Document Number: P3691R0
Date: 2025-05-19
Reply-to: Matthias Kretz <m.kretz@gsi.de>,
Abhilash Majumder <abmajumder@nvidia.com>,
Bryce Adelstein Lelbach <brycelelbach@gmail.com>,
Daniel Towner <daniel.towner@intel.com>,
Ilya Burylov <iburylov@nvidia.com>,
Mark Hoemmen <mhoemmen@nvidia.com>,
Ruslan Arutyunyan <ruslan.arutyunyan@intel.com>
Audience: LEWG
Target: C++26

Reconsider naming of the namespace for "std::simd"

ABSTRACT

There has been negative feedback on the names we chose for `std::datapar::basic_simd` and `std::datapar::basic_simd_mask`. The pushback is against the name of the namespace. This paper does *not reconsider the namespace itself*, it only reconsiders our naming options. Much of the discussion revolves around the question of whether it is acceptable to repeat a namespace name for an entity inside that namespace. This paper looks at this question and aims for a "policy-like" LEWG answer (independent of SIMD). Among authors there was no clear consensus for one better alternative. We therefore invite reflector discussion, as we expect many strong opinions on the matter (naming is hard and most importantly not always objective). For now, the paper recommends to rename the namespace to `simd` and subsequently drop the `simd_` part from `basic_simd_mask` and `simd_mask`.

CONTENTS

1 CHANGELOG

(placeholder)

2 STRAW POLLS

(placeholder)

3 CRITICISM AND DISCUSSION ON THE CURRENT NAME

It seems that the namespace name `datapar` doesn't appeal to some because "datapar" appears as a newly *invented name* and that's not a good fit for a namespace in the standard library. A namespace name should describe the contents of the namespace without requiring an explanation.

"Datapar" was orignally invented (and used for some time) for the `simd<T>` type leading up to the Parallelism TS 2. The rationale behind the name "datpar" was that the `simd<T>` type does more than just SIMD: it is an abstraction for expressing data-parallelism. As such "datapar" describes a superset of what SIMD is. This is why the combination `datapar::simd<T>` may look strange to some: SIMD is one concrete implementation of data parallelism, but the namespace name implies there is more. The combination of "datapar" and "SIMD" is the strange part about the status quo. Saying "data-parallel vector [of `int`]" / "data-parallel `int`" or a saying "SIMD vector [of `int`]" / "SIMD `int`" is fine. But a "data-parallel SIMD [vector] [of `int`]" is a strange thing to say/hear/read.

In the last LEWG discussion on [P3287R2], an alternative spelling for `datapar`, `dataparallel` was raised and voted on (with an even split on either). So what if we were to spell it out to `dataparallel` or `data_parallel`? Then we still have the issue from above, that "data-parallel SIMD" sounds strange because SIMD that is not data-parallel doesn't exist, and data parallelism is more than just SIMD. But more importantly, the term "data-parallel" is an adjective. We only ever used an adjective for "experimental" APIs:

`std::chrono`	`std::contracts`	`std::execution`
`std::experimental` (the exception)	`std::filesystem`	`std::linalg`
`std::literals`	`std::numbers`	`std::placeholders`
`std::pmr`	`std::ranges`	`std::regex_constants`
`std::rel_ops` (deprecated)	`std::this_thread`	`std::views`

However, the intent for `datapar` (in [P3287R2]) was to be an abbreviation of "Data-parallel types", the subclause heading in the current working draft. There is precedent with e.g. the `std::pmr` and `std::linalg` namespaces of using an abbreviation. The name "pmr" also needs an explanation. Does that make "pmr" a bad name or does it help to justify "datapar"?

Consequently, the argument that `std::datapar` has to be spelled out in a longer form as the adjective `std::data_parallel`, isn't necessarily correct. If instead we were to spell it out as `std::data_parallel_types`[1] or `std::dpt`, that argument doesn't hold anymore.

Thus, it appears that any argument for replacing the `datapar` status quo with `simd` has to be that *`datapar` is unclear and the clearer alternatives are too long*.

One argument that was used against `std::simd::simd` (and thus the only reason we considered a different namespace name) is the ambiguity between namespace and class name and whether one is looking at a constructor (`T::T`). What had not been mentioned in these discussions is that `simd` is actually not a class template name but alias template.[2] The default constructor, for example, therefore is spelled `std::simd::basic_simd::basic_simd()`.[3] Nevertheless, the repetition of "simd" looks odd and might be confusing to humans and simple tools. Examples of libraries that do something similar are Boost.Flyweight and Boost.Histogram:

```
namespace boost
{
  namespace flyweights
  {
    template</*...*/> class flyweight;
  }
  using flyweights::flyweight;

  namespace histogram
  {
    template<typename Axes, typename Storage> class histogram;
  }
}
```

So this has been done before and passed through Boost review. Some of us believe the fear, uncertainty, and doubts about repeating the name of the namespace in the alias template are possibly no more than that; and in reality the "problem" is little to non-existent.

4 A SURVEY OF EXISTING SIMD LIBRARIES

Interestingly, no C++ library uses the term "SIMD" in the name of the (unqualified) type. There are[4] two libraries that used "SIMD" in the namespace. Rust, however, uses `Simd` for the type name and

1 Other variations: `std::data_parallel_execution`, `std::data_parallel_algorithms`, `std::data_parallel_numerics`, `std::data_parallelism`
2 Granted, this makes little difference for some of us.
3 In diagnostic output it is typically spelled `std::simd::basic_simd<T, Abi>::basic_simd() [with T = ..., Abi = ...]`
4 "were", `boost::simd` doesn't exist anymore

just plain Mask for the mask type. Rust has no need for an additional namespace scope because all operations on Simd are implemented as operators or member functions[5].

4.1 Agner Fog's C++ Vector Class Library

The library does not define class templates but classes like Vec4f, Vec16c, Vec2uq, …The corresponding mask types have an additional b suffix in their name.

```
class Vecf4;   // ~simd<float, 4>
class Vecf4b;  // ~simd_mask<float, 4>
```

4.2 "Boost" SIMD

```
namespace simd
{
  template <class T> struct pack; // ~simd
  template <> struct pack<bool>;  // ~simd_mask
}
```

4.3 E.V.E

```
namespace eve
{
  template <class T, /*Cardinal*/> struct wide; // ~ simd
  template <class T> struct logical;            // ~ simd_mask
}
```

4.4 Highway

```
namespace hwy
{
  template <class T, size_t N> class Vec128;  // ~ simd
  template <class T, size_t N> class Mask128; // ~ simd_mask
  // ...
}
```

4.5 Vc

```
namespace Vc
{
  template <class T, class Abi> class Vector;  // ~ basic_simd
  template <class T, class Abi> class Mask;    // ~ basic_simd_mask
  using double_v = Vector<double>;
```

5 Rust does not enable SIMD-generic programming

```
  using double_m = Mask<double>;
  using int_v = Vector<int>;
  using int_m = Mask<int>;
  // ...
}
```

4.6 xsimd

```
namespace xsimd
{
  template <class T, class A> class batch;       // ~ basic_simd
  template <class T, class A> class batch_bool;  // ~ basic_simd_mask
}
```

4.7 Other programming languages

- Rust's experimental std::simd:

  ```
      pub struct Simd<T, const N: usize> ...; // ~ simd
      pub struct Mask<T, const N: usize> ...; // ~ simd_mask
  ```

- C#: `System.Numerics.Vector<T>`

- Java Panama Vector API:
  `jdk.incubator.vector.VectorSpecies<Float>`
  `jdk.incubator.vector.VectorMask<Float>`

- Swift SIMD module:
  `protocol SIMD<Scalar>`
  `struct SIMD8<Scalar>`
  `struct SIMDMask<Storage>`

- Julia SIMD.jl: `xs = Vec4,Float64(1)`

5 Names considered

The name of the namespace and the name of the class (and alias) templates need to be considered as a whole. It is not helpful to consider a namespace name or class name in isolation. The names of the class/alias templates also basically never appear without at least one template argument in user code. Consequently, the following presentation will include the namespace, class template, and first template argument.

5.1 options for basic_simd

class template alias template	obvious criticism

`std::simd::basic_simd<int>`
`std::simd::simd<int>`
	repetitive and human-ambiguous

`std::dpt::basic_simd<int>`
`std::dpt::simd<int>`
	what is “dpt”? (like what is “pmr”?)

`std::datapar::basic_simd<int>`
`std::datapar::simd<int>`
	see Section 3

`std::dataparallel::basic_simd<int>`
`std::dataparallel::simd<int>`
	see Section 3

`std::data_parallelism::basic_simd<int>`
`std::data_parallelism::simd<int>`
	too long? also see Section 3

`std::simd::basic_batch<int>`
`std::simd::batch<int>`	

`std::simd::basic_number<int>`
`std::simd::number<int>`
	it’s not a number, but set of numbers

`std::simd::basic_numbers<int>`
`std::simd::numbers<int>`
	we already have `std::numbers::*`
	what are “SIMD numbers”?

`std::simd::basic_pack<int>`
`std::simd::pack<int>`
	C++ already has parameter packs;
	another existing meaning: packed structs

`std::simd::basic_value<int>`
`std::simd::value<int>`
	too many variables are named `value`

`std::simd::basic_vec<int>`
`std::simd::vec<int>`
	sounds like a container

`std::simd::basic_vector<int>`
`std::simd::vector<int>`
	sounds even more like a container

`std::simd::basic_wide<int>`
`std::simd::wide<int>`
	like in `wchar_t`?
	“SIMD wide” and “SIMD basic wide” need an explanation

`std::simd::basic_chunk<int>`
`std::simd::chunk<int>`
	this is not a chunk out of a SIMD register

`std::datapar::basic_chunk<int>`
`std::datapar::chunk<int>`
	where did the “SIMD” name go?

`std::dataparallel::basic_chunk<int>`
`std::dataparallel::chunk<int>`
	ditto

`std::dataparallel::basic_numbers<int>`
`std::dataparallel::numbers<int>`
	long; any abbreviation becomes unclear

5.2 OPTIONS FOR BASIC_SIMD_MASK

class template alias template	obvious criticism

`std::simd::basic_mask<4>`
`std::simd::mask<int>`	

`std::simd::basic_simd_mask<4>`
`std::simd::simd_mask<int>`
	repetitive

`std::dpt::basic_mask<4>`
`std::dpt::mask<int>`
	relation to `dpt::simd` only via namespace

`std::dpt::basic_simd_mask<4>`
`std::dpt::simd_mask<int>`
	repetitive like `dpt::simd`

`std::datapar::basic_mask<4>`
`std::datapar::mask<int>`
	LEWG already voted against this

`std::datapar::basic_simd_mask<4>`
`std::datapar::simd_mask<int>`
	repetitive in a different way

`std::dataparallel::basic_simd_mask<4>`
`std::dataparallel::simd_mask<int>`
	ditto

`std::simd::basic_logical<4>`
`std::simd::logical<int>`	

`std::simd::basic_batch_bool<4>`
`std::simd::batch_bool<int>`	

`std::simd::basic_simd_bool<4>`
`std::simd::simd_bool<int>`	

`std::simd::basic_boolean<4>`
`std::simd::boolean<int>`	

6 DISCUSSION

There seems to be a strong desire to have “SIMD” in the name somewhere. This desire seems to be about findability and about trying not to invent a new name where the industry already recognizes an existing name. (Potentially also about buzzword compliance and delivering on a name we have been communicating before?)

If we were to use a different name for the namespace and class/alias templates, is the namespace or the class/alias name more important and thus needs to use the “SIMD” name? The class/alias name is the identifier that will always appear in the source code so choosing a class name which is ambiguous once the namespace is removed (e.g., with a using statement) may obscure the codes intent. We would end up with variables with types like `pack<int>`, `vec<int>`, `batch<int>`, `wide<int>`, and so on, and all hint of their SIMD behaviour is lost. The only viable alternative term that could potentially stand on its own (in terms of hinting at behavior) is “vector”. But that train has left the station decades ago.

On the other hand, if we consider the group of types and functions in the “SIMD” namespace to be a “SIMD-only” library (i.e., no other data-parallelism abstractions unrelated to the `simd` type), then shouldn’t the namespace have the “SIMD” name? If we expect common practice to use fully qualified names or use of a namespace alias (`namespace simd = std::simd`; there’s no other conceivable abbreviation other than maybe `namespace ss = std::simd`), then `simd` is already always part of the name. Consequently, we could then choose a name “SIMD <something>” to avoid the perceived ambiguity of `simd::simd`.

On the topic of ambiguity, does anyone feel one of the following functions is problematic? Is there any example that can create “visual confusion” to a human reader? I specifically combined `reduce` and the static data member `size` into one line of (non-sensical) code because I suspect that’s the most contentious syntax similarity.

```cpp
int f1() {
  std::simd::simd v = std::array {1, 2, 3, 4};
  return std::simd::reduce(v) + std::simd::simd<int>::size();
}

int f2() {
  namespace simd = std::simd;
  simd::simd v = std::array {1, 2, 3, 4};
  return simd::reduce(v) + simd::simd<int>::size();
}

int f3() {
  using std::simd::simd;
  simd v = std::array {1, 2, 3, 4};
  return std::simd::reduce(v) + simd<int>::size();
}

int f4() {
  using std::simd::simd;
  simd v = std::array {1, 2, 3, 4};
  return reduce(v) + simd<int>::size();
}

int f5() {
  using namespace std;
  simd::simd v = std::array {1, 2, 3, 4};
  return simd::reduce(v) + simd::simd<int>::size();
}

int f6() {
  using namespace std::simd;
  simd v = std::array {1, 2, 3, 4};
  return reduce(v) + simd<int>::size();
}
```

Naming the namespace of the `reduce` function after the type reinforces the connection between function and type it operates on, which improves clarity and cohesion. Whereas the status-quo of `std::datapar::reduce` used in isolation conveys less about its intended argument. The counter-argument here is that `simd::reduce` looks like a call to a static member function inside `simd`. (It can't be, though, because `simd` is an alias *template* and thus requires a template argument.)

7 MAIN POSITIONS

There seem to be three main positions that we were able to identify. These positions inform where the "over my dead body" and "any of these is fine" opinions originate.

7.1 ACCEPTABLE TO REPEAT NAME IN NAMESPACE AND TYPE

People who take no issue[6] with repeating the name of the namespace for a type inside that namespace tend to favor `std::simd::simd<T>` / `std::simd::mask<T>`. Those who take issue with such repetition fall into one of the following two categories.

7.2 THE NAMESPACE NAME CARRIES MORE WEIGHT

People on this position expect the namespace name to always be visible in code and diagnostics and therefore do the heavy lifting. Since there seems to be consensus on using the term SIMD, the namespcae name thus should be "simd". The name of the class/alias consequently needs to be something that reads as "SIMD <foo>", where "<foo>" must not contain "SIMD". Favored outcomes are `std::simd::pack`, `std::simd::vec`, `std::simd::batch`, `std::simd::wide`, …

7.3 THE CLASS/ALIAS NAME CARRIES MORE WEIGHT

Poeple on this position consider the namespace name optional in code. Therefore, the class/alias name must stand on its own. Again, with the consensus on using "SIMD", the class/alias name should be "simd" and the namespace name should be something else. Favored outcomes are `std::datapar::simd`, `std::dpt::simd`, `std::data_parallelism::simd`, …

8 CONCLUSION

We recommend to go back to the advertised "std::simd" name by naming the namespace `std::simd`. In order to avoid the unnecessary duplication in `std::simd::simd_mask` we also recommend to then rename `std::simd::basic_simd_mask` to `std::simd::basic_mask` and `std::simd::simd_mask` to `std::simd::mask`.

6 though, typically there's still a preference for different names — if there's an obvious set of names

Before further discussion or taking that poll, we would prefer a policy-like, general discussion about whether name repetition between namespace and enclosed entities should be considered bad practice (or outright banned). The outcome of that poll should inform us where `std::simd` can or cannot go.

If LEWG decides to ban name repetition between namespace and enclosed type, then we need to reconsider whether `std::datapar::simd` / `std::datapar::simd_mask` is really still better than `std::simd::pack`[7] / `std::simd::mask`.

9 WORDING

9.1 FEATURE TEST MACRO

No feature test macro is added or bumped.

9.2 ORDERING CONSTRAINTS ON LWG MOTIONS

TBD.

9.3 INSTRUCTIONS TO THE EDITOR

In 29.10 Data-parallel types [simd],

1. change every occurrence of namespace std::~~datapar~~simd {;
2. change every occurrence of using ~~datapar~~simd::;
3. change every occurrence of basic_~~simd_~~mask;
4. change every occurrence of ~~simd_~~mask.

A BIBLIOGRAPHY

[P3287R2] Matthias Kretz. *Exploration of namespaces for std::simd*. ISO/IEC C++ Standards Committee Paper. 2024. URL: https://wg21.link/p3287r2.

7 or one of the other alternatives mentioned; to be discussed on the reflector