## SIMD Example

Let's take the following function:

```
float Magnitude(float32x4_t & v) {
    return sqrtf(v[0] * v[0] + v[1] * v[1] + v[2] * v[2] + v[3] * v[3]);
}
```

This is, of course, Euclidean distance in 4D. The above source code includes 4 multiplies and 3 additions plus the sqrt. Add to this complexity, the number of loads needed to get the four components into registers.

#### No SIMD

The above function might look like this in hand-written ASM:

#### Magnitude:

```
ldr
        s0, [x0, 0]
ldr
        s1, [x0, 4]
        s2, [x0, 8]
ldr
        s3, [x0, 12]
ldr
fmul
        s0, s0, s0
fmul
        s1, s1, s1
fmul
        s2, s2, s2
fmul
        s3, s3, s3
fadd
        s0, s0, s1
fadd
        s0, s0, s2
fadd
        s0, s0, s3
        s0, s0
fsqrt
ret
```

All told this is:

- 4 references to memory for data (ignoring caching)
- 12 references to memory for instructions (ignoring caching)
- 4 multiplies
- 3 adds
- 1 sqrt

### Using SIMD

If you tweak this by hand you'll get:

#### MagnitudeV:

```
ldr q0, [x0] // 1
fmulx v0.4s, v0.4s, v0.4s // 2
faddp v0.4s, v0.4s, v0.4s // 3
faddp v0.4s, v0.4s, v0.4s // 4
```

```
fsqrt s0, s0 // 5 ret
```

#### Assumed input

For this example, let us assume the input is:

```
float32x4_t v = \{1, 2, 3, 4\};
```

#### Line 1

This instruction dereferences the *reference* to v which is a float32x4\_t (i.e. float a[4]). After this instruction, q0 contains all four floats from v.

```
q0 now contains {1, 2, 3, 4}.
```

#### Line 2

This instruction multiplies the four floats in the third operand by the four floats in the second operand and places the results in the first operand.

```
q0 now contains {1, 4, 9, 16}.
```

#### Line 3

The goal now is to add up all the v's to send to sqrtf().

This instruction is a pairwise addition. It does the following:

```
v[0] = v[2] = v[0] + v[1];

v[1] = v[3] = v[2] + v[3];

q0 now contains {5, 25, 5, 25}.
```

#### Line 4

The v's have been summed up in pairs, now call the same instruction again to complete the summation.

```
v[0] = v[2] = v[0] + v[1];

v[1] = v[3] = v[2] + v[3];

q0 now contains {30, 30, 30, 30}.
```

#### Line 5

This instruction takes the square root of the least significant single precision float (they're all the same value).

s0 contains the answer of: 5.47723.

# Summary

For many many algorithms, a sprinkling of SIMD instructions can provide a speed up, possibly a large one.