Creating
World
Changing
Technologies

Driving a New Era of Accelerated Computing
Intel® Fortran Compiler (IFX)

intel®

Our Fortran Solution 2022

Our Fortran Solution - Compilers

Intel® Fortran Compiler Classic (ifort)

Best-In-Class Fortran language features
and performance for CPU ***today!***

Intel® Fortran Compiler (ifx)

Driving a new era in accelerated computing!
Our Fortran compiler solution for Intel GPU offload
Committed to overall Best-in-Class Fortran ***for 2023 release***

Because you need advanced Fortran language features and the absolute best performance for your applications on Intel solutions

We deliver
CHOICE! Continuity! Features! Performance!

Our Fortran Solution – Compilers Details

- Two separate compilers. Same Intel Fortran Frontend (FFE). Both compilers in all packages. CHOICE! Continuity!

- ifort - Intel Fortran parser/analyzer + Intel optimizer/code generation
  - CPU only classic compiler. NO OFFLOAD TO GPU
  - Full F2018 support, best performance: Features! Performance!
  - Named “Intel® Fortran Compiler Classic”

- ifx – Intel Fortran parser/analyzer + LLVM optimizer and code generation (with Intel enhancements)
  - Supports OpenMP Offload to Intel GPUs Features!
  - F2003/2008 excluding Parameterized Derived Types & Coarrays
  - Binary compatible with DPCPP, ICX, ICC, IFORT
  - Named “Intel® Fortran Compiler”

A Fortran Solution – Complementary Compilers

- IFX is not a replacement for IFORT in 2022
- IFX provides outstanding OpenMP 5.x acceleration to Intel GPU
- IFORT is the best in class Fortran 2018 compiler for CPU
- Binary compatibility means you get the best of both

- Together you get the best Fortran SOLUTION for xPU in 2022

Intel® Compilers – Target & Packaging

Compilers are Binary Compatible and Linkable!

Intel Compiler	Driver	Target*	OpenMP CPU Support	OpenMP Offload Support
Intel® Fortran Compiler Classic	***ifort***	CPU	Yes	No
Intel® Fortran Compiler	***ifx***	CPU, GPU	Yes	Yes
Intel® C++ Compiler Classic	***icc***	CPU	Yes	No
Intel® oneAPI DPC++/C++ Compiler	***dpcpp***	CPU, GPU, FPGA	Yes	Yes and No*
	icx	CPU GPU	Yes	**Yes**

Creating
World
Changing
Technologies

IFX 2022

Status of features and performance in Intel® oneAPI 2022.x Products

intel®

IFX: Driving a New Era in Accelerated Computing

The **Intel® Fortran Compiler** (IFX) is driving a new era of accelerated computing across XPU architectures (CPU, GPU). IFX is our Fortran compiler going forward.

- Supports OpenMP* 5.x Standards to enable GPU offload from Fortran
  - *No need to call C/C++ or proprietary APIs for GPU acceleration!*
  - *An open, portable Standard to maintain your software investment*
- Same Fortran parser/analyzer (front end) you know and love from IFORT
  - **Supports legacy DEC extensions** and most of F03/F08* (see next slide) and… majority of IFORT compiler directives and options you have used for years.
  - And Microsoft Visual Studio* integration for Windows*
- HOWEVER … IFX still in development throughout 2022 with the goal to reach IFORT feature & *general performance parity** by the end of 2022.

* * *average performance of many apps, but not every app*

IFX Status, in Intel® oneAPI 2022.1.x

OpenMP 5.0 and 5.1 majority subset support (see Reference slides)

Complete Fortran 2003 and Fortran 2008 Standard features EXCEPT

- Fortran 2003 parameterized derived types
  - coming in oneAPI 2022.2 Update 1, ifx comp. version v2022.1.0
- Fortran 2008 coarrays coming in Update 2
- Fortran 2018 features coming, a lot in Update 2, finish at end of 2022

Performance: IFX may or may not match performance of ifort compiled applications. Improvements coming with each Update release

Each Update will provide more Fortran Language & OpenMP features AND performance improvement. Stay up to date!

Creating
World
Changing
Technologies

IFX 2022

Getting Started

intel

Important High-Level Understanding of IFX

- Although both IFORT and IFX use the same language parser (Fortran Front End)
  EVERYTHING after that is DIFFERENT

- Key takeaway – optimization, vectorization, optimization reports, inlining, unrolling, interprocedural optimization, profile guided optimization, floating point control, code generation is all different. SO …

- Examine the compiler options you are using, remove most of the "exotic" performance options.
  Start with simplest subset like
  `-O2 –xhost -flto`

Be aware that the default optimizations used by ifx are aggressive.

You may specify the following flags to ifx to turn off default optimizations:

- On Linux: `-O0 -fno-fast-math`
- On Windows: `/O0 /fp:precise`

Currently, To get Intel optimizations over LLVM defaults YOU MUST USE **–x<arch>**

AND you SHOULD use **–flto**

IFX Fortran Language Support

- First, check state of IFX Fortran Standard support
  - -stand [f90 | f95 | f03 | f08 | f18 | nostand] default is NOSTAND or no standards warnings
  - Use **`-stand f03`** if you want warnings for features not in F2003
  - Use **`-stand f03 –warn errors`** options to abort if any detected.

    Latest updates listed here

    https://software.intel.com/content/www/us/en/develop/articles/fortran-language-and-openmp-features-in-ifx.html

IFX Essentials – IFX Options Support

Undocumented IFORT options are not implemented and there is no plan to implement them.

IFX options that are implemented are accepted quietly (no msg)

IFX options that are not implemented generate this warning:

ifx: command line warning #10430: Unsupported command line options encountered
These options as listed are not supported.
For more information, use '-qnextgen-diag'.

ifx -qnextgen-diag

Prints a long list of IFORT options TO BE supported

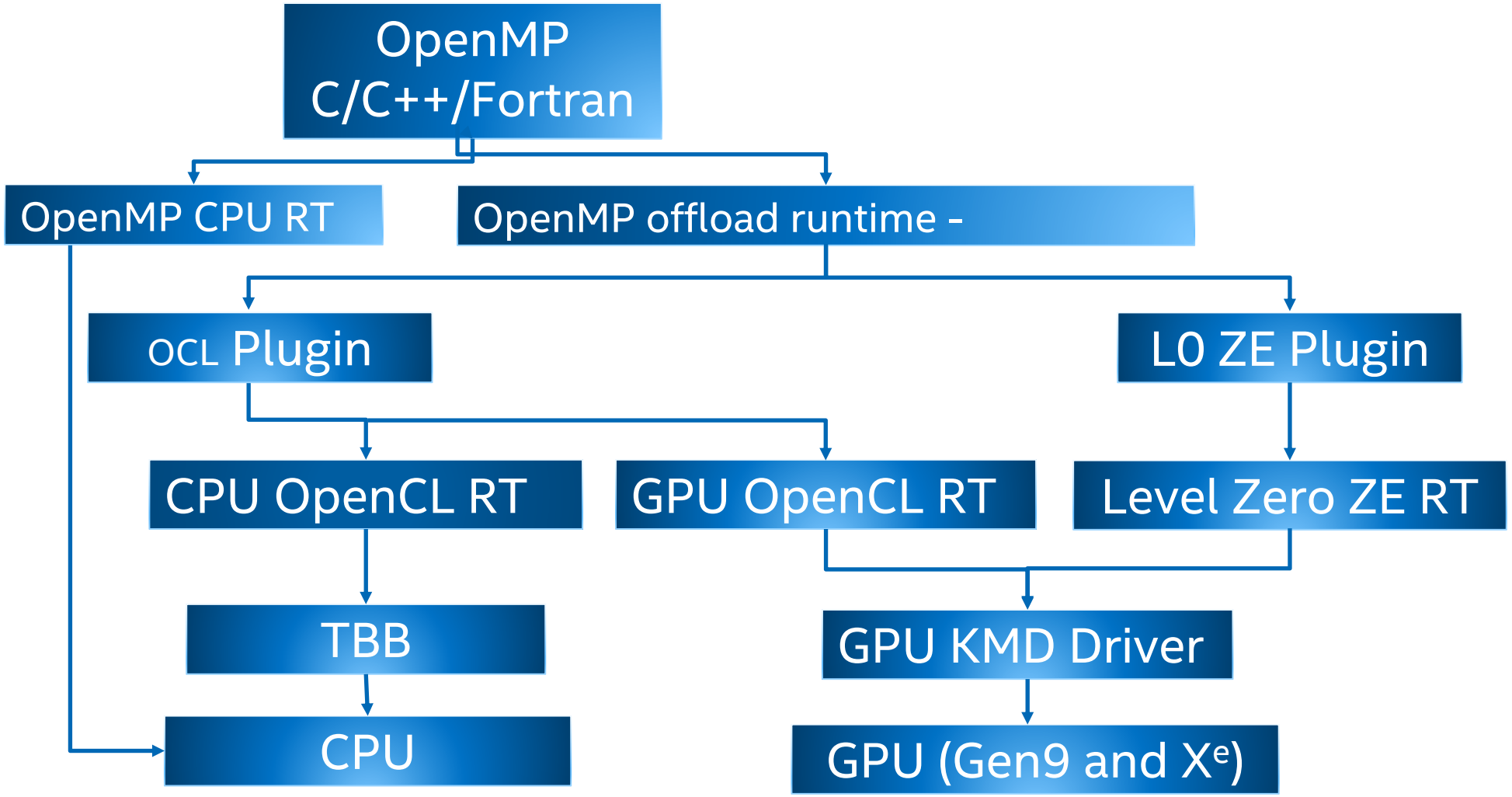
And prints a long list of IFORT options that are removed

Common Optimization Options

	Linux* ifx (ifort)
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 or -qopenmp (-qopenmp)

OpenMP

High level architecture



Prerequisites

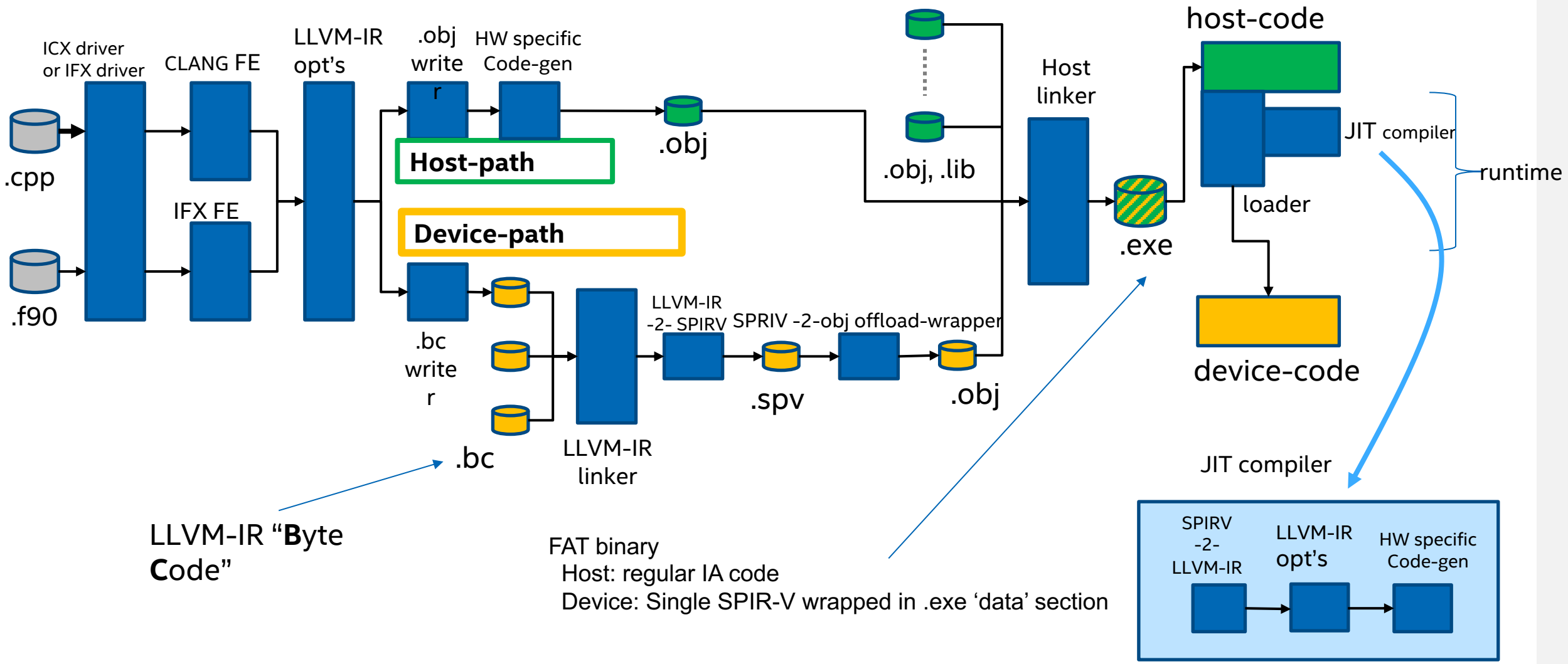
- OCL driver is installed separately, it's a prerequisite

https://dgpu-docs.intel.com/installation-guides/index.html

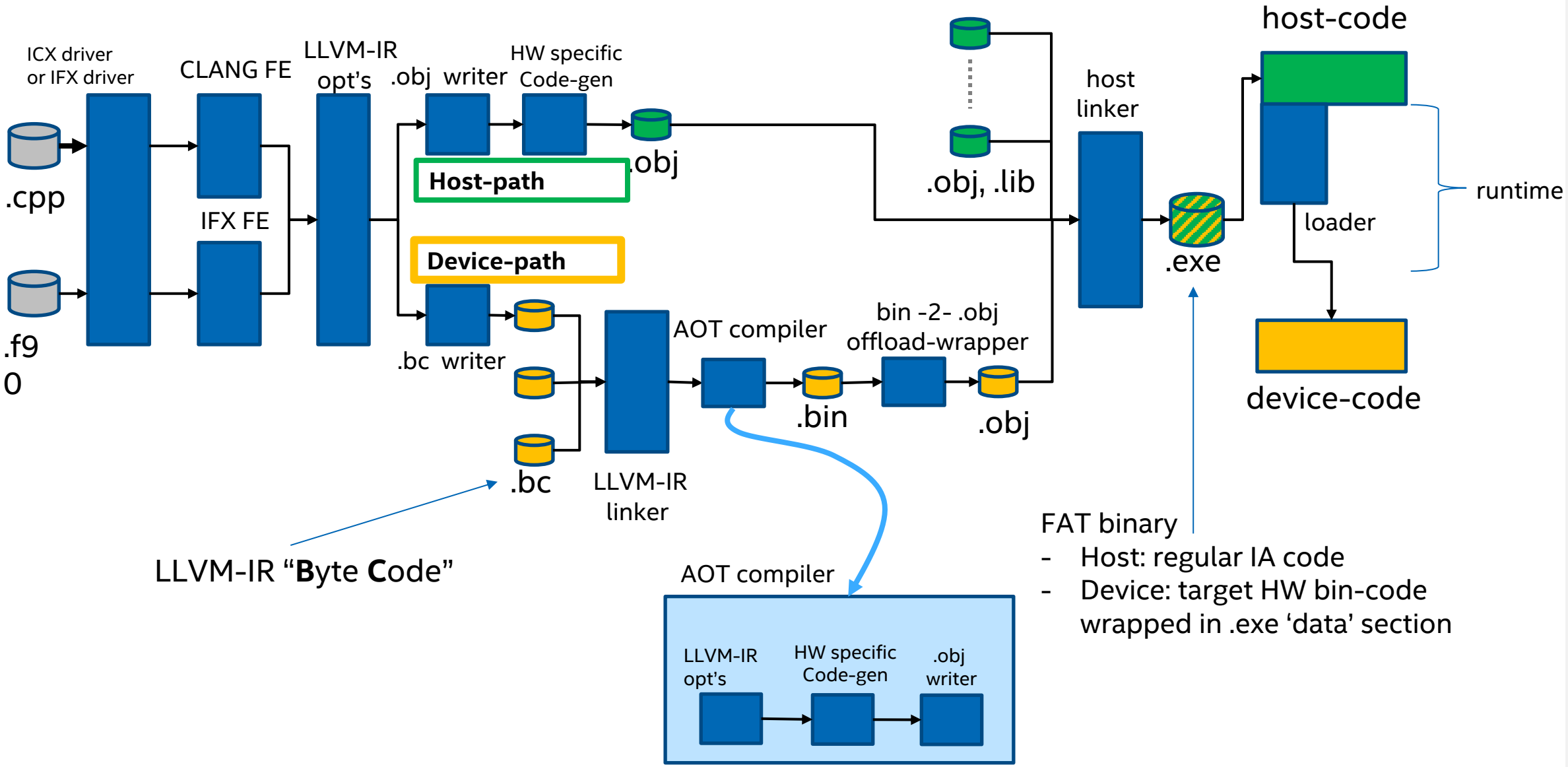
- TBB is needed if you want to target CPU target device.
  - From oneAPI Base Toolkit (downloads or pkg managers)
  - From Standalone Component Downloads
    - https://www.intel.com/content/www/us/en/developer/articles/tool/oneapi-standalone-components.html

https://www.intel.com/content/www/us/en/developer/articles/guide/installation-guide-for-oneapi-toolkits.html

Just-In-Time (JIT) Compilation Flow



Ahead-Of-Time (AOT) Compilation Flow



Intel Fat Binary Format

- Advantage: Fat binary has advantage that executable or libraries have both host and SPIRV or device code
- Disadvantage: Standard LLVM tools do not recognize this format
- Consequence: Intel provided replacements for working with Fat binaries:
- Use IFX as linker for any app/library with fat object files
- Use our LLVM bin tools replacements (next slide)

```
<root>/oneapi/compiler/[latest | <ver>]/bin-llvm
```

Intel Binary Tools

- Replacement utils for LLVM tools manipulating our fat binaries:
- Not in default path so as to not name collide with system LLVM and tools.
- Location:

<root>/oneapi/compiler/[latest | <ver>]/bin-llvm

- lld, llvm-ar, llvm-ranlib, llvm-link, etc.
- If you have ICX installed, full path can be printed:
  - icx --print-prog-name=llvm-ar

OpenMP Offload with Intel® Compilers

Built-in Support for Intel® Xe

OpenMP with Intel® Compilers

Drivers
icx (C/C++) ifx (Fortran)
Adheres to OpenMP spec directives to target for offload
OPTIONS

-fiopenmp

Selects Intel Optimized OMP
-fopenmp maps to –fiopenmp, deprecated will be removed
-qopenmp maps to –fiopenmp

-fopenmp-targets=spir64

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

JIT compilation

icpx –fiopenmp –fopenmp targets=spir64 source.cpp

ifx –fiopenmp –fopenmp targets=spir64 source.f90

AOT compilation – *Docs Coming soon*

icpx -fiopenmp -fopenmp-targets=**spir64_gen**
-Xopenmp-target-backend "-device <dev>" source.cpp

ifx -fiopenmp -fopenmp-targets=**spir64_gen**
-Xopenmp-target-backend "-device <dev>" source.f90

<dev> is your target, use 'ocloc compile –help' for list of targets

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

Example: Simple Matrix Multiply Offload

Transfer control and data from the host to the device

Syntax

!$omp target *[clause[[,] clause],…]*
structured-block

Clauses for TARGET

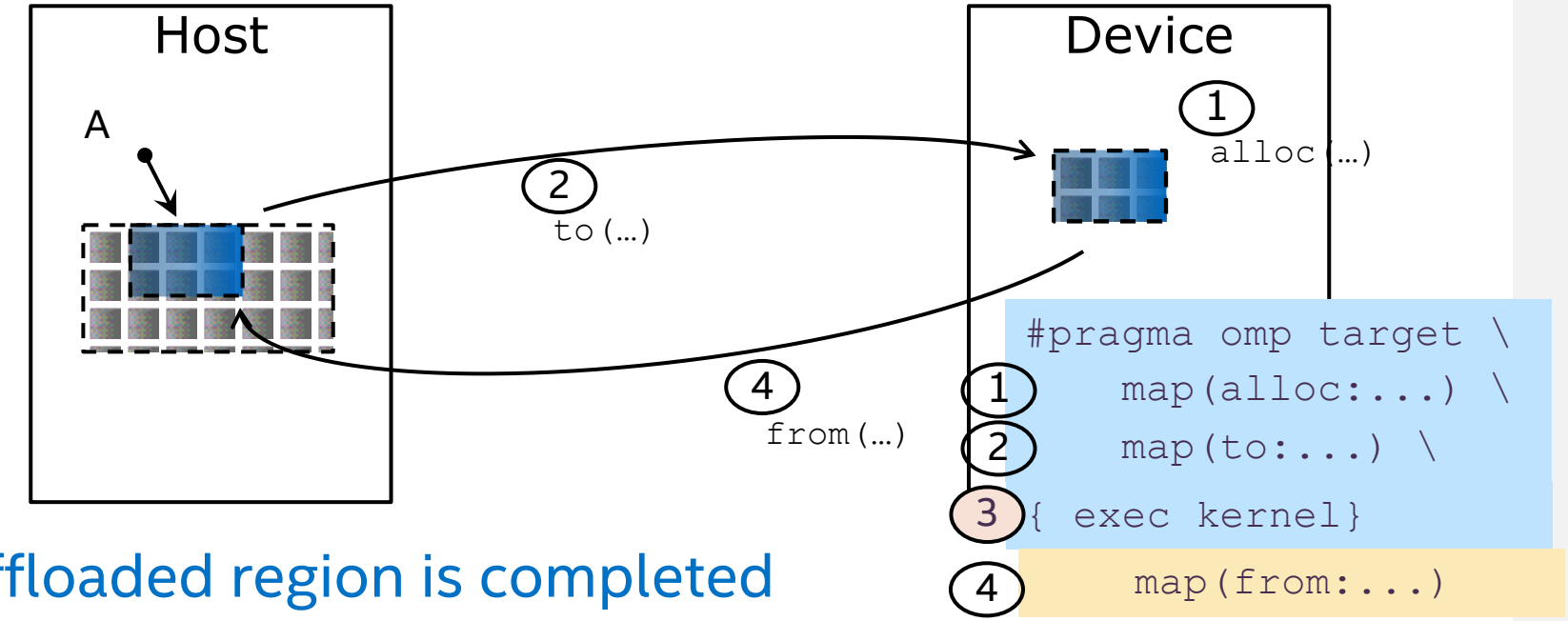
device(*scalar-integer-expression*)
map(*[{*alloc | to | from | tofrom*}:] list*)
if(*scalar-expr*)

These OMP pragmas cause the loop to execute on a target device (i.e., GPU)

```
program matrix_multiply
use omp_lib
implicit none
integer, parameter :: N=1000
integer :: i, j, k, my_thread_id
real, allocatable, dimension(:,:) :: a, b, c, c_validate

allocate( a(N,N), b(N,N), c(N,N), c_validate(N,N))
! Initialize the arrays A and B, set C to 0.0 (not shown)

!... offload data & compute matrix multiply on the GPU
!... send 'a' and 'b' but do not move them back (no change)
!... 'c' goes to GPU and brought back from GPU (changed)

!$omp target map(to: a, b ) map(tofrom: c )
!$omp parallel do

do j=1,N
  do i=1,N
    do k=1,N
      c(i,j) = c(i,j) + a(i,k) * b(k,j)
    enddo
  enddo
enddo
!$omp end parallel do
!$omp end target
```

Offloading and Device Data Mapping

- Use *target* construct to
  - Transfer control from the host to target device
  - Map variables between the host and target device data environments



- Host thread waits until offloaded region is completed
  - Use other OpenMP tasks for asynchronous execution
- The **`map`** clauses determine how an *original variable* in a data environment is mapped to a *corresponding variable* in a device data environment
- OpenMP also provides Unified Shared Memory IF you want the data mapping to be automatic (not shown, not in IFX as of Q1 2022, coming soon)

OpenMP Essential Environment Variables

- Helping you guide your OpenMP Runtime

Essential Environment Variables

- Select Target Device with Environment variable
  OMP_TARGET_OFFLOAD = mandatory | disabled | default
  - mandatory – The `target` region runs code on GPU or other accelerator
  - disabled – The `target` region code runs on CPU
  - default - The `target` region runs on GPU if device is available, else will fall back to the CPU

- Select Plugin/Driver
  LIBOMPTARGET_PLUGIN= [OPENCL | LEVEL0]
  LIBOMPTARGET_DEVICETYPE= gpu | cpu (only works for OpenCL)

- Performance profiling for tracking on GPU kernel start/complete time and data-transfer time.
  LIBOMPTARGET_PLUGIN_PROFILE

  Dumps offloading runtime debugging information.
  LIBOMPTARGET_DEBUG= [1 | 2]
  LIBOMPTARGET_INFO (see LLVM Runtimes document URL below)

  https://openmp.llvm.org//design/Runtimes.html

Essential Intel env Var LIBOMPTARGET_PROFILE

LLVM OpenMP Runtime ENV vars are accepted. Example

export LIBOMPTARGET_PLUGIN_PROFILE=T

performance profiling for tracking on GPU kernel start/complete time and data-transfer time.

```
GPU Performance (Gen9, export LIBOMPTARGET_PROFILE=T,usec)
… …
Kernel Name:
__omp_offloading_811_29cbc383__ZN12BlackScholesIdE12execute_partEiii_l368
iteration #0 ...
calling validate ... ok
calling close ...
execution finished in 1134.914ms, total time 0.045min
passed

LIBOMPTARGET_PROFILE:
-- DATA-READ: 16585.256 usec
-- DATA-WRITE: 9980.499 usec
-- EXEC-__omp_offloading_811_29cbc383__ZN12BlackScholesIfE12execute_partEiii_l368:
24048.503 usec
```

Debug RT env var LIBOMPTARGET_DEBUG

Export LIBOMPTARGET_DEBUG=1

Dumps offload runtime debug information. Default value is 0 indicates no offloading runtime debugging information dump.

```
./matmul

Libomptarget --> Loading RTLs...
Libomptarget --> Loading library 'libomptarget.rtl.nios2.so'...
Libomptarget --> Loading library 'libomptarget.rtl.x86_64.so'...
Libomptarget --> Successfully loaded library 'libomptarget.rtl.x86_64.so'!
Libomptarget --> Loading library 'libomptarget.rtl.opencl.so'...

Target OPENCL RTL --> Start initializing OpenCL
Target OPENCL RTL --> cl platform version is OpenCL 2.1 LINUX
Target OPENCL RTL --> Found 1 OpenCL devices
Target OPENCL RTL --> Device#0: Genuine Intel(R) CPU 0000 @ 3.00GHz

…. AND MUCH MORE …
```

Perfect for bug reports!

Some Quick Notes

- Classic Compilers (icc / ifort) support CPU OpenMP features
- LLVM-based Compilers needed for OMP Offload features
- USM ready for Intel® DPC++/C++ Compiler (icx/dpcpp)
- USM Coming Soon for Intel® Fortran Compiler (ifx)
  - We can go over USM in future, separate session on OpenMP Offload

Resources

Support for Compilers IFORT and IFX

- Summary: Same support model we have used for years:
- Current version fully supported
- 2 previous versions supported but only the last Update release to that version
- AND available but unsupported - next older version, last Update only, provided for download on Intel® Registration Center but not supported
- This means …
  - IFORT will continue to be supported per our usual model.
  - We will ensure you have Fortran compiler solutions that are Best-in-Class

  - https://www.intel.com/content/www/us/en/developer/articles/release-notes/intel-parallel-studio-xe-supported-and-unsupported-product-versions.html

IFX OpenMP Features and Support

IFX Fortran Language &
OpenMP Features Support
https://www.intel.com/content/www/us/en/developer/articles/technical/fortran-language-and-openmp-features-in-ifx.html

Porting Guide, ifort to ifx

Kept up to date with tips and techniques to help you move from ifort to ifx

https://www.intel.com/content/www/us/en/developer/articles/guide/porting-guide-for-ifort-to-ifx.html

Intel® Fortran Compiler Classic and Intel® Fortran Compiler Developer Guide and Reference

https://www.intel.com/content/www/us/en/develop/documentation/fortran-compiler-oneapi-dev-guide-and-reference/top/compilation/supported-environment-variables.html

System and Driver Prerequisites

System Requirements

https://software.intel.com/content/www/us/en/develop/articles/intel-oneapi-base-toolkit-system-requirements.html

- Driver downloads and installation guides

  https://dgpu-docs.intel.com/installation-guides/index.html

Installation guides

https://software.intel.com/content/www/us/en/develop/articles/installation-guide-for-intel-oneapi-toolkits.html


Questions?
Thank You for Attending!

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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. https://software.intel.com/en-us/articles/optimization-notice

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. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

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.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.


intel

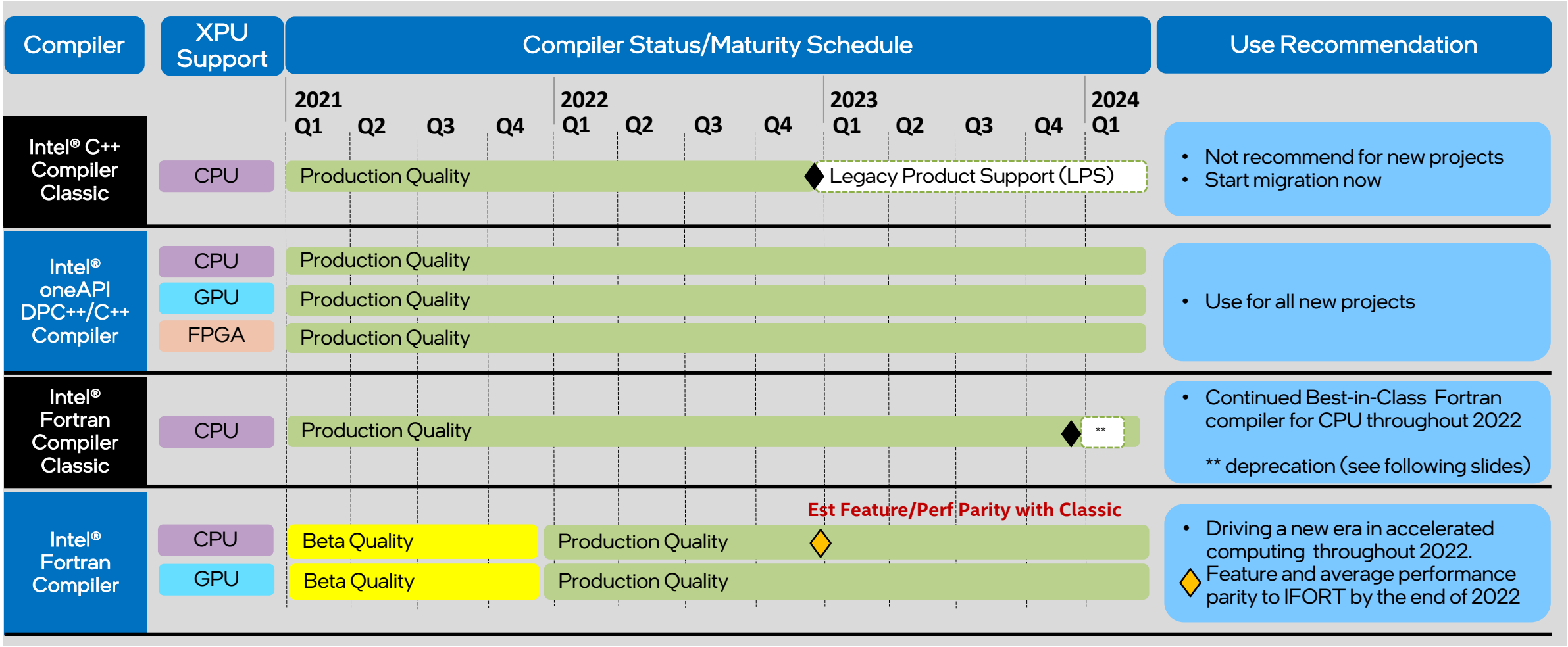
Backup Materials

Additional Material

For your further study

intel®

Intel Compilers Roadmap Q1 2022



CPU = Intel® Xeon® and Core™ processors
GPU = Intel® Integrated and discrete GPU's
FPGA = Intel® FPGA's (Stratix and Arria)

◆ ***IFX not "a replacement" for IFORT in 2022***

Our Fortran Solutions – Definition of Production Quality

- The Intel® Fortran Compiler v2022 (IFX) is Production Quality. What does this mean?
- Production Quality indicates that the compiler has passed our key performance metrics and validations and is ready for use. As always, language and compiler features will be continued to be developed and optimized and delivered in future releases.
  - IFX provides production quality Fortran OpenMP 5.x offload support
  - IFORT provides best in class Fortran 2018 for CPU
  - Binary compatibility provides a complete Fortran Solution for xPU
- Most important is for the customer to provide feedback to Intel on any technical issues that arise from the compilers so that they may be addressed accordingly.

Our Fortran Solutions – Definition of Deprecation

“Deprecation” similar to Language Standards definition:

> The act or process of marking the feature or product as obsolete, to discourage its use and warn users that it *may* be phased out in the future, but not removing the capability immediately, so as to allow for continued compatibility for a period of time.

Auto-Vectorization

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: ifx -xCORE-AVX2 test.f90
  ifx -xSKYLAKE test.f90

NEW! Not in docs in Q1 2022: **`ifort | ifx -xsapphirerapids`**

Basic Vectorization Options

-ax<code> COMING soon (summer/fall 2022)

- Multiple code paths: baseline and optimized/processor-specific
- Optimized code paths for Intel processors defined by <code>
- Not supported by icx/ifx in 1st half of 2022

-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
  - Host only: Target current CPU. Note: does nothing to identify/target GPUs

Some Notes on Processor Targeting

- -x/-ax SUGGESTION or REQUEST to compiler, not imperative

- Both compilers sometime use AVX2 instead of AVX512
- Force CLASSIC compiler to prefer AVX512 to AVX2 with **–qopt-zmm-usage=high**

```
ifort -xsapphirerapids –qopt-zmm-usage=high …
```

- LLVM compilers use –mprefer-vector-width=512

```
ifx –xsapphirerapids –mprefer-vector-width=512 …
```

IFX/ICX -x and Intel Optimizations

- Classic compilers IFORT/ICC performed Intel-specific optimizations at -O2. Additional vec optimizations done if –x used.

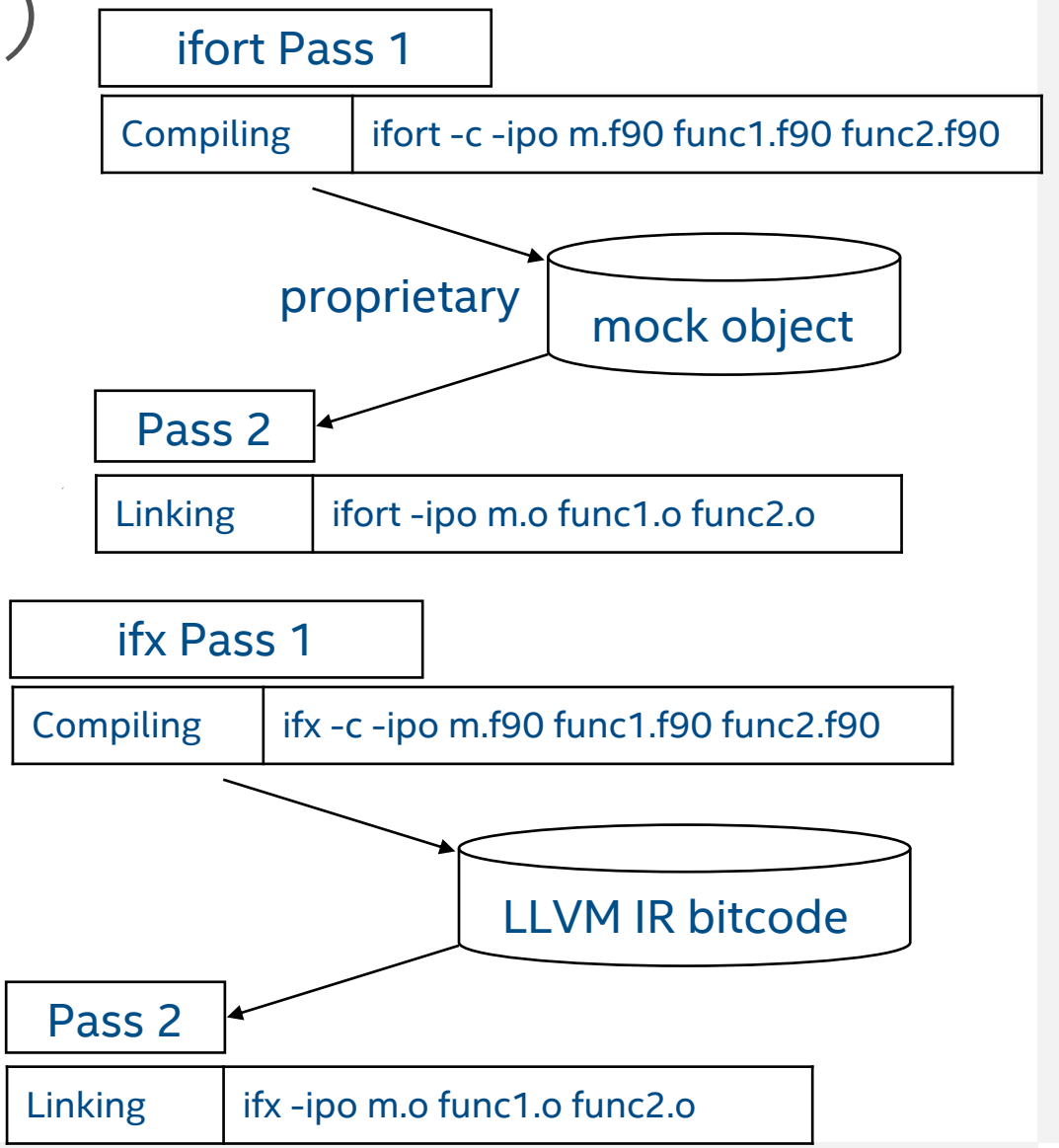
- ***LLVM compilers IFX/ICX will only do Intel specific optimizations with –x or –ax options***
  - Without –x or –ax you get default LLVM optimizations and vectorization
  - IMPLICATION: You will ONLY get the Intel optimizations and performance with IFX if and only if you use –x or –ax options. -O2 is NOT enough with IFX

Interprocedural Optimizations

Classic Compiler Interprocedural Optimizations (IPO)
IFX Link Time Optimization (LTO)

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
  - Classic Ifort creates proprietary object format
  - IFX creates bytecode LLVM IR



Interprocedural Optimizations

Extends optimizations across file boundaries

Ifort/icc	-ip	Only between modules of one source file
	-ipo	Modules of multiple files/whole application
Ifx/icx	-ipo | -flto (-ipo mapped to -flto)	Link Time Optimization in IFX / ICX

Without IPO

Compile & Optimize → file1.o

Compile & Optimize → file2.o

Compile & Optimize → file3.o

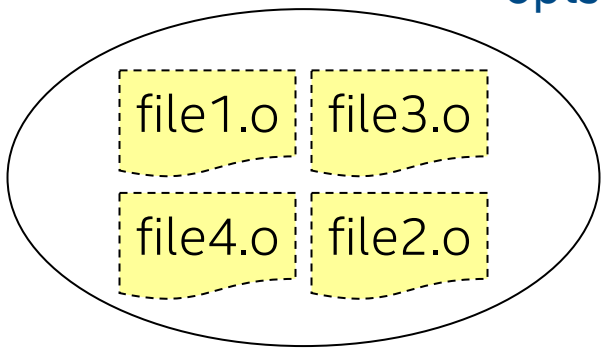
Compile & Optimize → file4.o

No opts across .o's

With IPO

Compile & Optimize

opts across all .o's



Cmake and IPO / LTO

- There is also a Cmake support for IPO and LTO:
- It's possible to enable LTO per default by setting CMAKE_INTERPROCEDURAL_OPTIMIZATION to TRUE:

```
set(CMAKE_INTERPROCEDURAL_OPTIMIZATION TRUE)
```

- Cmake – use version 3.21.3 or greater for oneAPI ICX / IFX support. DPCPP work in progress

IPO Objects Do Not Mix-and-Match with LTO Objects

- ifx uses Link Time Optimization (LTO) technology (-flto) and bitcode format
- -ipo should be added to both compilation and linking steps (or replace original linker with the '`lld –fuse-ld=lld`')
- Intel Classic and IPO binary format and tools 'xilink', 'xild', and 'xiar' are not compatible with LTO. Can't mix the 2

```
$ ifort -ipo -c hello.f90
$ ifx -ipo hello.o -o hello
/usr/bin/ld: hello.o:(.data+0x0): undefined reference to
`__must_be_linked_with_icc_or_xild'
```

```
$ ifx -ipo -c hello.f90
$ ifort  -ipo hello.o -o hello
ipo: warning #11003: no IR in object file hello.o; was the source file compiled with -ipo
```

```
$ ifort -ipo -c hello.f90
$ file hello.o
hello.o: ELF 64-bit LSB relocatable,
x86-64, version 1 (GNU/Linux), not
stripped
```

```
$ ifx -ipo -c hello.f90
$ file hello.o
hello.o: LLVM IR bitcode
```

https://llvm.org/docs/LinkTimeOptimization.html

Profile Guided Optimization

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

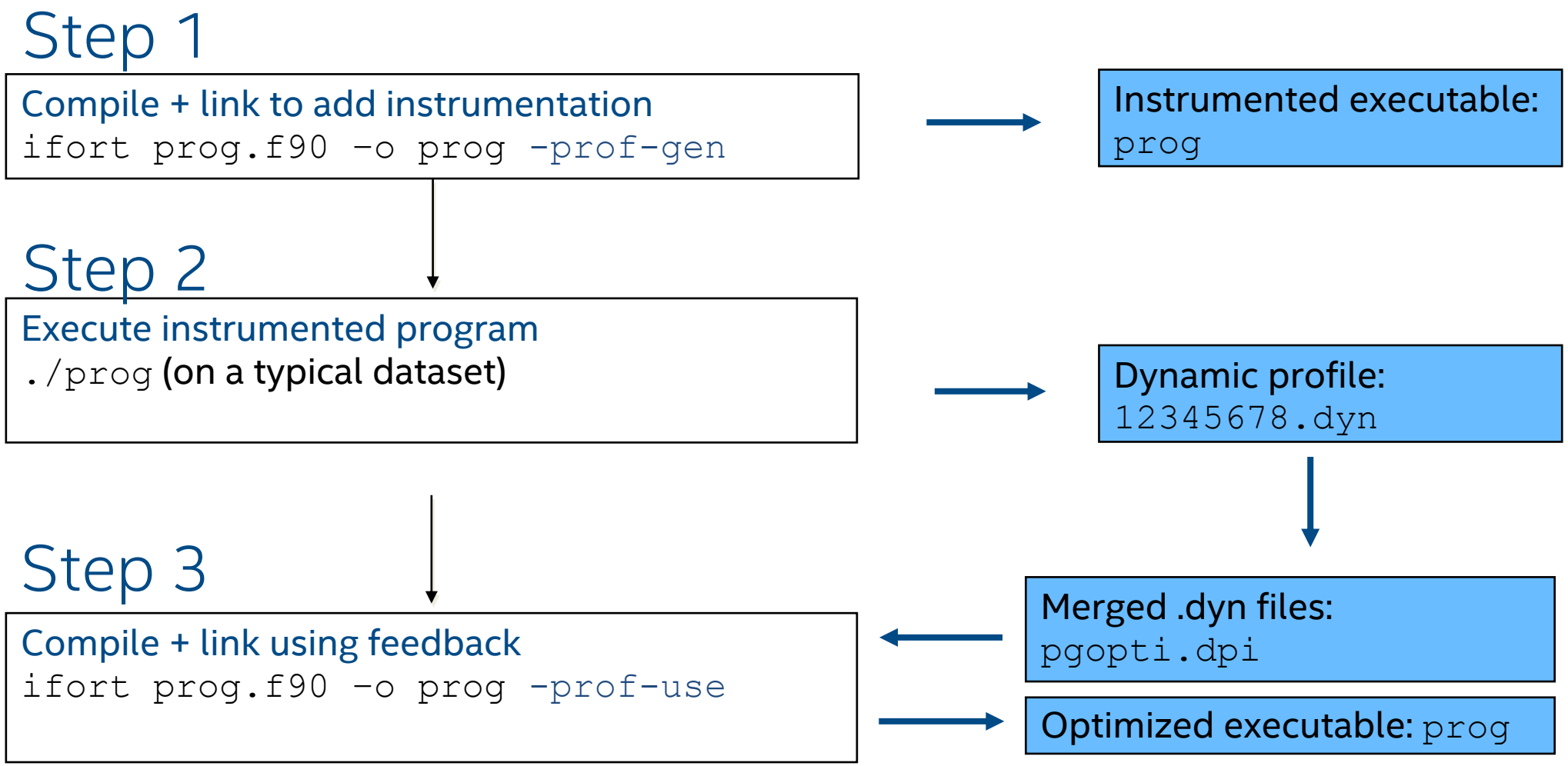
```
if (x > y)
do_this();
 else
do that();
```

```
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

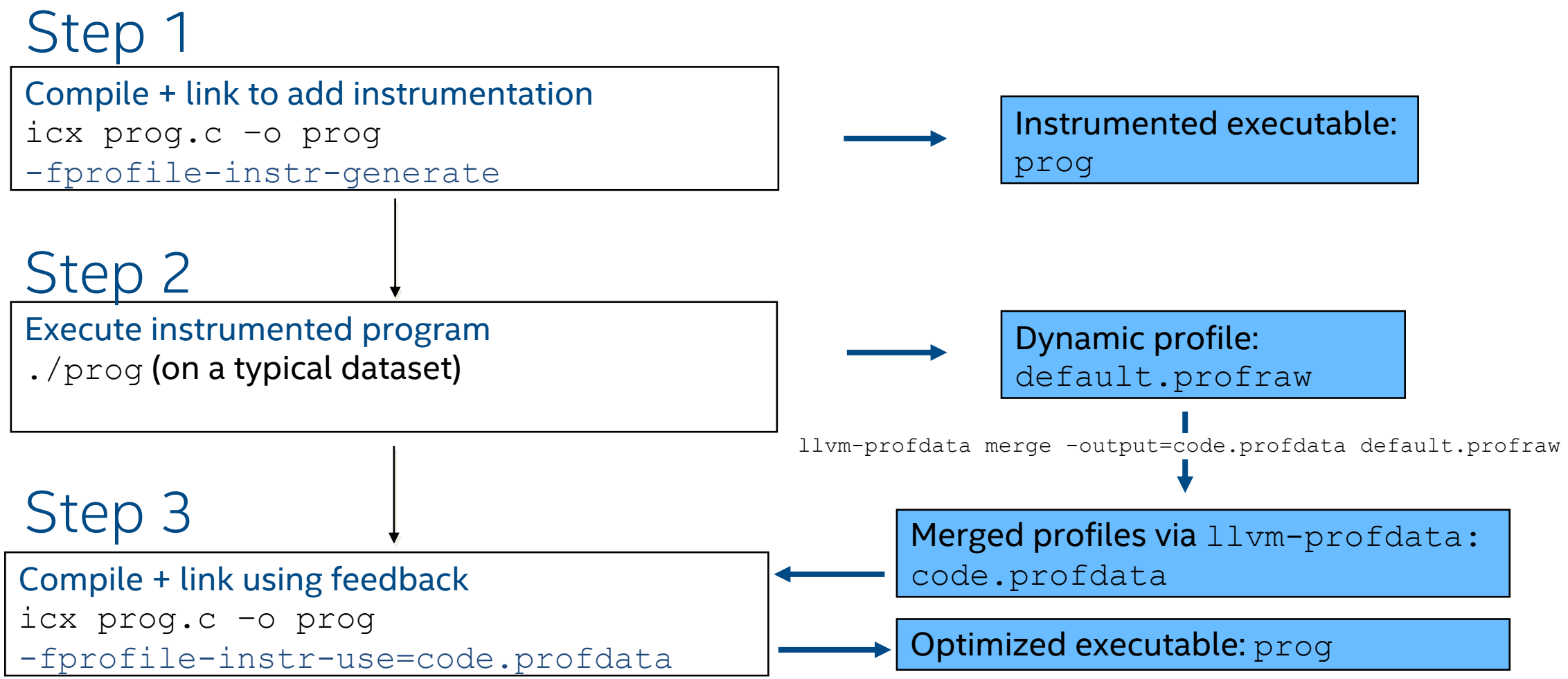
PGO Usage in Classic Compilers (icc/ifort)

Profiling with Instrumentation



PGO Usage in ICX. IFX Support TBD

Profiling with Instrumentation



tinyurl.com/clang-pgo

Optimization Reports

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.
  - **Ifx/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/IFX yet

Classic Compilers Optimization Report

```
$ ifort -qopt-report=3 -qopt-report-phase=vec -qopt-report-file=stdout test.f90
...
LOOP BEGIN at test.f90(3,5)
<Peeled loop for vectorization, Multiversioned v1>
LOOP END

LOOP BEGIN at test.f90(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
   ...
```

LLVM-based Compilers Optimization Report

$ ifx **-qopt-report=3 –O2 –xhost matmul-offload.f90**

Global loop optimization report for : _MAIN

LOOP BEGIN at matmul-offload.f90 (25, 5)

remark #15553: loop was not vectorized: outer loop is not an auto-vectorization candidate at -O2. Consider using -O3.

LOOP BEGIN at matmul-offload.f90 (24, 9)

remark #15300: LOOP WAS VECTORIZED

remark #15305: vectorization support: vector length 16

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: 0

remark #15451: unmasked unaligned unit stride stores: 2

remark #15456: masked unaligned unit stride loads: 0

remark #15457: masked unaligned unit stride stores: 0

remark #15458: masked indexed (or gather) loads: 0

remark #15459: masked indexed (or scatter) stores: 0

remark #15462: unmasked indexed (or gather) loads: 0

remark #15463: unmasked indexed (or scatter) stores: 0

remark #15554: Unmasked VLS-optimized loads (each part of the group counted separately): 0

remark #15555: Masked VLS-optimized loads (each part of the group counted separately): 0

remark #15556: Unmasked VLS-optimized stores (each part of the group counted separately): 0

remark #15557: Masked VLS-optimized stores (each part of the group counted separately): 0

remark #15474: --- end vector loop memory reference summary ---

LOOP END

LOOP END

Note: opt-report for IFX goes to STDERR.
Redirect to save to file using shell redirect or
‘ … |& tee <filename>’