Vector SIMD Instructions

Contents
- basic instructions 2-5
- useful building blocks
  - load/store 6-7
    (gather/scatter)
  - reordering 8-9
  - matrix-vector product 10

Information

vector indexing: \[ a = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} \]
in most intrinsics, the order of operands matter

memory:

SIMD extensions timeline: SSE1, SSE2, SSE3, SSSE3, SSE4

We focus on single precision float, 4-way
1 vector = 128 bits = 16 B, data type -- m128

Unless stated otherwise, instructions are SSE or later
Load and Store

\[ a = \text{mm-load-ps}(m); \quad \text{aligned} \]
\[ a = \text{mm-loadu-ps}(m); \quad \text{unaligned} \quad \text{(avoid)} \]
\[ a = \text{pći}; \quad \text{if } p \text{ is } \quad \text{-m128 +p} \]

\[ m, 88 \text{ aligned} \]

\[ a = \text{mm-loadl-pi}(a, m); \quad \text{(keeps upper half)} \]
\[ b = \text{mm-loadh-pi}(b, m); \quad \text{(keeps lower half)} \]

\[ m, 46 \text{ bit aligned} \]

\[ a = \text{mm-load-ss}(m) \]

\[ \text{set to zero} \]

\[ s\text{tores are analogous} \]

Constants

\[ c: \begin{bmatrix} 4.0 & 10.2 & 0.1 & 0.0 \end{bmatrix} \]
\[ c = \text{mm-set-ps}(4.0, 3.0, 2.0, 1.0); \]
\[ \alpha: \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix} \]
\[ \alpha = \text{mm-setl-ps}(1.0); \]
\[ \varepsilon: \begin{bmatrix} 0.0 & 0.0 & 1.0 \end{bmatrix} \]
\[ \varepsilon = \text{mm-set-ss}(1.0); \]
\[ f: \begin{bmatrix} 0.0 & 0.0 & 0.0 \end{bmatrix} \]
\[ f = \text{mm-setzero-ps}(1); \]
Vector arithmetic

\[ \begin{align*}
\text{c} &= \text{mm_addl_ps}(a, b); \quad "a+b" \\
\text{analogous:} \\
\text{c} &= \text{mm_subl_ps}(a, b); \quad "a-b" \\
\text{c} &= \text{mm_mul_l_ps}(a, b); \quad "a\cdot b"
\end{align*} \]

Scalar arithmetic

\[ \begin{align*}
\text{c} &= \text{mm_addlss}(a, b); \\
\text{AddSub (SSSE3 and later)}
\end{align*} \]
Reorder Instructions

```
shuffle:

a   b   c

align: (SSE3 and later)

a   b   c

shuffle: (SSE3 and later)

a   b   c

blend: (SSE4.0 and later)

a   b   c

c = mm_unpcklo-ps(a, b)
c = mm_unpckhi-ps(a, b)
c = mm_shuffle-ps(a, b, mm_shuffle_epi8(c, d, e))

c0 = a.i

c1 = b.j

c2 = d.k

c3 = e.l

"any 4 consecutive elements of the concatenation of a and b go into c"

-mm-alignmask-epi8  use with
-mm-castps128-ps

"c is filled in each position with an element from a or b from the same position"

-mm-blend-ps
```
Dot product (SSE4 and later)

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
</table>
```

(example)

"computes the pointwise product of a and b and writes an arbitrary sum of the resulting numbers into selected elements of c — the others are set to zero"

`mm_dp_ps(a, b, mask)`
Load 4 real numbers from arbitrary memory locations

7 instructions, this is the right way

Note:
- whenever possible avoid this by restructuring the algorithm or data to have aligned vector loads _mm_load_ps
- the above should be equivalent to the following but a) the above is safer; b) be aware that the below are 7 instructions

```c
float f[20] = [...] ;
m128 vf = _mm_set_ps(f[0], f[5], f[10], f[15]);
```

Don't do this:

```c
float f[20] = [...] ;
decspecl align(16) g[4] ;
m128 vf:
g[0] = f[0];
g[1] = f[4];
g[2] = f[8];
g[3] = f[12];
```

(mem -> register -> mem round trip) ➞ expensive

(same problem with unions and pointers)
Store 4 real numbers to arbitrary memory locations

7 instructions, shorter critical path than load

Load 4 complex numbers (4 pairs of real numbers)

6 instructions
store analogous

same with consecutive data:

4 instructions
store analogous
Shift by 1

2 instructions
SSE3 and later: \(-mm\_shuffle\_ps\ + \text{casts} \)

Reverse vectors

Interleaved complex \(\rightarrow\) split complex

Split complex \(\rightarrow\) interleaved complex
Transposition: 4x4 matrix

4x4 matrix:
\[
\begin{bmatrix}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 \\
12 & 13 & 14 & 15 \\
\end{bmatrix} = A
\]

in memory:
4 aligned loads
4 shuffles
4 shuffles
4 aligned stores
in memory:
\[
\begin{bmatrix}
15 & 14 & 13 & 12 \\
10 & 9 & 8 & 7 \\
6 & 5 & 4 & 3 \\
2 & 1 & 0 & 1 \\
\end{bmatrix}
\]

as matrix:
\[
\begin{bmatrix}
0 & 4 & 8 & 12 \\
1 & 5 & 9 & 13 \\
2 & 6 & 10 & 14 \\
3 & 7 & 11 & 15 \\
\end{bmatrix} = A^T
\]

\(\otimes\) done by the macro \(-\text{MM\_TRANSPOSE\_4\_PS}\(\ a, b, c, d\);\)

8 instructions
Matrix-vector product

\[
\begin{pmatrix}
  a & b \\
  c & d
\end{pmatrix}
\begin{pmatrix}
  x
\end{pmatrix}
\]

1. step: 4 vector products ax, bx, cx, dx (4 instructions)

\[
\begin{array}{cccc}
  a & b & c & d \\
  x & & & \\
\end{array}
\]

result:

\[
\begin{array}{cccc}
  a & b & c & d \\
  x & x & x & x \\
\end{array}
\]

SSE:

2. step: transpose (8 instructions)

result:

SSE2:

3. step: sum rows (3 instructions)

Total: 15 instructions

SSE3:

2. step: tree reduction

\[
\begin{array}{cccc}
  a & b & c & d \\
  & & & \\
\end{array}
\]

7 instructions

SSE4: has dot product instruction, but still 7 instructions are needed (exercise)