SIMD Libraries in C++

Jeff Garland

Created: 2023-05-08

A regular c++ program uses a tiny percentage of the machine silicon

- *Sean Parent*
  - *(paraphrased) CPPNorth 2022, c++Now Keynote 2012*
  - *https://youtu.be/iGenpw2NeKQ?t=24m13s*

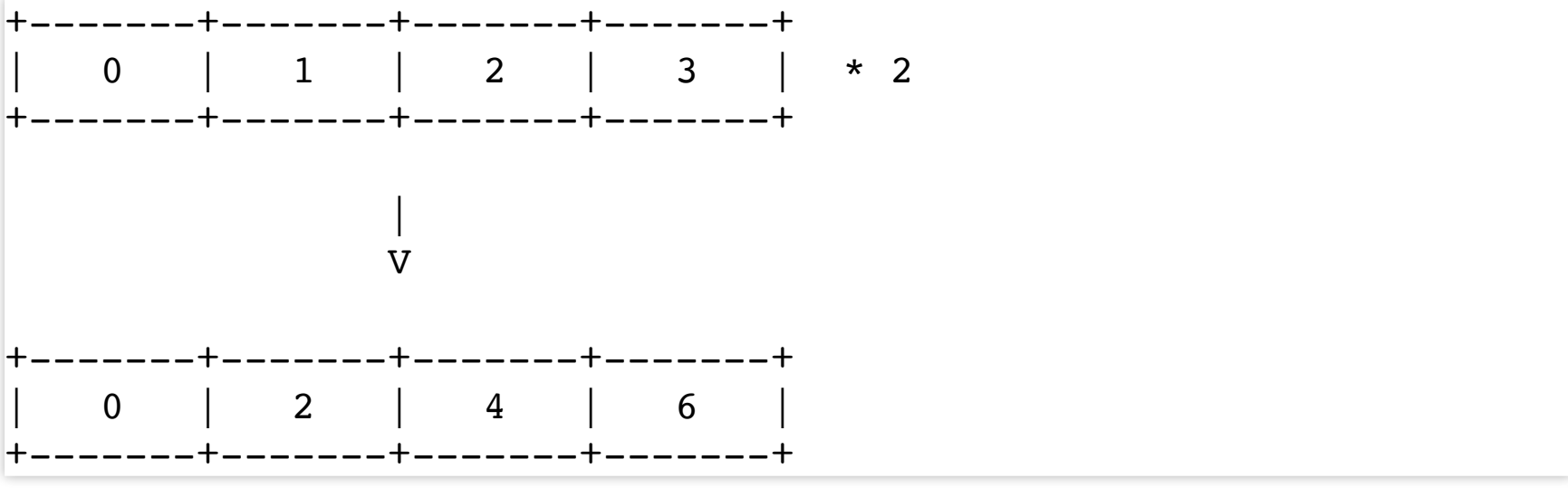
Talk Outline

- intro to simd
- why simd isn't for free
- simd libraries
  - high level
  - development libraries
- `std::simd`

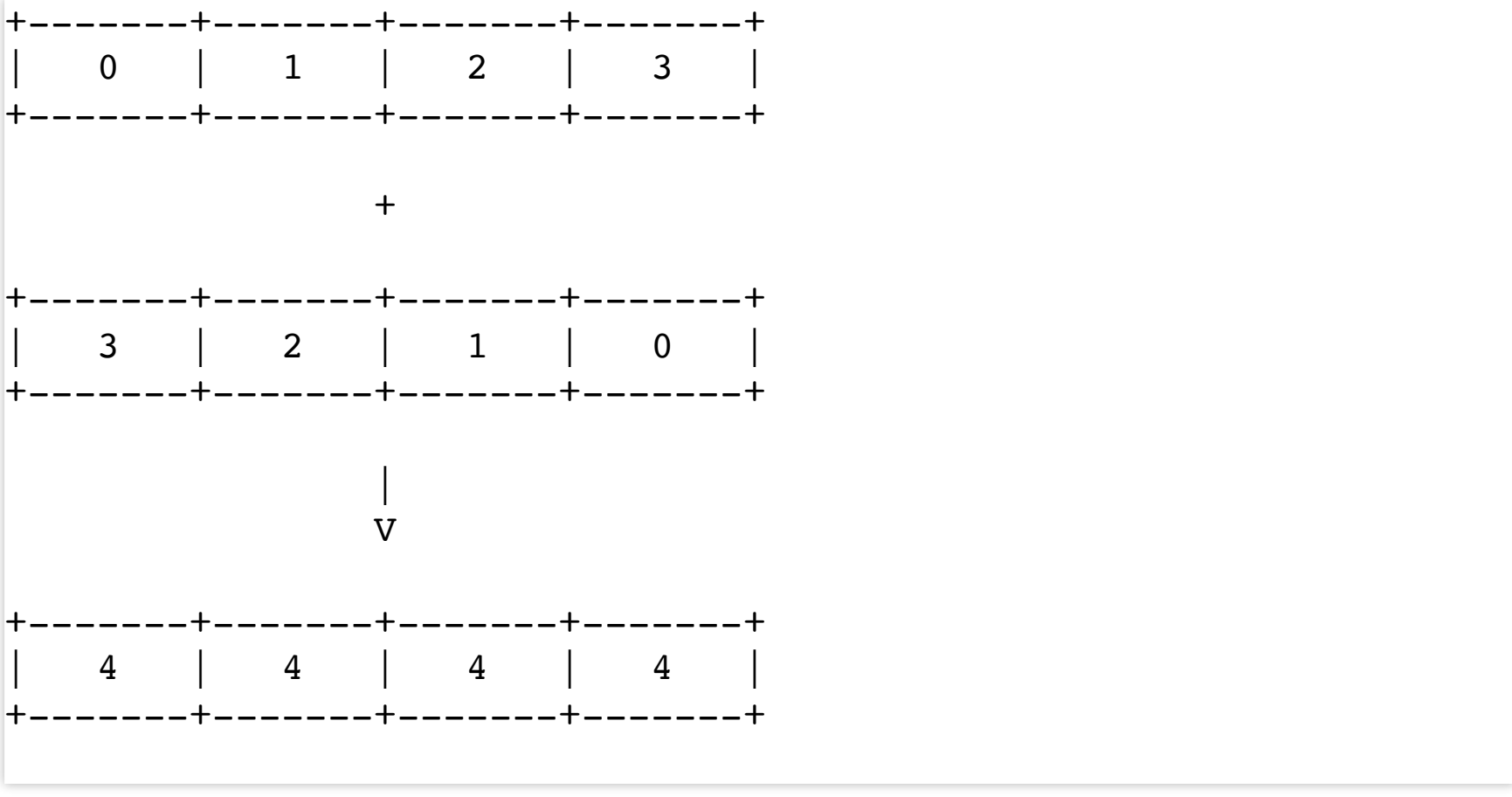
What is SIMD

- single instruction multiple data
- vector processing
- math applications are obvious domain

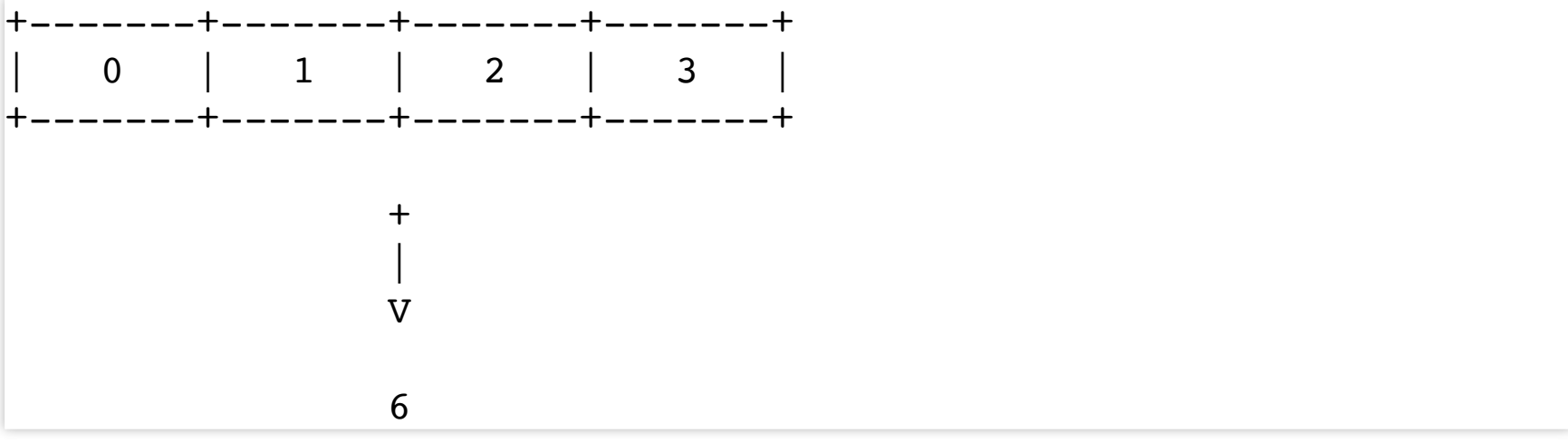
vector <op> single value –> vector



vector <op> vector –> vector



vector <op> –> single value



Why do we care?

- performance!
- handy form of parallelism
- 'easy' 4-16x speedup
  - only if code is easy and clear
  - only if portable
- tends to be branchless
  - branchless == fast

How can we get it?

- compiler unrolling loops - maybe?
- compiler intrinsics - yuk!
  - not necessarily portable
- open mp #pragma flags - yuk, yuk!
- high level interface - yes and yes

C array - can the compiler just do it?

```
void v_add8( int* c, int* a, int* b )
{
  for (int i=0; i<=7; i++)
     c[i] = a[i] + b[i];
};
int main()
{
    int c[] = {0, 0, 0, 0, 0, 0, 0, 0};
    int a[] = {1, 2, 3, 4, 5, 6, 7, 8};
    int b[] = {7, 6, 5, 4, 3, 2, 1, 0};
    v_add8(c, a, b);
    print("{}", c); //[8, 8, 8, 8, 8, 8, 8, 8]
}
```

C array - can the compiler just do it?

```
main:
        sub     rsp, 120
        vpxor   xmm0, xmm0, xmm0
        vmovdqa XMMWORD PTR [rsp+16], xmm0
        vmovdqa XMMWORD PTR [rsp+32], xmm0
        vmovdqa xmm0, XMMWORD PTR .LC4[rip]
        lea     rdi, [rsp+16]
        lea     rdx, [rsp+80]
        lea     rsi, [rsp+48]
        vmovdqa XMMWORD PTR [rsp+48], xmm0
        vmovdqa xmm0, XMMWORD PTR .LC5[rip]
        vmovdqa XMMWORD PTR [rsp+64], xmm0
        vmovdqa xmm0, XMMWORD PTR .LC6[rip]
        vmovdqa XMMWORD PTR [rsp+80], xmm0
        vmovdqa xmm0, XMMWORD PTR .LC7[rip]
        vmovdqa XMMWORD PTR [rsp+96], xmm0
        call    v_add8(int*, int*, int*)
```

C array - can the compiler just do it?

```
v_add8(int*, int*, int*):
        xor     eax, eax
.L21:
        mov     ecx, DWORD PTR [rdx+rax]
        add     ecx, DWORD PTR [rsi+rax]
        mov     DWORD PTR [rdi+rax], ecx
        add     rax, 4
        cmp     rax, 32
        jne     .L21
        ret
```

- sad trombones

std::array - can the compiler just do it?

```
using array8i = std::array<int, 8>;

void
arr_add( array8i& c, const array8i& a, const array8i& b )
{
  for (int i=0; i<=7; i++)
     c[i] = a[i] + b[i];
};
```

std::array - can the compiler just do it?

```
using array8i = std::array<int, 8>;

void
arr_add( array8i& c, const array8i& a, const array8i& b )
{
  for (int i=0; i<=7; i++)
     c[i] = a[i] + b[i];
};
```

- sorry charlie...

std::array - can the compiler just do it?

```
using array8i = std::array<int, 8>;

array8i
arr_add( const array8i& a, const array8i& b )
{
  array8i c;
  for (int i=0; i<=7; i++)
     c[i] = a[i] + b[i];
  return c;
};
```

std::array - can the compiler just do it?

```
using array8i = std::array<int, 8>;

array8i
arr_add( const array8i& a, const array8i& b )
{
  array8i c;
  for (int i=0; i<=7; i++)
     c[i] = a[i] + b[i];
  return c;
};
```

- we have a winner!

std::array - can the compiler just do it?

```
arr_add(std::array<int, 8ul> const&, std::array<int, 8ul> const&):
        vmovdqu ymm0, YMMWORD PTR [rdx]
        mov     rax, rdi
        vpaddd  ymm0, ymm0, YMMWORD PTR [rsi]
        vmovdqu YMMWORD PTR [rdi], ymm0
        vzeroupper
        ret
```

- https://godbolt.org/z/3nGvGxcPs

Exploiting SIMD

- requires new data structures
- requires new algorithms
- aka: programmer intervention
- std::simd is proposed for c++ 26

'No magical compiler coming'

- *James Rainders (formally Intel) 2016*

SIMD in C++

SIMD typical applications

- math (on vectors/arrays)
- CRC zip compression
- hashing - google highway hash
- sorting

Application libraries

- simd json https://simdjson.org/
  - parse gigabytes json per second
- simd in the standard library
  - parallel algorithms
  - mdspan
- simd crc https://github.com/neurolabusc/simd_crc
- simd sort https://github.com/intel/x86-simd-sort
- math https://bitbucket.org/blaze-lib/blaze/src/master/

Developer libraries

- xsimd - c++11
  - 'batch type' is vector type
  - https://xsimd.readthedocs.io/en/latest/
- eve (formerly boost.simd) - c++20
  - Expressive Vector Engine
  - 'wide type' is vector type
  - https://jfalcou.github.io/eve/index.html
- Agner Fog VCL
  - vector class library
  - series of vector types
  - https://www.agner.org/optimize/vectorclass.pdf

xsimd example

```
include "xsimd/xsimd.hpp"
namespace xs = xsimd;

template <class T, class Arch>
xs::batch<T, Arch> mean(xs::batch<T, Arch> lhs, xs::batch<T, Arch> rhs)
{
    return (lhs + rhs) / 2;
}
int main()
{
    xs::batch<double, xs::avx> a = { 1.5, 2.5, 3.5, 4.5 };
    xs::batch<double, xs::avx> b = { 2.5, 3.5, 4.5, 5.5 };
    auto mean = (a + b) / 2;
    std::cout << mean << std::endl;
}
```

Eve Example

```
void check_polar()
{
  eve::wide<float> x1{4};
  eve::wide<float> y1{[](auto i, auto ) { return 1.5f*(i+1); }};

  std::cout << x1 << " " << y1 << " => " << rho(x1,y1) << "\n";

  float data[] = {1.5f, 3, 4.5f, 6, 7.5f, 9, 10.5f, 12, 13.5, 15,
                  16.5, 18, 19.5, 21, 22.5, 24};
  eve::wide<float> y2{&data[0]};

  std::cout << x1 << " " << y2 << " => " << theta(x1,y2) << "\n";
}
```

Agner Fog VCL - vector class library

- series of 'vector classes'
  - Vec8i, Vec4f, Vec16c, ...
- half precision floating point
- has a decimal to ascii conversion util

class	bitsize	signed	elements per vec	total bits	min recor instru
Vec32us	16	unsigned	32	512	AVX5
...					

Agner Fog vec8

```
#include <vectorclass.h>

void
v_add( Vec8i& c, const Vec8i& a, const Vec8i& b )
{
     c = a + b;
};

int main()
{
    Vec8i c = {0, 0, 0, 0, 0, 0, 0, 0};
    Vec8i a = {1, 2, 3, 4, 5, 6, 7, 8};
    Vec8i b = {7, 6, 5, 4, 3, 2, 1, 0};

    v_add(c, a, b);
    for (int i=0; i<7; i++)
      print("{} ", c[i]); //8 8 8 8 8 8 8
}
```

Agner Fog vec8 asm

```
v_add(Vec8i&, Vec8i const&, Vec8i const&):
        vmovdqa ymm0, YMMWORD PTR [rsi]
        vpaddd  ymm0, ymm0, YMMWORD PTR [rdx]
        vmovdqa YMMWORD PTR [rdi], ymm0
        vzeroupper
        ret
```

- winner!

Agner Fog

```
Vec8i
v_add(const Vec8i& a, const Vec8i& b )
{
     return a + b;
};
```

```
v_add(Vec8i const&, Vec8i const&):
        vmovdqa ymm0, YMMWORD PTR [rdi]
        vpaddd  ymm0, ymm0, YMMWORD PTR [rsi]
        ret
```

std::simd

concept: data parallel types

- `simd<T>` and `fixed_size_simd<T,N>`
- T arithmetic type, called the element type
- number of elements is a constant for each data-parallel type and called the `width`
- element is `vectorizable` type
  - char, unsigned char, integers, float, double, …
- `simd_mask` - specialization with bool element type

native versus fixed abi

- native abi
  - best for target platform
  - will be default
- fixed abi
  - specified by programmer

`std::simd` add 2 vectors

```
#include <experimental/simd>

namespace stdx = std::experimental;
using intv  = stdx::fixed_size_simd<int,8>;

intv add_v(const intv& a, const intv& b)
{
    return a + b;
}

int main()
{
    //note brace init doesn't work :(
    //intv a = {1, 2, 3, 4, 5, 6, 7, 8};
    //note constructor doesn't support
    //intv a{1, 2, 3, 4, 5, 6, 7, 8};
```

- a bit disappointing

`std::simd` add 2 c arrays

```
#include <experimental/simd>
namespace stdx = std::experimental;
using intv  = stdx::fixed_size_simd<int,8>;

intv add_v(const intv& a, const intv& b)
{
    return a + b;
}
int main()
{
    int a_data[] = {1, 2, 3, 4, 5, 6, 7, 8};
    intv a;
    a.copy_from(a_data.begin(), stdx::vector_aligned);
    int b_data[] = {7, 6, 5, 4, 3, 2, 1, 0};
    intv b;
    b.copy_from(b_data.begin(), stdx::vector_aligned);
    intv c = add_v( a, b );

    for (int i=0; i< c.size(); i++)
       print("{} ", c[i]);
```

- `copy_from` to construct

`std::simd` add 2 c arrays

```
add_v(std::experimental::parallelism_v2::simd<int, std::experimental::p
        vmovdqa ymm0, YMMWORD PTR [rdx]
        mov     rax, rdi
        vpaddd  ymm0, ymm0, YMMWORD PTR [rsi]
        vmovdqa YMMWORD PTR [rdi], ymm0
        vzeroupper
        ret
```

fixed_vec

```
namespace stdx = std::experimental;
using intv  = stdx::fixed_size_simd<int,8>;

template<class T, int N>
class fixed_vec : public stdx::fixed_size_simd<T,N>
{
public:
   //inherit constructors
   using stdx::fixed_size_simd<T,N>::fixed_size_simd;

   fixed_vec(std::span<T,N> data)
   {
     this->copy_from(&data[0], stdx::vector_aligned);
   }
   fixed_vec(std::array<T,N> data)
   {
     this->copy_from(&data[0], stdx::vector_aligned);
   }
};
```

- https://godbolt.org/z/bMGba78fq

`std::simd` library outline

`std::simd` constructors

```
template<class U>
constexpr simd(U&& value) noexcept;

template<class U, class UAbi>
constexpr explicit simd(const simd<U, UAbi>&) noexcept;

template<class G> constexpr explicit simd(G&& gen) noexcept;

template<contiguous_iterator It, class Flags = element_aligned_tag>
constexpr simd(const It& first, Flags = {})

template<class U, class Flags = element_aligned_tag>
void copy_from(const U* mem, Flags = {}) &&

template<class U, class Flags = element_aligned_tag>
void copy_to(U* mem, Flags = {}) const &&;
```

generator function construction

```
#include <experimental/simd>
namespace stdx = std::experimental;
using intv  = stdx::fixed_size_simd<int,8>;

std::random_device rd;  // a seed source for the random number engine
std::mt19937 gen(rd()); // mersenne_twister_engine seeded with rd()
std::uniform_int_distribution<> distrib(1, 20);

intv a([&gen, &distrib](int){ return distrib(gen);});
// use a ...
```

- https://godbolt.org/z/8WG6qq78s

algorithms

- `any_of`, `all_of`, `none_of`
- `reduce`, `hmin`, `hmax`, `abs`

reduce

```
namespace stdx = std::experimental;
using intv  = stdx::fixed_size_simd<int,8>;

int main()
{
    std::array<int,8> a_data = {-1, 2, 3, 4, 5, 6, 7, -8};
    intv a;
    a.copy_from( a_data.begin(), stdx::vector_aligned );

    //returns fixed_size_simd<int,8>
    auto a_abs = abs(a);
    int sum = reduce( a_abs );
    int min = hmin( a_abs );
    int max = hmax( a_abs );

    // 36 1 8
    print("{} {} {}", sum, min, max);
}
```

- https://godbolt.org/z/7W4sxaoEq

`std::simd` - comparison

```
// 36 1 8
// comparison works: true true true true true true true true
// 39.6 1.1 8.8
int main()
{
    int a_data[] = {1, 2, 3, 4, 5, 6, 7, 8};
    fixed_vec<int,8> a(a_data);
    print_stats(a);
    fixed_vec<int,8> b = a; //copy

    auto c = a == b;
    if ( all_of( c ) )
    {
        print("comparison works: ");
        for (int i=0; i < c.size(); i++)
           print("{} ", c[i]);
        print("\n");
    }

    std::array<float,8> fa = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8};
    fixed_vec<float, 8> fv{fa};
    print_stats(fv);
}
```

- https://godbolt.org/z/vqaqorWMY

`std::simd` design open items

- type support std::complex (P2663)
- permutation algorithms
- parallel algorithm updates
- span and array as input (see P1928R3)
- initializer list
- ranges support

Wrap up & References

"They told me computers could only do arithmetic"

- *Grace Hopper 1951*
- ***after** inventing the first compiler*

references

- merges std::simd https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p1928r3.p
- complex for std::simd https://isocpp.org/files/papers/P2663R2.html
- permutation for simd https://isocpp.org/files/papers/P2664R2.html