Concat and Split on simd<> objects

Document Number	P1118R0
Date	2018-06-08
Reply-to	Tim Shen <timshen91@gmail.com> Matthias Kretz <m.kretz@gsi.de>
Audience	LWG

Abstract

We propose improvements on the `concat()` and `split()` functions in the Parallelism v2 simd<> library.

`concat()` doesn't support `std::array`

We propose it for being consistent with `split()`. Users may take the array from split(), do some operations, and concat back the array. It'd be hard for them to use the existing variadic parameter `concat()`.

Wording

Add the following to [parallel.simd.synopsis]:

```
template<class T, class... Abis>
simd_mask<T, simd_abi::deduce_t<T, (simd_size_v<T, Abis> + ...)>>
concat(const simd_mask<T, Abis>&...);

template <class T, class Abi, size_t N>
resize_simd<simd_size_v<T, Abi> * N, simd<T, Abi>>
concat(const array<simd<T, Abi>, N>& arr) noexcept;

template <class T, class Abi, size_t N>
resize_simd<simd_size_v<T, Abi> * N, simd_mask<T, Abi>>
concat(const array<simd_mask<T, Abi>, N>& arr) noexcept;
```

Add the following after [parallel.simd.casts] p28:

```
template <class T, class Abi, size_t N>
resize_simd<simd_size_v<T, Abi> * N, simd<T, Abi>>
concat(const array<simd<T, Abi>, N>& arr) noexcept;

template <class T, class Abi, size_t N>
```

```
resize_simd<simd_size_v<T, Abi> * N, simd_mask<T, Abi>>
concat(const array<simd_mask<T, Abi>, N>& arr) noexcept;
```

29 *Returns:* A data-parallel object, the i^{th} element of which is initialized by `arr[i / simd_size_v<T, Abi>][i % simd_size_v<T, Abi>]`.

`split()` is sometimes verbose to use

It is sometimes verbose and not intuitive to use the array version of `split()`, e.g.

```
  template <typename T, typename Abi>
  void Foo(simd<T, Abi> a) {
    auto arr = split<simd<T, fixed_size<a.size() / 4>>>(a);
    // auto arr = split_by<4>(a) is much better.
    /* … */
  }
```

and it's even more verbose for non-`fixed_size` types. We propose to add `split_by()` that splits the input by an `n` parameter.

Wording

Add the following to [parallel.simd.synopsis]:

```
template <class V, class Abi>
array<V, simd_size_v<typename V::value_type, Abi> / V::size()> split(const
simd_mask<typename V::value_type, Abi>&);

template <size_t N, class T, class A>
array<resize_simd<simd_size_v<T, A> / N, simd<T, A>>, N>
split_by(const simd<T, A>& x) noexcept;

template <size_t N, class T, class A>
array<resize_simd<simd_size_v<T, A> / N, simd_mask<T, A>>, N>
split_by(const simd_mask<T, A>& x) noexcept;
```

Add the following after [parallel.simd.casts] p26:

```
template <size_t N, class T, class A>
array<resize_simd<simd_size_v<T, A> / N, simd<T, A>>, N>
split_by(const simd<T, A>& x) noexcept;

template <size_t N, class T, class A>
array<resize_simd<simd_size_v<T, A> / N, simd_mask<T, A>>, N>
split_by(const simd_mask<T, A>& x) noexcept;
```

27 *Returns:* An array `arr`, where `arr[i][j]` is initialized by `x[i * (simd_size_v<T, A> / N) + j]`.

28 *Remarks:* The functions shall not participate in overload resolution unless `simd_size_v<T, A>` is an integral multiple of `N`.