Optimizing zlib for arm

A deflated story

Adenilson Cavalcanti
BS. MSc.
Staff Engineer - Arm San Jose (CA)
THE CONFERENCE MORNING SESSION

DAY 1
7:00am
Welcome, everyone!

DAY 2
7:00am
Sorry, I haven't had my coffee yet...

DAY 3
7:00am
(Awkward silence)

LAST DAY
7:00am
I couldn't find an earlier flight.

Thanks for attending.
What to optimize in Chromium
What to optimize in Chromium

- Too big.
- Too many areas.
- What would be helpful?
What to optimize in Chromium

Bulk of content still is:
- Text.
- Images.
What to optimize in Chromium

Bulk of content still is:

- Text.
- Images.
What to optimize in Chromium

Bulk of content still is:
- Text.
- Images.
PNG

- Powerful format: Palette, pre-filters, compressed.
- Encoder affects behavior.
- Libpng and zlib are ‘Bros!’.
Parrots are not created equal
Parrots are not created equal

Original: 2.7MB
Palette: 0.8MB
Zopfli: 2.6MB
Features affect hotspots

<table>
<thead>
<tr>
<th>Lib</th>
<th>Command</th>
<th>SharedObj</th>
<th>method</th>
<th>CPU (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>zlib</td>
<td>TileWorker</td>
<td>libblink</td>
<td>inflate_fast</td>
<td>1.96</td>
</tr>
<tr>
<td>zlib</td>
<td>TileWorker</td>
<td>libblink</td>
<td>adler32</td>
<td>0.88</td>
</tr>
<tr>
<td>blink</td>
<td>TileWorker</td>
<td>libblink</td>
<td>ImageFrame::setRGBAPremultiply</td>
<td>0.45</td>
</tr>
<tr>
<td>blink</td>
<td>TileWorker</td>
<td>libblink</td>
<td>png_read_filter_row_up</td>
<td>0.03*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lib</th>
<th>Command</th>
<th>SharedObj</th>
<th>method</th>
<th>CPU (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>zlib</td>
<td>TileWorker</td>
<td>libblink</td>
<td>inflate_fast</td>
<td>3.06</td>
</tr>
<tr>
<td>zlib</td>
<td>TileWorker</td>
<td>libblink</td>
<td>adler32</td>
<td>1.36</td>
</tr>
<tr>
<td>blink</td>
<td>TileWorker</td>
<td>libblink</td>
<td>ImageFrame::setRGBAPremultiply</td>
<td>0.70</td>
</tr>
<tr>
<td>blink</td>
<td>TileWorker</td>
<td>libblink</td>
<td>png_read_filter_row_up</td>
<td>0.48*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lib</th>
<th>Command</th>
<th>SharedObj</th>
<th>method</th>
<th>CPU (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>libpng</td>
<td>TileWorker</td>
<td>libblink</td>
<td>cr_png_do_expand_palette</td>
<td>0.88</td>
</tr>
<tr>
<td>zlib</td>
<td>TileWorker</td>
<td>libblink</td>
<td>inflate_fast</td>
<td>0.62</td>
</tr>
<tr>
<td>blink</td>
<td>TileWorker</td>
<td>libblink</td>
<td>ImageFrame::setRGBAPremultiply</td>
<td>0.49</td>
</tr>
<tr>
<td>zlib</td>
<td>TileWorker</td>
<td>libblink</td>
<td>adler32</td>
<td>0.31</td>
</tr>
</tbody>
</table>
NEON: Advanced SIMD (Single Instruction Multiple Data)

- Optional in Armv7.
- Mandatory in Armv8.
Registers@Arm v7

- 16 registers@128 bits: Q0 - Q15.
- 32 registers@64bits: D0 - D31.
- Varied set of instructions: load, store, add, mul, etc.
Registers@Armv8 (SIMD&FP, V0 - V31)

- 32 registers@128 bits: Q0 - V31.
- 32 registers@64 bits: D0 - D31.
- 32 registers@32 bits: S0 - S31.
- 32 registers@8 bits: H0 - H31.
- Varied set of instructions: load, store, add, mul, etc.
An example: VADD.I16 Q0, Q1, Q2
Candidates

- Inflate_fast: zlib.
- Adler32: zlib.
- ImageFrame: Blink.
- png_do_expand_palette: libpng.
## Why zlib?

<table>
<thead>
<tr>
<th>Zlib</th>
<th>Context</th>
<th>Problem statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used everywhere (libpng, Skia, freetype, <strong>cronet</strong>, blink, chrome, linux kernel, etc). Old code base released in 1995. Written in K&amp;R C style.</td>
<td>Lacks any optimizations for ARM CPUs.</td>
<td>Identify potential optimization candidates and verify positive effects in Chromium.</td>
</tr>
</tbody>
</table>
Potential problems

- Viability of optimization.
- Positive effects.
- **Upstreaming.**
Implementation
Adler-32

\[ A = 1 + D_1 + D_2 + \ldots + D_n \pmod{65521} \]

\[ B = (1 + D_1) + (1 + D_1 + D_2) + \ldots + (1 + D_1 + D_2 + \ldots + D_n) \pmod{65521} \]

\[ = n\times D_1 + (n-1)\times D_2 + (n-2)\times D_3 + \ldots + D_n + n \pmod{65521} \]

\[ Adler-32(D) = B \times 65536 + A \]

https://en.wikipedia.org/wiki/Adler-32
Adler-32: simplistic implementation

```c
// 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;
}
```
Problems

- Zlib’s Adler-32 was more than \textbf{7x faster} than naive implementation.
- It is hard to vectorize the following computation:

```c
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];
    }
}
```
Problems: how to represent pair[1] or ‘B’?

```c
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];
    }
}
```
Problems: how to represent pair[1] or ‘B’?

\[ A = 1 + D_1 + D_2 + \ldots + D_n \pmod{65521} \]
\[ B = (1 + D_1) + (1 + D_1 + D_2) + \ldots + (1 + D_1 + D_2 + \ldots + D_n) \pmod{65521} \]
\[ = n \times D_1 + (n-1) \times D_2 + (n-2) \times D_3 + \ldots + D_n + n \pmod{65521} \]

**Adler-32(D) =** \[ B \times 65536 + A \]
Highly technical drawing (Jan 2017)
‘Taps’ to the rescue

Assembly:
https://godbolt.org/g/KMeBAJ

```c
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(adler, d0);
        adler = vpaddlq_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 = vpaddq_u16(adacc, adler);
        s2acc = vpaddlq_u16(s2acc, sum2);
        len -= 2;
        buf += 32;
    }
}
```
Happy end! Up to 18% performance gain in PNG

commit d400b38e450a71e9ed75bd4c1dda6329267e9ce0
Author: Adenilson Cavalcanti <adenilson.cavalcanti@arm.com>
Date:   Mon Jan 30 15:30:38 2017 -0800

NEON implementation for Adler32

The checksum is calculated in the uncompressed PNG data and can be made
much faster by using SIMD.

Tests in ARMv8 yielded an improvement of about 3x (e.g. walltime was
350ms x 125ms for a 4096x4096 bytes executed 30 times).

This alone yields a performance boost for PNG decoding ranging
from 5% to 18% depending on a few factors (SoC, battery status,
big/little, etc).

https://bugs.chromium.org/p/chromium/issues/detail?id=688601
Inffast (Simon Hosie)

- Second candidate in the perf profiling was `inflate_fast`.
- Very high level idea: perform long loads/stores in the byte array.
- Major gains: up to 30% faster!

https://bugs.chromium.org/p/chromium/issues/detail?id=697280
Libpng (Richard Townsend)

- NEON optimization in libpng.
- From 10 to 30% improvement.
- Depends on png using a palette.

https://bugs.chromium.org/p/chromium/issues/detail?id=706134
Impact

Combined effect of 3 patches
Chrome trace: patched (73ms) 1.6x improvement
Comparing Arm x Intel

Source: https://commons.wikimedia.org/wiki/File:Apple_and_Orange_-_they_do_not_compare.jpg
Keeping in mind

- Snapdragon™ 805 @2014.
- 2.7Ghz Krait™ 450.
- 2MB L2 cache
- 28nm lithography.
- Cellphone.
- EAS kernel.

- 5Y10C launched @2015.
- 2Ghz Intel m5.
- 4MB cache.
- 14nm lithography.
- Ultrabook.
- Regular linux kernel.
Chrome trace: Intel m5@2016 (66ms)
Effect of NEON optimization in Zlib

2012 Arm core x 2014 Intel processor (ms)

- ARM
- Intel

vanilla
NEON
Lessons learned

- **arm** cores can benefit **a lot** from NEON optimizations.
- Performance gains of 2 generations of silicon.
- It pays off to work in a lower software layer (e.g. zlib/libpng).
Happy end? Not yet...

- Requested to perform a study comparing zlib forks.
- Upstream ARM optimizations.
- Move Chromium to a new/better maintained zlib.
Happy end? Not yet...

- Requested to perform a study comparing zlibs forks. Done!
  - [https://goo.gl/ZUoy96](https://goo.gl/ZUoy96)
- Upstream ARM optimizations. Done!
  - [https://github.com/Dead2/zlib-ng/commit/ec02ecf104e1d3f1836a908a359f20aa93494df5](https://github.com/Dead2/zlib-ng/commit/ec02ecf104e1d3f1836a908a359f20aa93494df5)
- Move Chromium to a new/actively maintained zlib.
  - Upgraded/moved PDFium to Chromium’s zlib.
  - Zlib-ng **didn’t release** a stable release.
January: Initial investigation
February: Zlib forks benchmarking
April: Upstreaming to zlib-ng
... August: Still no zlib-ng release

PDFium zlib
Let's optimize zlib for ARM

Adenilson Cavalcanti

Zlib is a compression library used by Chromium codebase and its dependencies (skia, libpng, pdfium, freetype, etc) for quite a few tasks ranging from image handling, loading extensions and accessing compressed content (i.e. Content-Encoding: gzip).

It is an impressive feat of engineering considering that it is 22 years old and is used all over the place (e.g. Linux kernel). Due to its history and the need to support long gone compilers and operating systems, its main focus has been portability than performance.
NEON inffast: featured in M62

https://bugs.chromium.org/p/chromium/issues/detail?id=697280
cronet: NEON != ARMv6

Source: https://xkcd.com/1172/
After re-landing… An internal app was broken.

Source: https://xkcd.com/1172/
Second revert (i.e. revert-revert-revert)

Misha Efimov@Google found the bug in the Java app client last Wednesday (Sep 27th).
Re-re-landed on Thur 28th
What comes next

- Land Adler-32 optimization* (Noel Gordon@Google implemented the same algorithm for Intel).
- Land the libpng optimization.
- CRC32: Armv8 instruction is about 10x faster.
- Compression comes next.

*Just landed last Friday: https://chromium-review.googlesource.com/c/chromium/src/+/660019
Adler-32 landed on Fri 29th

https://goo.gl/RTgkGe
What comes next

- Land the libpng optimization.
- CRC32: ARMv8 instruction is about 10x faster.
- Fix infback corner case.
- Compression comes next.

Zlib users should consider migrating to Chromium’s zlib.
Special Thanks

- Igalia for the invite (Xabier Rodriguez Calvar).
- Arm for sponsoring the trip.
- Chris Blume@Google.
- Team Arm@UK: Dave Rodgman, Matteo Franchin, Richard Townsend, Stephen Kyle.
- Team Arm@US: Amaury Leleyzour, Simon Hosie.
- Compiler explorer: https://godbolt.org
Questions?
The Arm Trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

https://www.arm.com/company/policies/trademarks