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

\section*{ABSTRACT}

This paper discusses the set of traits we want to ship with \texttt{simd<T>}.

\section*{CONTENTS}

1 Introduction 1
2 Motivation 1
3 Proposed Wording 1
A Bibliography 3
1

Kretz [P0214R8] defines the trait \texttt{abi\_for\_size<T, N>}, allowing users to find an “implementation-recommended” ABI tag for a given \texttt{value\_type} and number of elements. Shen [P0820R1] discusses a use for considering involved ABI tags in the “recommendation”. SG1 polled in Albuquerque about

\begin{tabular}{|c|c|c|c|c|}
\hline
SF & F & N & A & SA \\
\hline
1 & 7 & 7 & 0 & 0 \\
\hline
\end{tabular}

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 \texttt{rebind\_abi<T, N, Abis...>}. 

2

MOTIVATION

I believe the name \texttt{rebind\_abi} in Shen [P0820R1] is misleading, since no rebinding is taking place, but rather a type for implementing a rebind of a given \texttt{simd<T, Abi>} to a different \texttt{value\_type U} is made possible. Therefore, I propose to

1. not rename the \texttt{abi\_for\_size} trait in Kretz [P0214R8], and
2. extend \texttt{abi\_for\_size} to consider input ABI tags in its decision, and
3. introduce a new trait \texttt{rebind\_simd<U, V>}, which deduces a \texttt{simd<U, Abi>} instantiation from a given \texttt{simd} \texttt{type V} and requested \texttt{value\_type U}.

In addition to \texttt{rebind\_simd}, SG1 should consider whether a \texttt{resize\_simd\_t} trait should be added. \texttt{resize\_simd\_t<N, simd <T, Abi0> is an alias for a simd<T, Abi> so that simd\_size\_v<T, Abil> == N, and resize\_simd\_t<N, simd\_mask <T, Abi0> is an alias for a simd\_mask<T, Abil> so that simd\_size\_v<T, Abil> == 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:

\begin{verbatim}
modify §8.2
\end{verbatim}
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 ]

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:

```cpp
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 \( \text{simd}<T, \text{Abi0}> \) or \( \text{simd}_{-}\text{mask}<T, \text{Abi0}> \), where \( T \) and \( \text{Abi}0 \) are deduced from \( V \).

Where present, the member typedef type shall name \( \text{simd}<T, \text{Abi1}> \) if \( V \) is \( \text{simd}<T, \text{Abi0}> \) or \( \text{simd}_{-}\text{mask}<T, \text{Abi1}> \) if \( V \) is \( \text{simd}_{-}\text{mask}<T, \text{Abi0}> \). \text{Abi1} \) is equal to \( \text{abi}_\text{for}\_\text{size}_t<T, N, \text{Abi0}> \).

---

### BIBLIOGRAPHY
