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

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)`

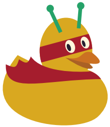
Performance

[Kon20]

Description	Time (in µs)
Regular floating point math	439
SSE dpps instruction	181
AVX vdpps instruction	103

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

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



2. Overview

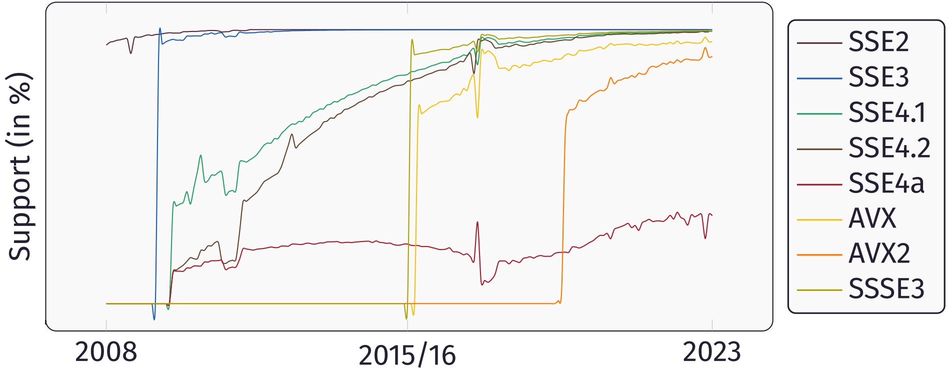
Forms of computing systems

[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!

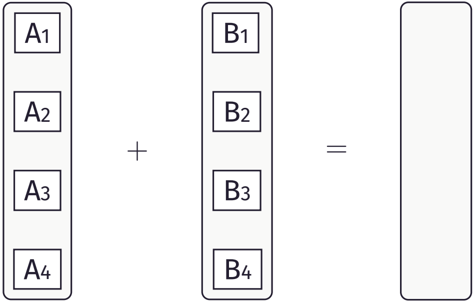
3. How to use SIMD

3.1 Overview

It’s hard

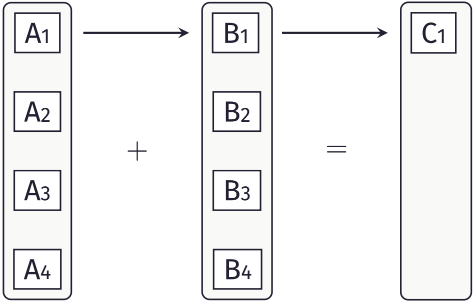
The idea behind SIMD

Without SIMD:



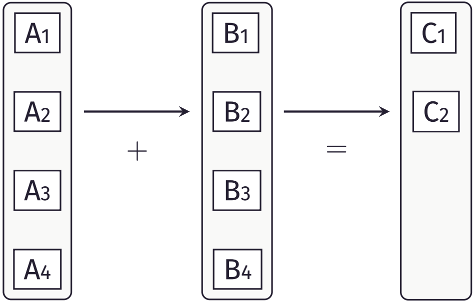
The idea behind SIMD

Without SIMD:



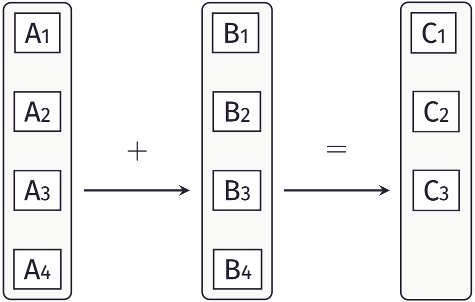
The idea behind SIMD

Without SIMD:



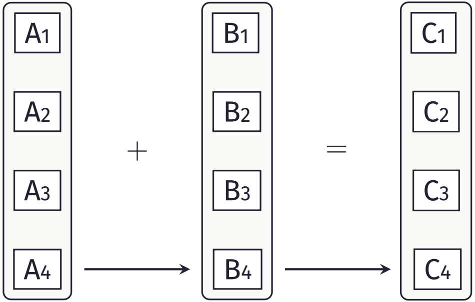
The idea behind SIMD

Without SIMD:



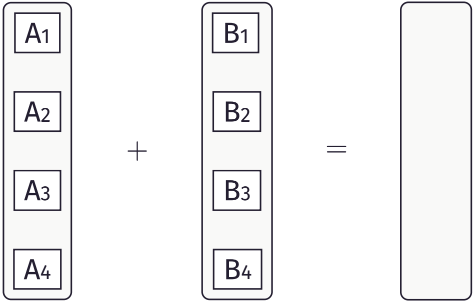
The idea behind SIMD

Without SIMD:



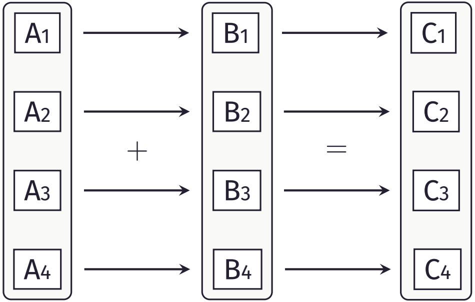
The idea behind SIMD

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
32 Bit float	__m128	__m256
64 Bit double	__m128d	__m256d
32/64 Bit integer	__m128i	__m256i

SSE2 → (64 Bit double)

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)

$$\_mm\_\boxed{}\_\boxed{} = \begin{cases} \_mm\_add\_ps \\ \_mm\_mul\_ps \\ \_mm\_min\_ss \\ \ldots \end{cases}$$

```
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

Well, it's the same. Just append `_epi8` or `_epi16`

`_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

5. References

References

[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