# Portable optimisation hints

Timur Doumler (papers@timur.audio)

| | |
|---|---|
| Document #: | P1774R1 |
| Date: | 2019-10-06 |
| Project: | Programming Language C++ |
| Audience: | Evolution Working Group |

**Abstract**

We propose a standard facility providing the semantics of existing compiler intrinsics such as `__builtin_assume` (Clang) and `__assume` (MSVC, Intel). It gives the programmer a way to allow the compiler to assume that a given C++ expression is true, without evaluating it, and to optimise based on this assumption. This is very useful for high-performance and low-latency applications in order to generate both faster and smaller code.

## 1 Motivation

All major compilers offer built-ins that give the programmer a way to allow the compiler to assume that a given C++ expression is true, and to optimise based on this assumption. They are very useful for high-performance and low-latency applications in order to generate both faster and smaller code. Use cases include more efficient code generation for mathematical operations, better vectorisation of loops, elision of unnecessary branches, function calls, and more. This is existing practice, but it would be much more accessible and easy-to-use if it were a standardised, portable C++ facility.

### 1.1 History and context

Adding such portable optimisation hints was already proposed once [N4425] and discussed by EWG in 2015 in Lenexa[1]. The paper was rejected. EWG's guidance was that this functionality should be provided within the proposed contracts facility, and not as a separate feature.

Unfortunately, contracts as merged into the C++20 working draft in June 2018 in Rapperswil [P0542R5], actually failed to provide such portable optimisation hints [P1773R0]. Later, in July 2019 in Cologne, contracts were pulled from C++20 altogether.

Regardless of whether contracts will eventually make it into a future C++ standard, and whether or not assumptions might be a feature of such contracts, we need an independent low-level "assume" facility. The present paper focuses on proposing exactly this low-level facility. Its purpose is to introduce a standard way to provide an optimisation hint to the compiler, as an implementation detail of a C++ program, locally, with clearly defined semantics that are expressed in C++ code, independent of any build modes, build flags, etc.

In case contracts or other higher-level features will make use of such assumptions in the future, it can then be implemented in terms of this low-level facility.

[1]`https://cplusplus.github.io/EWG/ewg-closed.html#179`

## 1.2 Existing practice

The major compilers offer the following built-ins providing this functionality:

— MSVC and Intel provide `__assume(`*`expression`*`);`

— Clang provides `__builtin_assume(`*`expression`*`);`

— GCC does not provide an analogous built-in directly, but the same can be achieved with `if (!`*`expression`*`) __builtin_unreachable();`

See [N4425] for a more thorough discussion.

## 1.3 Examples

Consider the following function:

```
int divide_by_32(int x)
{
  __builtin_assume(x >= 0);
  return x/32;
}
```

Without the assumption, the compiler has to generate code that works correctly for all possible input values. With the assumption, it can implement the calculation using a single instruction (shift right by 5 bits). Here is the output generated by clang (trunk) with `-O3`:

Without `__builtin_assume`:

```
mov eax, edi
sar eax, 31
shr eax, 27
add eax, edi
sar eax, 5
ret
```

With `__builtin_assume`:

```
mov eax, edi
shr eax, 5
ret
```

Another example: consider looping over an array of numbers and performing math on the elements. Often, there are invariants on the array size such as: it's a power of two, it's a multiple of the SIMD register size, etc (all very common e.g. in audio processing code). Telling the optimiser about such invariants leads to a much better optimisation and vectorisation of the loop:

```
void limiter(float* buffer, size_t size)
{
  __builtin_assume(size % 8 == 0);
  for (size_t i = 0; i < size; ++i)
     data[i] = std::clamp(data[i], -1.0f, 1.0f);
}
```

For this function, clang (trunk) with `-O3` generates 70 lines of assembly without the assumption, and only 42 lines with it.

See [Regehr2014] for more examples and use cases.

## 2 Proposed solution

### 2.1 Semantics

The design goal is to provide a portable facility closely following the compiler built-ins `__assume` and `__builtin_assume`, therefore standardising existing practice. The facility should be implementable with the existing built-ins on those compiler implementations who have them, without unnecessarily constraining implementations who do not. Therefore, we propose the following semantics:

— It is a statement with a single argument, which is a C++ expression contextually convertible to `bool`.

— The expression is guaranteed to be unevaluated. Therefore, expressions with side effects are allowed (which is useful, consider `++ptr != end`), and any such side effects are discarded.

— However, the optimiser may analyse the form of the expression, and deduce from that information used to optimise the program.

— The behaviour is undefined if the expression would evaluate to `false` (this allows the optimiser to optimise the program based on the assumption that the expression always evaluates to `true`).

— Simply ignoring the whole statement is a conforming implementation, i.e. the optimiser is not required in any way to make use of that assumption.

## 2.2 Syntax

We propose two alternatives how to spell this statement, each with its pros and cons which are discussed below. We leave it up to EWG to give guidance on which should be preferred.

Regardless of which is chosen, we propose that the word "assume" is the name used to spell it. This is the name already used in existing built-ins, therefore choosing it means standardising existing practice. This name will be least surprising and most self-explanatory to the user. It is also the most descriptive choice.

### 2.2.1 Alternative 1: Function-style syntax

We could introduce the feature as a "magic" library function, so `__builtin_assume(`*`expression`*`)` becomes:

```
std::assume(expression);
```

An advantage of this spelling is that no core language change is required to introduce it. It would then also be consistent with the closely related `std::assume_aligned` [P1007R3], which was adopted for C++20. `std::assume_aligned` was changed by EWG from an initially proposed attribute syntax [P0886R0] to a function-style syntax to avoid a core language change.

The drawback is that this spelling would introduce a weird novelty: something that is syntactically a function call, yet does not evaluate its operand.

### 2.2.2 Alternative 2: Attribute syntax

Alternatively, we could use an attribute syntax, so `__builtin_assume(`*`expression`*`)` instead becomes:

```
[[assume(expression)]]
```

This syntax (using parentheses) is chosen such that it is fully compatible with standard attribute syntax and therefore backwards-compatible with a compiler that does not support this feature. Making this an attribute also makes it clear to the user that ignoring this statement does not change the observable semantics of a valid program.

While an attribute spelling would be inconsistent with the closely related `std::assume_aligned`, it would be consistent with other optimisation hints that do use attribute syntax as well, such as `[[likely]]`/`[[unlikely]]` [P0479R5] and `[[carries_dependency]]`.

However, by introducing a new attribute, this feature would require an addition to the core language.

### 2.3 Wording

The formal wording for this proposal will be provided after EWG guidance on the two syntax alternatives.

## Document history

— **R0**, 2019-06-17: Initial version.

— **R1**, 2019-10-06: Updated text to reflect removal of Contracts from C++20; made proposed attribute syntax backwards-compatible by replacing colon with parentheses.

## References

[N4425] Hal Finkel. Generalized Dynamic Assumptions. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4425.pdf, 2015-04-07.

[P0479R5] Clay Trychta. Proposed wording for likely and unlikely attributes (Revision 5). http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0479r5.html, 2018-03-16.

[P0542R5] Gabriel Dos Reis, Jose Daniel Garcia, John Lakosand Alisdair Meredith, Nathan Myers, and Bjarne Stroustrup. Support for contract based programming in C++. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0542r5.html, 2018-06-08.

[P0886R0] Timur Doumler. The assume aligned attribute. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0886r0.pdf, 2018-02-12.

[P1007R3] Timur Doumler. std::assume_aligned. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0627r3.pdf, 2018-11-07.

[P1773R0] Timur Doumler. Contracts have failed to provide a portable “assume”. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p1773r0.pdf, 2019-06-17.

[Regehr2014] John Regehr. Assertions Are Pessimistic, Assumptions Are Optimistic. https://blog.regehr.org/archives/1096, 2014-02-05.