March 5, 2012

SUB-WORD HANDLING IN DATA-PARALLEL MAPPING

Georgia Psychou, Robert Fasthuber, Jos Hulzink, Jos Huisken and Francky Catthoor

University of Patras, Greece; Holst Center/IMEC, The Netherlands; IMEC, Belgium;
120 data items

Fixed-point operands
- Calculations make operands change their word-length size
- Bit handling is required
- Performance/Energy/Area trade-offs
- Selection of when to change word-lengths is decisive choice in the trade-off space

MOTIVATION
CONTRIBUTION

• Systematic exploration of handling data word-lengths on different SIMD platform options (given the minimal wordlengths)
  - Time multiplexed instruction-set processors
  - Hard- and Soft-SIMD platforms
  - Architectural components: data memory, register file, arithmetic operators, sub-word rearrangement units

• Apply the methodology on a realistic embedded application and illustrate the different mapping results in terms of performance, energy, area
DRIVER APPLICATION

- Critical part
- Scans the whole image
- Big amount of constant multiplications
APPLICATION CODE

Four operations of the optimized Gauss Loop (1D filter):

```c
for (x = 1; x < N - 1; x++){
    for (y = 2; y < M - 1; y++) {
        M0 = imsub[x-1][y] + imsub[x+1][y];
        M7 = M0 * Gauss[0][1];
        M4 = imsub[x][y] * Gauss[0][2];
        imgauss_x[x][y] = M4 + M7;
    }
}
```

<table>
<thead>
<tr>
<th>Signal</th>
<th>Bitwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>imsub</td>
<td>7</td>
</tr>
<tr>
<td>M0</td>
<td>8</td>
</tr>
<tr>
<td>M7</td>
<td>13</td>
</tr>
<tr>
<td>M4</td>
<td>13</td>
</tr>
<tr>
<td>imgauss_x</td>
<td>12</td>
</tr>
</tbody>
</table>

Alternative multiplication options:

- Multiplier unit
- Shift-add unit (strength reduction)
- Soft-SIMD in shift-add way

(see paper for other options)
SOFTSIMD IN SHIFT-ADD WAY

\[ M_7 = M_0 \times \text{Gauss}[0][1] \]

\[
((-M_0 >> 4) + M_0) \\
(>>3 + M_0) \\
(>>2 + M_0)
\]

M7 = \((-M_0 >> 4) + M_0\) \(>>3 + M_0\) \(>>2 + M_0\)

FloatP : 0.08
FixedP: 655
Binary : 1010001111
CSD : 101001000-

12 bits
16 bits
16 bits
SOFTSIMD IN SHIFT-ADD WAY

\[ M7 = ((-(M0>>4) + M0) >>3+M0)>>2+M0 \]
HYBRID SOLUTIONS

- Shifts
  - e.g. M0 >> 4

M0  10101101 0000
M0 >> 4  000010101101

Prevent moving into right-hand subword

10 bits
8 bits

Cut-off bits

HYBRID

Resize

Pre-allocate

Data memory

Register file

Pre-allocate

Hardware

Software instruction

Provide extra space and guard bits
Mapping of 4 operations of the optimized Gauss Loop (1D filter)

```c
for (x = 1; x < N - 1; x++) {
    for (y = 2; y < M - 1; y++) {
        M0 = imsub[x-1][y] + imsub[x+1][y];
        M7 = M0 * Gauss[0][1];
        M4 = imsub[x][y] * Gauss[0][2];
        imgauss_x[x][y] = M4 + M7;
    }
}
```

- 8-bit subwords stored in data memory - 1 guard bit (MSB)
- **HYBRID**
  - In total: Six 8-bit subwords operating in parallel
  - Extended to 12-bit with the shuffler - 1 guard bit (MSB)
  - 1 bit cut-off (LSB)
  - 4 guard bits (LSB)
- **HYBRID**
  - In total: Four 12-bit subwords operating in parallel
MAPPING OF APPLICATION

Mapping of 4 operations of the optimized Gauss Loop (1D filter)

```c
for (x = 1; x < N - 1; x++){
    for (y = 2; y < M - 1; y++) {
        M0= imsub[x-1][y] + imsub[x+1][y];
        M7= M0 * Gauss[0][1];
        M4= imsub[x][y] * Gauss[0][2];
        imgauss_x[x][y] = M4 + M7;
    }
}
```

- 12-bit subwords resized to 16-bit subwords - 1 guard bit (MSB)
  - 3 guard bits (LSB)
- Cut-off 2 bits during the last shift (LSB)
- In total: Three 16-bit subwords operating in parallel
ALTERNATE IMPLEMENTATIONS

1. Soft-SIMD, 3 wl
2. Hard-SIMD, 3 wl
3. Hard-SIMD, 2 wl
4. SIMD SAS, 1 wl
5. HardSIMD Mult, 3 wl
6. SIMD Mult, 1 wl
ALTERNATE IMPLEMENTATIONS

1. Soft-SIMD, 3 wl
2. Hard-SIMD, 3 wl
3. Hard-SIMD, 2 wl
4. SIMD SAS, 1 wl
5. HardSIMD Mult, 3 wl
6. SIMD Mult, 1 wl
RESULTS

- One Gauss iteration (data-path: 48 bits)
- Algorithm manually scheduled and optimized
- TSMC 40 nm std. cell

<table>
<thead>
<tr>
<th>Data-path option</th>
<th>Area</th>
<th>Energy</th>
<th>Throughput</th>
<th>Through/Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoftSIMD</td>
<td>1.63</td>
<td>0.80</td>
<td>1.69</td>
<td>2.12</td>
</tr>
<tr>
<td>HardSAS, 3 wl</td>
<td>1.48</td>
<td>0.72</td>
<td>1.27</td>
<td>1.76</td>
</tr>
<tr>
<td>HardSAS, 2 wl</td>
<td>1.28</td>
<td>0.72</td>
<td>1.33</td>
<td>1.85</td>
</tr>
<tr>
<td>SIMD SAS, 1 wl</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Hardmult, 3 wl</td>
<td>7.84</td>
<td>2.05</td>
<td>2.00</td>
<td>0.96</td>
</tr>
<tr>
<td>SIMD mult, 1 wl</td>
<td>6.63</td>
<td>2.00</td>
<td>2.29</td>
<td>1.15</td>
</tr>
</tbody>
</table>
STATE OF THE ART

Mapping solutions:

1. Instantiations (SIMD in software)
   - [Lambrechts et al, 2007]
   - [Novo et al, 2010b]
     pointed to specific implementations

2. Vector compiler literature
   - [Larsen et al, 2000]
   - [Krall et al, 2000]
   - [Tenllado et al, 2005]
   - [Xue et al, 2007]
     - data organization in memory and application code
     - minimal number of sub-word lengths

Differentiating elements:

- Either focused in software or in hardware-based SIMD, not both
- Do not exploit broad search space with hybrids
- No systematic selection process
CONCLUSIONS

• Solution space of organizing parallel data with minimal word-length requirements in domain-specific processors
• Systematic way can save design time, prohibit suboptimal choices, reveal hybrids
• Wide range of mapping options, wide range of quantitative results
Thank you for your attention!
BACK-UP SLIDES
<table>
<thead>
<tr>
<th>Cycles</th>
<th>VWR</th>
<th>R1</th>
<th>R4</th>
<th>SA unit</th>
<th>Shuffler unit</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rd</td>
<td>Wr</td>
<td></td>
<td>Sh</td>
<td>Add</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+/&gt; &gt;&gt; +/&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>M0_12_a &gt;&gt; &amp; +</td>
<td>M0_12_c</td>
<td>M7_12_a = SADD (M0_12_a, M0_12_a)</td>
<td>&gt;&gt; 4</td>
<td>-</td>
<td>M0_12_c = REPACK (M0_8_b)</td>
<td>(4)</td>
</tr>
<tr>
<td>15</td>
<td>M0_12_b &gt;&gt; &amp; +</td>
<td></td>
<td>M7_16_a</td>
<td>M7_12_b = SADD (M0_12_b, M0_12_b)</td>
<td>&gt;&gt; 4</td>
<td>-</td>
<td>M7_16_a = REPACK (M7_12_a)</td>
</tr>
<tr>
<td>16</td>
<td>M0_16_a + &amp; R1</td>
<td>M0_16_a</td>
<td>M7_16_a &gt;&gt;/ M7_16_b mxd</td>
<td>M7_16_a = SADD (M7_16_a, M0_16_a)</td>
<td>&gt;&gt; 3</td>
<td>+</td>
<td>M7_16_b mxd = REPACK (M7_12_a, M7_12_b)</td>
</tr>
<tr>
<td>17</td>
<td>M0_16_b mxd</td>
<td>R1</td>
<td>M7_16_b mxd &gt;&gt;/ M7_16_a</td>
<td>M7_16_b mxd = SADD (M7_16_b mxd, M0_16_b mxd)</td>
<td>&gt;&gt; 3</td>
<td>+</td>
<td>M7_16_a &gt;&gt; 2</td>
</tr>
<tr>
<td>18</td>
<td>M7_16_a</td>
<td>M0_16_a +/&gt;</td>
<td>M7_16_a &gt;&gt;/ M7_16_b mxd</td>
<td>M7_16_a = SADD (M7_16_a, M0_16_a)</td>
<td>&gt;&gt; 0</td>
<td>+</td>
<td>M7_16_b mxd &gt;&gt; 2</td>
</tr>
</tbody>
</table>

© imec 2012
NEED FOR UNROLLING THE LOOP
HYBRID SOLUTIONS

- **Shifts**
  - e.g. M0 >> 4

  \[
  \begin{align*}
  M0 &= 1101100100000 \\
  M0 >> 4 &= 0000110110010
  \end{align*}
  \]

  Prevent moving into right-hand subword

- **Additions**
  - e.g. M0 + M0

  \[
  \begin{align*}
  M0 &= 0 110110010 \\
  M0 &= 0 110110010 \\
  M0 + M0 &= 1101100100
  \end{align*}
  \]

  Move into left-hand subword