Finding the right set of traits for \texttt{simd<T>}

\textbf{ABSTRACT}

This paper makes the set of traits for \texttt{simd<T>} more complete.

\section*{CONTENTS}

1 \textbf{Introduction} \hfill 1
2 \textbf{Motivation} \hfill 1
3 \textbf{Proposed Wording} \hfill 2
4 \textbf{Changelog} \hfill 3
5 \textbf{Straw Polls} \hfill 3
A \textbf{Bibliography} \hfill 4
1 INTRODUCTION

[N4744] defines the trait `simd_abi::deduce<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)

<table>
<thead>
<tr>
<th>SF</th>
<th>F</th>
<th>N</th>
<th>A</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The poll result implies that SG1 prefers users to be able to spell out the ABI tags that are determined as return types.

2 MOTIVATION

As Shen [P0820R1] shows, there is a use case for deducing an ABI tag type from a `value_type`, a width, and additionally zero or more “input” ABI tags. The latter tells the deduction logic what ABI tags are used in the input types to produce an object of the requested `value_type` and width. This enables an implementation design choice of staying within a certain SIMD register subset.

From the user’s perspective, the ABI tag deduction is most often necessary in the following two cases:

• Given a certain `simd` type, what is the best `simd` type for a different `value_type` (e.g. mixed precision calculations).

• Given a certain `simd` type, what is the best `simd` type for a different width (e.g. split, concat, shuffle).

Therefore, I propose to

1. extend `simd_abi::deduce` to consider input ABI tags in its decision,

2. 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`, and

3. introduce a new trait `resize_simd<N, V>`, which deduces a `simd<T, Abi>` instantiation from a given `simd` type `V` with `value_type T` and requested width `N`. 
Apply the following change to the Parallelism TS 2 [N4744]:

```cpp
// modify [parallel.simd.synopsis]
template <class T, size_t N> struct deduce { using type = see below; };  
template <class T, size_t N, class... Abis> struct deduce { using type = typename deduce<T, N, Abis...>::type; }

// add to [parallel.simd.synopsis]
inline constexpr size_t memory_alignment_v = memory_alignment<T, U>::value;

// modify [parallel.simd.abi]
template <class T, size_t N> struct deduce { using type = see below; };  
template <class T, size_t N, class... Abis> struct deduce { using type = see below; };  

12 The member type is present if and only if
• T is a vectorizable type, and
• simd_abi::fixed_size<N> is supported (see 9.2.1), and
• every type in the Abis pack is an ABI tag.

13 Where present, the member typedef type names an ABI tag type that satisfies
• simd_size_v<T, type> == N, and
• simd<T, type> is default constructible (see 9.3.1),
If N is 1, the member typedef type is simd_abi::scalar. Otherwise, if there are multiple ABI tag types
that satisfy the constraints, the member typedef type is implementation-defined. [ Note: It is expected
that extended ABI tags can produce better optimizations and thus are preferred over simd_abi::fixed_size<N>. Implementations can base the choice on Abis, but can also ignore the Abis arguments. — end note ]

// add at the end of [parallel.simd.traits]
template <class T, class V> struct rebind_simd { using type = see below; };  

15 The member type is present if and only if
• V is either simd<U, Abi0> or simd_mask<U, Abi0>, where U and Abi0 are deduced from V, and
• T is a vectorizable type, and
  • simd_abi::deduce<
      T, simd_size_v<
      U, Abi0>, Abi0>
      has a member type type.

Let Abi1 denote the type
deduce_t<T, simd_size_v<U, Abi0>, Abi0>. Where present, the member
typedef type names

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

The member type is present if and only if
• V is either simd<T, Abi0> or simd_mask<T, Abi0>, where T and Abi0 are deduced from V, and
• simd_abi::deduce<
      T, N, Abi0>
      has a member type type.

Let Abi1 denote the type
deduce_t<T, N, Abi0>. Where present, the member typedef type names

```cpp
simd<T, Abi1>
```

if V is simd<T, Abi0> or
```cpp
simd_mask<T, Abi1>
```
if V is simd_mask<T, Abi0>.

---

## 4 Changelog

### 4.1 changes from r1

Previous revision: [P0964R1].

• Editorial changes to the wording: “denotes” instead of “identify”, remove incorrect “shall”.

### 4.2 changes from r0

Previous revision: [P0964R0].

• Adjusted to changes between [P0214R8] and [N4744].

• Make resize_simd a non-optional part of the requested changes (after SG1 discussion).

• Update motivation after resolving different naming preferences with Tim.

---

## 5 Straw Polls

Poll: Proceed to LEWG?
→ unanimous consent
A Bibliography

5.2 lewg at rapperswil 2018

Poll: Proceed to LWG?

<table>
<thead>
<tr>
<th>SF</th>
<th>F</th>
<th>N</th>
<th>A</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A Bibliography


