AN EMPIRICAL STUDY OF SOME X86 SIMD INTEGER EXTENSIONS

Isabelle Hurbain and Georges-André Silber
École des mines de Paris
Centre de recherche en informatique
CPC’06 - January, 9th 2006
Motivations

Overview of SSE2 instructions

Benchmarking SSE2 instructions

Guidelines for optimization

Conclusion
Motivations

- What speedup is attainable with integer SSE2 instructions?
- Measure the best-case impact of 32 SSE2 integer instructions in terms of performance
- Study the code generation for those instructions in state-of-the-art compilers
SSE2 EXTENSION: AN OVERVIEW

- Processor instructions introduced to improve multimedia applications and floating-point operations
- 8 new 128-bit multi-purpose registers
- Integer, floating point, memory instructions
- Similar extensions: Altivec (PowerPC), VIS (Sparc), 3DNow! (Athlon)
Integer Operations

- Basic operations (+, -, *)
- Basic operations with saturation
- Comparison operations (min, max, >, =)
- Miscellaneous operations
Programming with SSE2 instructions

- Writing assembly code
- Using “intrinsics”
- Using the compiler to semi-automatically vectorize the code
- Letting the compiler automatically vectorize the code
**EXAMPLE: PADDDB**

```
for (i=0; i<16;i++){
    c[i] = a[i] + b[i]
}
```

```c
*c = _mm_add_epi8(*(__m128i *) a, *(__m128i *) b);
```

```assembly
movl    12(%ebp), %eax
movdqa  (%eax), %xmm0
movl    8(%ebp), %eax
paddb   (%eax), %xmm0
movl    16(%ebp), %eax
movdqa  %xmm0, (%eax)
```
10 executions of 30,000,000 calls
for (i = 0; i<16; i++) {
    c[i] = a[i] + b[i];
}

- GCC 4.1: 6.79
- GCC 4.1 autovect: 4.16
- GCC 4.1 intrinsics: 1.08
- ICC 9.0: 8.09
- ICC 9.0 autovect: 2.33
- ICC 9.0 intrinsics: 2.23

Speedup: 6.3

Speedup: 3.5
for(i = 0; i<8; i++)
    c[i] = 0;
for(i = 0; i<16; i++)
    tmparray[i] = abs(a[i] - b[i])
for(i=0; i<8; i++){
    c[0] += tmparray[i];
    c[4] += tmparray[i+8];
}
PSADBW: A CASE THAT IS NOT DETECTED

- GCC 4.1: 21.17
- GCC 4.1 autovect: 22.55
- ICC 9.0 intrinsics: 0.99
- ICC 9.0: 14.93
- ICC 9.0 autovect: 18.37
- ICC 9.0 intrinsics: 1.84

Speedup: 21.50
Speedup: 8.13
RESULTS

- Number of detected vectorizable code:
  - ICC : 26 / 32
  - GCC : 8 / 32

- Quality of the semi-automatically vectorized code: ICC is a clear winner
## RESULTS

<table>
<thead>
<tr>
<th>Operation</th>
<th>Base (s)</th>
<th>Auto vect</th>
<th>Speedup</th>
<th>Status</th>
<th>Intrinsics</th>
<th>Speedup</th>
<th>Base (s)</th>
<th>Auto vect</th>
<th>Speedup</th>
<th>Status</th>
<th>Intrinsics</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>paddb</td>
<td>6.79</td>
<td>4.16</td>
<td>1.63</td>
<td>✔️</td>
<td>1.08</td>
<td>6.26</td>
<td>8.09</td>
<td>2.33</td>
<td>3.48</td>
<td>✔️</td>
<td>2.29</td>
<td>3.53</td>
</tr>
<tr>
<td>psubb</td>
<td>7.25</td>
<td>3.83</td>
<td>1.89</td>
<td>✔️</td>
<td>1.09</td>
<td>6.68</td>
<td>7.70</td>
<td>2.34</td>
<td>3.28</td>
<td>✔️</td>
<td>2.14</td>
<td>3.60</td>
</tr>
<tr>
<td>paddsb</td>
<td>14.56</td>
<td>14.87</td>
<td>0.98</td>
<td>✗</td>
<td>1.08</td>
<td>13.79</td>
<td>15.06</td>
<td>2.29</td>
<td>6.29</td>
<td>✔️</td>
<td>2.01</td>
<td>7.50</td>
</tr>
<tr>
<td>psubsb</td>
<td>14.55</td>
<td>13.79</td>
<td>1.06</td>
<td>✗</td>
<td>0.96</td>
<td>15.21</td>
<td>14.87</td>
<td>1.85</td>
<td>8.06</td>
<td>✔️</td>
<td>1.78</td>
<td>8.35</td>
</tr>
<tr>
<td>pcmpgtb</td>
<td>17.61</td>
<td>14.26</td>
<td>1.23</td>
<td>✗</td>
<td>0.87</td>
<td>20.31</td>
<td>13.05</td>
<td>1.82</td>
<td>7.16</td>
<td>✔️</td>
<td>1.93</td>
<td>6.75</td>
</tr>
<tr>
<td>pminub</td>
<td>10.60</td>
<td>12.11</td>
<td>0.88</td>
<td>✗</td>
<td>1.02</td>
<td>10.41</td>
<td>10.38</td>
<td>11.66</td>
<td>0.89</td>
<td>✗</td>
<td>1.82</td>
<td>5.71</td>
</tr>
<tr>
<td>pavgb</td>
<td>9.95</td>
<td>9.79</td>
<td>1.02</td>
<td>✗</td>
<td>1.33</td>
<td>7.50</td>
<td>10.98</td>
<td>1.78</td>
<td>6.18</td>
<td>✔️</td>
<td>1.82</td>
<td>6.04</td>
</tr>
<tr>
<td>pavgw</td>
<td>6.18</td>
<td>5.88</td>
<td>1.05</td>
<td>✗</td>
<td>0.93</td>
<td>6.62</td>
<td>5.48</td>
<td>2.08</td>
<td>2.63</td>
<td>✔️</td>
<td>2.11</td>
<td>2.59</td>
</tr>
<tr>
<td>psadbw</td>
<td>21.17</td>
<td>22.54</td>
<td>0.94</td>
<td>✗</td>
<td>0.99</td>
<td>21.50</td>
<td>14.93</td>
<td>18.37</td>
<td>0.81</td>
<td>✗</td>
<td>1.84</td>
<td>8.13</td>
</tr>
<tr>
<td>pmaddwd</td>
<td>5.00</td>
<td>5.39</td>
<td>0.93</td>
<td>✗</td>
<td>1.22</td>
<td>4.10</td>
<td>6.39</td>
<td>6.39</td>
<td>0.75</td>
<td>✗</td>
<td>2.07</td>
<td>2.31</td>
</tr>
</tbody>
</table>
• When possible, use intrinsics

• Direct assembly code is a bit faster but much more error-prone

• Autovectorization: the compiler needs to be “helped”
CONCLUSION

- As of today, automatic use of SSE2 instructions by compilers does not work
  - needs interprocedural informations about alignment, overlaps...
- Source to source transformation of the code (producing intrinsics) is a good approach
- Created a set of benchmarks that could be used as testcases for compilers
QUESTIONS