P2667R0

Vector Allocator For SBO

Bengt Gustafsson

Kona - 2022

Presentation contents

- Rationale
- Rules for vector to follow
- A few new allocator traits
- Some allocators to implement static and sbo vectors
- Convenient type aliases
- Extension of copy/move to allow different allocators
- Freestanding?
- Conclusions

Rationale

- We see demand for static_vector and sbo-enabled vector
- Why don’t we use the **vector** we have?
- Generic code templated on **vector<T, A>** just works.
- Shorter standard text (?)
- New allocators can be used for some other containers.

- Iterators potentially invalidated by move/copy (but this case already exists for assigmnent).

Minimum rules for vector to follow

- Use **allocate_at_least** when allocating
- Don’t allocate unless capacity is too low
- Handle move/copy when source data is in SBO buffer. This uses a new allocator trait **buffer_capacity**.

Optimizations vector can do

- Specialize on **can_allocate** trait of allocator.
- Only store a size member allocator can't allocate.
- Adjust type of size member depending on buffer size.

New allocators

- One **buffered_allocator** which has a backing allocator
- One or more non-allocating allocators for **static_vector**

```
Template<typename T, size_t SZ, typename Backing> std::buffered_allocator {
    // Contains a buffer and forwards overflowing allocatíons to Backing allocator
};


template<typename T> struct terminating_allocator;   // Terminate in alloc()
template<typename T> struct throwing_allocator;          // Throw bad_alloc in alloc()
template<typename T> struct unchecked_allocator;         // Return nullptr from alloc().
```

Convenient type aliases

```
template<typename T, size_t SZ> using sbo_vector;           // Bike shedding needed!
template<typename T, size_t SZ> using static_vector;        // Bike-shedded!
```

What is the overflow policy of **static_vector** if there is only one such type alias?

Move/Copy with different allocators

- If T is same we should be able to copy/move between vectors even if allocators differ.
- More important with SBO buffers of different sizes.
- If Backing allocators are equal moves can be optimized.
- This needs another trait:

```
template<typename<Alloc>
using **backing_allocator_of**<Alloc> = Alloc;

template<typename T, size_t SZ, typename Backing>
using backing_allocator<buffered_allocator<T, SZ, Backing> = Backing;   // I wish!
```

Take aways from the demo implementation

- Optimizing for !can_alloc is fairly easy.
- A mechanism for conditional data members would be nice: *constraints on members would solve this.*
- **if constexpr** is very handy to select implementation.
- Move and copy is the only somewhat complex functionality to implement.

Concerns for freestanding

- static_vector can easily be part of ’freestanding’.
- It is unclear whether this is possible for vector, even if a non-throwing allocator is used (and T that doesn’t throw during copy).
- It seems like ”conditionally freestanding vector” may be hard as there may be throw/catch constructs needed to handle T constructor exceptions.
- Are throw/try-catch allowed in a non-taken if constexpr branch when compiling without exceptions?
- New trait for non-throwing allocator may allow some optimizations.

Conclusion, status, discussions.

- It is possible to reuse vector for static- and sbo cases.
- Where to put the new traits? (cf P2652).
- The new allocators compose well.
- Useful for anything allocating like a vector.
- Implementation sketch at: github.com/BengtGustafsson/isocpp-proposals
- Is this a way forward or should we stick to static_vector and maybe later sbo_vector?
- Thoughts on freestanding?