Issues with current `flat_map` proposal

Document #:
Date: 2019-05-20
Project: Programming Language C++
Library Working Group
Reply-to: Denis Yaroshevskiy
<denis.yaroshevskij@gmail.com>

Contents

1 Revision history

Revision 0 Original revision of the paper for 2019 Cologne meeting.

2 Introduction

[P0429] introduces `flat_map` into the standard as a container adaptor based on two containers which requires that reference type to be a proxy object. This has been attempted on multiple occasions for other use-cases such as an infamous `vector<bool>` [VECTOR_BOOL], multi-span (decided against it) [P0009], standard audio proposal [P1386] (also decided against it following multi-span) and others. From the last revision of the ranges proposal [P1035R4] the standard comittee removed `zip_view` - a general purpose view to iterate over multiple containers - which is a more general solution to what `flat_map` needs to do for iteration. The goal of this paper is to object to inclusion of `flat_map` with a knowingly flawed design into the standard at least until `zip_view` [P1035R4], because `zip_view` would be a decision point: how should proxy references be implemented and what is an acceptable level of comlexity for them.

3 Listing of examples of how current design is problematic

In the flat_map paper reference type is defined as `pair<const key_type&, value_type&>`. This breaks many expectations and typical patterns for writing C++ code. (In all of the examples `decltype(it)` is `flat_map::iterator`).

```
//--------------------
auto x = std::move(*it);  // `x` is moved from `*it`. Unless it's a `flat_map` where
                          // this creates a reference. Copy does not work either.

//--------------------
auto& x = *it;  // Create a mutable reference to the value pointed by it.
                // Except for flat_map where this does not compile.

//--------------------
[x = *it] { do_smth(x); }  // Capture an element by value. Unless flat_map,
                           // in which case capture is by reference.

//--------------------
auto foo() {              // This does not dangle unless used with flat_map's iterators.
   // ...
   return *it;
}

//--------------------
template <typename T>
void bar(T mine) {        // bar does not modify input parameters.
  sink(std::move(mine));  // Unless called with a flat_map reference.
}
```

4 Lack of reference implementations

There is no production (or even a complete reference) implementation for `flat_map` that is based on two containers that the author could find. All popular open source implementations (boost, folly, eastl, chromium) use a single container. The only library that we are aware of that could provide a similar experience to using two containers flat map is `zip` utility from ranges-v3. The proposal to add `zip_view` [P1035R4] (a version of that utility) was not yet accepted for standardisation.

5 Lack of `zip_view` limits the usage of `flat_map`

- erase_remove_if idiom is not implementable since there is no `zip_view` in the standard. Also current version does not have mutating access to keys/values which also disallows this.
- One of the most typical use-cases for flat_map is building a buffer and then converting it into a map. Without standard `zip_view` we cannot use algorithms to populate such buffers.

These are very important use-cases and current proposal does not address them.

6 Performance implications of two containers on sort

The goal of separating keys and values into two arrays is to increase the lookup speed by packing keys together in the cache. However, this might have detrimental consequences for sort.

Sort is absolutely crucial for flat containers - for example most of the time spent in Chromium on every key-stroke is flat_set construction [CHROMIUM_EXAMPLE].

However, at least with `ranges::zip` (which is the only known example of two-container sort), it brings a significant overhead [QUICKBENCH_SORT]. At this point the author does not know whether this is due to the quality of an implementation, benchmarking artefact or a fundamental problem.

7 How does this relate to std::ref/std::string_view/std::span?

In C++ standard there are a few "reference-like" types that are widely and succesfully used in production. This paper is not against such types it is just against using them as `iterator::reference`. All of the language constructs (such as auto/argument deduction etc) as well as library components (such as algorithms) are expected to work with them as `Semiregular` types and to not perform any conversions. Examples:

- It would be unexpected that captured by value `string_view` was converted into `std::string`
- If the user has a function that returns a `std::span` and result of that would be stored into a local variable with a deduced type that variable should be `std::span` type and not `std::vector`.

On the contrary these exaples not what we expect from `iterator::reference` types that we have now (see listing). If the standard was to inroduce `iterator::reference` that were to behave differently - all of the examples in the listing should be reworked in some different way.

The problem is not a "reference-like" type but an iterator that tries to use it as a `reference`.

8 How this relates to "Proxy Iterators for the Ranges Extensions" [D0022]

Eric Niebler in [D0022] solved a very important part of the proxy-iterator problem: how proxy-iterators fit into iterator concept and how to design algorithms for them. However that paper does not address type deduction which is also a big part of the problem.

9 Proposed actions

This paper suggests to postpone `flat_map` at least until the standardization of `zip_view`. If with `zip_view` the standard committee decides that proxy references are an acceptable practice in C++ than `flat_map` can use them too and the C++ community will have to learn to be extremely cautious in a much bigger number of use-cases than now. But this should be a considered decision and not something done on the back of `flat_map`. Ideally would be to find a better solution for proxy references that is intuitive and unintrusive. And then use that for `flat_map`.

10 References

[CHROMIUM_EXAMPLE] Example where sort is important in Chromium.
https://cs.chromium.org/chromium/src/components/omnibox/browser/url_index_private_data.cc?l=657&rcl=7bfecf258f220ea375ffd376608d6ec71ca9d8ce

[D0022] Eric Niebler. Proxy Iterators for the Ranges Extensions.
https://ericniebler.github.io/std/wg21/D0022.html

[P0009] 2019. Polymorphic Multidimensional Array Reference.
https://wg21.link/p0009

[P0429] Zach Laine. 2019. A Standard `flat_map`.
https://wg21.link/p0429

[P1035R4] Input range adaptors (revision 4).
https://wg21.link/p1035r4

[P1386] 2019. A Standard Audio API for C++: Motivation, Scope, and Basic Design.
https://wg21.link/p1386

[QUICKBENCH_SORT] Measuring sort of two containers vs one container.
http://quick-bench.com/V7zQvXo5R13DVMvmuRzj9GtoEMM

[VECTOR_BOOL] Howard Hinnant. 2012. On `vector<bool>`.
https://howardhinnant.github.io/onvectorbool.html