New features in Intel C/C++ Compiler 16.0 Beta

Judy Ward, Intel C++ front end engineer

Judy.Ward@intel.com

Agenda

Compile time improvements

SSE operators

Honoring Parenthesis

Current state of C++14 and C11 support (15.0)

C++14 and C11 support (16.0 beta)

Feature macros

Comparison to GNU/Microsoft

Future plans and C/C++ standard features

Q & A

Compile Time Improvements

Intrinsic headers provided by Intel

emmintrin.h, immintrin.h, etc.

Prototypes are now automatically disabled in headers, i.e.:

```
extern __m128  _mm_shuffle_ps(__m128, __m128,unsigned int);
extern __m128  _mm_unpackhi_ps(__m128, __m128);
```

Use –D__INTEL_COMPILER_USE_INTRINSIC_PROTOTYPES to restore old behaviour (enhanced type checking)

SIMD Operator Support

Using operators with the SSE integer types will now work

- These operators now supported: + - * / & | ^ += -= *= /= &= |= ^= == != > < >= <=
- Example:
  - `__m128i x,y,z;`
  - `X = y + z;`

Restrictions:

- Only supports 128 and 256 bit SIMD types
- Only supports the two operand operators listed above
- The operands must have the same type (i.e. Our SSE types cannot be combinable with GNU types declared with vector_size attribute)

Honoring Parentheses

Some Intel floating point optimizations (like –fp-model fast) do not require parentheses to be honored in real and complex expressions, i.e.

```
double a , b =1;
a = (1 + b) + 3;
```

Switch:

-f[no-]protect-parens (Linux) –Qprotect-parens[-] (Windows)

enable/disable(DEFAULT) a reassociation optimization for REAL and COMPLEX expression evaluations by not honoring parenthesis

Current Support (15.0)

Full C++11 support

Very limited C11 support (only binary literals)

Minimal C++14 support (decltype auto, lambda init capture, deduced return types for routines)

Comparison to GNU and Microsoft can be found at:

- https://software.intel.com/en-us/articles/c0x-features-supported-by-intel-c-compiler

C11 Features

_Alignment features (_Alignas, _Alignof)

_Static_assert

_Thread_local keyword

Type generic selections (_Generic)

```
#define  pow(X) _Generic((X), long double: powl, \
                default: pow, \
                float: powf)(X)
```

Noreturning functions (_Noreturn)

Unicode strings

C11 anonymous unions

_Atomic keyword*** NOT IN 16.0 (used in stdatomic.h)

C++14 Features in 16.0

Generic Lambdas

- auto glambda = [] (auto a) { return a; };

Generalized lambda captures

```
int x = 4;
int z = [&r = x, y = x+1] {
        r += 2; // set x to 6; "R is for Renamed Ref"
        return y+2; // return 7 to initialize z
    }(); // invoke lambda
```

Digit Separators

- auto million = 1’000’000;

C++14 Features in 16.0

The [[deprecated]] attribute

```
[[deprecated]] void foo() {;}  // warning on call to foo()
```

Function return type deduction, i.e.:

```
auto foo(int i) {
     if (i ==1) return i;
     else return foo(i-1)+i;
}
```

C++14 features not yet implemented in 16.0

Relaxed constexpr restrictions, i.e.:

- Local variable declarations (except for static or __thread_local)
- if, switch, for, while, do-while statements (not goto)
- constexpr member functions are not implicitly const

Variable templates, i.e.:

- template<typename T>
- constexpr T pi = T(3.14159265358979323385);
- template<class T> T area_of_circle_with_radius(T r) { return pi<T> * r * r; }

Feature Test Macros

Testing for existence of header files:

```
#if __has_include("shared_mutex")  // use standard header
#elif __has_include("boost/shared_mutex.h") // use BOOST header
#endif
```

Testing for existence of compiler features:

```
#ifndef __cpp_constexpr
// no constexpr functionality available
#elif __cpp_constexpr == 200704
// c++11 constexpr functionality available
#else
// c++14 relaxed constexpr functionality available
#endif
```

GNU Compatibility

To enable c11 or c++14 support you need to use –std=c11 or –std=c++14 switch

We currently support all C++14 features used in the GNU 5.0 versions of the headers enabled when you use the switch and all c11 features except _Atomic which is used in <stdatomic.h>

Depending upon the GNU on your system (i.e. g++ in your PATH) you may get different features enabled

Microsoft Compatibility

Microsoft has been slower than GNU in implementing new C++11, C++14, and C11 features although MSVC++ 2015 has added many of them.

- Still missing are C++11 standard attributes, char16_t and char32_t, full constexpr, unicode string literals, expression SFINAE, and most of the C++14 features

No special command line switch to access C11 or C++11 or C++14 functionality

Intel® C++ compiler is compatible by default

- Whatever features are provided by the reference Microsoft compiler are available with Intel Windows* compiler

To get additional C++14 functionality with our compiler use Intel-specific /Qstd=c++11 or /Qstd=c++14 or /Qstd=c11 switches

Intel C++ Compiler is fully compatible with the Microsoft Visual C++ 2015 with respect to C++14 and C11 functionality

Future C++ Features (C++17 & beyond)

resumable/await

Terse range-based for loop

Nested namespaces

Removal of trigraphs

Addition of a default text message for static_assert

Attributes for namespaces and enumerators

UTF-8 character literals

Constant evaluation for all non-type template arguments

C++ Technical Reports in Progress

Concepts

File System Utilities

Networking Utilities

Library Fundamentals

Array Extensions

Extensions for Parallelism

Extensions for Concurrency

Transactional Memory

References

C++14 FDIS

http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2013/n3690.pdf

Individual papers

http://www.open-std.org/JTC1/SC22/WG21/docs/papers/

Technical Report on Feature Macros:

https://isocpp.org/std/standing-documents/sd-6-sg10-feature-test-recommendations

C++ FAQ

http://www.stroustrup.com/bs_faq.html

Legal Disclaimer & Optimization Notice

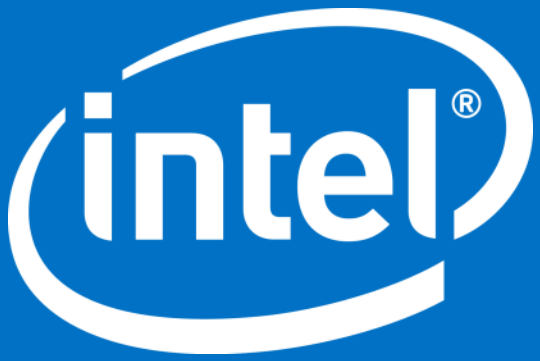
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804


intel®