16, 15, 14, 13, 12, 11, 10, 9,
        8, 7, 6, 5, 4, 3, 2, 1 };

    uint32x2_t adacc2, s2acc2, as;
    uint8x16_t t0 = vld1q_u8(taps), t1 = vld1q_u8(taps + 16);

    uint32x4_t adacc = vdupq_n_u32(0), s2acc = vdupq_n_u32(0);
    adacc = vsetq_lane_u32(s[0], adacc, 0);
    s2acc = vsetq_lane_u32(s[1], s2acc, 0);

    while (len >= 2) {
        uint8x16_t d0 = vld1q_u8(buf), d1 = vld1q_u8(buf + 16);
        uint16x8_t adler, sum2;
        s2acc = vaddq_u32(s2acc, vshlq_n_u32(adacc, 5));
        adler = vpaddlq_u8(d0);
        adler = vpadalq_u8(adler, d1);
        sum2 = vmull_u8(vget_low_u8(t0), vget_low_u8(d0));
        sum2 = vmlal_u8(sum2, vget_high_u8(t0), vget_high_u8(d0));
        sum2 = vmlal_u8(sum2, vget_low_u8(t1), vget_low_u8(d1));
        sum2 = vmlal_u8(sum2, vget_high_u8(t1), vget_high_u8(d1));
        adacc = vpadalq_u16(adacc, adler);
        s2acc = vpadalq_u16(s2acc, sum2);
        len -= 2;
        buf += 32;
    }
}
```



# Adler-32: Intel got some love too!

**author** Noel Gordon <noel@chromium.org> Fri Sep 29  
**committer** Commit Bot <commit-bot@chromium.org> Fri Sep 29  
**tree** [a25de9dd3212b49c1d903e72289e424b72127c3e](#)  
**parent** [6baf6221674f5a075f12f83e4262a4751b5d445b](#) [diff]

zlib Adler32.c

Add SSSE3 implementation of the Adler32 checksum, suitable for both large workloads, and small workloads commonly seen during PNG image decoding. Add a NEON implementation.

Speed is comparable to the serial Adler32 computation but near 64 bytes of input data, the SIMD code paths begin to be faster than the serial path: 3x faster at 256 bytes of input data, to ~8x faster for 1M of input data (~4x on ARMv8 NEON).

For the PNG 140 image corpus, PNG decoding speed is ~8% faster on average on the desktop machines tested, and ~2% on an ARMv8 Pixel C Android (N) tablet, <https://crbug.com/762564#c41>

Update x86.{c,h} to runtime detect SSSE3 support and use it to enable the Adler32\_simd code path and update inflate.c to call x86\_check\_features(). Update the name mangler file names.h for the new symbols added, add FIXME about simd.patch.

Ignore data alignment in the SSSE3 case since unaligned access is no longer penalized on current generation Intel CPU. Use it in the NEON case however to avoid the extra costs of unaligned memory access on ARMv8/v7.

NEON credits: the v\_s1/s2 vector component accumulate code was provided by Adenilson Cavalcanti. The uint16 column vector sum code is from libdeflate with corrections to process NMAX input bytes which improves performance by 3% for large buffers.

<https://bugs.chromium.org/p/chromium/issues/detail?id=688601>

# fast\_chunk

- Second candidate in the perf profiling was **inflate\_fast**.
- Very **high level** idea: perform long loads/stores in the byte array.
- Average **20% faster!**
- Shipping on M62.
- Original patch by Simon Hosie.

```

+                                     */
+                                     out = chunkcopy_safe(out, from, len, limit);
+                                     }
+                                 }
+                                 else {
-                                     from = out - dist;          /* copy direct from output */
-                                     do {                        /* minimum length is three */
-                                         *out++ = *from++;
-                                         *out++ = *from++;
-                                         *out++ = *from++;
-                                         len -= 3;
-                                     } while (len > 2);
-                                     if (len) {
-                                         *out++ = *from++;
-                                         if (len > 1)
-                                             *out++ = *from++;
-                                     }
+                                     /* Whole reference is in range of current output.  No
+                                     range checks are necessary because we start with room
+                                     for at least 258 bytes of output, so unroll and roundoff
+                                     operations can write beyond `out+len` so long as they
+                                     stay within 258 bytes of `out`.
+                                     */
+                                     out = chunkcopy_lapped_relaxed(out, dist, len);
+                                 }

```

<https://bugs.chromium.org/p/chromium/issues/detail?id=697280>

# CRC-32

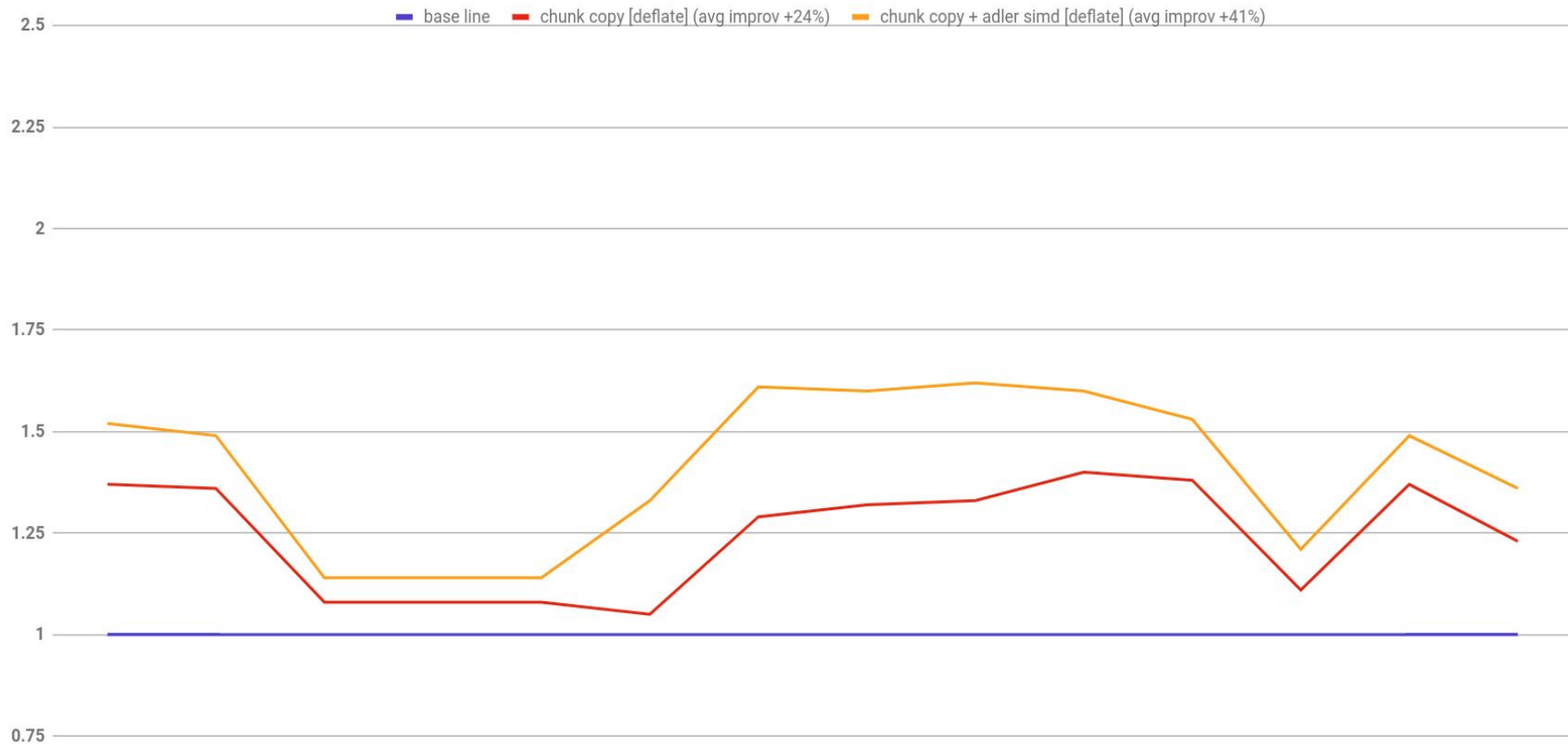
- YMMV on PNGs (from 1 to 5%).
- Remember it is used while **decompressing** web content (29% boost for gzipped content).
- ARMv8-a has a crc32 instruction (from 3 to 10x faster than zlib's crc32 C code).
- Shipping on M66.

<https://bugs.chromium.org/p/chromium/issues/detail?id=709716>

# Results: Chromium's zlib\*

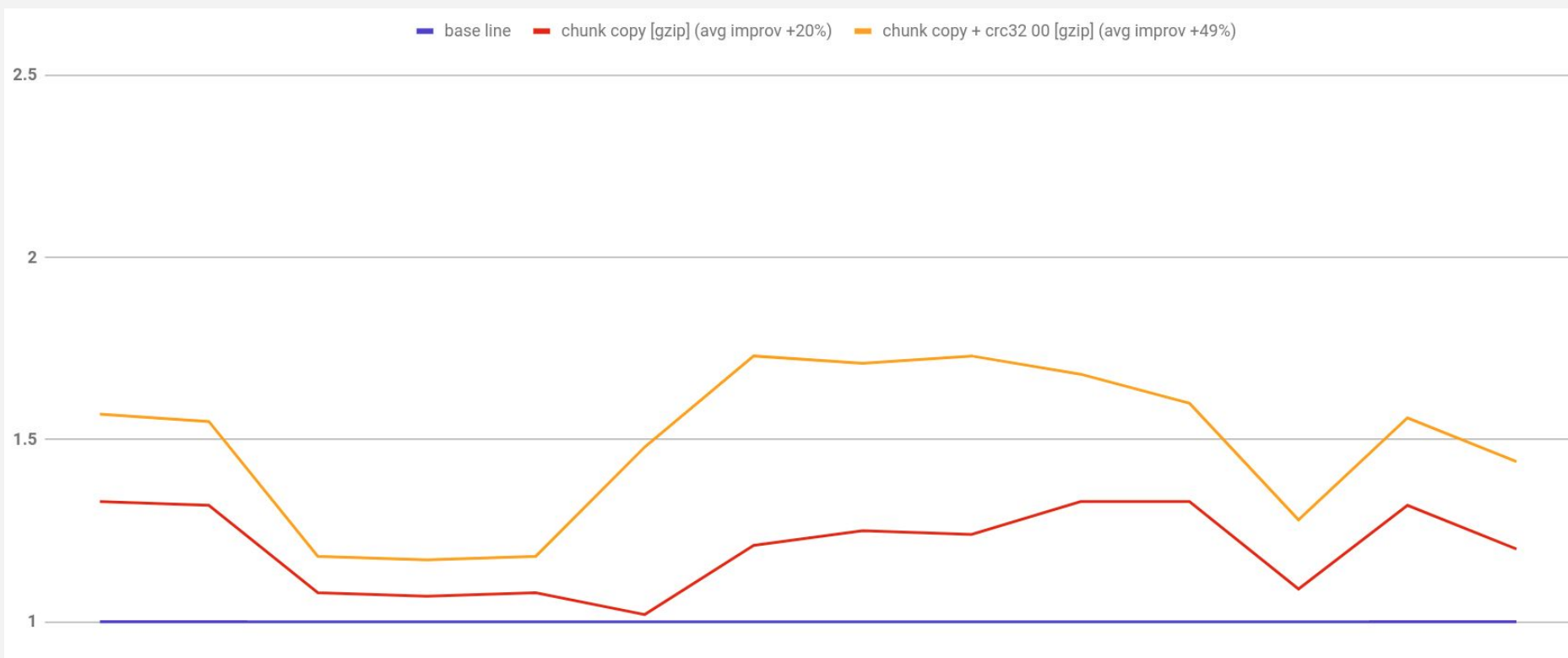
\* c-zlib

# Arm: zlib format 1.4x



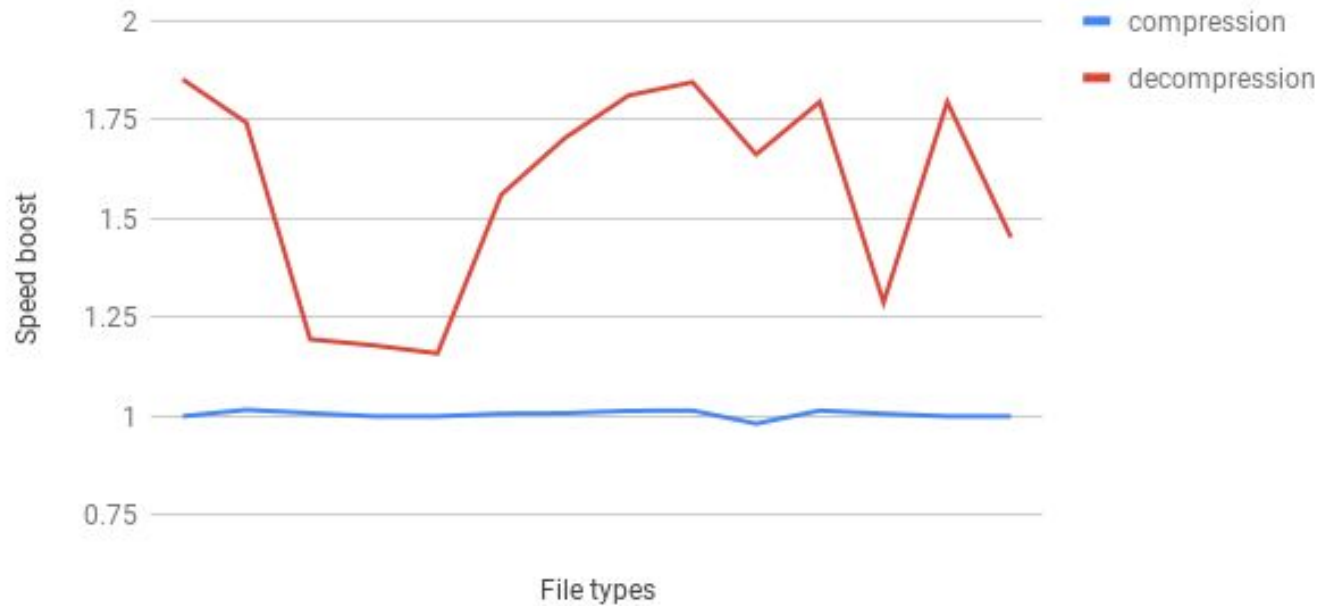


# Arm: gzip format 1.5x



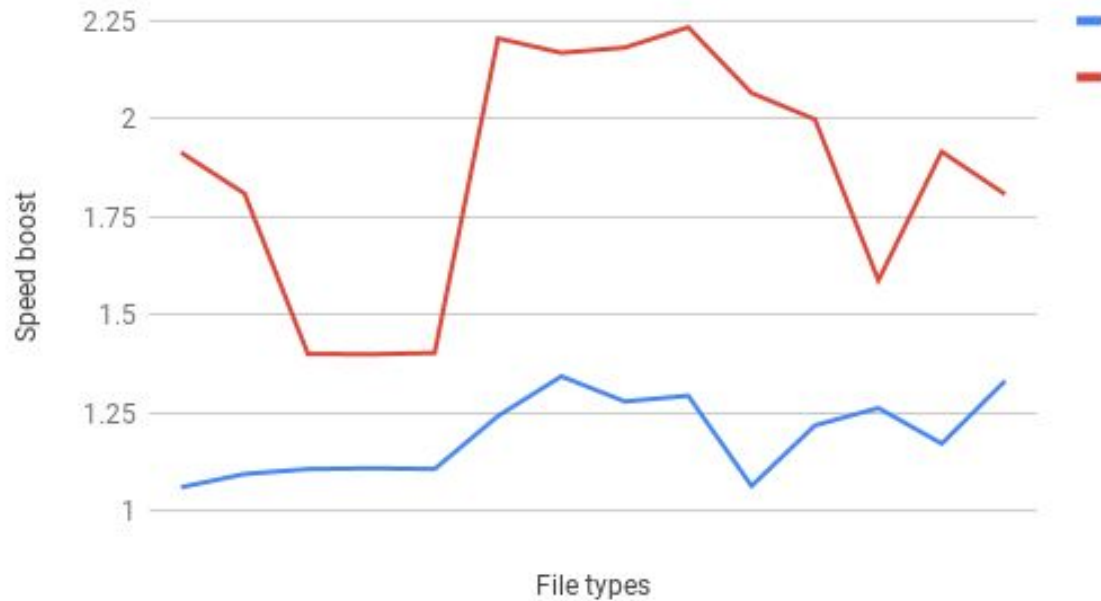
# Arm: c-zlib X Vanilla

ARM: decompression vs compression speed



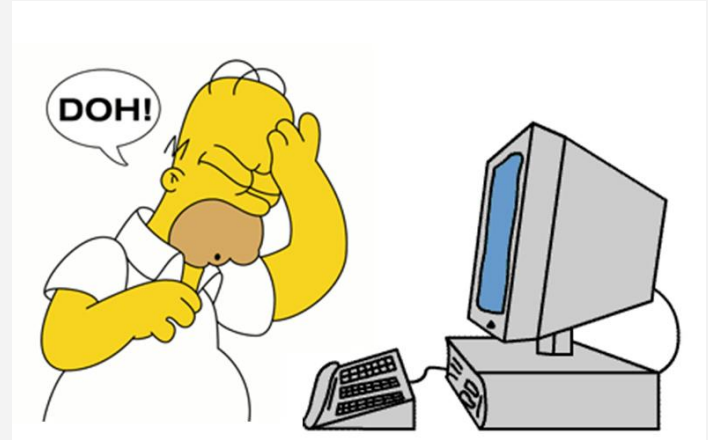
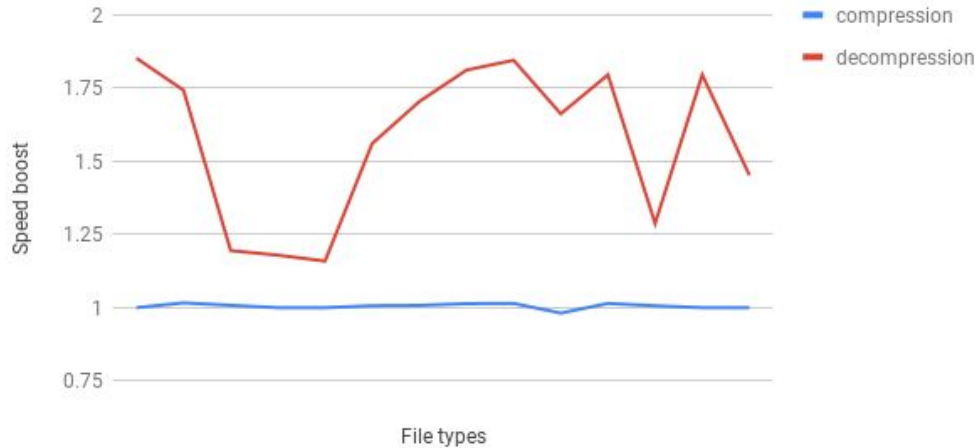
# x86: c-zlib X Vanilla

x86: decompression vs compression speed



# We were missing compression...

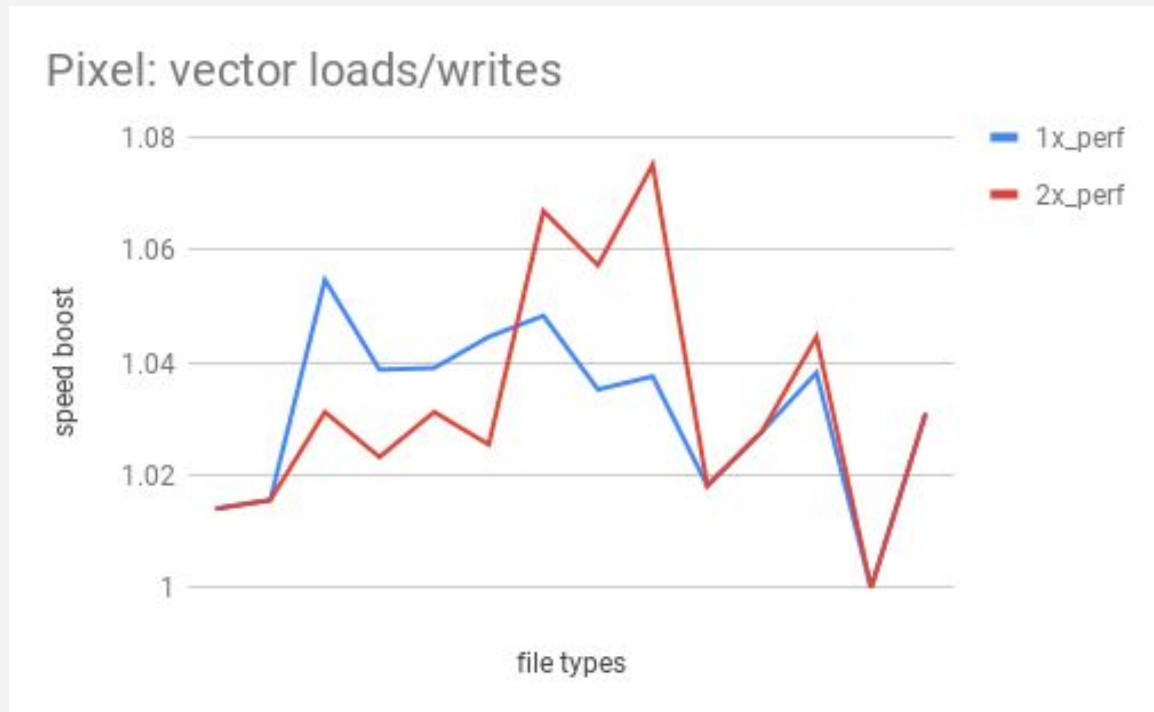
ARM: decompression vs compression speed



# Bonus: Compression on Arm

# Slide-hash: NEON

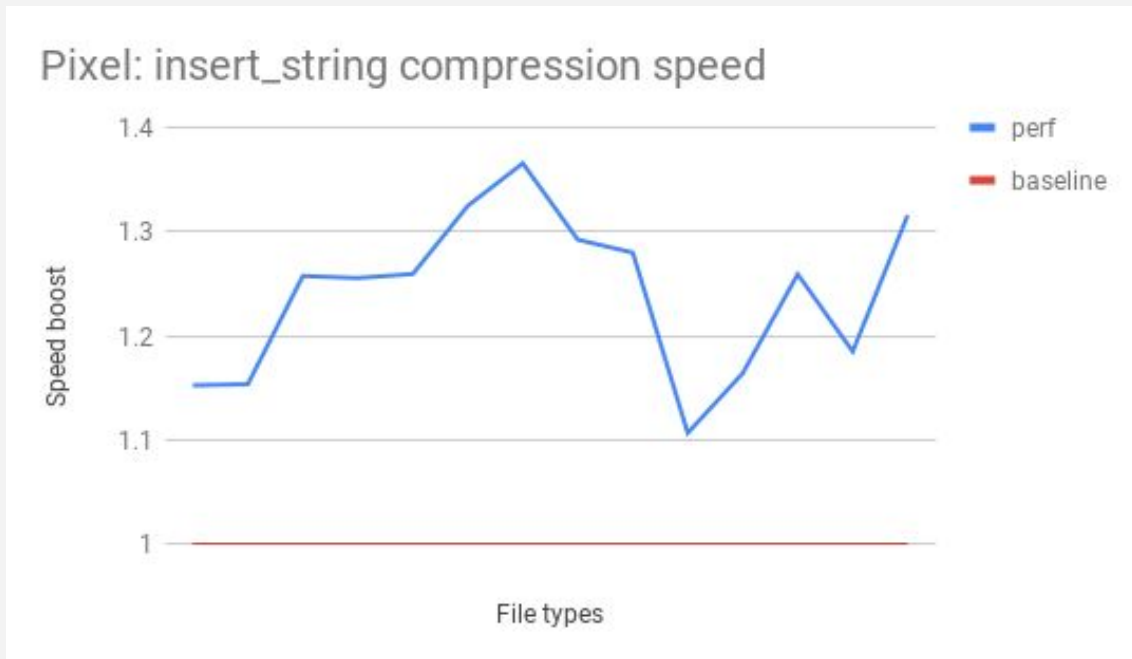
- Using NEON instruction vqsubq.
- Works on 8x 16bits chunks.
- Perf gain of 5%.





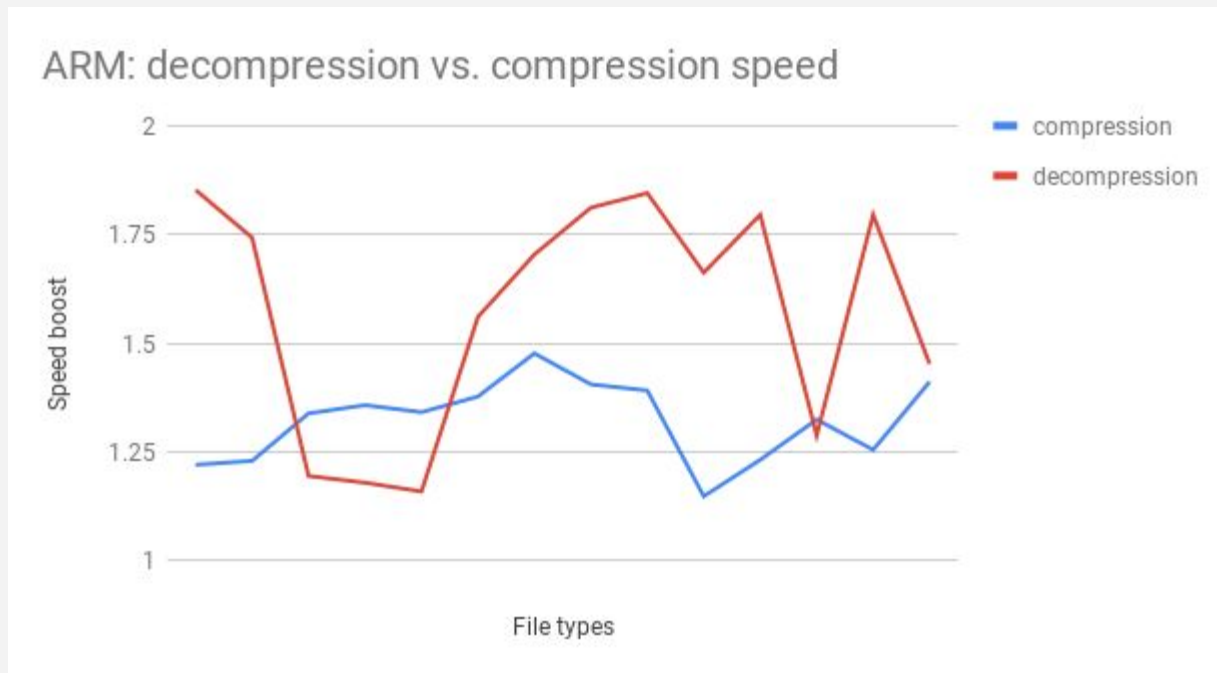
# insert-string: crypto CRC-32

- Using ARMv8-a instruction crc32.
- Works on 1x 32bits chunks.
- Perf gain of 24%.



# Arm: current state

- Compression: average 1.36x faster, but 1.4x faster for HTML.
- Decompression: average 1.6x faster (gzip), but 1.8x faster for HTML.



# Conclusions

# Conclusions

- There is plenty of life left even in an old code base.
- NEON optimizations can yield a \*huge\* impact.
- Benchmark + statistics: only way to handle variation\*.
- It pays off to work in a lower layer.
- OSS love: Intel got it too.

\*Power, scheduler, big.LITTLE, etc.

# Chromium's zlib: c-zlib

- Decompression: 1.7x to 2x faster than vanilla.
- Compression: 1.3x to 1.4x faster than vanilla.
- Both Arm & x86 are supported.
- Highly tested (i.e. cronet, fuzzers).
- Widely deployed (over 1 billion users).
- Open to performance & security patches.

# Chromium's zlib: c-zlib

- Decompression: 1.7x to 2x faster.
- Compression: 1.3x to 1.4x faster.
- Both Arm & x86 are supported.
- Highly tested (i.e. cronet, fuzzers).
- Widely deployed (over 1 billion users).
- Open to performance & security patches.

**zlib users should consider moving to Chromium's zlib if they care about performance.**



# Resources

- a) Slides: <https://goo.gl/vaZA9o>
- b) Performance benchmarks: <https://goo.gl/qLVdvh>
- c) Code:  
[https://cs.chromium.org/chromium/src/third\\_party/zlib/](https://cs.chromium.org/chromium/src/third_party/zlib/)

# Final words

*“This is how the open-source model works: building upon the work of others is far more efficient than rewriting everything.”*

Jean-loup Gailly (zlib author)

<https://slashdot.org/story/00/03/10/1043247/jean-loup-gailly-on-gzip-go-and-mandrake>

# Questions





# OPEN SOURCE SUMMIT

THE LINUX FOUNDATION