# Debug performance is improving

*...what now?*

CppCon 2022
Lightning Talk

**Vittorio Romeo**
mail@vittorioromeo.com
@supahvee1234

Bloomberg
Engineering

TechAtBloomberg.com
Careers

# Debug performance matters

- Debugging with optimizations enabled is a nightmare
  - Control flow of the code might be lost
  - Information on local variables and the stack might be unavailable
  - Many functions will be inlined
- Performance matters in many fields
  - Simulations, game development, virtual reality, VFX, real-time, etc...
- Developers working in those fields need perfomant debug builds
- “Zero-cost abstractions” are a *lie*
  - Or rather, a misnomer...
  - They **rely** on compiler optimizations
- People write C-like code to avoid debug performance overhead
  - Unabstracted code leads to more bugs...
  - ...which leads to more debugging

 

# Abstractions are important

- I want to encourage C++ developers to write and use abstractions
  - Code becomes simpler, easier to understand, less bug-prone
- But I cannot, because “zero-cost abstractions” have costs
  - Debug performance is the topic of this talk
  - But compilation time also matters!

```cpp
#include <cstddef>

using byte_type = std::byte;
// using byte_type = char;

byte_type example()
{
    byte_type b{123};
    b <<= 1;
    return b;
}
```

on godbolt

 

# Moving an `int` is slow (#1)

```
int accumulate_range(int* begin, int* end)
{
    return std::accumulate(begin, end, 0);
}
```

- When compiling(*) with no optimizations:
  - `accumulate_range` runs 1.4x slower in C++20 compared to C++17

(*) Using GCC and Clang from a few months ago, or MSVC.

- Why?

 

# Moving an `int` is slow (#2)

```
 1 template <class _InputIterator, class _Tp>
 2 _Tp
 3 accumulate(_InputIterator __first,
 4              _InputIterator __last,
 5              _Tp __init)
 6 {
 7     for (; __first != __last; ++__first)
 8 #if _LIBCPP_STD_VER > 17
 9         __init = std::move(__init) + *__first;
10 #else
11         __init = __init + *__first;
12 #endif
13     return __init;
14 }
```

 

# Moving an `int` is slow (#3)

```
 1 template <class _InputIterator, class _Tp>
 2 _Tp
 3 accumulate(_InputIterator __first,
 4              _InputIterator __last,
 5              _Tp __init)
 6 {
 7     for (; __first != __last; ++__first)
 8 #if _LIBCPP_STD_VER > 17
 9         __init = std::move(__init) + *__first;
10 #else
11         __init = __init + *__first;
12 #endif
13     return __init;
14 }
```

- Wait, is `std::move` adding run-time overhead?
- Isn't `std::move` just a cast?

 

# Moving an `int` is slow (#4)

```
template <class _Tp>
[[nodiscard]] inline constexpr
std::remove_reference_t<_Tp>&& move(_Tp&& __t) noexcept
{
    return static_cast<std::remove_reference_t<_Tp>&&>(__t);
}
```

- Semantically, it is just a cast
  - To the compiler, it is just another *function call*
- I.e. overhead unless inlining happens
  - It doesn't in `-O0`
- The same issue applies to other functions:
  - `std::forward`
  - `std::as_const`
  - `std::as_underlying`
  - `std::vector<T>::iterator::operator*`
  - `std::unique_ptr<T>::operator*`
  - *etc...*

 

# What can we do?

- `-Og` doesn't cut it
  - Sometimes optimizes too much
  - For Clang, it's the same as `-O1`
  - MSVC doesn't have an equivalent

- Some people resort to macros:

```
#define MOV(...) \
    static_cast< \
        std::remove_reference_t< \
            decltype(__VA_ARGS__)>&&>(__VA_ARGS__)

#define FWD(...) \
    static_cast<decltype(__VA_ARGS__)&&>(__VA_ARGS__)
```

*(From https://www.foonathan.net/2020/09/move-forward/)*

- Some people tried addressing the issue in the language
  - E.g. P1221: "Parametric Expressions"
  - Nothing was accepted, and it wouldn't fix existing issues

# What has been done?

- GCC and Clang recently started eliding some functions in the frontend

> *Improved `-O0` code generation for calls to `std::move`, `std::forward`, `std::move_if_noexcept`, `std::addressof`, and `std::as_const`.*
>
> *These are **now treated as compiler builtins** and implemented directly, rather than instantiating the definition from the standard library.*

- MSVC hasn't delivered yet...
- And many other functions and types are still a source of overhead
  - E.g. `std::byte` or iterators

# Call to action

- Let your voice be heard:
  - GCC: Bugzilla Report
  - Clang: GitHub Issue
  - MSVC: Developer Community Feedback
- Think about how to solve this problem in general
  - Special attribute?
  - New language feature for hygenic macros?
  - `[[gnu::always_inline]]` in Standard Library implementations?
- This is an important problem
- C++'s reputation in many prominent fields is at risk
- People with a lot of influence are crusading against abstractions
- New programmers are encouraged to write unsafe C-like code
- **Let's make (some) abstractions *truly* zero-cost!**

 

# Thanks!

- **https://github.com/vittorioromeo/cppcon2022**

- Pragmatic Simplicity
  - *Actionable Guidelines To Tame Complexity*
- Thursday 15 @ 15:15 MDT
  - Aurora A / Online A

- *"Embracing Modern C++ Safely"* book signing
  - by J. Lakos, R. Khlebnikov, A. Meredith, & other contributors
- Tuesday 13 @ 12:00 MDT
  - Aurora A / Online A

- Let's keep in touch!
  - https://vittorioromeo.com
  - mail@vittorioromeo.com
  - @supahvee1234