Document Number: P3319R0
Date: 2024-05-22
Reply-to: Matthias Kretz <m.kretz@gsi.de>
Audience: LEWG
Target: C++26

Add an iota object for simd (and more)

Abstract

There is one important constant in SIMD programming: `0, 1, 2, 3, ...`. In the standard library we have an algorithm called `iota` that can initialize a range with such values. For `simd` we want to have simple to spell constants that scale with the SIMD width. This paper proposes a simple facility that can be generalized.

Contents

1 CHANGELOG

(placeholder)

2 STRAW POLLS

(placeholder)

3 MOTIVATION

The 90% use case for simd generator constructors is a simd with values 0, 1, 2, 3, … potentially with scaling and offset applied. However, often it would be more easier and more readable to use an "iota" `simd` object instead.

generator ctor	iota

`std::simd<int> a([](int i) { return i; };`
`std::simd<int> b([](int i) { return 2 + 3 * i; };`
	`auto a = std::iota_v<std::simd<int>>;`
	`auto b = 2 + 3 * std::iota_v<std::simd<int>>;`

The minimal definition I propose for `basic_simd` can look like this:

```
template <class T>
  inline constexpr T
  iota_v;

template <class T>
  requires(std::is_arithmetic_v<T>)
  inline constexpr T
  iota_v<T> = T();

template <detail::simd_type T>
  inline constexpr T
  iota_v<T> = T([](auto i) { return static_cast<typename T::value_type>(i); });
```

4 GENERALIZATION

If we define a (`constexpr`) variable template `std::iota_v<T>` where `T` must be a `basic_simd` type, we're simply filling a sequence of values. We can create such an object for any type with static extent. This is especially interesting for the degenerate case in SIMD-generic programming, where `T` could e. g. be an `int`. A `std::iota_v<int>` is nothing other than an object `int` with value `0`. We can easily generalize to `iota_v<std::array<T, N>>` and `iota_v<T[N]>`. And the next step then is to allow any type that

- has a static extent,
- has a value_type member type,
- can be list-initialized with N numbers of type value_type, where N equals the static extent of the type, and
- where value_type() + 1 is an constant expression and convertible to value_type.

Consequently you could write

```
auto x = std::iota_v<float[5]>;
auto y = std::iota_v<std::array<my_fixed_point, 8>>;
// ...
```

5 RELATION TO LIST-INITIALIZATION OF SIMD

If we add a constructor to basic_simd that enables list-initialization, then many users might use that in place of a generator constructor. This leads to code that doesn't scale with the vector width anymore. Therefore we should provide a simple facility that works better and is more portable.

6 PROPOSED POLLS

Poll: We want an iota facility for basic_simd

SF	F	N	A	SA

Poll: The iota facility should be generalized to scalars

SF	F	N	A	SA

Poll: The iota facility should be generalized to any sequence of static extent

SF	F	N	A	SA

7 WORDING

TBD after deciding on the preferred solution.