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: \[ a: \begin{array}{cccc}
3 & 2 & 1 & 0 \\
\end{array} \]
in most intrinsics, the order of operands matters
memory:

\[ \begin{array}{cccccccc}
\uparrow & & & & & & & \\
\text{address} & & & & & & & \\
\text{address increases} & & & & & & & \\
\end{array} \]

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

We focus on single precision, float, 4-way
1 vector = 128 bit = 16 bytes, data type \(-m128\)
Unless stated otherwise, instructions are SSE or later
Load and Store

\[ a \downarrow \downarrow \downarrow \downarrow \\]
\[ \text{aligned} \]
\[ a = \text{mm-local-ps}(m) \]
\[ \text{unaligned (avoid)} \]
\[ a = \text{pi}(j); \text{if } p \text{ is: } -m128 \times p \]
\[ \text{m, 88 aligned} \]
\[ a \]
\[ \text{aligned} \]
\[ a = \text{mm-load-l-pi}(a, m) \]
\[ \text{keeps upper half} \]
\[ b \]
\[ \text{aligned} \]
\[ b = \text{mm-load-b-pi}(b, m) \]
\[ \text{keeps lower half} \]
\[ \text{m, 46 bit aligned} \]
\[ a \]
\[ \text{set to zero} \]
\[ a = \text{mm-load-ss}(m) \]

Stores are analogous

Constants

\[ e: \begin{bmatrix} 4.0 & 1.0 & 2.0 & 1.0 \end{bmatrix} \]
\[ c = \text{mm-set-ps}(4.0, 1.0, 2.0, 1.0); \]
\[ d: \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix} \]
\[ o = \text{mm-set-l-ps}(1.0); \]
\[ e: \begin{bmatrix} 0.0 & 0.0 & 0.0 \end{bmatrix} \]
\[ r = \text{mm-set-ss}(1.0); \]
\[ f: \begin{bmatrix} 0.0 & 0.0 & 0.0 \end{bmatrix} \]
\[ h = \text{mm-set-zero-ps}(1.0); \]
Vector arithmetic

\[ c = \text{\_mm\_add\_ps}(a, b); \quad \text{"a + b"} \]

*analogous:*

\[ c = \text{\_mm\_sub\_ps}(a, b); \quad \text{"a - b"} \]

\[ c = \text{\_mm\_mul\_ps}(a, b); \quad \text{"a \cdot b"} \]

Scalar arithmetic

\[ c = \text{\_mm\_add\_ss}(a, b); \]

AddSub (SSE3 and later)
Reorder Instructions

shuffle:

align: (SSE3 and later)

shuffle: (SSE3 and later)

blend: (SSE4.0 and later)
Dot product (SSE4 and later)

"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"

\[ \text{mm_dp_vs}(a, b, \text{mask}) \]
Load 4 real numbers from arbitrary memory locations

f

m

n

floats

_mm_load_ss

_mm_shuffle_ps

_mm_shuffle_ps

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

float f[20] = [. . . 3];
_mm128 vf = _mm_set_ps(f[23], f[53], f[13], f[13]);

Don't do this:

float f[20] = [. . . 3];
__declspec(aligned(16)) g[4];
_mm128 vf;

\[
g[0] = f[13];
g[1] = f[5];
g[2] = f[53];
g[3] = f[13];
\]

{ mem \to \text{register} \to \text{mem \ rouding} \Rightarrow \text{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

\[ a \{3, 6, 5, 4 \} \quad b \{3, 2, 1, 0 \} \]

- \_mm\_shuffle\_ps
- \_mm\_shuffle\_ps

2 instructions

SSE3 and later: \_mm\_align\_epi\_8 + casts Instruction

Reverse vector

\[ a \{1, 1, 1, 1 \} \quad b = \_mm\_shuffle\_ps(9, 9, \_mm\_shuffle(0, 1, 2, 3)) \]

Interleaved complex \rightarrow split complex

\[ a \{4, 1, 1, 4 \} \quad b \{4, 1, 1, 4 \} \quad c = \_mm\_shuffle\_ps(6, 8, \_mm\_shuffle(3, 1, 3, 1)) \];
\[ d = \ldots \ldots \ldots \ldots \ldots \ldots \ldots \_mm\_shuffle(2, 0, 2, 0) \];

Split complex \rightarrow interleaved complex

\[ a \{4, 1, 1, 4 \} \quad b \{4, 1, 1, 4 \} \quad c = \_mm\_unpackhi\_ps(6, 8) \];
\[ d = \_mm\_unpacklo\_ps(6, 8) \]
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:

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

\( \otimes \) done by the macro -NTRANPOSE4-PS (a,b,c,d);
8 instructions
Matrix-vector product

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

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

   result: \[
   \begin{bmatrix}
     ax \\
     bx \\
     cx \\
     dx
   \end{bmatrix}
   \]

SSE:

2. step: transpose (8 instructions)

   result: \[
   \begin{bmatrix}
     ax \\
     bx \\
     cx \\
     dx
   \end{bmatrix}
   \]

3. step: sum rows (3 instructions)

   total: 15 instructions

SSE2:

2. step: tree reduction

   total: 7 instructions

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