Intel® oneAPI DPC++/C++ Compiler

Transition from Intel C++ Compiler Classic (ICC)

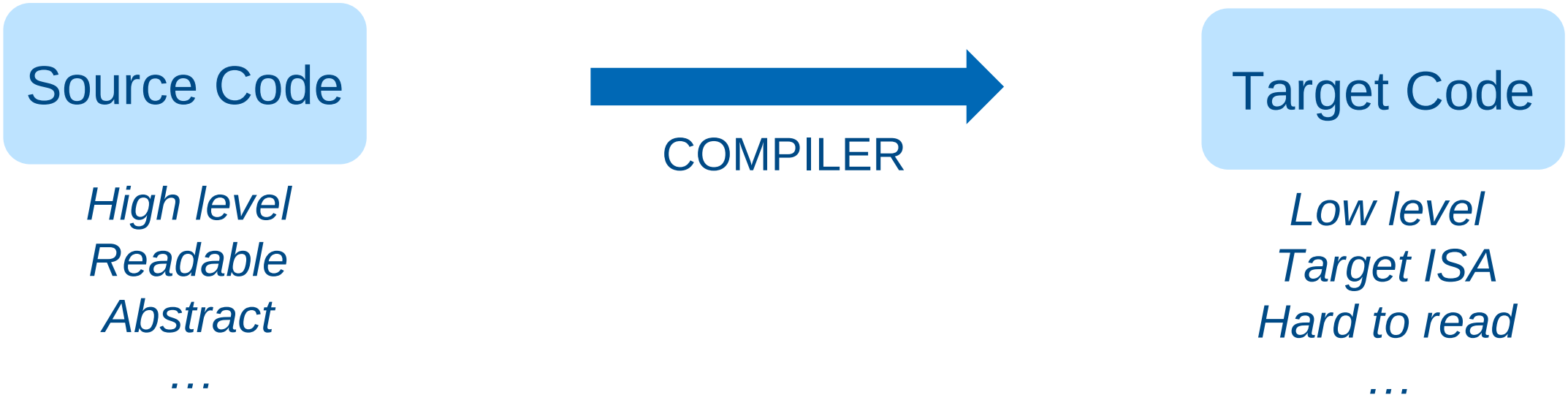
March 2022

Agenda

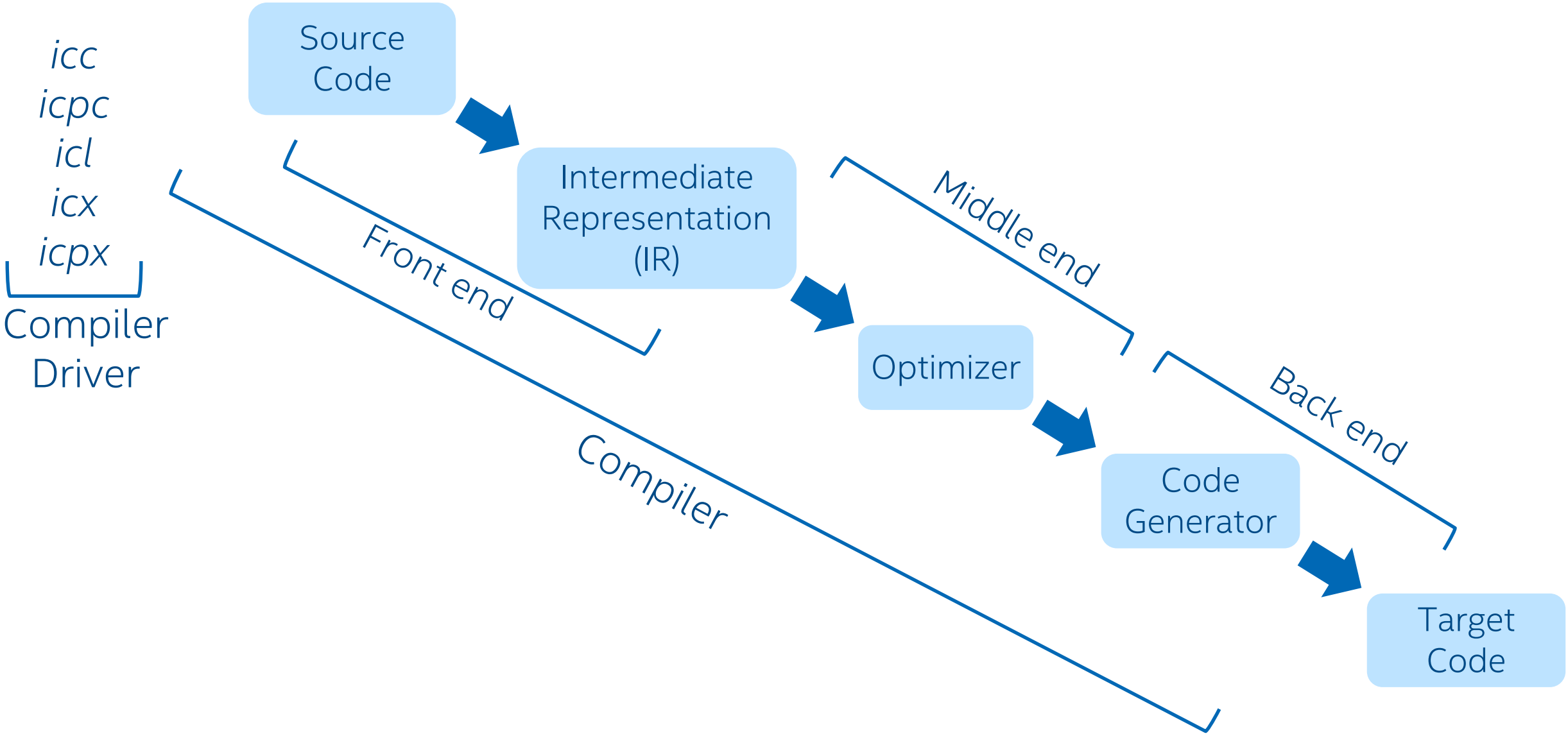
- Introduction
  - Compiler Architecture
  - What is ICX?
  - Major Changes Overview
  - Versioning and Packaging
  - Compatibility
- Compiler Options, IPO, PGO
- Changes in Diagnostics
- Floating Point Reproducibility Controls
- Auto-vectorization, Optimization Reports
- Intrinsic Usage Model
- OpenMP support
- Performance
- Miscellaneous

Introduction

Introduction to Compilers



Compiler Architecture – Simplified View



What is ICX?

- Close collaboration with Clang*/LLVM* community
- ICX is Clang front-end (FE), LLVM infrastructure
  - PLUS Intel proprietary optimizations and code generation
- Clang FE pulled down frequently from open source, kept current
  - Always up to date in ICX
  - We contribute! Pushing enhancements to both Clang and LLVM
- Enhancements working with community – better vectorization, opt-report, for example

Major Changes Overview

tinyurl.com/icc-to-icx-migration-guide

- Written for ICC to ICX transition
- LLVM is a different compilation technology. EXPECT differences
- Options: undocumented IL0 options all ignored
  - icx -qnextgen-diag option to get a list of supported and unsupported options
- IPO/PGO - new LLVM LTO/PGO instead
- FP Model is not the same
- Intrinsics are handled very differently
- C/C++ Pragmas – a lot of Intel proprietary ones not supported
  - enable -Wunknown-pragmas to warn on unsupported pragmas
- __INTEL_LLVM_COMPILER is defined instead of __INTEL_COMPILER

Major Changes Overview

- Not supported features:
  - Auto-parallelization (-parallel option)
  - Intel Cilk™ Plus (pragma simd) replaced by OpenMP pragmas
  - -ax not implemented yet
- Optimization reports
- Analyzers may be affected
  - Advisor needs opt reports for Vectorization Advisor
  - VTune can't get code lines/clear symbols for OpenMP regions
- no macOS support

Intel® C++ Compilers

Intel Compiler	Target	OpenMP Support	OpenMP Offload Support	Included in oneAPI Toolkit
Intel® C++ Compiler, IL0 ***icc/icpc/icl***	CPU	Yes	No	HPC
Intel® oneAPI DPC++/C++ Compiler, LLVM ***dpcpp***	CPU, GPU, FPGA*	Yes	Yes	Base
Intel® oneAPI DPC++/C++ Compiler, LLVM ***icx/icpx***	CPU GPU*	Yes	Yes	Base

Cross Compiler Binary Compatible and Linkable!

tinyurl.com/oneapi-standalone-components

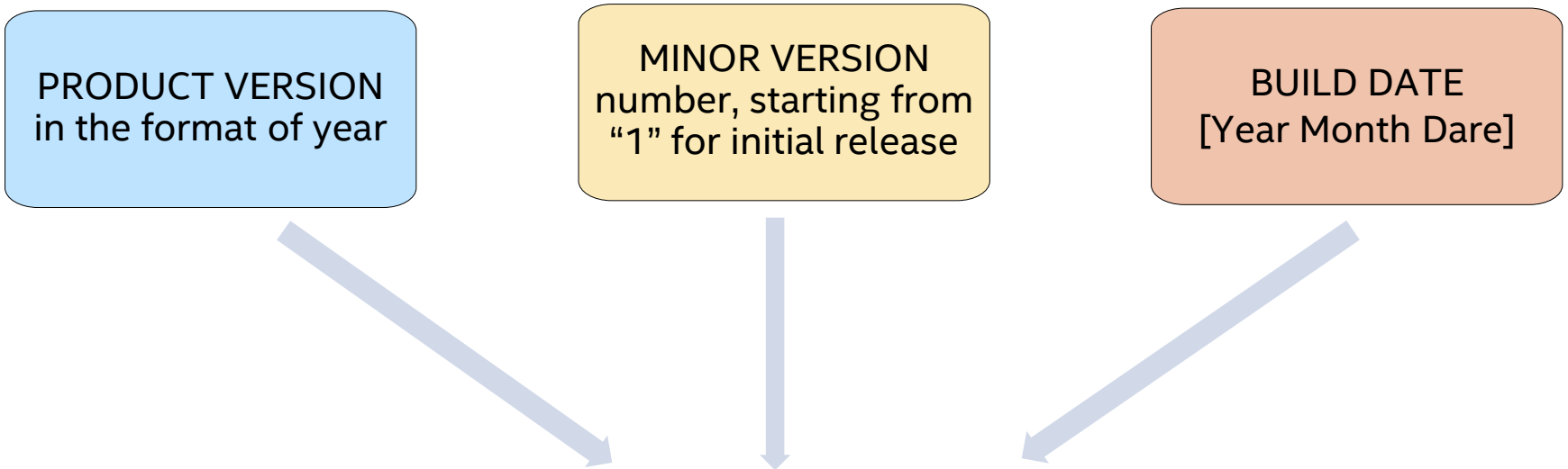
Packaging of C/C++ Compilers

oneAPI Base Toolkit *PLUS* oneAPI HPC Toolkit

- Existing IL0 compilers ICC, ICPC in HPC Toolkit
- v2021.5 code base for IL0 compilers

- **Compilers based on LLVM* framework**
- Compiler Drivers: icx/icpx and dpcpp
- v2022.0 in oneAPI 2022.1

Versioning Convention



icx --version

Intel(R) oneAPI DPC++/C++ Compiler 2022.0.0 (2022.0.0. 20211123)

icx for C code

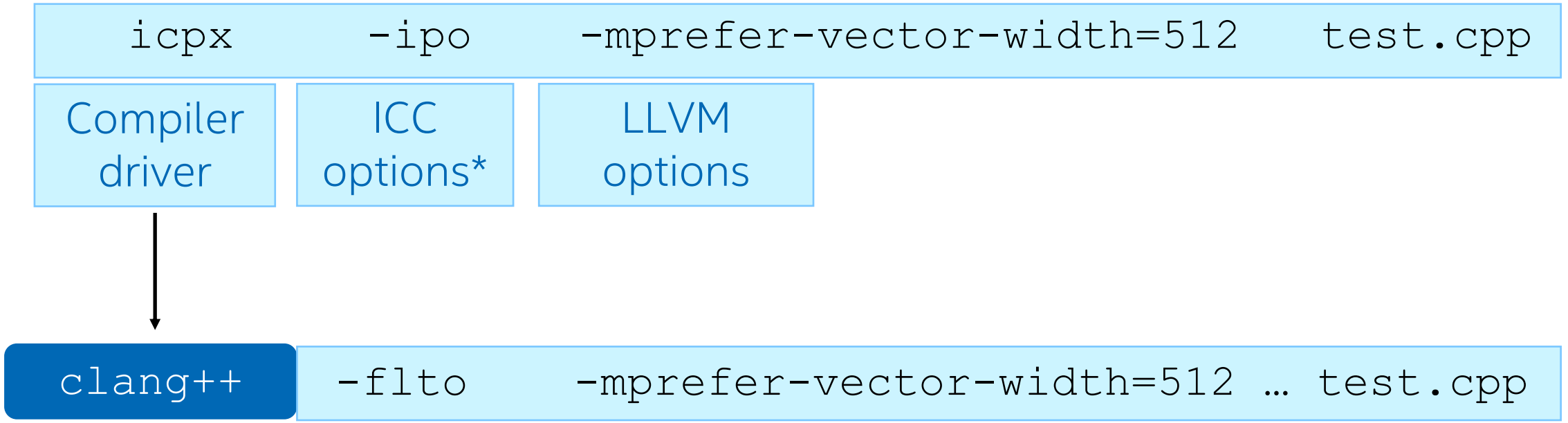
icpx for C++ code

ICC and ICX Compatibility

- ABI compatible
  - IPO
  - -vecabi option
- Safe to mix binaries with attention to options

Compiler Options

Options Mapping



*Not all ICC Classic options are accepted and/or implemented in ICX.

-# is useful ‘dryrun’ option

Not Supported Options

- Not all ICC Classic options are accepted and/or implemented in ICX
  - Undocumented options from ICC Classic are NOT implemented
  - Use `-qnextgen-diag` to emit a long list of ICC Classic options that are NOT accepted by ICX
- All Clang*/LLVM options for the Clang version included in ICX are accepted and implemented.
  - Use `-Xclang` to pass Clang options to ICX (Windows, Linux)
- GNU* and Microsoft* compatible options are accepted by ICC Classic and ICX.

Common optimization options

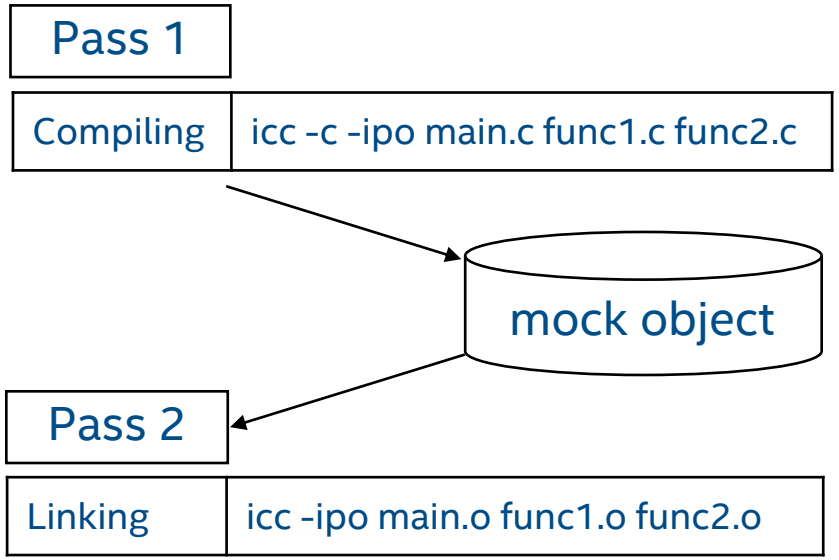
	Linux* icx (icc)
Disable optimization	-O0
Optimize for speed (no code size increase)	-O1
Optimize for speed (default)	-O2
High-level loop optimization	-O3
Create symbols for debugging	-g
Multi-file inter-procedural optimization	-ipo
Profile guided optimization (multi-step build)	-fprofile-generate (-prof-gen) -fprofile-use (-prof-use)
Optimize for speed across the entire program (“prototype switch”)	-fast same as “-ipo -O3 -static -fp-model fast” (-ipo -O3 -no-prec-div –static -fp-model fast=2 -xHost)
OpenMP support	-fiopenmp (-qopenmp)

Interprocedural Optimizations

Interprocedural Optimizations (IPO)

Multi-pass Optimization

- Interprocedural optimizations performs a static, topological analysis of our application
- Enabled optimizations:
  - Procedure inlining (reduced function call overhead)
  - Interprocedural dead code elimination, constant propagation and procedure reordering
  - Enhances optimization when used in combination with other compiler features
  - Much of ip (including inlining) is enabled by default at option O2



Interprocedural Optimizations

Extends optimizations across file boundaries

icc

-ip	Only between modules of one source file
-ipo	Modules of multiple files/whole application

icx

-ipo (mapped to -flto)	Link Time Optimization in ICX

Without IPO

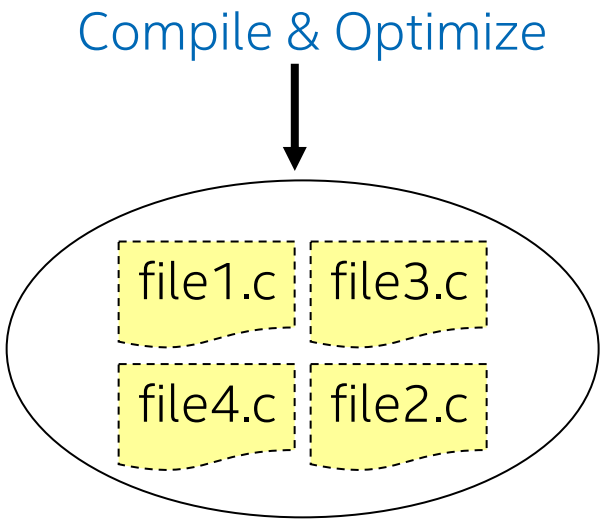
Compile & Optimize → file1.c

Compile & Optimize → file2.c

Compile & Optimize → file3.c

Compile & Optimize → file4.c

With IPO



Link Time Optimization (LTO): tinyurl.com/clang-lto

Interprocedural Optimizations

- icx uses Link Time Optimization (LTO) technology (-flto)
- -ipo should be added to both compilation and linking steps (or replace original linker with the 'lld –fuse-ld=lld')
- Intel tools 'xilink', 'xild', and 'xiar' are removed from ICX and should be replaced in projects settings, makefiles, etc. with equivalents

```
$ icpc -ipo -c hello.cpp
$ icpx -ipo hello.o -o hello
/usr/bin/ld: hello.o:(.data+0x0): undefined reference to `__must_be_linked_with_icc_or_xild'
clang-13: error: linker command failed with exit code 1 (use -v to see invocation)

$ icpx -ipo -c hello.cpp
$ icpc hello.o -o hello
hello.o: file not recognized: file format not recognized
```

Profile-Guided Optimizations

Profile-Guided Optimizations (PGO)

- Static analysis leaves many questions open for the optimizer like:
  - How often is x > y
  - What is the size of count
  - Which code is touched how often

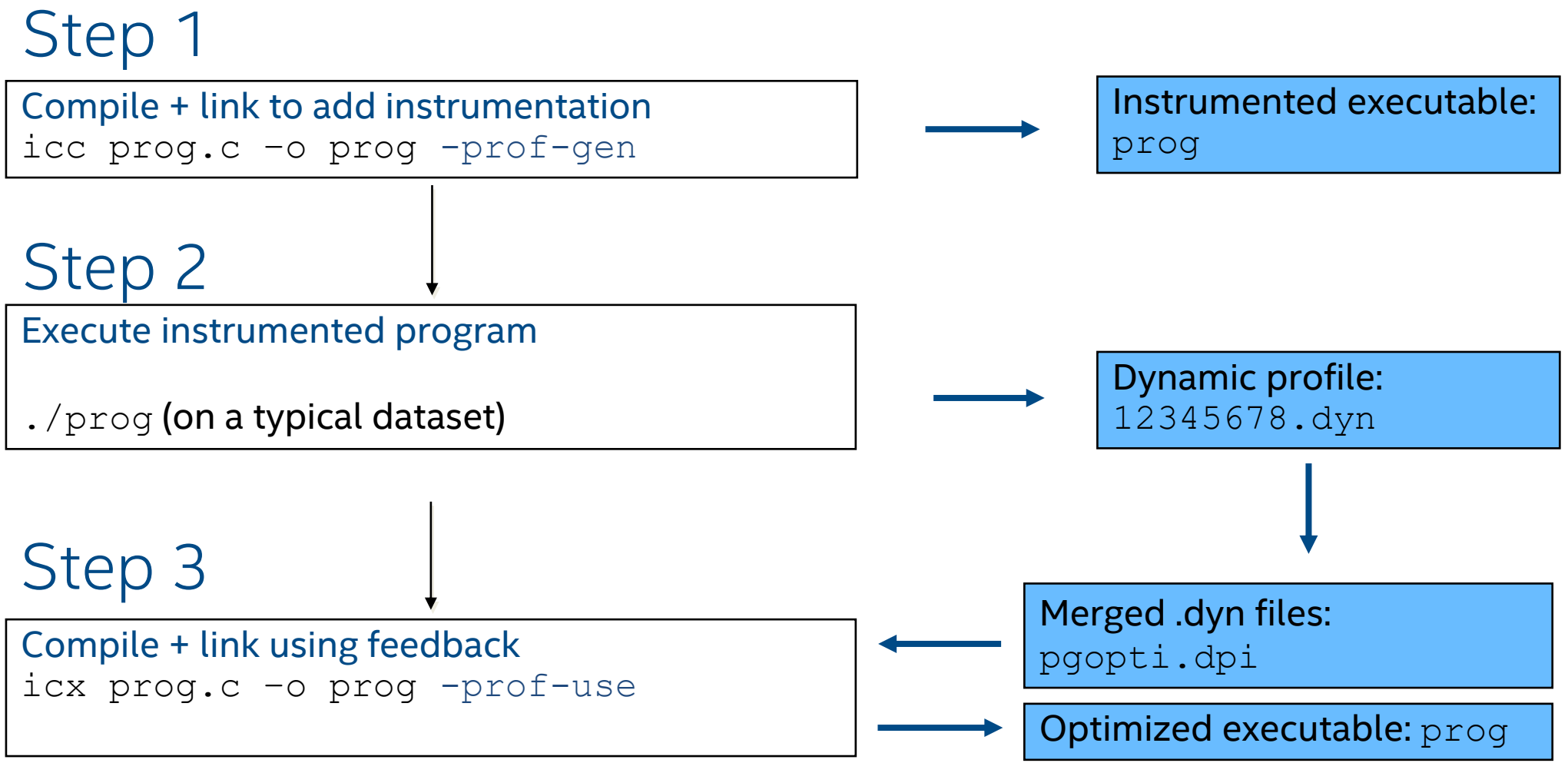
```
if (x > y)
     do_this();
 else
     do that();
```

```
for(i=0; i<count; ++i)
do_work();
```

- Use execution-time feedback to guide (final) optimization
- Enhancements with PGO:
  - More accurate branch prediction
  - Basic block movement to improve instruction cache behavior
  - Better decision of functions to inline (help IPO)
  - Can optimize function ordering
  - Switch-statement optimization
  - Better vectorization decisions

PGO Usage in ICC

Profiling with Instrumentation



PGO Usage in ICX

Profiling with Instrumentation

Step 1

Compile + link to add instrumentation

```
icx prog.c –o prog
-fprofile-instr-generate
```

→ Instrumented executable: `prog`

Step 2

Execute instrumented program

`./prog` (on a typical dataset)

→ Dynamic profile: `default.profraw`

```
llvm-profdata merge -output=code.profdata default.profraw
```

Step 3

Compile + link using feedback

```
icx prog.c –o prog
-fprofile-instr-use=code.profdata
```

← Merged profiles via `llvm-profdata`: `code.profdata`

→ Optimized executable: `prog`

tinyurl.com/clang-pgo

Changes in Diagnostics

Changes in Diagnostics

- icc uses -diag options to control the display of diagnostic information during compilation.

```
#include <iostream>

int main() {
    int X;
    std::cout << "Hello World!";
    return 0;
}
```

```
$ icpc -diag-enable=177 hello.cpp
hello.cpp(4): warning #177: variable "X" was declared but never referenced
    int X;
        ^
```

Changes in Diagnostics

- ICPX classifies diagnostic messages using descriptive phrases.

```
$ icpx -Wall hello.cpp
hello.cpp:4:9: warning: unused variable 'X' [-Wunused-variable]
    int X;
        ^
1 warning generated.
$ icpx -Werror=unused-variable hello.cpp
hello.cpp:4:9: error: unused variable 'X' [-Werror,-Wunused-variable]
    int X;
        ^
1 error generated.
```

- The clang diagnostics are continuously improving

Pragmas support

Do NOT assume ICC or GCC pragmas are supported by ICX!

- Use `-Wunknown-pragmas` to check for the unsupported pragmas

```
$ cat unknown-pragmas.c
int main(void) {
   float arr[1000];

   #pragma totallybogus           // pragma; does NOT exist in ICC Classic, GCC, or ICX => causes warning
   #pragma simd                   // ICC Classic pragma; NOT supported by ICX => causes warning
   #pragma vector                 // is recognized and implemented by ICX
   for (int k=0; k<1000; k++) {
      arr[k] = 42.0;
   }
}

$ icx -Wunknown-pragmas unknown-pragmas.c
unknown-pragmas.c:4:9: warning: unknown pragma ignored [-Wunknown-pragmas]
#pragma totallybogus
        ^
unknown-pragmas.c:5:9: warning: unknown pragma ignored [-Wunknown-pragmas]
#pragma simd
        ^
2 warnings generated.
```

Floating Point Reproducibility Controls

Floating-Point (FP) Programming Objectives

- Accuracy
  - Produce results that are “close” to the correct value
    - Measured in relative error, possibly in ulp
- Performance
  - Produce the most efficient code possible
- Reproducibility
  - Produce consistent results
    - From one run to the next
    - From one set of build options to another
    - From one compiler to another
    - From one platform to another

These objectives usually conflict!
Use of compiler options lets you control the tradeoffs.
Different compilers have different defaults.

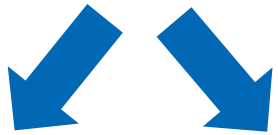
Floating Point Reproducibility Controls

- Default FP model: -fp-model fast=1 -fma
- No -fp-model consistent model option
  - Use -fp-model=precise -fimf-arch-consistency=true -no-fma
- No support for #pragma fenv_access, -fimf-use-svml, etc.
- Math library related features supported, e.g. -fimf-precision, -fimf-max-error, etc.

Auto-vectorization

Auto-vectorization

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 512; i++)
       sth[i] = sin(theta[i]+3.1415927);
}
```



```
vmovups   zmm1, ZMMWORD PTR [r12+rdi*4]                 #5.21
vextractf64x4 ymm2, zmm1, 1                             #5.21
vcvtps2pd zmm3, ymm1                                    #5.21
vaddpd    zmm0, zmm16, zmm3                             #5.30
vcvtps2pd zmm4, ymm2                                    #5.21
vaddpd    zmm17, zmm16, zmm4                            #5.30
call      QWORD PTR [__svml_sin8@GOTPCREL+rip]          #5.17
vmovaps   zmm18, zmm0                                   #5.17
vmovaps   zmm0, zmm17                                   #5.17
call      QWORD PTR [__svml_sin8@GOTPCREL+rip]          #5.17
vcvtpd2ps ymm2, zmm18                                   #5.17
vcvtpd2ps ymm1, zmm0                                    #5.17
vinsertf64x4 zmm3, zmm2, ymm1, 1                        #5.17
vmovups   ZMMWORD PTR [rsi+rdi*4], zmm3                 #5.8
```

```
vcvtss2sd xmm16, xmm16, DWORD PTR [r12+r14*4]           #5.17
vaddsd    xmm0, xmm16, QWORD PTR .L_2il0floatpacket.1[rip] #5.17
call      sin                                           #5.17
vcvtsd2ss xmm0, xmm0, xmm0                              #5.8
vmovss    DWORD PTR [r13+r14*4], xmm0
```

godbolt.org/z/dWvEh7GGc

Basic Vectorization Options

-x<code>

- Might enable Intel processor specific optimizations
- Processor-check added to “main” routine:
  Application errors in case SIMD feature missing or non-Intel processor with appropriate/informative message

<code> indicates a feature set that compiler may target (including instruction sets and optimizations)

- Microarchitecture code names: BROADWELL, HASWELL, SKYLAKE, SKYLAKE-AVX512, ICELAKE-SERVER, etc.
- SIMD extensions: CORE-AVX512, CORE-AVX2, CORE-AVX-I, AVX, SSE4.2, etc.
- Example: icx -xCORE-AVX2 test.c
  icx -xSKYLAKE test.c

Basic Vectorization Options

-ax<code>

- Multiple code paths: baseline and optimized/processor-specific
- Optimized code paths for Intel processors defined by <code>
- Not supported by icx yet

-m<code>

- No check and no specific optimizations for Intel processors:
  Application optimized for both Intel and non-Intel processors for selected SIMD feature
- Missing check can cause application to fail in case extension not available

- -xHost

Optimization Report

Optimization Report

- -qopt-report[=n]: tells the compiler to generate an optimization report
  - n: (Optional) Indicates the level of detail in the report (0-5).
  - Level 5 produces the greatest level of detail.
  - icx has n=3 as a max level and includes Loop Optimizations, OpenMP parallelization and Register Allocation messages

- -qopt-report-phase[=list] specifies one or more optimizer phases for which optimization reports are generated.
  - loop: the phase for loop nest optimization
  - vec: the phase for vectorization
  - par: the phase for auto-parallelization
  - all: all optimizer phases

- -qopt-report-filter=string: specified the indicated parts of your application, and generate optimization reports for those parts of your application.

Not supported in ICX yet

Optimization Report

```
$ icc -qopt-report=3 -qopt-report-phase=vec -qopt-report-file=stdout test.cpp
...
LOOP BEGIN at test.cpp(3,5)
<Peeled loop for vectorization, Multiversioned v1>
LOOP END

LOOP BEGIN at test.cpp(3,5)
<Multiversioned v1>
   remark #15300: LOOP WAS VECTORIZED
   remark #15442: entire loop may be executed in remainder
   remark #15448: unmasked aligned unit stride loads: 1
   remark #15449: unmasked aligned unit stride stores: 1
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 8
   remark #15477: vector cost: 1.500
   remark #15478: estimated potential speedup: 4.860
   ...
```

Optimization Report

$ icx **-qopt-report=3** test.cpp

Global loop optimization report for : _Z6vecAddPfS_S_i

LOOP BEGIN at test.cpp (3, 5)

LOOP END

$ icx **-qopt-report=3 -xcore-avx2** test.cpp

Global loop optimization report for : _Z6vecAddPfS_S_i

LOOP BEGIN at test.cpp (3, 5)

   remark: The loop has been multiversioned

LOOP END

LOOP BEGIN at test.cpp (3, 5)

<Multiversioned loop>

   remark #15300: LOOP WAS VECTORIZED

   remark #15305: vectorization support: vector length 8

   remark #15475: --- begin vector loop cost summary ---

   remark #15482: vectorized math library calls: 0

   remark #15484: vector function calls: 0

   remark #15485: serialized function calls: 0

   remark #15488: --- end vector loop cost summary ---

   remark #15447: --- begin vector loop memory reference summary ---

   remark #15450: unmasked unaligned unit stride loads: 2

   ...

Clang Optimization Report

-Rpass* flags for opt report in clang:

```
clang++ -O3 -Rpass-analysis=loop-vectorize -Rpass=loop-vectorize -Rpass-missed=loop-vectorize
-c example.cpp
```

```
 void vecAdd(float* a, float* b, float *c, int size)
 {
     for (int i = 0; i < size; i++)
         c[i] =a[i] + b[i];
 }

 #1 with x86-64 clang 12.0.0
 example.cpp:3:5: remark: vectorized loop (vectorization width: 4, interleaved count: 2)
 [-Rpass=loop-vectorize]
     for (int i = 0; i < size; i++)
     ^
```

-fsave-optimization-record and opt-viewer

clang++ -O3 -fsave-optimization-record -c example.cpp

opt-viewer.py example.opt.yaml

tinyurl.com/llvm-opt-viewer

Line	Hotness	Optimization	
1			`void vecAdd(float* a, float* b, float *c, int size)`
2			`{`
		prologepilog	0 stack bytes in function
		asm-printer	40 instructions in function
3			`for (int i = 0; i < size; i++)`
		loop-unroll	unrolled loop by a factor of 4 with run-time trip count
		asm-printer	+ BasicBlock:
		asm-printer	+ BasicBlock:
		asm-printer	+ BasicBlock:
		asm-printer	+ BasicBlock:
4			`c[i] =a[i] + b[i];`

Optimization Report

-qopt-report to generate YAML-formatted opt report:

icx -qopt-report=3 example.cpp

example.opt.yaml is generated

opt-viewer.py example.opt.yaml

Line	Hotness	Optimization	
1			`void vecAdd(float* a, float* b, float *c, int size)`
		regalloc	5 virtual registers copies 1.492647e+00 total copies cost generated in function
2			{
		prologepilog	0 stack bytes in function
		asm-printer	100 instructions in function
3			`for (int i = 0; i < size; i++)`
		loop-vectorize	Disabling scalable vectorization, because target does not support scalable vectors.
		loop-vectorize	vectorized loop (vectorization width: 4, interleaved count: 2)
		slp-vectorizer	List vectorization was possible but not beneficial with cost 0 >= 0
		loop-unroll	unrolled loop by a factor of 4 with run-time trip count
		loop-unroll	unrolled loop by a factor of 2 with run-time trip count

Note: -mllvm –inline-report=n 7 to get full IPO report

Intrinsic Usage Model

Intrinsic Usage Model

- Type checking for arguments to intrinsics
- Specific processor/architecture specific compiler option required
  - Use the compiler option -march or -m, -x
- immintrin.h header file

LLVM Intrinsics Handling Differences

Example

```
#include <immintrin.h>
#define CACHE_LINE_SIZE 64
__attribute__((always_inline))
inline void Prefetch_Block(const void* addr, size_t sz, int hint)
{
    char* pref_addr = (char*)addr;
    size_t pref_iters = (sz + CACHE_LINE_SIZE - 1) / CACHE_LINE_SIZE;
    for (int i = 0; i < pref_iters; i++)
    {
        _mm_prefetch(pref_addr, hint/*_MM_HINT_T1*/);
        pref_addr += CACHE_LINE_SIZE;
    }
}
```

```
$ icc -c sample_mm_prefetch.c
$ icx -c sample_mm_prefetch.c
sample_mm_prefetch.c:10:9: error: argument to '__builtin_prefetch' must be a constant integer
        _mm_prefetch(pref_addr, hint/*_MM_HINT_T1*/);
        ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/opt/intel/oneapi/compiler/2021.2.0/linux/lib/clang/12.0.0/include/xmmintrin.h:2103:31: note: expanded from macro '_mm_prefetch'
#define _mm_prefetch(a, sel) (__builtin_prefetch((const void *)(a), \
                              ^
1 error generated.
```

LLVM Intrinsics Handling Differences

Example

```
#include <immintrin.h>

double reduce_vector(__m256d input) {
  __m256d temp = _mm256_hadd_pd(input, input);
  return ((double*)&temp)[0] + ((double*)&temp)[2];
}
```

```
icx -c test.cpp
test.cpp(3,23): error: unknown type name '__m256d'
double reduce_vector1(__m256d input) {
                      ^

test.cpp(4,3): error: unknown type name '__m256d'
  __m256d temp = _mm256_hadd_pd(input, input);
  ^
2 errors generated.

icx -c -xAVX test.cpp
```

LLVM Intrinsics Handling Differences

Example

```
#include <immintrin.h>

__forceinline __m256 operator+( __m256 aValue1, __m256 aValue2)
{
        return _mm256_add_ps(aValue1,aValue2);
}

__forceinline __mmask16     operator<( __m512 aValue1, __m512 aValue2)
{
        return _mm512_cmplt_ps_mask(aValue2, aValue1);
}
```

```
$ icpc -c op_oveload.cpp
$ icpx -c op_oveload.cpp
op_oveload.cpp:3:22: error: overloaded 'operator+' must have at least one parameter of class or enumeration type
__forceinline __m256 operator+( __m256 aValue1, __m256 aValue2)
                     ^
op_oveload.cpp:8:29: error: overloaded 'operator<' must have at least one parameter of class or enumeration type
__forceinline __mmask16     operator<( __m512 aValue1, __m512 aValue2)
                            ^
2 errors generated.
```

LLVM Intrinsics Handling Differences

Example

```
#include "immintrin.h"
int main(){
    __m128i blocks = _mm_aesenc_si128(_mm_setzero_si128(), _mm_setzero_si128());
    return 0;
}
```

```
$ icc -xHASWELL test.c
$ icx -xHASWELL test.c
test.c:4:22: error: always_inline function '_mm_aesenc_si128' requires target feature 'aes', but would be
inlined into function 'main' that is compiled without support for 'aes'
    __m128i blocks = _mm_aesenc_si128(_mm_setzero_si128(), _mm_setzero_si128());
                     ^
1 error generated.

$ icx -xHASWELL -mintrinsic-promote test.c
test.c:3:5: warning: Function target features +aes added due to call to an intrinsic [-Wintrinsic-promote]
int main(){
    ^
test.c:4:22: note: Intrinsic called here
    __m128i blocks = _mm_aesenc_si128(_mm_setzero_si128(), _mm_setzero_si128());
                     ^
1 warning generated.
```

OpenMP* Support

OpenMP with Intel® C++ Compiler

- Full OpenMP* 5.0 TR4 support + some OpenMP 5.1 features
- Options

-fiopenmp, -fopenmp, -qopenmp

- Selects Intel Optimized OMP

-fopenmp-targets=spir64

- Needed for OMP Offload
- Generates SPIRV code fat binary for offload kernels

Get Started with OpenMP* Offload Feature to GPU: tinyurl.com/intel-openmp-offload

Performance

Looking for Best Compiler Options?

It depends!

- workload, hw, OS, compiler version, memory allocation, etc.
- take a look on benchmark results and options for reference:

ICC:

SPECint®_rate_base_2017: *-xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-mem-layout-trans=4*

SPECfp®_rate_base_2017: *-xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -ffinite-math-only -qopt-mem-layout-trans=4*

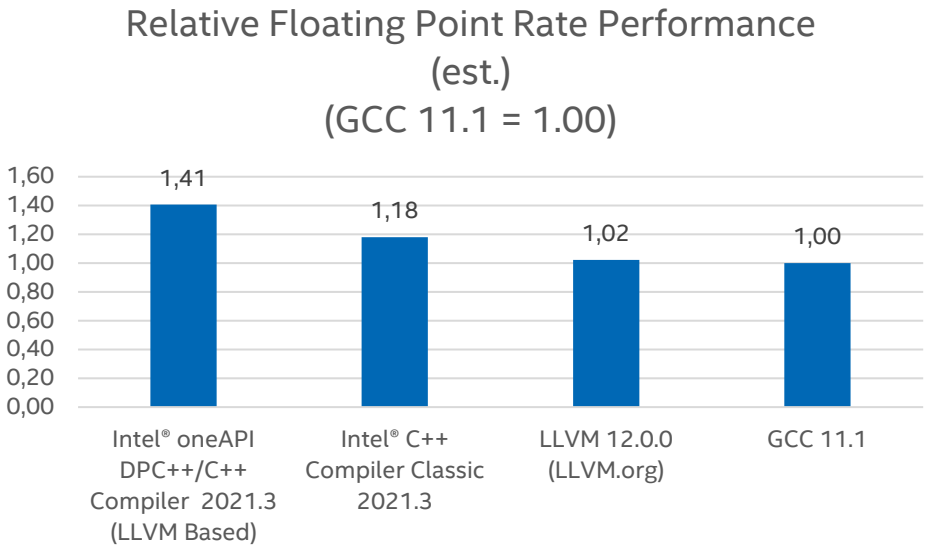
SPEC HPC2021: -qopt-zmm-usage=high -Ofast -xCORE-AVX512 -qopenmp -ipo -qopt-multiple-gather-scatter-by-shuffles -fimf-precision=low:sin,sqrt [for IFORT: -align array64byte -nostandard-realloc-lhs]

ICX:

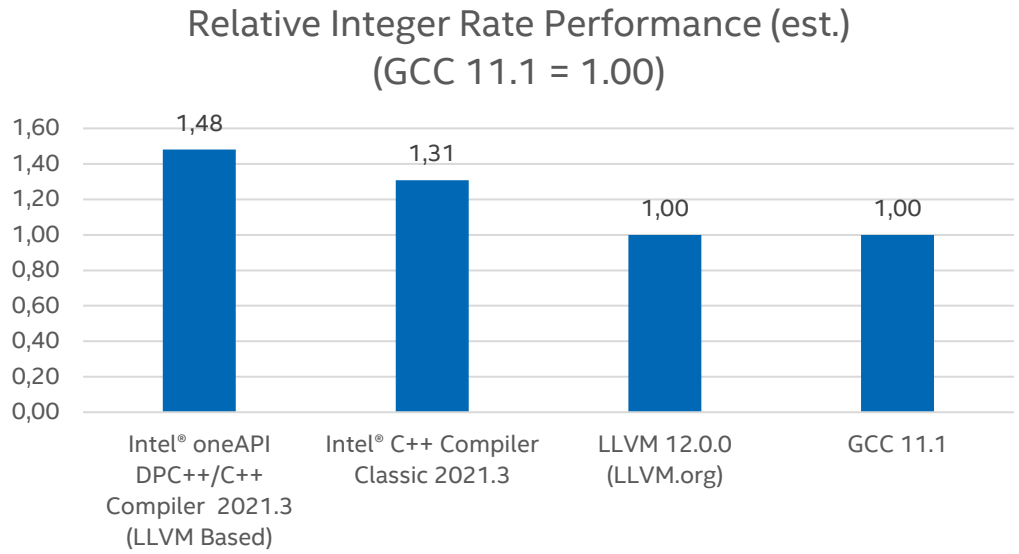
SPEC HPC2021: -mprefer-vector-width=512 -Ofast -xCORE-AVX512 -ffast-math -fiopenmp -flto -fimf-precision=low:sin,sqrt -funroll-loops [for IFX: -align array64byte -nostandard-realloc-lhs]

Intel® Compilers Boost C++ Application Performance on Linux*

Performance advantage relative to other compilers on Intel® Xeon Platinum 8280 Processor



Estimated: internal measurement of the geometric mean of the C/C++ workloads from the SPECrate* 2017 Floating Point suite



Estimated: internal measurement of the geometric mean of the C/C++ workloads from the SPECrate* 2017 Integer suite

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

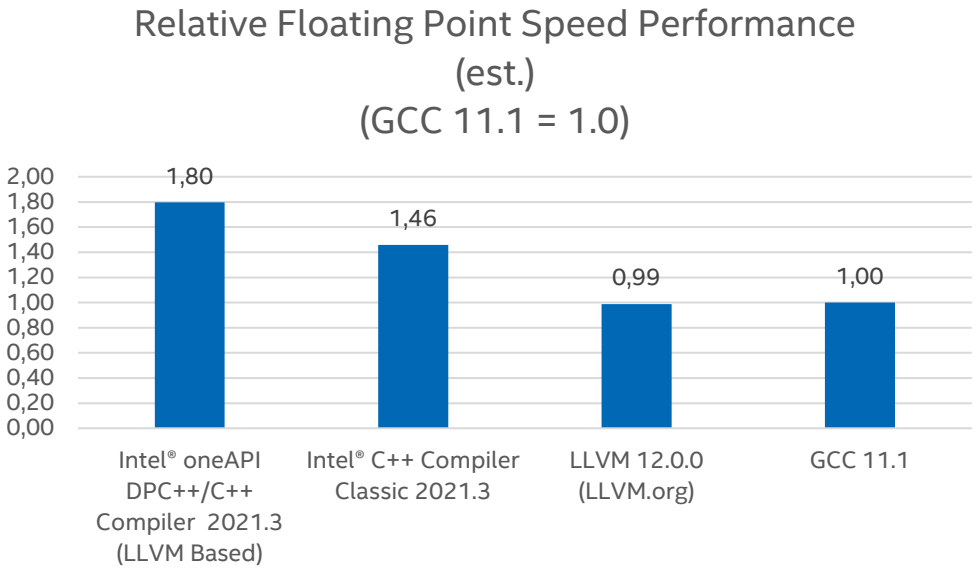
Your costs and results may vary. Intel technologies may require enabled hardware, software, or service activation.

More information on the SPEC benchmarks can be found at: http://www.spec.org

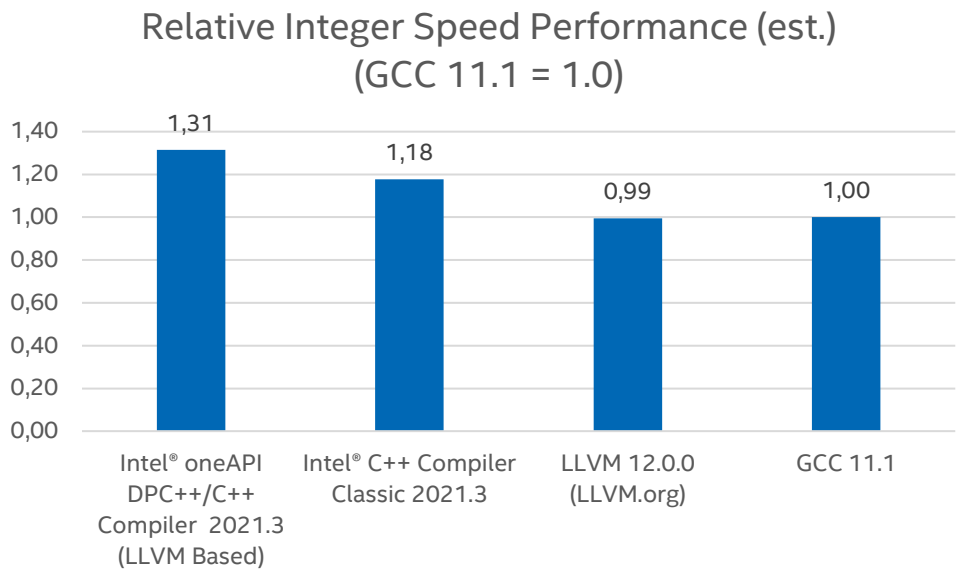
Configuration: Testing by Intel as of Jun 10, 2021 -Intel(R) Xeon(R) Platinum 8380 CPU @ 2.30GHz, 2 socket, Hyper Thread on, Turbo on, 32G x12 DDR4 3200 (1DPC). Software: Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2021.2.0 Build 20210317, Intel(R) C++ Compiler Classic for applications running on Intel(R) 64, Version 2021.3.0 Build 20210604_000000, GCC 11.1, Clang/LLVM 12.0.0. Red Hat Enterprise Linux release 8.2 (Ootpa), 4.18.0-193.el8.x86_64. SPECint®_rate_base_2017 compiler switches: Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2021.2.0 Build 20210317: -xCORE-AVX512 -O3 -ffast-math -flto -mfpmath=sse -funroll-loops -qopt-mem-layout-trans=4. qkmalloc used. Intel(R) C++ Compiler Classic for applications running on Intel(R) 64, Version 2021.3.0 Build 20210604_000000: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-mem-layout-trans=4 -qopt-multiple-gather-scatter-by-shuffles. qkmalloc used. GCC 11.1 -march=znver1 -mfpmath=sse -Ofast -funroll-loops -flto. LLVM 12.0.0: -march=core-avx2 -mfpmath=sse -Ofast -funroll-loops –flto. SPECfp®_rate_base_2017 compiler switches: Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2021.1 Build 20201113: -xCORE-AVX512 -Ofast -ffast-math -flto -mfpmath=sse -funroll-loops -qopt-mem-layout-trans=4. jemalloc 5.0.1 used.Intel(R) C++ Compiler Classic for applications running on Intel(R) 64, Version 2021.3.0 Build 20210604_000000:-xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -ffinite-math-only -qopt-multiple-gather-scatter-by-shuffles -qopt-mem-layout-trans=4. jemalloc 5.0.1 used. GCC 11.1: -march=skylake-avx512 -mfpmath=sse -Ofast -fno-associative-math -funroll-loops -flto. jemalloc 5.0.1 used. LLVM 12.0.0: - march=znver1 -mfpmath=sse -Ofast -funroll-loops –flto. jemalloc 5.0.1 used.

Intel® Compilers Boost C++ Application Performance on Linux*

Performance advantage relative to other compilers on Intel® Xeon Platinum 8280 Processor



Estimated: internal measurement of the geometric mean of the C/C++ workloads from the SPECspeed* 2017 Floating Point suite



Estimated: internal measurement of the geometric mean of the C/C++ workloads from the SPECspeed* 2017 Integer suite

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Your costs and results may vary. Intel technologies may require enabled hardware, software, or service activation.

More information on the SPEC benchmarks can be found at: http://www.spec.org

Configuration: Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2021.1 Build 20201113: -xCORE-AVX512 -O3 -ffast-math -flto -mfpmath=sse -funroll-loops -qopt-mem-layout-trans=4 -fiopenmp. jemalloc 5.0.1 used. Intel(R) C++ Compiler Classic for applications running on Intel(R) 64, Version 2021.3.0 Build 20210604_000000: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-mem-layout-trans=4 -qopt-multiple-gather-scatter-by-shuffles -qopenmp. jemalloc 5.0.1 used. GCC 11.1: - march=znver1 -mfpmath=sse -Ofast -funroll-loops -flto -fopenmp. jemalloc 5.0.1 used. LLVM 12.0.0: -march=core-avx2 -mfpmath=sse -Ofast -funroll-loops -flto -fopenmp=libomp. jemalloc 5.0.1 used. SPECfp®_speed_base_2017 compiler switches: Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2021.1 Build 20201113:-xCORE-AVX512 -Ofast -ffast-math -flto -mfpmath=sse -funroll-loops -qopt-mem-layout-trans=4 -fiopenmp. jemalloc 5.0.1 used. Intel(R) C++ Compiler Classic for applications running on Intel(R) 64, Version 2021.3.0 Build 20210604_000000: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -ffinite-math-only -qopt-multiple-gather-scatter-by-shuffles -qopenmp. jemalloc 5.0.1 used. GCC 11.1: -march=skylake-avx512 -mfpmath=sse -Ofast -fno-associative- math -funroll-loops -flto -fopenmp. jemalloc 5.0.1 used. LLVM 12.0.0: -march=skylake-avx512 -mfpmath=sse -Ofast -funroll-loops -flto -fopenmp=libomp.jemalloc 5.0.1 used.

Notices & Disclaimers

- This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice.
- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
- 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. For more complete information visit www.intel.com/benchmarks.
- 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.
-

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