Introduction to SIMD

Thomas Schaub
Native Instruments

20. October 2015

What is SIMD?

Scalar/SISD

$a = b + c$

Vector/SIMD

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} + \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

- x86: SSE/AVX
- ARM: Neon

Why?

https://smoothspan.wordpress.com/2007/09/06/a-picture-of-the-multicore-crisis/

Intrinsics

Expectation

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} + \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

Reality

```
__mm128 b, c;
auto a = _mm_add_ps(b, c);
```

Intrinsics in Action

```
-> __m128 b = _mm_set_ps(4, 3, 2, 1);
-> __m128 c = _mm_set_ps(10, 20, 30, 40);
-> __m128 a = _mm_add_ps(b, c);
->
```

```
b = { ?,  ?,  ?,  ?}
c = { ?,  ?,  ?,  ?}
a = { ?,  ?,  ?,  ?}
```

Example: Matrix · Vector

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ m_{1,0} & m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,0} & m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

$$u_1 = m_{1,0} v_0 + m_{1,1} v_1 + m_{1,2} v_2 + m_{1,3} v_3$$

$$\begin{aligned} u_i &= m_{i,0} v_0 + m_{i,1} v_1 + m_{i,2} v_2 + m_{i,3} v_3 \\ &= \sum_j m_{i,j} v_j \end{aligned}$$

Matrix · Vector (Scalar)

```
void mul(float* m, float* v, float* u)
{
  for (auto i = 0; i < 4; ++i)
  {
    float a = 0;
    for (auto j = 0; j < 4; ++j)
    {
      a += m[4*i + j] * v[j];
    }
    u[i] = a;
  }
}
```

Three easy Steps

- **#include** <xmmintrin.h>
- https://software.intel.com/sites/landingpage/IntrinsicsGuide/
- Profit

How to NOT use SIMD

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ m_{1,0} & m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,0} & m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

```
{m_1_0, m_1_1, ...} * {v_0, v_1, ...} = {m_1_0*v_0, m_1_1*v_1, ...}
```

```
void mul(float* m, float* v, float* u) {
  for (i = 0; i < 4; ++i) {
    auto row = _mm_load_ps(m + 4*i);
    auto col = _mm_load_ps(v);
    auto t0 = _mm_mul_ps(row, col);
    // quick reduction, then lunch auto t1 = _mm_shuffle_ps(t0, t0, 0x44);
    auto t2 = _mm_shuffle_ps(t0, t0, 0xbb);
    auto t3 = _mm_add_ps(t1, t2);
    auto t4 = _mm_shuffle_ps(t3, t3, 0xb1);
    auto t5 = _mm_add_ps(t3, t4);
    u[i] = _mm_cvtss_f32(t5);
} }
```

Single Program, Multiple Data

$$\begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} \begin{pmatrix} v_0' \\ v_1' \\ v_2' \\ v_3' \end{pmatrix} \begin{pmatrix} v_0'' \\ v_1'' \\ v_2'' \\ v_3'' \end{pmatrix} \begin{pmatrix} v_0''' \\ v_1''' \\ v_2''' \\ v_3''' \end{pmatrix} \cdots \qquad \Rightarrow \qquad \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{pmatrix} \begin{pmatrix} u_0' \\ u_1' \\ u_2' \\ u_3' \end{pmatrix} \begin{pmatrix} u_0'' \\ u_1'' \\ u_2'' \\ u_3'' \end{pmatrix} \begin{pmatrix} u_0''' \\ u_1''' \\ u_2''' \\ u_3''' \end{pmatrix} \cdots$$

SPMD Example (Scalar)

```
void mul(float* m, float* v, float* u)
{
  for (k = 0; k < n; ++k)
  {
    for (i = 0; i < 4; ++i)
    {
      auto a = 0;
      for (j = 0; j < 4; ++j)
      {
        a += m[4*i + j] * v[4*k + j];
      }
      u[4*k + i] = a;
    }
  }
}
```

Scalar Instances

$$k = 0 k = 1 k = 2 k = 3 k = 4 k = 5 k = 6 k = 7 k = \ldots$$

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{pmatrix} \begin{pmatrix} u_0' \\ u_1' \\ u_2' \\ u_3' \end{pmatrix} \begin{pmatrix} u_0'' \\ u_1'' \\ u_2'' \\ u_3'' \end{pmatrix} \begin{pmatrix} u_0''' \\ u_1''' \\ u_2''' \\ u_3''' \end{pmatrix} \begin{pmatrix} u_0'''' \\ u_1'''' \\ u_2'''' \\ u_3'''' \end{pmatrix} \begin{pmatrix} u_0''''' \\ u_1''''' \\ u_2''''' \\ u_3''''' \end{pmatrix} \begin{pmatrix} u_0'''''' \\ u_1'''''' \\ u_2'''''' \\ u_3'''''' \end{pmatrix} \begin{pmatrix} u_0''''''' \\ u_1''''''' \\ u_2''''''' \\ u_3''''''' \end{pmatrix} \ldots$$

SIMD Instances

$$k = (0, 1, 2, 3) k = (4, 5, 6, 7) k = \ldots$$

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{pmatrix} \begin{pmatrix} u_0' \\ u_1' \\ u_2' \\ u_3' \end{pmatrix} \begin{pmatrix} u_0'' \\ u_1'' \\ u_2'' \\ u_3'' \end{pmatrix} \begin{pmatrix} u_0''' \\ u_1''' \\ u_2''' \\ u_3''' \end{pmatrix} \begin{pmatrix} u_0'''' \\ u_1'''' \\ u_2'''' \\ u_3'''' \end{pmatrix} \begin{pmatrix} u_0''''' \\ u_1''''' \\ u_2''''' \\ u_3''''' \end{pmatrix} \begin{pmatrix} u_0'''''' \\ u_1'''''' \\ u_2'''''' \\ u_3'''''' \end{pmatrix} \begin{pmatrix} u_0''''''' \\ u_1''''''' \\ u_2''''''' \\ u_3''''''' \end{pmatrix} \ldots$$

SPMD: Scalar $\rightsquigarrow$ SIMD

$$
\begin{array}{ccc}
a \cdot (v_0, v_1, v_2, v_3) + b & \rightsquigarrow & (av_0 + b, av_1 + b, av_2 + b, av_3 + b) \\
m[(v_0, v_1, v_2, v_3)] & \rightsquigarrow & (m[v_0], m[v_1], m[v_2], m[v_3])
\end{array}
$$

SPMD Example (Scalar $\rightsquigarrow$ SIMD)

```
void mul(float* m, float* v, float* u)
{
  for (auto k = 0; k < n; ++k)
  {
    for (auto i = 0; i < 4; ++i)
    {
      auto a = 0;
      for (auto j = 0; j < 4; ++j)
      {
        a += m[4*i + j] * v[4*k + j];
      }
      u[4*k + i] = a;
    }
  }
}
```

Memory Layout

$$\begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} \begin{pmatrix} v_0' \\ v_1' \\ v_2' \\ v_3' \end{pmatrix} \begin{pmatrix} v_0'' \\ v_1'' \\ v_2'' \\ v_3'' \end{pmatrix} \begin{pmatrix} v_0''' \\ v_1''' \\ v_2''' \\ v_3''' \end{pmatrix} \cdots$$

0x00	v_0	v_1	v_2	v_3	v_0'	v_1'	v_2'	v_3'
0x20	v_0''	v_1''	v_2''	v_3''	v_0'''	v_1'''	v_2'''	v_3'''
0x40	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Hybrid Struct of Arrays

$$\begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} \begin{pmatrix} v_0' \\ v_1' \\ v_2' \\ v_3' \end{pmatrix} \begin{pmatrix} v_0'' \\ v_1'' \\ v_2'' \\ v_3'' \end{pmatrix} \begin{pmatrix} v_0''' \\ v_1''' \\ v_2''' \\ v_3''' \end{pmatrix} \cdots$$

0x00	v_0	v_0'	v_0''	v_0'''	v_1	v_1'	v_1''	v_1''
0x20	v_2	v_2'	v_2''	v_2''	v_3	v_3'	v_3''	v_3''
0x40	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

SPMD + SIMD w/ actual Intrinsics

```
void mul(float* m, float* v, float* u)
{
  for (auto k = 0; k < n; k += 4)
  {
    for (auto i = 0; i < 4; ++i)
    {
      auto a = _mm_set_ps1(0);
      for (auto j = 0; j < 4; ++j)
      {
        auto tempM = _mm_set_ps1(m[4*i + j]);
        auto tempX = _mm_load_ps(v + 4*k + 4*j);
        a = _mm_add_ps(a, _mm_mul_ps(tempM, tempX));
      }
      _mm_store_ps(u + 4*k + 4*i, a);
    }
  }
}
```

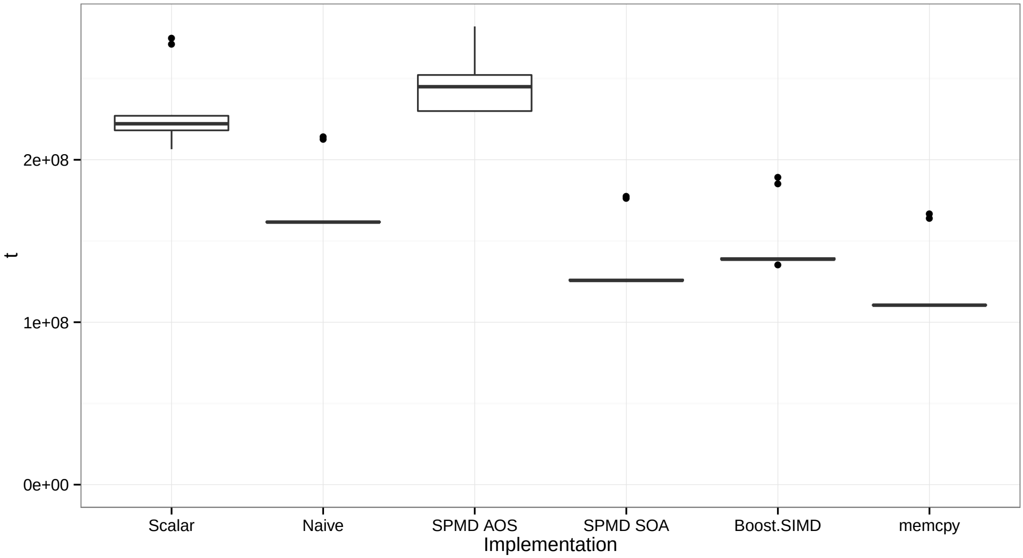
Easy, but ...

- Verbose
- AVX?
- ARM?
- Scalar version required for $4 \nmid n$
- _mm_load_ps: "mem_addr must be aligned on a 16-byte boundary"

Boost.SIMD

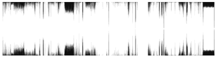
```
template <unsigned vecWidth>
void mul_boost(float* matrix, float* xs, float* ys)
{
  using pack = boost::simd::pack<float, vecWidth>;

  for (unsigned k = 0; k < n; k += vecWidth)
  {
    for (unsigned i = 0; i < 4; ++i)
    {
      pack a = pack(0);
      for (unsigned j = 0; j < 4; ++j)
      {
        pack tempM = pack(matrix[4*i + j]);
        pack tempX = boost::simd::aligned_load<pack>(xs + 4*k + 4*j);
        a = a + tempM * tempX;
      }
      boost::simd::aligned_store<pack>(a, ys + 4*k + 4*i);
    }
  }
}
```

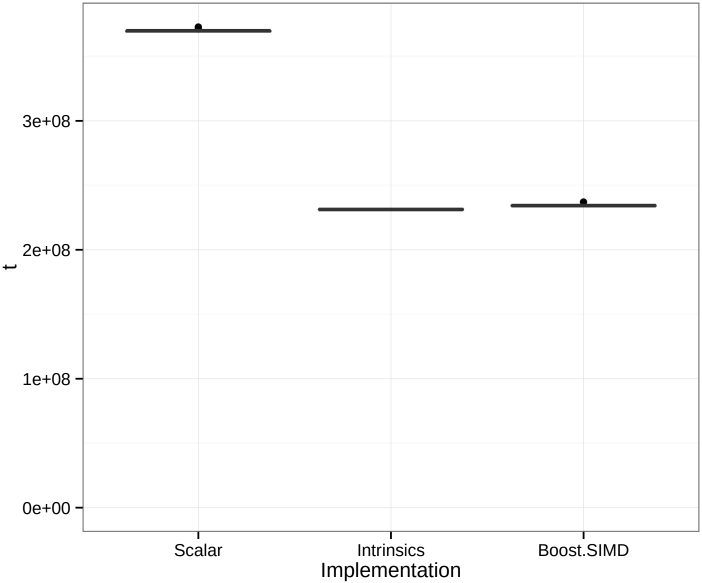
Benchmark: Matrix · Vector



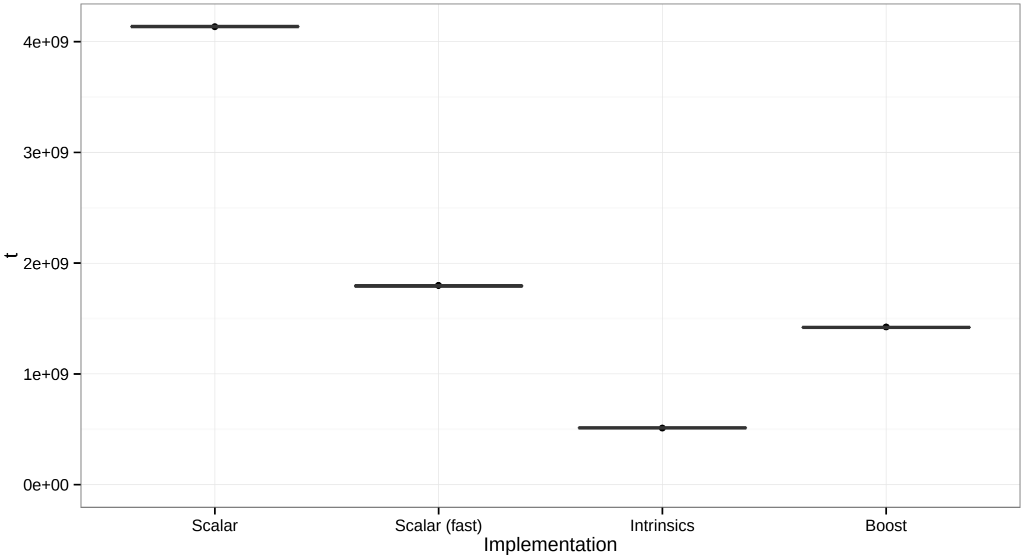
More Benchmarks: Waveform

Output:

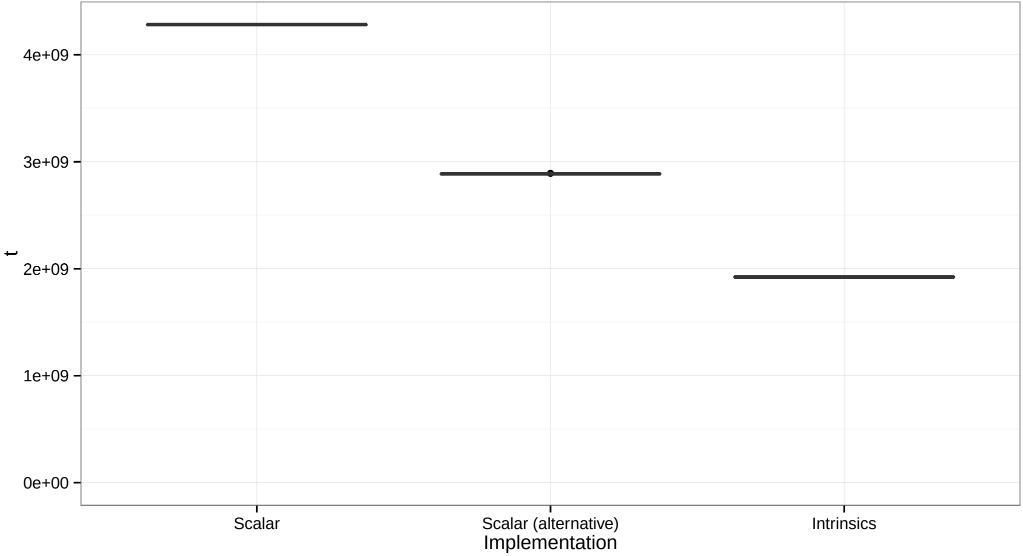




Benchmarks: Oscilators



Benchmarks: Convolution



SPMD Control Flow (Scalar)

```
for (auto k = 0; k < n; ++k)
{
  if (x[k] > 0)
    y[k] = 123;
  else
    y[k] = 42;
}
```

SPMD Control Flow (SIMD attempt)

```
for (auto k = 0; k < n; ++k)
{
  if (load(x, k) > pack(0) /* ??? */)
    store(y, k, ???);
  else
    store(y, k, ???);
}
```

Linearized SPMD Control Flow

```
for (auto k = 0; k < n; ++k)
{
  auto mask = x[k] > 0;
  auto y_true = 123;
  auto y_false = 42;
  y[k] = (mask & y_true) | (~mask & y_false)
}
```

Blend

```
(mask & y_true) | (~mask & y_false)
```

```
(0xFF & 0x7B) | (~0xFF & 0x2A)
(0xFF & 0x7B) | ( 0x00 & 0x2A)
(0xFF & 0x7B) |   0x00
(0xFF & 0x7B)
        0x7B
```

```
(0x00 & 0x7B) | (~0x00 & 0x2A)
 0x00         | (~0x00 & 0x2A)
                (~0x00 & 0x2A)
                ( 0xFF & 0x2A)
                         0x2A
```

So it's 2015...

- ISPC
  - only SSE/AVX
- OpenCL
  - Performance Lottery
- Rust
  - Unstable

Conclusion and Advice

- Speedups vary from 60% (Matrix, Waveform) to 250% (Osc)
- Not always applicable

- Prefer SPMD
- Think about memory layout early
- Avoid control flow
- ...or come up with custom solutions
- Use Boost.SIMD

Questions?

Follow up

- Handmade Hero Day 115 - SIMD Basics: `https://www.youtube.com/watch?v=YnnTb0AQgYM`
- Performance Optimization, SIMD and Cache: `https://www.youtube.com/watch?v=Nsf2_Au6KxU`
- SIMD at Insomniac Games: `https://deplinenoise.wordpress.com/2015/03/06/slides-simd-at-insomniac-games-gdc-2015/`
- Whole-Function Vectorization: `http://www.intel-vci.uni-saarland.de/uploads/tx_sibibtex/10_01.pdf`
- Data-Oriented Design: `http://www.dataorienteddesign.com/dodmain/`

References

- Intrinsics Reference:
  `https://software.intel.com/sites/landingpage/IntrinsicsGuide/`
- Boost.SIMD:
  `http://nt2.metascale.fr/doc/html/the_boost_simd_library.html`
- ISPC: `https://ispc.github.io/`

Boost.SIMD Evaluation

- Pain points:
  - Documentation
  - Uneven number of instances
  - Control flow
  - Compile Time
  - Error Messages
- Good:
  - Availability
  - Performance
- Unknown:
  - Debugability

SIMD vs. Multi Core

- Orthogonal
- SIMD: One consecutive chunk of memory
- Multi Core: Split memory into chunks

DSP and SIMD