Document	P1413R0
Date	2019-01-17
Author	CJ Johnson <johnsoncj@google.com>
Audience	Library Evolution Working Group (LEWG)

A safer interface for std::aligned_storage

Background

As a developer on the Abseil team at Google, I've had the opportunity to learn about and work on generic container utilities in C++. One of the more interesting types provided by the standard for this use case, in my opinion, is `std::aligned_storage`. While it is true that strict aliasing is a concern, in practice, placement-new over an instance does work and is useful for decoupling storage duration from object lifetime.

That said, the API for creating an instance of `std::aligned_storage` is rather unsafe, or at least easy to misuse. Not only is it on the user to pass in a particular size, the alignment parameter has a default value, meaning there's nothing in the type to enforce that the alignment is sufficient for the type with which it is intended to be used. Since the committee has already elected to include `std::aligned_storage` in the standard library, I believe a safer interface for creating class template instances of it should also be provided, making it easier for users of the type to suitibly align their storage buffers for their types.

NOTE: For this paper, the term aligned_storage is used in addition to `std::aligned_storage`. It is intended to highlight the fact that some usage of the construct is of `std::aligned_storage` while others are of types that have the same API and behavior as `std::aligned_storage` but are defined outside of the standard library, for various reasons (notably including code that targets versions of the language older than C++11/14).

Proposed change

Currently the standard provides `std::aligned_storage` and `std::aligned_storage_t`. These templates take in as non-type template parameters at least one `size_t` for the size of the storage and an optional second `size_t` for the alignment of the storage. If the second parameter is not provided, they default to an implementation-defined alignment value.

In addition to these two symbols, the standard library should provided two more symbols in the form of typedefs that take in a single template type parameter and, on behalf of the user, deduce the size and alignment of that type, passing in the values to `std::aligned_storage`. The symbols should be `std::aligned_storage_for` and `std::aligned_storage_for_t`. Like `std::aligned_storage` and `std::aligned_storage_t`, they should be available in the `<type_traits>` header of the standard library. See the below code for an example implementation.

```
// Assume `std::aligned_storage` is visible

namespace std {

template <typename T>
using aligned_storage_for
  = std::aligned_storage<sizeof(T), alignof(T)>;

template <typename T>
using aligned_storage_for_t
  = typename std::aligned_storage_for<T>::type;

}  // `namespace std`
```

Precedent for the spelling

The spelling `*_for` is not new for the standard when it comes to typedefs that help in the creation of `*`-named types. Already found in the standard library is `std::index_sequence_for` which aids in template-instantiating `std::index_sequence` (which is itself a typedef of `std::integer_sequence` with `size_t` specified as the type).

Further, recently added by Facebook, the library Folly provides `folly::aligned_storage_for_t` [1] which behaves in the same way as this paper's proposed `std::aligned_storage_for_t`.

Precedent for the behavior

Utilities for taking in a type and deducing the size and alignment for aligned_storage's template arguments already exist, just not in the standard library. The usecase is common enough that library maintainers have found the abstraction useful.

As noted above, there is `folly::aligned_storage_for_t` [1] which provides this abstraction using the `*_t` naming convention (where the dependent `::type` is already resolved).

In Boost there are two definitions of aligned_storage that take in a single type and use the provided type's size and alignment for the aligned_storage, avoiding the need for the user to pass them in. [2][3]

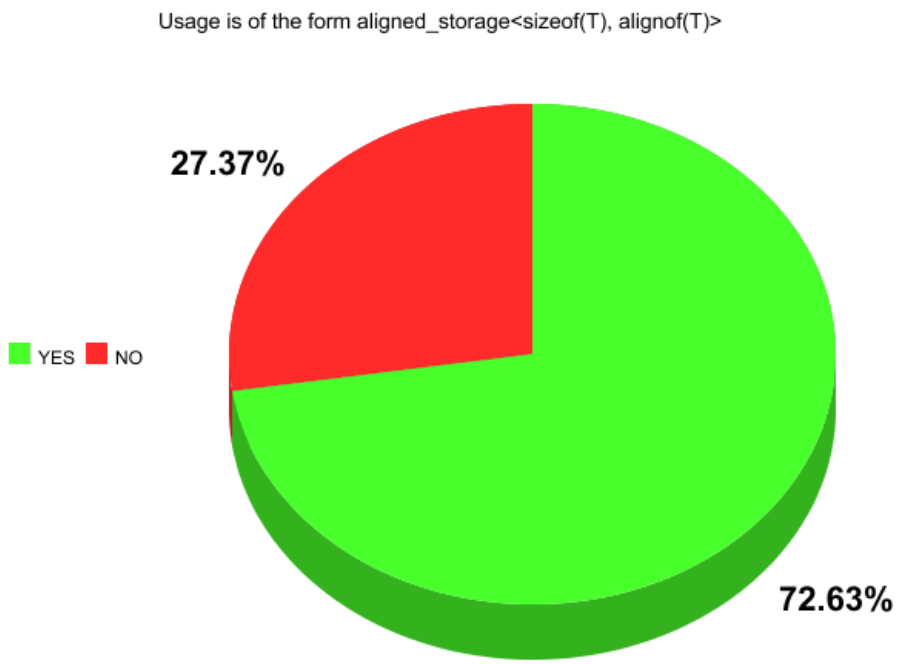
Boost also includes `pod_value_holder` which has the same behavior as the two above-mentioned aligned_storage definitions but goes by a different name. [4]

Existing usage

The type aligned_storage is niche, being most useful for container types provided by utility libraries. Thus, to find how it is used in practice, I scraped the implementation details of Folly, Boost and Abseil. Guideline Support Library, being much smaller in scope, does not include any use of aligned_storage.

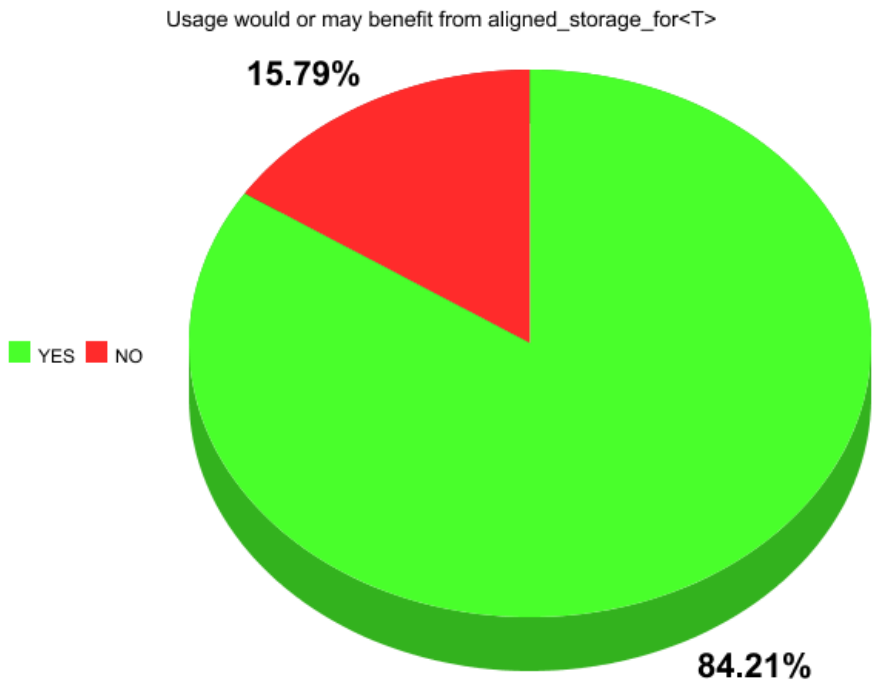
NOTE: All metrics are gathered from the source at head of the official Github repository for each library as of the time of writing this paper.

Across the three libraries inspected, there are 95 distinct uses of aligned_storage. Nearly **73%** (69/95) of those uses are of the form `aligned_storage<sizeof(T), alignof(T)>` (or some alternate spelling thereof). The vast majority of aligned_storage's use is simply repeating the same process of taking a type and deducing its size and alignment manually to meet the API.



By library it's **58%** (11/19) for Folly, **75%** (50/67) for Boost and **89%** (8/9) for Abseil. Looking at Boost.Container specifically leaves us with **88%** (23/26), bringing it in line with Abseil. All three libraries slightly or overwhelmingly favor the form `aligned_storage<sizeof(T), alignof(T)>` when using aligned_storage.

In addition to the callsites that already use the sizeof-alignof-T pattern, there are a handful that depend on the default alignment, passing in only a size to aligned_storage. I don't have enough context about these to diagnose if they are bugs, but choosing to not pass in an alignment to aligned_storage is, if nothing else, a red flag. If we choose to consider all of them bugs, the argument could be made that they too would benefit in the form of alignment-safety by being replaced with `std::aligned_storage_for`. Summing the callsites that definitely could use the proposed interface (69) with the ones that may be bugs (11), we get a whopping **84%** (80/95) of all usage of aligned_storage across Folly, Boost and Abseil that would or may benefit from access to `std::aligned_storage_for`.



Additional usage

In addition to the main utility libraries of the world, the `aligned_storage<sizeof(T), alignof(T)>` pattern appears in most use of aligned_storage across the internet. The CppReference `static_vector` example for `std::aligned_storage` features `typename std::aligned_storage<sizeof(T), alignof(T)>::type` at the top. [5] Comments on StackOverflow, when using aligned_storage, commonly follow the same sizeof-alignof-T pattern, as well. [6][7][8][9] The pattern appears in wiki pages [10], blog posts [11][12][13] and in the source of other libraries [14] and languages. [15]

Conclusion

With code of the pattern sizeof-alignof-T being so common, I feel it is justified to include in the standard library an abstraction, `std::aligned_storage_for` and `std::aligned_storage_for_t`, for users to avoid code duplication and enforce that storage buffers be not only sized to a given type but aligned for it as well.

References

[1] `folly::aligned_storage_for_t`

[2] First aligned_storage taking a typename in Boost

[3] Second aligned_storage taking a typename in Boost

[3] `pod_value_holder` in Boost

[4] Gathered data

Library	Is of the form aligned_storage<sizeof(T), alignof(T)>?	Depends on default alignment?	Source
Folly	YES	NO	HazptrHolder.h:220
Folly	YES	NO	HazptrHolder.h:338
Folly	YES	NO	LifoSem.h:125
Folly	YES	NO	small_vector.h:1142
Folly	YES	NO	Replaceable.h:640
Folly	YES	NO	dynamic.h:782
Folly	YES	NO	F14Table.h:567
Folly	YES	NO	F14Policy.h:1208
Folly	YES	NO	UnboundedQueue.h:761
Folly	YES	NO	MPMCQueue.h:1442
Folly	YES	NO	AtomicUnorderedMap.h:366
Folly	NO	YES	Fiber.h:124
Folly	NO	YES	Fiber.h:164
Folly	NO	YES	Function.h:255
Folly	NO	YES	PolyDetail.h:384
Folly	NO	NO	ExceptionWrapper.h:223-224
Folly	NO	NO	small_vector.h:1154-1155
Folly	NO	NO	F14Table.h:330-332
Folly	NO	NO	Memory.h:109
Boost.Accumulators	NO	YES	droppable_accumulator.hpp:249
Boost.Asio	NO	YES	allocator.hpp:49
Boost.Beast	NO	NO	type_traits.hpp:88-90
Boost.Container	YES	NO	advanced_insert_int.hpp:131
Boost.Container	YES	NO	advanced_insert_int.hpp:154
Boost.Container	YES	NO	advanced_insert_int.hpp:291
Boost.Container	YES	NO	advanced_insert_int.hpp:401
Boost.Container	YES	NO	flat_tree.hpp:925
Boost.Container	YES	NO	flat_tree.hpp:937
Boost.Container	YES	NO	flat_tree.hpp:948
Boost.Container	YES	NO	flat_tree.hpp:960
Boost.Container	YES	NO	flat_tree.hpp:997
Boost.Container	YES	NO	flat_tree.hpp:1008
Boost.Container	YES	NO	flat_tree.hpp:1019

Library	Is of the form aligned_storage<sizeof(T), alignof(T)>?	Depends on default alignment?	Source
Boost.Container	YES	NO	flat_tree.hpp:1030
Boost.Container	YES	NO	node_handle.hpp:114-116
Boost.Container	YES	NO	varray.hpp:228-231
Boost.Container	YES	NO	varray.hpp:1072-1073
Boost.Container	YES	NO	varray.hpp:1115-1116
Boost.Container	YES	NO	varray.hpp:1625-1628
Boost.Container	YES	NO	small_vector.hpp:385-386
Boost.Container	YES	NO	tree.hpp:137-138
Boost.Container	YES	NO	list.hpp:79
Boost.Container	YES	NO	slist.hpp:84
Boost.Container	YES	NO	stable_vector.hpp:150-151
Boost.Container	YES	NO	string.hpp:176-177
Boost.Container	NO	NO	copy_move_algo.hpp:1024-1025
Boost.Container	NO	NO	copy_move_algo.hpp:1054-1055
Boost.Container	NO	NO	static_vector.hpp:73
Boost.Coroutine2	YES	NO	pull_control_block_cc.hpp:33
Boost.Coroutine2	YES	NO	pull_control_block_cc.hpp:74
Boost.Fiber	YES	NO	shared_state.hpp:160
Boost.Fiber	YES	NO	buffered_channel.hpp:41
Boost.Fiber	YES	NO	buffered_channel.hpp:536
Boost.Fiber	YES	NO	unbuffered_channel.hpp:585
Boost.Flyweight	YES	NO	archive_constructed.hpp:70
Boost.Flyweight	NO	NO	key_value.hpp:157-164
Boost.Foreach	NO	YES	foreach.hpp:611
Boost.Geometry	YES	NO	varray.hpp:165-168
Boost.Geometry	YES	NO	varray.hpp:1052
Boost.Geometry	YES	NO	varray.hpp:1101
Boost.Geometry	YES	NO	varray.hpp:1613-1616
Boost.Geometry	YES	NO	serialization.hpp:50
Boost.Hana	NO	YES	traits.hpp:176
Boost.Hana	NO	NO	traits.hpp:169
Boost.Hof	NO	YES	construct.hpp:106
Boost.Interprocess	NO	NO	offset_ptr.hpp:69-72

Library	Is of the form aligned_storage<sizeof(T), alignof(T)>?	Depends on default alignment?	Source
Boost.Iostreams	YES	NO	optional.hpp:108
Boost.Lockfree	NO	NO	spsc_queue.hpp:429-431
Boost.Log	YES	NO	thread_id.cpp:131
Boost.Log	YES	NO	threadsafe_queue.hpp:78
Boost.MultiIndex	YES	NO	archive_constructed.hpp:74
Boost.MultiIndex	YES	NO	seq_index_ops.hpp:134-137
Boost.MultiIndex	YES	NO	seq_index_ops.hpp:141-149
Boost.Optional	YES	NO	old_optional_implementation.hpp:87
Boost.Optional	YES	NO	optional.hpp:120
Boost.Parameter	NO	YES	maybe.hpp:37-39
Boost.PolyCollection	YES	NO	value_holder.hpp:61
Boost.Pool	YES	NO	singleton_pool.hpp:196
Boost.Python	NO	YES	referent_storage.hpp:59-61
Boost.Serialization	YES	NO	stack_constructor.hpp:38-41
Boost.Signals2	NO	NO	auto_buffer.hpp:1061-1063
Boost.Spirit	YES	NO	static.hpp:103-104
Boost.Unordered	YES	NO	implementation.hpp:751-752
Boost.Unordered	YES	NO	implementation.hpp:2753-2754
Boost.Utility	YES	NO	value_init.hpp:93
Boost.Variant	NO	NO	variant.hpp:366-369
Abseil	YES	NO	mutex_nonprod.inc:255
Abseil	YES	NO	fixed_array.h:399-400
Abseil	YES	NO	inlined_vector.h:1248-1249
Abseil	YES	NO	inlined_vector.h:1250-1251
Abseil	YES	NO	low_level_alloc.cc:222-223
Abseil	YES	NO	raw_hash_set.h:1156-1157
Abseil	YES	NO	raw_hash_set.h:1626-1627
Abseil	YES	NO	raw_hash_set.h:542-543
Abseil	NO	NO	symbolize_win32.inc:54-55

[5] `std::aligned_storage` on CppReference.com

[6] StackOverflow comment by user743382

[7] StackOverflow post by Andrew Tomazos

[8] StackOverflow post by nwp

[9] StackOverflow comment by Andrew

[10] "More C++ Idioms/Nifty Counter" on WikiBooks.org

[11] "Pimpl idiom without dynamic memory allocation" blog post

[12] "À propos de l'alignement en mémoire" blog post

[13] "Strange behavior of std::aligned_storage" blog post

[14] `WTF::NeverDestroyed` on opensource.apple.com

[15] "Multiple Producer Single Consumer Lockless Q" on haskell.org