

# SIMD

Single instruction, multiple data

---

Lukas Pietzschmann

lukas.pietzschmann@uni-ulm.de

Institute of Software Engineering and  
Programming Languages  
Institute of Distributed Systems  
University of Ulm

07/04/2023



# AGENDA

1. Motivation

2. Overview

3. How to use SIMD

3.1 Overview

3.2 Data types

3.3 Instructions

3.4 Example

4. Summary

5. References

# 1. MOTIVATION

# HOW NOT TO DO IT

```
void mul4(float* arr) {  
    for(int i=0; i < 4; ++i) {  
        const float f = arr[i];  
        arr[i] = f * f;  
    }  
}
```

## Why is it that bad?

- Short loops are bad
  - Branch prediction will be wrong often
- We could have multiplied way more than two floats

# HOW TO MAKE IT BETTER

```
void mul4(float* vec) {  
    __m128 f=_mm_loadu_ps(vec);  
    f = _mm_mul_ps(f, f);  
    _mm_storeu_ps(vec, f);  
}
```

## Why is it better?

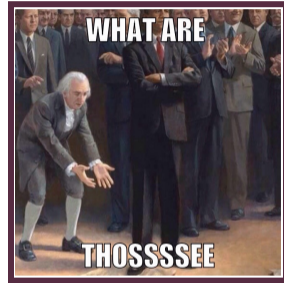
- No loops
- No branches to predict
- Nice machine code
- We square all floats “at once”

# HOW TO MAKE IT BETTER

Or just compile with optimisations

# HOW TO MAKE IT BETTER

```
void mul4(float* vec) {  
    __m128 f=_mm_loadu_ps(vec);  
    f = _mm_mul_ps(f, f);  
    _mm_storeu_ps(vec, f);  
}
```



- `__m128`
- `_mm_loadu_ps(float*)`
- `_mm_mul_ps(__m128, __m128)`
- `_mm_store_ps(float*, __m128)`

# PERFORMANCE

[Kon20]

Description	Time (in $\mu\text{s}$ )
Regular floating point math	439
SSE dpps instruction	181
AVX vdpss instruction	103

Time it takes to compute the dot product of two vectors with a length of 256,000

- ▷ SSE: 2.5x speed increase
- ▷ AVX: 4x speed increase







## 2. OVERVIEW

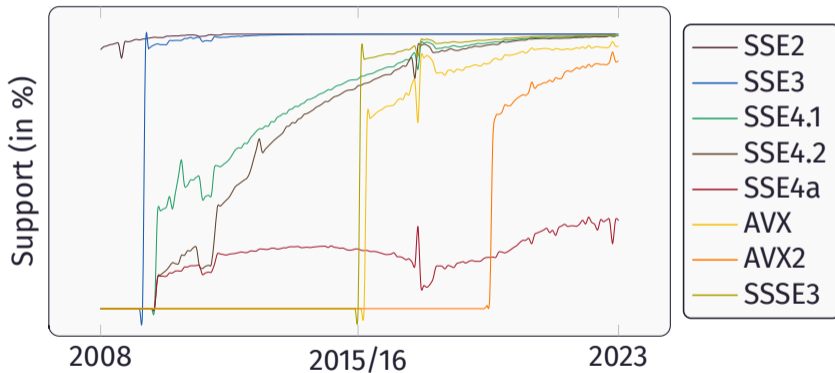
# FORMS OF COMPUTING SYSTEMS

[Fly72]

	Single data stream	Multiple data stream
Single instruction	SISD	 SIMD 
Multiple Instructions	MISD	MIMD

# SIMD SUPPORT

According to Steam



Note that Steam did not start collecting data for all extensions at the date of their release!

# 3. HOW TO USE SIMD

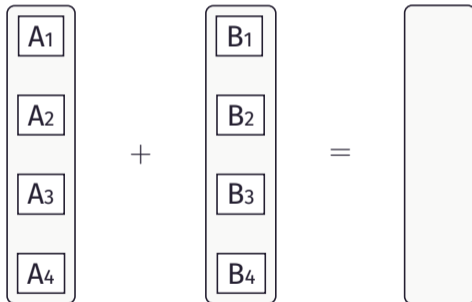
# 3.1 OVERVIEW

# It's hard



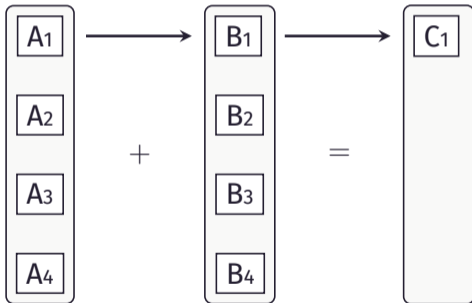
# THE IDEA BEHIND SIMD

Without SIMD:



# THE IDEA BEHIND SIMD

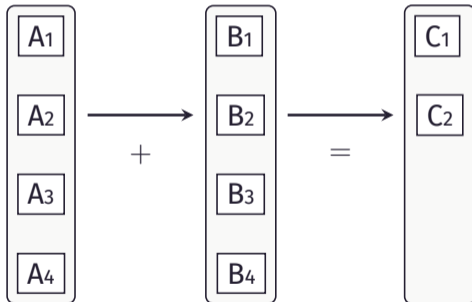
Without SIMD:





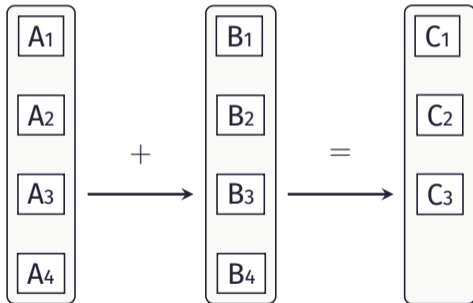
# THE IDEA BEHIND SIMD

Without SIMD:



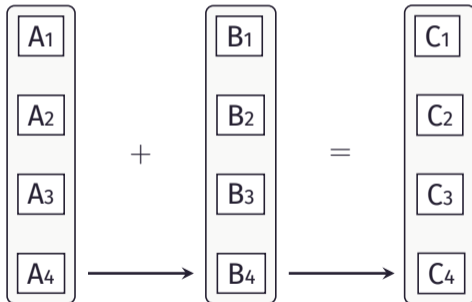
# THE IDEA BEHIND SIMD

Without SIMD:



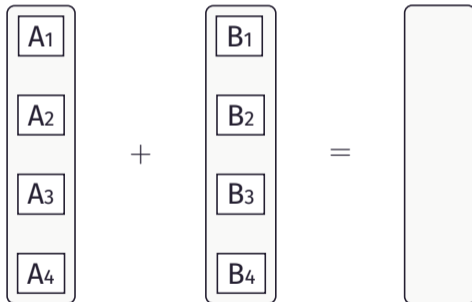
# THE IDEA BEHIND SIMD

Without SIMD:



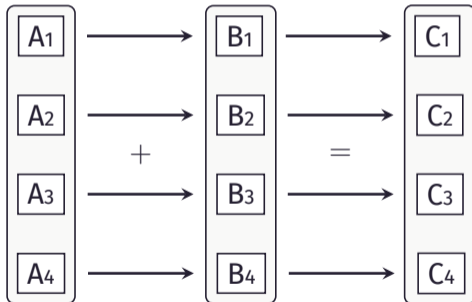
# THE IDEA BEHIND SIMD

With SIMD:



# THE IDEA BEHIND SIMD

With SIMD:



# APPROACH

1. Operate directly on memory

or

1. Load data to registers

2. Do as much as possible while it's in registers

3. Store results ...

- into memory
- in general purpose registers

# SIMD IN C++

## Intrinsics

### Intrinsics

### In... what?

- Usually implemented “inside” the compiler
- Allow for better optimisations than raw inline assembly
- Intrinsics provide access to instructions that cannot be generated using the standard constructs

## 3.2 DATA TYPES



# REGISTER TYPES

	16 Bytes	32 Bytes	
SSE2 →	32 Bit float	__m128	__m256
	64 Bit double	__m128d	__m256d
	32/64 Bit integer	__m128i	__m256i

Note that:

- The CPU doesn't distinguish between **\_\_m128**, **\_\_m128d** and **\_\_m128i**
  - This information is only used for type checking
- The compiler automatically assigns the values to registers
  - Be aware that there are only 16 (8+8) registers underneath the compiler

# REGISTER TYPES

Gotcha!



A SIMD register does not store a single scalar value



but multiple values that are interpreted like a vector.

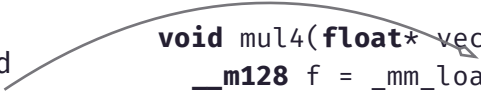
## 3.3 INSTRUCTIONS

# LOADING FROM MEMORY

We can load ...

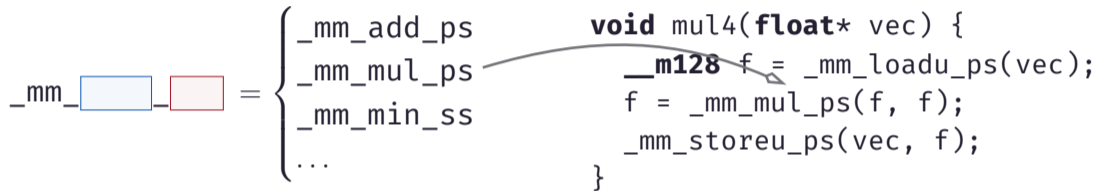
- four values aligned
- four values unaligned
- four values in reverse
- ...

```
void mul4(float* vec) {  
    __m128 f = _mm_loadu_ps(vec);  
    f = _mm_mul_ps(f, f);  
    _mm_storeu_ps(vec, f);  
}
```



# ARITHMETIC OPERATIONS

For floats (and doubles)




```
void mul4(float* vec) {  
    __m128 f = _mm_loadu_ps(vec);  
    f = _mm_mul_ps(f, f);  
    _mm_storeu_ps(vec, f);  
}
```

- add
- sub
- mul
- div
- sqrt
- min
- max
- ...

- ss
- ps
- sd
- pd

# ARITHMETIC OPERATIONS

For integers

\_mm\_ 

- add
- sub
- mul
- avg
- min
- max
- ...


- epi8
- epi16
- epu8

# STORING TO MEMORY

We can store ...

- four values aligned
- four values unaligned
- four values in reverse
- ...

```
void mul4(float* vec) {  
    m128 f = _mm_loadu_ps(vec);  
    f = _mm_mul_ps(f, f);  
    _mm_storeu_ps(vec, f);  
}
```



# MISCELLANEOUS

## Copy values to general purpose registers

```
_m128i → int32_t  
  _mm_cvtsi128_si32  
int32_t → _m128i  
  _mm_cvtsi32_si128  
_m128 → float  
  _mm_cvtss_f32
```

## Cryptography

- AES de- and encryption
- SHA computation

## String manipulation (SSE 4.2)

- Compare strings with ...
  - known length
  - unknown length



## 3.4 EXAMPLE

## MORE ADDING

[Click for code](#)

```
float* add(const float* a, const float* b, size_t size) {
    float* result = new float[size];
    const auto numof_vectorizable_elements = size - (size % 4);
    unsigned i = 0;
    for (; i < numof_vectorizable_elements; i += 4) {
        __m128 a_reg = _mm_loadu_ps(a + i);
        __m128 b_reg = _mm_loadu_ps(b + i);
        __m128 sum = _mm_add_ps(a_reg, b_reg);
        _mm_storeu_ps(result + i, sum);
    }
    for (; i < size; ++i)
        result[i] = a[i] + b[i];
    return result;
}
```

# 4. SUMMARY

# SHOULD YOU EVEN CARE?

You shouldn't if ...

- you don't write performance sensitive code
- you're code is not CPU bound
- if most of the math you do is implemented in libraries
- if your favourite language does not support SIMD



# WHAT I DIDN'T COVER

## Instructions

- Casting
- Converting
- Comparing
- Shuffling
- Shifting
- Logic operations
- Bitwise operations
- Prefetching

## Libraries

- `std::experimental::simd`
- Eigen
- DirectXMath

A lot



# 5. REFERENCES

## REFERENCES (I)

- [Fly72] Michael J. Flynn. “Some Computer Organizations and Their Effectiveness”. In: *IEEE Transactions on Computers* 9 (1972), pp. 948–960. DOI: 10.1109/TC.1972.5009071.
- [Kon20] Konstantin. *Improving performance with SIMD intrinsics in three use cases*. 2020. URL: <https://stackoverflow.blog/2020/07/08/improving-performance-with-simd-intrinsics-in-three-use-cases>.
- [Int21] Intel. *Intrinsics*. 2021. URL: <https://www.intel.com/content/www/us/en/docs/cpp-compiler/developer-guide-reference/2021-8/intrinsics.html>.

**Lukas Pietzschmann**

Ulm, 07/04/2023

lukas.pietzschmann@uni-ulm.de