Document Number: P0964R0
Date: 2018-02-12
Reply-to: Matthias Kretz <m.kretz@gsi.de>
Audience: SG1, LEWG

Finding the right set of traits for `simd<T>`

ABSTRACT

This paper discusses the set of traits we want to ship with `simd<T>`.

CONTENTS

1 Introduction

Kretz [P0214R8] defines the trait `abi_for_size<T, N>`, allowing users to find an "implementation-recommended" ABI tag for a given `value_type` and number of elements. Shen [P0820R1] discusses a use for considering involved ABI tags in the "recommendation". SG1 polled in Albuquerque about

Poll: `abi_for_size_t` (SF) vs. *implementation-defined* (SA)

SF	F	N	A	SA
1	7	7	0	0

The poll result implies that SG1 prefers users to be able to spell out the ABI tags that are determined as return types. The poll was not about a specific name to use for the trait. Shen [P0820R1] suggests to rename the trait to `rebind_abi<T, N, Abis...>`.

2 Motivation

I believe the name `rebind_abi` in Shen [P0820R1] is misleading, since no rebinding is taking place, but rather a type for implementing a rebind of a given `simd<T, Abi>` to a different `value_type U` is made possible. Therefore, I propose to

1. not rename the `abi_for_size` trait in Kretz [P0214R8], and
2. extend `abi_for_size` to consider input ABI tags in its decision, and
3. introduce a new trait `rebind_simd<U, V>`, which deduces a `simd<U, Abi>` instantiation from a given `simd` type `V` and requested `value_type U`.

In addition to `rebind_simd`, SG1 should consider whether a `resize_simd` trait should be added. `resize_simd_t<N, simd <T, Abi0>>` is an alias for a `simd<T, Abi1>` so that `simd_size_v<T, Abi1> == N`, and `resize_simd_t<N, simd_mask <T, Abi0>>` is an alias for a `simd_mask<T, Abi1>` so that `simd_size_v<T, Abi1> == N`. Since the implementation burden is minimal and the trait can simplify user code, I recommend to add it to the Parallelism TS 2.

3 Proposed Wording

Apply the following change to the Parallelism TS 2 before finalization:

modify §8.2

~~`template <class T, size_t N> struct abi_for_size { using type = see below; };`~~

~~`template <class T, size_t N> using abi_for_size_t = typename abi_for_size<T, N>::type;`~~

`template <class T, size_t N, class... Abis> struct abi_for_size { using type = see below; };`

```
template <class T, size_t N, class... Abis> using abi_for_size_t = typename abi_for_size<T, N, Abis...>::type;

template <class T, class V> struct rebind_simd { using type = see below; };
template <class T, class V> using rebind_simd_t = typename rebind_simd<T, V>::type;
```

---

modify §8.2.2

```
template <class T, size_t N> struct abi_for_size { using type = see below; };
template <class T, size_t N, class... Abis> struct abi_for_size { using type = see below; };
```

5 The member `type` shall be omitted unless

- `T` is a cv-unqualified type, and
- `T` is a vectorizable type, and
- `simd_abi::fixed_size<N>` is supported (see [simd.abi]), and
- every type in the `Abis` pack is an ABI tag.

6 Where present, the member typedef `type` shall name an ABI tag type that satisfies

- `simd_size_v<T, type> == N`, and
- `simd<T, type>` is default constructible (see [simd.overview]),

`simd_abi::scalar` takes precedence over `simd_abi::fixed_size<1>`. The precedence of implementation-defined ABI tags over `simd_abi::fixed_size<N>` is implementation-defined. [*Note:* It is expected that implementation-defined ABI tags can produce better optimizations and thus take precedence over `simd_abi::fixed_size<N>`. Implementations may want to base the choice on `Abis`, but may also ignore the `Abis` arguments. — *end note*]

```
template <class T, class V> struct rebind_simd { using type = see below; };
```

7 The member `type` shall be omitted unless

- `T` is a cv-unqualified type, and
- `T` is a vectorizable type, and
- `V` is either `simd<U, Abi0>` or `simd_mask<U, Abi0>`, where `U` and `Abi0` are deduced from `V`.

8 Where present, the member typedef `type` shall name `simd<T, Abi1>` if `V` is `simd<U, Abi0>` or `simd_mask<T, Abi1>` if `V` is `simd_mask<U, Abi0>`. `Abi1` is equal to `abi_for_size_t<T, simd_size_v<U, Abi0>, Abi0>`.

If `resize_simd` is accepted, add the following right after `rebind_simd_t`:

modify §8.2

```
template <class T, class V> using rebind_simd_t = typename rebind_simd<T, V>::type;

template <int N, class V> struct resize_simd { using type = see below; };
template <int N, class V> using resize_simd_t = typename resize_simd<N, V>::type;
```

And the following after paragraph 8 in §8.2.2:

modify §8.2.2

```
template <int N, class V> struct resize_simd { using type = see below; };
```

9 The member `type` shall be omitted unless

- `N > 0`, and
- `V` is either `simd<T, Abi0>` or `simd_mask<T, Abi0>`, where `T` and `Abi0` are deduced from `V`.

10 Where present, the member typedef `type` shall name `simd<T, Abi1>` if `V` is `simd<T, Abi0>` or `simd_mask<T, Abi1>` if `V` is `simd_mask<T, Abi0>`. `Abi1` is equal to `abi_for_size_t<T, N, Abi0>`.

A BIBLIOGRAPHY

[P0214R8] Matthias Kretz. *P0214R8: Data-Parallel Vector Types & Operations*. ISO/IEC C++ Standards Committee Paper. 2018. URL: https://wg21.link/p0214r8.

[P0820R1] Tim Shen. *P0820R1: Feedback on P0214R5*. ISO/IEC C++ Standards Committee Paper. 2017. URL: https://wg21.link/p0820r1.