How fast goes the light?

Euro LLVM 2015

Arnaud de Grandmaison
Scope

- Speed of light: the fastest implementation of a function on a given cpu (Cortex-A57)

- The function under test is a typical image processing kernel:
  - Color space conversion from RGB to YIQ (see [http://en.wikipedia.org/wiki/YIQ](http://en.wikipedia.org/wiki/YIQ))

\[
\begin{bmatrix}
Y \\
n \\
q
\end{bmatrix} =
\begin{bmatrix}
Y_r & Y_g & Y_b \\
I_r & I_g & I_b \\
Q_r & Q_g & Q_b
\end{bmatrix}
\begin{bmatrix}
r \\
g \\
b
\end{bmatrix}
\]

- That’s the most basic computation out there, so we’d better get it right…
RGB2YIQ in C, with 16-bits integer coefficients

```c
void rgb2yiq(uint8_t *restrict In, uint8_t *restrict Out, unsigned N) {
    for (unsigned pixel = 0; pixel < N; pixel++) {
        uint8_t r = *In++, g = *In++, b = *In++;

        uint8_t y = ((YR * r) + (YG * g) + (YB * b) + HALF_LSB) >> S;
        int8_t i = ((IR * r) + (IG * g) + (IB * b) + HALF_LSB) >> S;
        int8_t q = ((QR * r) + (QG * g) + (QB * b) + HALF_LSB) >> S;

        *Out++ = y, *Out++ = i, *Out++ = q;
    }
}
```

No aliasing

Matrix x vector

Rounding
Expectations

- 9 or 10 coefficients loading
- 9 Multiply-accumulate
- Vectorization
A first shot…

rgb2yiq_ref:

```assembly
  cbz w2, .LBB0_3
  movz w8, #0x4c8b
  movz w9, #0x9646
  movz w10, #0x1d2f
  movn w11, #0x3b0e
  movn w12, #0x44ef
  movz w13, #0x33e2
  movz w14, #0x4c1d

.LBB0_2:
  ldrb w15, [x0]
  ldrb w16, [x0, #1]
  mul w18, w15, w8
  mul w3, w16, w9
  ldrb w17, [x0, #2]
  lsl w5, w15, #15
  sub w5, w5, w15
  mul w15, w15, w13
  mul w4, w17, w10
  add w18, w18, w3
  mul w3, w16, w11
  sub w16, w16, w16, lsl #15
```

7 coefficients

`.LBB0_2:

```assembly
  add w15, w15, w16
  add w16, w18, w4
  add w3, w5, w3
  mul w5, w17, w12
  add w16, w16, #8, lsl #12
  mul w17, w17, w14
  lsr w16, w16, #16
  add w18, w3, w5
  add w15, w15, w17
  add w17, w18, #8, lsl #12
  add w15, w15, #8, lsl #12
  lsr w17, w17, #16
  lsr w15, w15, #16
  strb w16, [x1]
  strb w17, [x1, #1]
  strb w15, [x1, #2]
  sub w2, w2, #1
  add x0, x0, #3
  add x1, x1, #3
  cbnz w2, .LBB0_2

.LBB0_3:
  ret
```

Immediate half LSB

No multiply-accumulate, no vectorization!
### Performances (reference)

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Code size</th>
<th>Data size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First shot (reference)</td>
<td>1.0</td>
<td>1.0</td>
<td>0</td>
</tr>
</tbody>
</table>
RGB2YIQ v2 : fight the compiler!

```c
int Coefs[3][3] = {{YR, YG, YB}, {IR, IG, IB}, {QR, QG, QB}};
int Half_LSB = HALF_LSB;

void rgb2yiq(uint8_t *restrict In, uint8_t *restrict Out, unsigned N) {
    int yr = Coefs[0][0], yg = Coefs[0][1], yb = Coefs[0][2];
    int ir = Coefs[1][0], ig = Coefs[1][1], ib = Coefs[1][2];
    int qr = Coefs[2][0], qg = Coefs[2][1], qb = Coefs[2][2];
    int half_lsb = Half_LSB;

    for (unsigned pixel = 0; pixel < N; pixel++) {
        uint8_t r = *In++, g = *In++, b = *In++;

        uint8_t y = ((yr * r) + (yg * g) + (yb * b) + half_lsb) >> S;
        int8_t i = ((ir * r) + (ig * g) + (ib * b) + half_lsb) >> S;
        int8_t q = ((qr * r) + (qg * g) + (qb * b) + half_lsb) >> S;

        *Out++ = y, *Out++ = i, *Out++ = q;
    }
}
```

- Place coefficients in memory
- Make sure it does not alias with `In` or `Out`, and is hoisted of the loop
rgb2yiq:

```assembly
stp x20, x19, [sp, #16]!
cbz w2, .LBB0_3
adrp x16, Coeffs
add x16, x16, :lo12:Coeffs
adrp x17, Half_LSB
ldp w8, w9, [x16]
ldp w10, w11, [x16, #8]
ldp w12, w13, [x16, #16]
ldp w14, w15, [x16, #24]
ldr w16, [x16, #32]
ldr w17, [x17, :lo12:Half_LSB]

.LBB0_2:

ldrb w18, [x0]
ldrb w3, [x0, #1]
mul w5, w3, w9
madd w7, w18, w8, w17
ldrb w4, [x0, #2]
mul w19, w3, w12
mul w3, w3, w15
mul w6, w4, w10
add w5, w7, w5

.LBB0_3:

ldp x20, x19, [sp], #16
ret
```

9 coefficients + half lsb

3 MACs!
Performances (lower is better)

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Code size</th>
<th>Data size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First shot (reference)</td>
<td>1.0</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Second try</td>
<td>1.03</td>
<td>1.0</td>
<td>40</td>
</tr>
</tbody>
</table>
Let’s ignore the compiler…

```
rgb2yiq:

    cbz w2, .LBB0_3
    adrp x16, Coeffs
    add x16, x16, :lo12:Coeffs
    adrp x17, Half_LSB
    ldp w8, w9, [x16]
    ldp w10, w11, [x16, #8]
    ldp w12, w13, [x16, #16]
    ldp w14, w15, [x16, #24]
    ldp w16, [x16, #32]
    ldp w17, [x17, :lo12:Half_LSB]

.LBB0_2:

    ldrb w3, [x0]
    ldrb w4, [x0, #1]
    ldrb w5, [x0, #2]

    madd w6, w3, w8, w17
    madd w6, w4, w9, w6
    madd w6, w5, w10, w6

    madd w7, w3, w11, w17
    madd w7, w4, w12, w7
    madd w7, w5, w13, w7
```

Shift

Load coefficients

Multiply-add

```
.madd w18, w3, w14, w17
.madd w18, w4, w15, w18
.madd w18, w5, w16, w18

.lsr w6, w6, #16
.lsr w7, w7, #16
.lsr w18, w18, #16

.strb w6, [x1]
.strb w7, [x1, #1]
.strb w18, [x1, #2]

.add x0, x0, #3
.add x1, x1, #3
.sub w2, w2, #1
.cbz w2, .LBB0_2

.retn
```
Performances (lower is better)

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Code size</th>
<th>Data size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First shot (reference)</td>
<td>1.0</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Second try</td>
<td>1.03</td>
<td>1.0</td>
<td>40</td>
</tr>
<tr>
<td>Hand written straight asm (scalar)</td>
<td>0.94</td>
<td>0.80</td>
<td>40</td>
</tr>
</tbody>
</table>
Performances (lower is better)

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Code size</th>
<th>Data size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First shot (reference)</td>
<td>1.0</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Second try</td>
<td>1.03</td>
<td>1.0</td>
<td>40</td>
</tr>
<tr>
<td>Hand written straight asm (scalar)</td>
<td>0.94</td>
<td>0.80</td>
<td>40</td>
</tr>
<tr>
<td>Hand written scheduled asm (scalar)</td>
<td>0.79</td>
<td>0.80</td>
<td>40</td>
</tr>
</tbody>
</table>
What about vectorization?

1. Load 8 pixels from memory to neon registers

   Load 8 pixels from memory to neon registers

   - `ld3 {v0, v1, v2}, [x0], #24`

   

2. Expand to 32 bits (`utxlt`, `utxlt2`)

   Expand to 32 bits (`utxlt`, `utxlt2`)

   - `v0 r3 r2 r1 r0`
   - `v1 g3 g2 g1 g0`
   - `v2 b3 b2 b1 b0`
   - `v3 r7 r6 r5 r4`
   - `v4 g7 g6 g5 g4`
   - `v5 b7 b6 b5 b4`
What about vectorization (cont.)

3. Bunch of `mul / mla` with the coefficients

4. Round shift right the `y, i, q` results to 16 bits (`rshrn, rshrn2`)

<table>
<thead>
<tr>
<th></th>
<th>y7</th>
<th>y6</th>
<th>y5</th>
<th>y4</th>
<th>y3</th>
<th>y2</th>
<th>y1</th>
<th>y0</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td>i7</td>
<td>i6</td>
<td>i5</td>
<td>i4</td>
<td>i3</td>
<td>i2</td>
<td>i1</td>
<td>i0</td>
</tr>
<tr>
<td>v2</td>
<td>q7</td>
<td>q6</td>
<td>q5</td>
<td>q4</td>
<td>q3</td>
<td>q2</td>
<td>q1</td>
<td>q0</td>
</tr>
</tbody>
</table>

5. Extract and compact the 8 LSB from the `y, i, q` results (`xtn`)

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6. And store with `st3 {v0, v1, v2}, [x1], #24`

<table>
<thead>
<tr>
<th>memory</th>
<th>...</th>
<th>y0</th>
<th>i0</th>
<th>q0</th>
<th>y1</th>
<th>i1</th>
<th>q1</th>
<th>y2</th>
<th>i2</th>
<th>...</th>
</tr>
</thead>
</table>
Performances (lower is better)

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th>Code size</th>
<th>Data size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First shot (reference)</td>
<td>1.0</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Second try</td>
<td>1.03</td>
<td>1.0</td>
<td>40</td>
</tr>
<tr>
<td>Hand written straight asm (scalar)</td>
<td>0.94</td>
<td>0.80</td>
<td>40</td>
</tr>
<tr>
<td>Hand written scheduled asm (scalar)</td>
<td>0.79</td>
<td>0.80</td>
<td>40</td>
</tr>
<tr>
<td>Hand written asm (vector)</td>
<td>0.49</td>
<td>1.88</td>
<td>48</td>
</tr>
</tbody>
</table>
Thank you!