How To SIMD

A Light Introduction To SIMD Intrinsics

whoami

Philippe Michael Groarke

Autodesk

Games, Game Engines, Rendering Pipelines, GPU things and 3D… A LOT of 3D.

philippe.groarke@gmail.com

pgroarke on Cpplang Slack

Any views or opinions presented in this presentation are solely those of the author and do not necessarily represent those of his employer.

OUTLINE

I hate outlines

A Note On Performance and Optimizations

pgroarke 5:19 PM
Hey buddy, do you think a SIMD presentation would work at cppmtl?

[…]

gabriel_al 1:49 PM
When you say better than hoping your compiler vectorizes,
do you mean you've done manual optimizations before profiling? 😛

Reenactment

A Note On Performance and Optimizations

"Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered.

We should forget about small efficiencies, say about 97% of the time: ***premature optimization is the root of all evil.****Yet we should not pass up our opportunities in that critical 3%."*

Donald Knuth, **1974**

Structured Programming With Go To Statements

Which Can Lead To

- Complacency.
- An excuse to ignore performance best practices.
- Bad engineering.
- Bad architecturing.
- Ignorance of hardware constraints.
- Under-utilization of expensive hardware.
- Slow software.

You Wouldn't Do This...

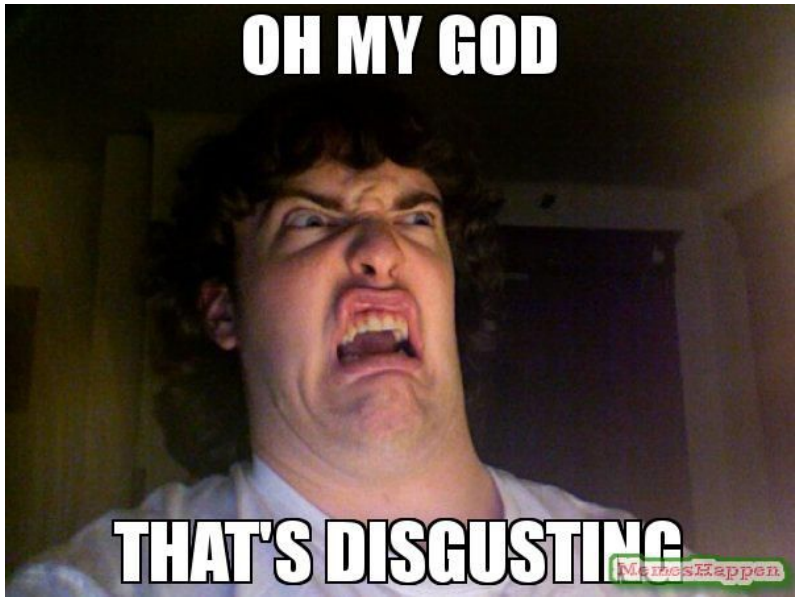
```
struct big_data {
    double data[1024] = {};
};

void do_something(big_data performance_is_evil) {
    /* ... */
}
```

Would You?

```
struct big_data {
    double data[1024] = {};
};

void do_something(big_data performance_is_evil) {
    /* ... */
}
```



A More Sensible Approach

DO think about performance when designing and architecturing your software.

DO think about performance when choosing containers.

DO think about the bottlenecks of your particular problem space.

PREPARE for eventual performance doomsday.

DON’T waste time optimizing before benchmarking or before there is a need for it.

If you don’t need any performance at all, you should not be using C++.

Let’s Get Technical

What is SIMD?

Single Instruction Multiple Data

MMX, SSE, SSSE3, AVX, AVX-512, etc.

XMM : Wide register

Ex : 128 bits wide register

2 doubles, 4 floats, 8 shorts, 16 chars!

Process “chunks” of data with 1 CPU instruction.

Just A CPU Instruction

Normal CPU Instruction

```
add     rax, 0x01
```

SIMD

```
addps   xmm1, xmm0
```

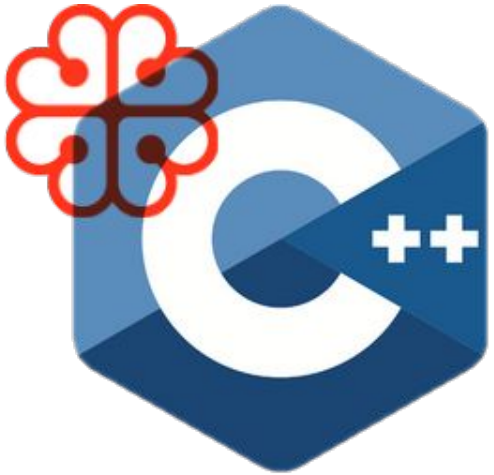
Float Addition

Audio Gain

audio channel	0.53	0.521	0.5	0.49	*
mulps gain	1.5	1.5	1.5	1.5	
audio result	0.795	0.7815	0.75	0.735	=

Color Multiply Blend

rgba0	0.8	0.8	0.8	1.0	*
mulps rgba1	0.2	0.0	0.2	1.0	=
rgba1	0.16	0.0	0.16	1.0	



macOS strlen

string	c	p	p	m	t	l	\0
							==
pcmpeqb zero	\0	\0	\0	\0	\0	\0	\0
							=
compare_result	0	0	0	0	0	0	0xFF

```
size_t len = __builtin_ctz(compare_result); // BitScanForward on MSVC
```


NOT BAD BRO
BRILLIANT!
NOT BAD.
makeameme.org

How To Use SIMD?

Auto-Vectorization, aka “hope for the best!”

- Tedious, unreliable, may break silently. Cross-platform? Good luck ;)

Use a library, often least common denominator.

- A very good option if you only need select instructions.
- Upcoming Parallelism TS proposal : cppreference.com/w/cpp/experimental/simd/simd

Use intrinsics, write your algorithm within the constraints of SIMD.

- Full control and exposed logic.
- **More complicated and harder to read. Prepare yourself for intense code reviews!**
- Most importantly, fun.

When To Use SIMD

Lots of homogenous data.

- Offset the cost of loading and unloading data in SSE registers.
- Ex: 3D meshes, image processing, video codecs, financial data, audio processing, bitcoin?

Computation bound processing. The "core" of your application.

- Used for small specific hotspots in your software.
- Repeated operations on data.
- Few branches, simple or no logic.

Other optimizations have failed, or are inappropriate.

- Multi-threading doesn't lend itself well to **all** problems.
- As seen in a previous cppmtl talk, GPU compute is often inadequate.

When To Use SIMD

You can use SIMD inside threads for ridiculous performance gains.

- Assuming your algorithm lends itself well to both multi-threading and SIMD.

For recent standards, you will have to check availability of feature set at runtime.

- MSDN __cpuid example : https://msdn.microsoft.com/en-us/library/hskdteyh.aspx
- SSE2 is available on all x64 architectures, and is always enabled on MSVC 64bit targets.

Good example

- Compute normals of mesh faces.

Bad examples

- Database interactions, File IO.

Just Another Tool

SIMD is just a tool, use it as such.

Keep it in mind when architecturing your software.

It can do great things or suck greatly. It all depends on your problem and data layout.

How To Organise Your Data

Forget lists ever existed, be suspicious of maps.

If possible, do not use objects or perform operations on heterogenous values.

Favor SoA types for your core data.

- Side effect : Your data will be cache-friendly as well.
- **Requires pretty huge refactoring and may be impossible on legacy codebases.**
- Tough sell…

Structure of Arrays

```
struct vec3 {
    float x;
    float y;
    float z;
};
std::vector<vec3> my_vecs;
```

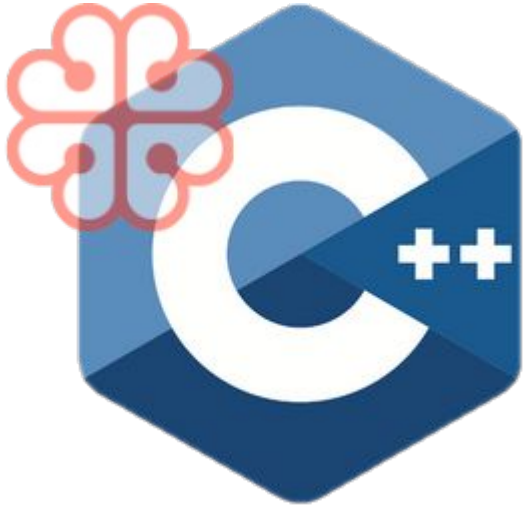
```
struct vec3_soa {
    std::vector<float> x;
    std::vector<float> y;
    std::vector<float> z;
};
```

Conventional Alpha Blend

blend(source, dest) = (source.rgb * source.a) + (dest.rgb * (1 - source.a))

What about alpha?

rgba0	0.8	0.8	0.8	1.0	*
mulps alpha0	?	?	?	?	=
rgba1	?	?	?	?	



Conventional Alpha Blend

Structure of Arrays to the rescue!

Source Colors

	Src 1	Src 2	Src 3	Src 4
4x red	0.8	0.6	0.0	1.0
4x green	0.8	1.0	1.0	0.6
4x blue	0.8	0.2	0.0	1.0
4x alpha	0.5	1.0	1.0	0.6

Destination Colors

	Dst 1	Dst 2	Dst 3	Dst 4
4x red	0.1	0.2	0.3	0.4
4x green	0.1	0.2	0.3	0.4
4x blue	0.1	0.2	0.3	0.4
4x alpha	0.1	0.2	0.3	0.4

Conventional Alpha Blend

blend(source, dest) = **(source.rgb * source.a)** + (dest.rgb * (1 - source.a))

4x red_src	0.8	0.6	0.0	1.0	*
mulps 4x alpha_src	0.5	1.0	1.0	0.6	
4x red_src_result	0.4	0.6	1.0	0.6	=

Conventional Alpha Blend

blend(source, dest) = (source.rgb * source.a) + **(dest.rgb * (1 - source.a))**

4x red_dst	0.1	0.2	0.3	0.4	*
mulps (4x 1.0	1.0	1.0	1.0	1.0	-
subps 4x alpha_src **)**	0.5	1.0	1.0	0.6	=
4x red_dst_result	0.05	0.0	0.0	0.16	

Conventional Alpha Blend

blend(source, dest) = (source.rgb * source.a) + (dest.rgb * (1 - source.a))

4x red_src_result	0.4	0.6	1.0	0.6	+
addps 4x red_dst_result	0.05	0.0	0.0	0.16	=
4x red_result	0.45	0.6	1.0	0.76	

Repeat for green and blue.

blend_alpha(source, dest) = source.a + (dest.a * (1 - source.a))

SIMD Basics - SSE Data Types

- `__m128` - Holds four 32-bit floats
- `__m128d` - Holds two 64-bit doubles
- `__m128i` - Holds either sixteen 8-bit, eight 16-bit, four 32-bit or two 64-bit integers.

Includes

```
#include <xmmintrin.h> // SSE
#include <emmintrin.h> // SSE2
#include <pmmintrin.h> // SSE3
#include <tmmintrin.h> // SSSE3
#include <smmintrin.h> // SSE4.1
#include <nmmintrin.h> // SSE4.2
```

SIMD Basics - SSE Intrinsics Naming Scheme

```
_mm_<intrin_op>_<suffix>  // ex : _mm_add_ps
```

Suffix Prefix

- `p` - packed
- `ep` - extended packed
- `s` - scalar

Suffix Suffix

- `s, d` - float, double
- `i8, i16, i32, i64, i128` - n-bit signed integer
- `u8, u16, u32, u64` - n-bit unsigned integer

Example

Lets try to beat libc++!

Anyone Uses chars?

String types are particularly easy to accelerate, as you get 16x or 8x data throughput, minus overhead.

The data is naturally contiguous.

String operations aren’t (usually) overly complex.

A certain sponsor buys pizza every month, I have a feeling they use strings a lot ;)

Let's beat std::replace!

Boring Replace

```
template <size_t N>
void simple_replace(char (&data)[N], char original, char replacement) {
        for (size_t i = 0; i < N; ++i) {
                if (data[i] == original)
                        data[i] = replacement;
        }
}
```

We can do better!

The Algorithm

Fast SIMD algorithms usually boil down to creative ways of repurposing conditionals.

How can we get rid of the "if" statement?

SSE has a compare function, which returns 0xFF on a match.

Chars are just numbers, we don't have to replace the char, we can increment or decrement it.

All we need then, is a mask to apply the transformation on our desired characters.

The Algorithm

An example using one character. Replace 'A' with 'C', 65 with 67.

1. Find the difference between chars : 67 - 65 = +2
2. Compare the input character, store the answer : True is 0xFF, false is 0x00.
3. Mask the answer with 0b00000001 : True is 1, false is 0.
4. Multiply the mask by the character difference (+2) : True is +2, false is 0.
5. Add the value to the original char : 65 + 2 = 67

simd_replace

```
template <size_t N>
void simd_replace(char (&data)[N], char original, char replacement) {

    // Load the searched for character.
    const __m128i xmm_search_char = _mm_set1_epi8(original);

    // ASCII difference of characters.
    const __m128i xmm_diff = _mm_set1_epi8(replacement - original);

    // Mask upper 7bits.
    const __m128i xmm_mask_7 = _mm_set1_epi8(0x01);
```

char original = 'E'; // 69
char replacement = 'A' // 65

69	69	69	69	...

-4	-4	-4	-4	...

0b00000001	0b00000001	...

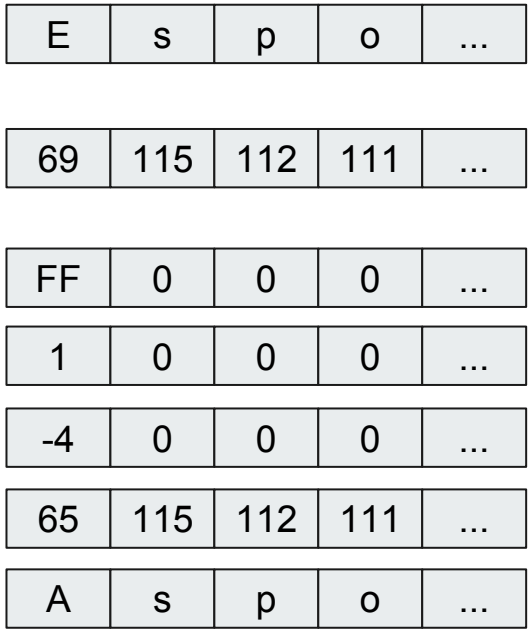
...

simd_replace

```
size_t i;
for (i = 0; i < N / 16 * 16; i += 16) {
        // Load unaligned.
        const __m128i xmm_str =
                _mm_loadu_si128((const __m128i*)(&data[i]));

        // Find char.
        __m128i xmm_result = _mm_cmpeq_epi8(xmm_str, xmm_search_char);
        // Mask all upper 7bits to get 0x01. We could bit shift instead.
        xmm_result = _mm_and_si128(xmm_result, xmm_mask_7);
        // Multiply 1 by character difference.
        xmm_result = _mm_mullo_epi8(xmm_result, xmm_diff);
        // Add the character difference.
        xmm_result = _mm_add_epi8(xmm_result, xmm_str);
        // Store the results in string.
        _mm_storeu_si128((__m128i*)(&data[i]), xmm_result);
}

// Final pass : N - i != 0
```



Profiling & Demo

May the demo gods be with us.

In Closing

SIMD is an alternative parallel idiom that complements threads.

Use SIMD when processing big data.

Images, videos, 3d meshes, audio, DSP, financial data, scientific data etc.

SIMD cannot make you a sandwich, even if you ask nicely.

Thank you for attending!

Resources

- Intel Intrinsics Guide
  - software.intel.com/sites/landingpage/IntrinsicsGuide/
  - Highly recommended!
- x86 Intrinsics Cheat Sheet
  - https://db.in.tum.de/~finis/x86-intrin-cheatsheet-v2.2.pdf
- SIMD tag on StackOverflow
- Reddit r/simd
- GDC Vault : SIMD at Insomniac Games
  - Andreas Fredriksson
  - Highly recommended as well!