Transition of Intel® C/C++ Compilers

Hardware Complexity Driving Compiler Opportunity

Hardware complexity

- Modern compute complexity
- Accelerator compute complexity
- Domain specific compute complexity

Need for innovation in modern compilers and programming languages

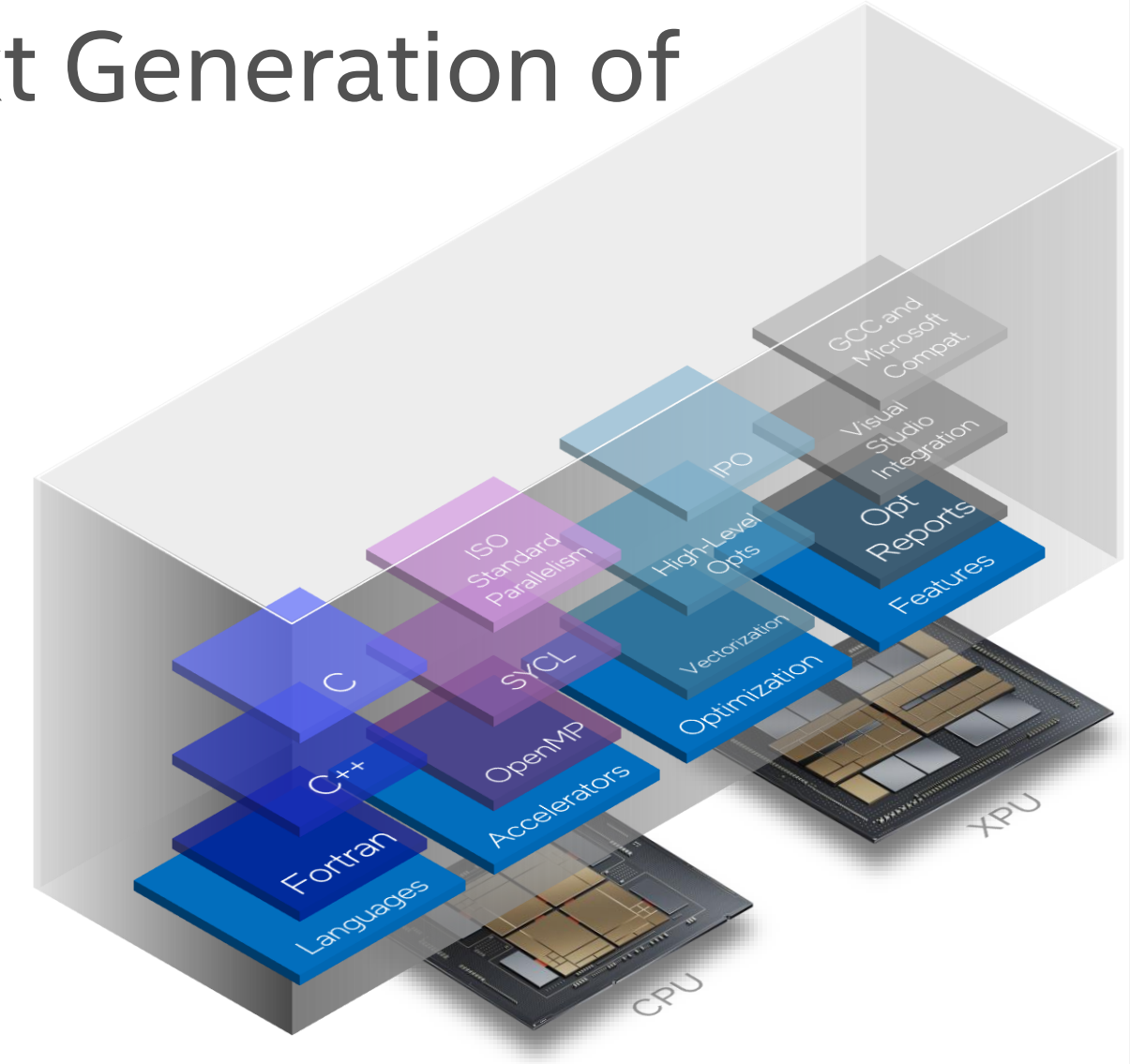
- Hardware and accelerator abstractions
- Domain specific programming models
- Quality, reliability, scalability and performance

LLVM Powering the Next Generation of Compilers

intel

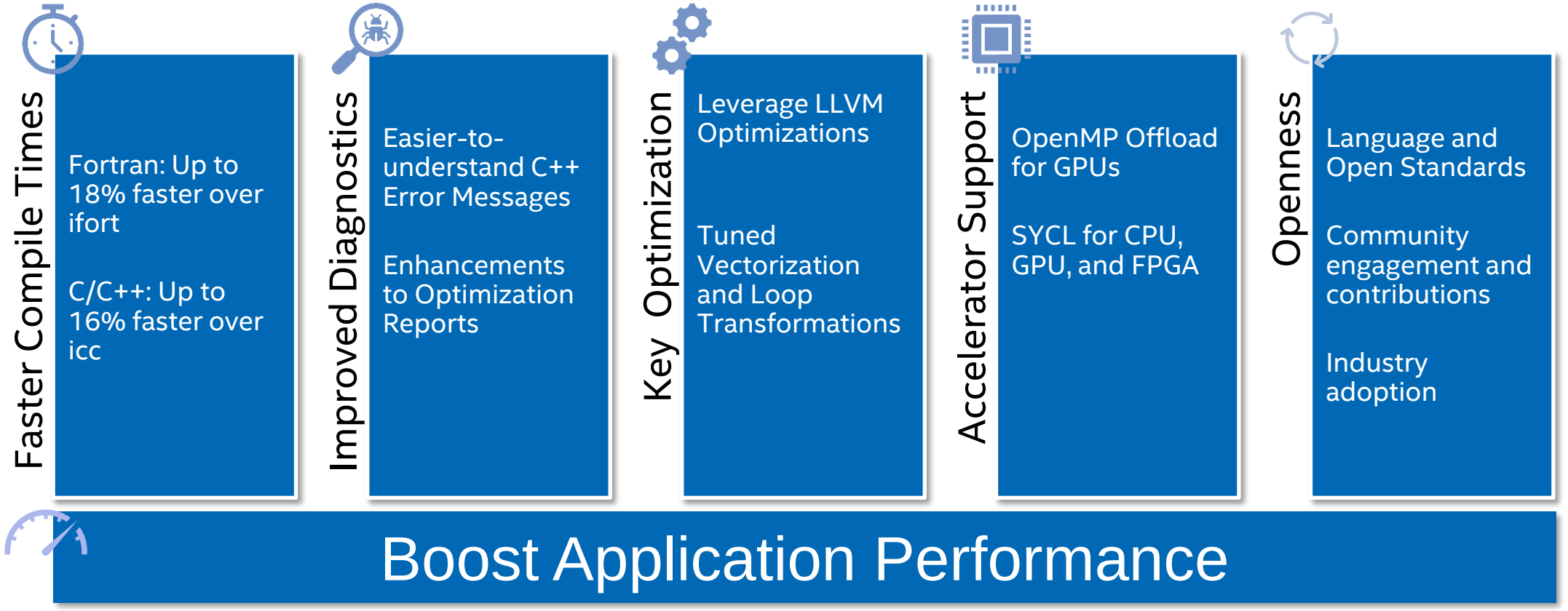
+

=



Motivation

Why did we re-design our compilers leveraging LLVM?



Leveraging & Contributing to LLVM

Why LLVM?

Power of the Community

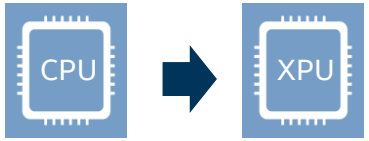
Security

Flexibility

Modern Infrastructure

Why For Intel?

Active Member & Upstream



Inflection Point to XPU Future

Effective Use of Resources Develop Faster

Why For Customers?

intel

Expertise of Intel & Community

Faster Time to Standards

Intel Support & Commitment

Faster Time to Performance & Architectures

Key Knowledge for Intel® Compilers Going Forward

- New underlying back-end compilation technology based on LLVM
- Shipping today in Intel® oneAPI Base & HPC Toolkit for C/C++, SYCL, and Fortran
- Existing Intel proprietary "IL0" (icc, ifort) compilation technology compilers provided alongside new compilers – names using "Compiler Classic" to distinguish from new LLVM-based compilers
- **Offload compute only with new LLVM-based compilers**

Intel® C++ Compiler Classic has been deprecated as of Q3 2022 and is targeted to be removed from the oneAPI package in Q4 2023. Start migration from ICC to ICX now.

What's New: Intel® oneAPI DPC++/C++ Compiler

Intel oneAPI DPC++/C++ Compiler (icx/dpcpp) – based on modern LLVM technology

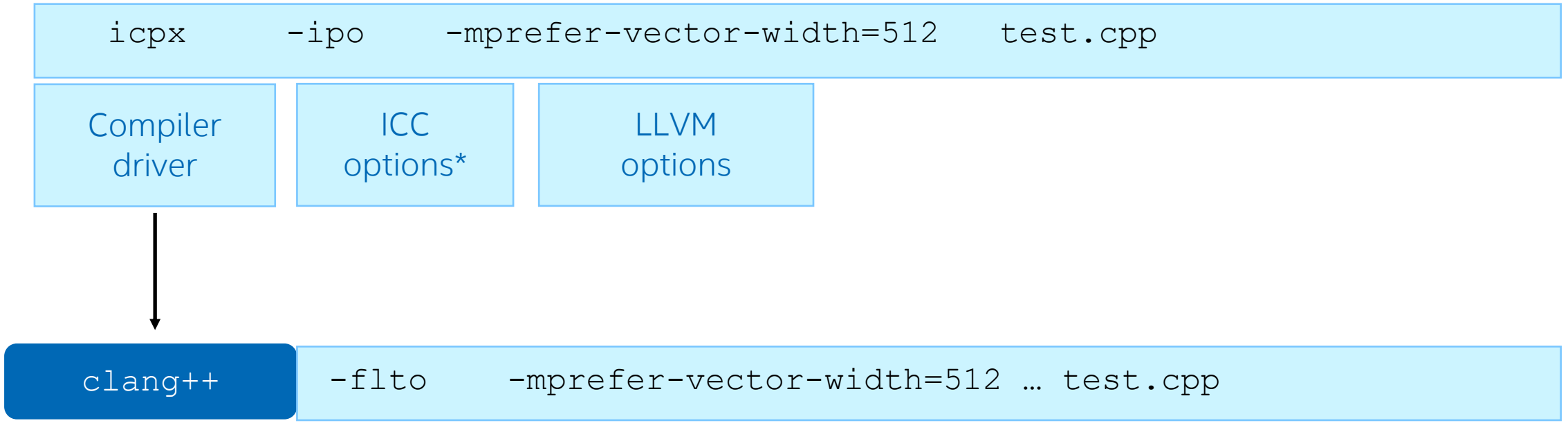
- The Intel® oneAPI DPC++/C++ Compiler further improves accelerated computing support through the addition of newly added SYCL 2020 and OpenMP 5.x features.
- Support for the Intel® Data Center GPU Flex/Max Series (formerly Ponte Vecchio).
- Backend code generation and tuning for the 4th Gen. Intel® Xeon® Scalable Processors, Max Series CPUs (formerly Sapphire Rapids).
- Intel oneAPI DPC++/C++ Compiler now defaults to the more recent ISO C++17 language support.
- New standard features have been added and enhanced for C23, C++20, C++23.
- Intel® oneAPI DPC++/C++ Compiler plugin architecture allowed Codeplay to add 3rd party GPU support

Intel® C++ Compiler Classic (icc)

- The Intel C++ Compiler Classic (icc) has been deprecated and has entered Long-Term Support with 2023.0. Please start using Intel® oneAPI DPC++/C++ Compiler.
- The Intel C++ Compiler Classic (icc) has been updated to include recent versions of 3rd party components, which include functional and security updates.

Each icx/dpcpp update will provide more performance, C/C++ and SYCL language, OpenMP, and new platform support

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

- 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 equivalent
- Binaries compiled with icc and icx and IPO are not compatible

```
$ 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
```

- Use llvm-ar for libraries
- Make sure tools from bin-llvm folder are used

Floating Point Reproducibility Controls

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

Looking for Best Compiler Options?

It depends!

- workload, hw, OS, compiler version, memory allocation, etc.

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 ]`