Debugging assembly functions

Martin Storsjö, VideoLAN Developer Days 2024

Pinpointing mismatches

- Checkasm is passing for my function, but the real decoder is giving the wrong output in some cases
  - How to debug this?

How to find the bug?

```
void dav1d_avg_8bpc_neon(pixel *dst, ptrdiff_t dst_stride,
                         const int16_t *tmp1, const int16_t *tmp2,
                         int w, int h);

void mc_dsp_init_arm(Dav1dMCDSPContext *c) {
    if (flags & DAV1D_ARM_CPU_FLAG_NEON) {
       c->avg = dav1d_avg_8bpc_neon;
```

How to find the bug?

```
void dav1d_avg_8bpc_neon(pixel *dst, ptrdiff_t dst_stride,
                         const int16_t *tmp1, const int16_t *tmp2,
                         int w, int h);
void dav1d_avg_8bpc_c(pixel *dst, ptrdiff_t dst_stride,
                      const int16_t *tmp1, const int16_t *tmp2,
                      int w, int h);

static void avg_8bpc_wrap(pixel *dst, ptrdiff_t dst_stride,
                          const int16_t *tmp1, const int16_t *tmp2,
                          int w, int h) {
    pixel ref[64*64];
    dav1d_avg_8bpc_c(ref, 64, tmp1, tmp2, w, h);
    dav1d_avg_8bpc_neon(tmp, stride, tmp1, tmp2, w, h);
    for (int y = 0; y < h; y++)
        if (memcmp(&dst[y*dst_stride], &ref[y*64], w)) {
            // Found a mismatch! Dump the inputs, the outputs, and abort
            printf(..., tmp1, tmp2, dst);
            abort();
        }
}

void mc_dsp_init_arm(Dav1dMCDSPContext *c) {
    if (flags & DAV1D_ARM_CPU_FLAG_NEON) {
       c->avg = dav1d_avg_8bpc_wrap;
```

Ways to debug your assembly function

1. Don’t write incorrect code in the first place!
2. Stare at the code until you understand what’s wrong
    - `checkasm -v` can be very helpful for figuring out where to look and what to look for
3. Use a debugger (GDB or LLDB)
    - Set a breakpoint, check register contents, step through the function
        - Works, but requires a lot of work
        - What if the interesting case only happens on the 500th iteration?
4. Use printf debugging
    - Very easy - with the right tools!

What does it look like?

```
8:
        ld1             {v2.16b},  [x5],  #16
        ld1             {v1.16b},  [x2],  #16
        ld1             {v0.8b},   [x0]
        ld1             {v0.d}[1], [x8]
        DUMP_SIMD_U8("tmp", 2, 2)
        DUMP_SIMD_U8("mask", 5, 5)
        DUMP_SIMD_U8("dst", 0, 0)
        sub             v3.16b,  v4.16b,  v2.16b
        DUMP_GPR("h", 4, 4)
        subs            w4,  w4,  #2
        umull           v5.8h,   v1.8b,   v2.8b
        umlal           v5.8h,   v0.8b,   v3.8b
        umull2          v6.8h,   v1.16b,  v2.16b
        umlal2          v6.8h,   v0.16b,  v3.16b
        DUMP_SIMD_U16("sum", 5, 5)
        rshrn           v5.8b,   v5.8h,   #6
        rshrn           v6.8b,   v6.8h,   #6
        DUMP_SIMD_U8("dst", 5, 6)
        st1             {v5.8b},   [x0],  x1
        st1             {v6.8b},   [x8],  x1
        b.gt            8b
        ret
```

What does it look like?

```
471: tmp:
v02 = { 28, 63, 47, 48, 26, 14, 10, 62, 59, 32, 33, 33, 44, 18, 38, 24 }
472: mask:
v05 = { 184, 26, 86, 49, 188, 14, 176, 35, 48, 36, 187, 57, 154, 24, 120, 56 }
473: dst:
v00 = { 131, 155, 86, 157, 199, 103, 80, 240, 17, 200, 238, 172, 64, 28, 232, 37 }
475: h:
x04 = 4
481: sum:
v05 = { 6340, 3935, 13400, 6160, 9148, 7166, 5480, 11640 }
484: dst:
v05 = { 99, 61, 209, 96, 143, 112, 86, 182, 0, 0, 0, 0, 0, 0, 0, 0 }
v06 = { 128, 121, 141, 134, 70, 77, 120, 56, 0, 0, 0, 0, 0, 0, 0, 0 }
```

How?

```
.macro call_c func, line, msg, first, last, ptr, offset
        stp             x29, x30, [sp, #-16]!
        stp             x27, x28, [sp, #-16]!
        [...]
        stp             x1,  x2,  [sp, #-16]!
        mrs             x30, nzcv
        stp             x30, x0,  [sp, #-16]!

        stp             q30, q31, [sp, #-32]!
        [...]
        stp             q0,  q1,  [sp, #-32]!

        add             x4,  \ptr, #\offset
        ldr             x0,  =\line
        adr             x1,  777f
        mov             x2,  #\first
        mov             x3,  #\last
        bl              X(\func)

        ldp             q0,  q1,  [sp], #32
        [...]
        ldp             q30, q31, [sp], #32

        ldp             x30, x0,  [sp], #16
        msr             nzcv, x30
        ldp             x1,  x2,  [sp], #16
        [...]
        ldp             x27, x28, [sp], #16
        ldp             x29, x30, [sp], #16
.endm
```

How?

```
#define CALL_C(func, msg, first, last, ptr, offset) \
        call_c func, __LINE__, msg, first, last, ptr, offset ; \
        b 778f ; \
    777: \
        .asciz msg ; \
        .align 2 ; \
    778:

#define DUMP_SIMD_S8(msg, first, last)  CALL_C(dump_simd_s8,  msg, first, last, sp, 0)
#define DUMP_SIMD_U8(msg, first, last)  CALL_C(dump_simd_u8,  msg, first, last, sp, 0)
#define DUMP_SIMD_S16(msg, first, last) CALL_C(dump_simd_s16, msg, first, last, sp, 0)
#define DUMP_SIMD_U16(msg, first, last) CALL_C(dump_simd_u16, msg, first, last, sp, 0)
#define DUMP_SIMD_S32(msg, first, last) CALL_C(dump_simd_s32, msg, first, last, sp, 0)
#define DUMP_GPR(msg, first, last)      CALL_C(dump_gpr,      msg, first, last, sp, 520)
```

How?

```
void dav1d_dump_simd_s16(int line, const char *msg, int first, int last,
                         const int16_t *ptr) {
    printf("%d: %s:\n", line, msg);
    for (int v = first; v <= last; v++) {
        const int16_t *vec = &ptr[8*v];
        printf("v%02d = {", v);
        for (int i = 0; i < 8; i++)
            printf("%s %d", i > 0 ? "," : "", vec[i]);
        printf(" }\n");
    }
}
```

Where?

- Current PoC patches available online
  - https://code.videolan.org/mstorsjo/dav1d/-/commits/arm-asm-dump
  - https://github.com/mstorsjo/ffmpeg/commit/arm-asm-dump