Creating World Changing Technologies

intel®

Creating World Changing Technologies

Driving a New Era of Accelerated Computing using OpenMP* with Intel® oneAPI Compilers

Varsha Madananth, Ron Green

Intel Compiler Engineering and Technical Consulting

With technical contributions from:

Xinmin Tian, Senior Principal Engineer

Karl Qi, Application Engineer, Intel Accelerated Computing Systems and Graphics Group

intel.

Agenda

- Key Concepts for Intel® oneAPI Compilers
- Intel’s Compiler and Software Stack
  - Architecture
  - JIT and AOT
- OpenMP Offload with Intel® Compilers
- Essential Environment Variables
- OpenMP USM Support
- OpenMP Feature Support
- What we Learned & Call to Action

Key Concepts for Intel® oneAPI Compilers

Moving forward AND keeping you productive

Evolution of OpenMP Programming Model

- CPUs and All forms of accelerators/coprocessors, GPU, APU, GPGPU, FPGA, and DSP
- Heterogenous consumer devices
  - ✓ Kitchen appliances, drones, signal processors, medical imaging, auto, telecom, automation, not just graphics engines

Key Concepts for Intel® oneAPI Compilers

- New Compilation Technology based on LLVM
- New compiler technology available today in oneAPI Toolkits for DPC++, C++ and Fortran (DPCPP, ICX, IFX)
- Existing Intel proprietary "IL0" (ICC, IFORT) Compilation Technology compilers provided alongside new compilers
  - ***CHOICE! Continuity!***
- ***BUT Offload with OpenMP supported only with new LLVM-based compilers - ICX, IFX***

Cross Compiler Binary Compatible and Linkable!

Intel® Compilers – Target & Packaging

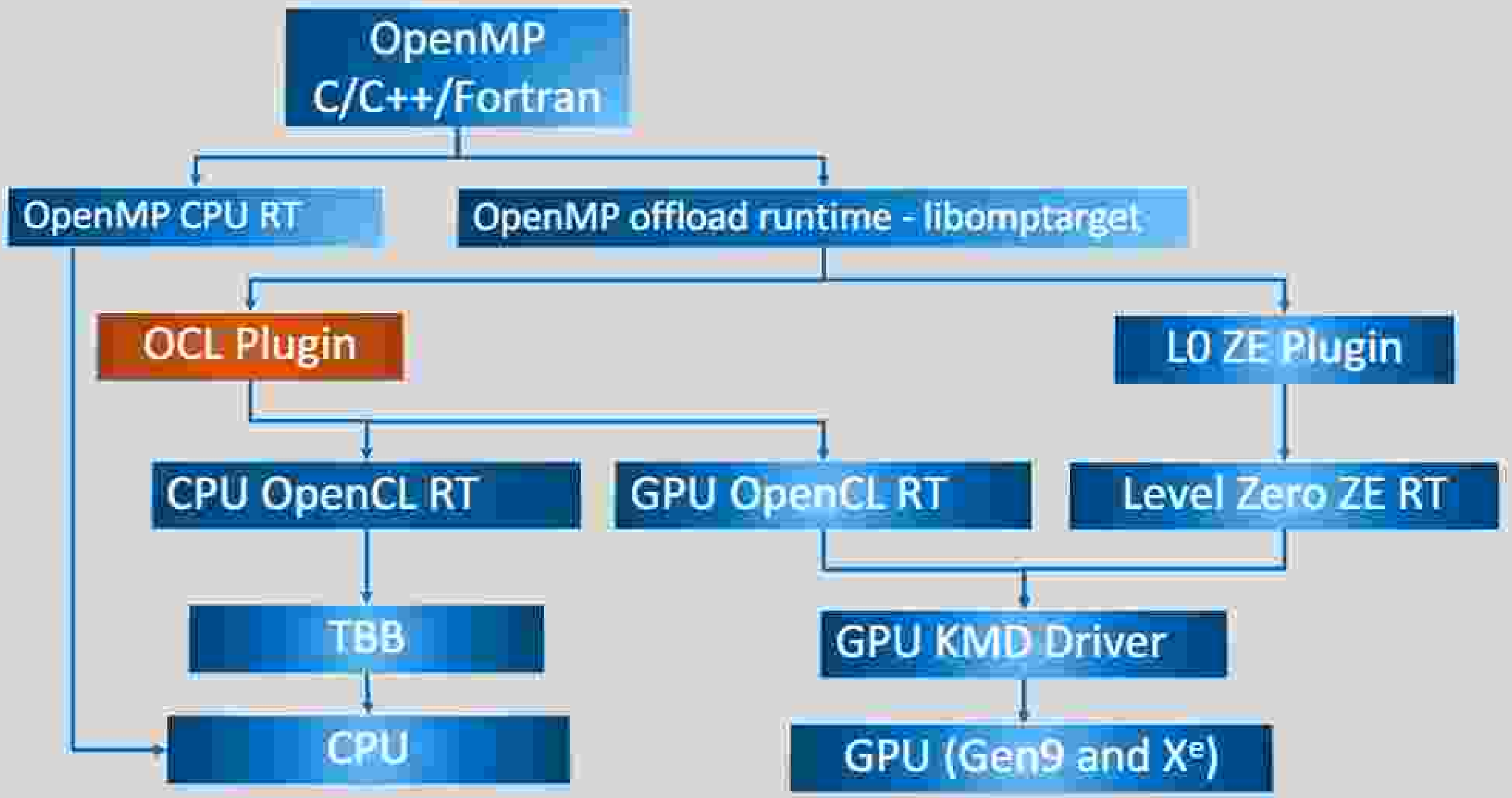
Intel Compiler	Driver	Target*	OpenMP CPU Support	OpenMP Offload Support
Intel® C++ Compiler Classic	***icc***	CPU	Yes	No
Intel® oneAPI DPC++/C++ Compiler	***dpcpp***	CPU, GPU, FPGA	Composable with constraints	Composable with constraints
	icx	CPU GPU	Yes	Yes
Intel® Fortran Compiler Classic	***ifort***	CPU	Yes	No
Intel® Fortran Compiler (Beta)	***ifx***	CPU, GPU	Yes	Yes

Cross Compiler Binary Compatible and Linkable!

Intel's Compiler and Software Stack

Supporting OpenMP Standard, Intel's OMP RT Implementation

High level architecture



Just-In-Time (JIT) and Ahead-of-Time (AOT) Compilation

JIT compilation

```
icpx -fiopenmp  -fopenmp-targets=spir64 source.cpp
ifx -fiopenmp  -fopenmp-targets=spir64  source.f90
```

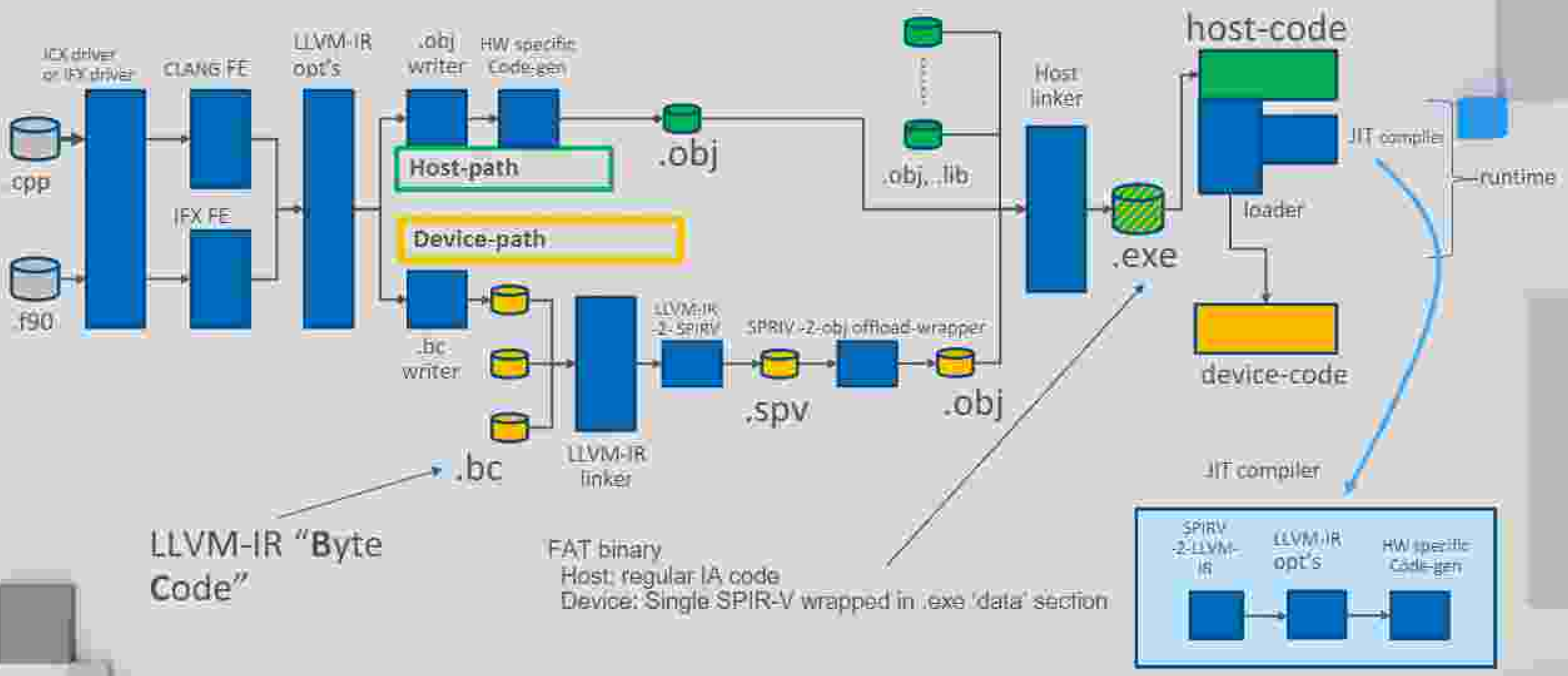
AOT compilation

```
icpx -fiopenmp -fopenmp-targets=spir64_gen -Xopenmp-target-backend "-device <dev>" src.cpp
ifx -fiopenmp -fopenmp-targets=spir64_gen -Xopenmp-target-backend "-device <dev>" src.f90
```

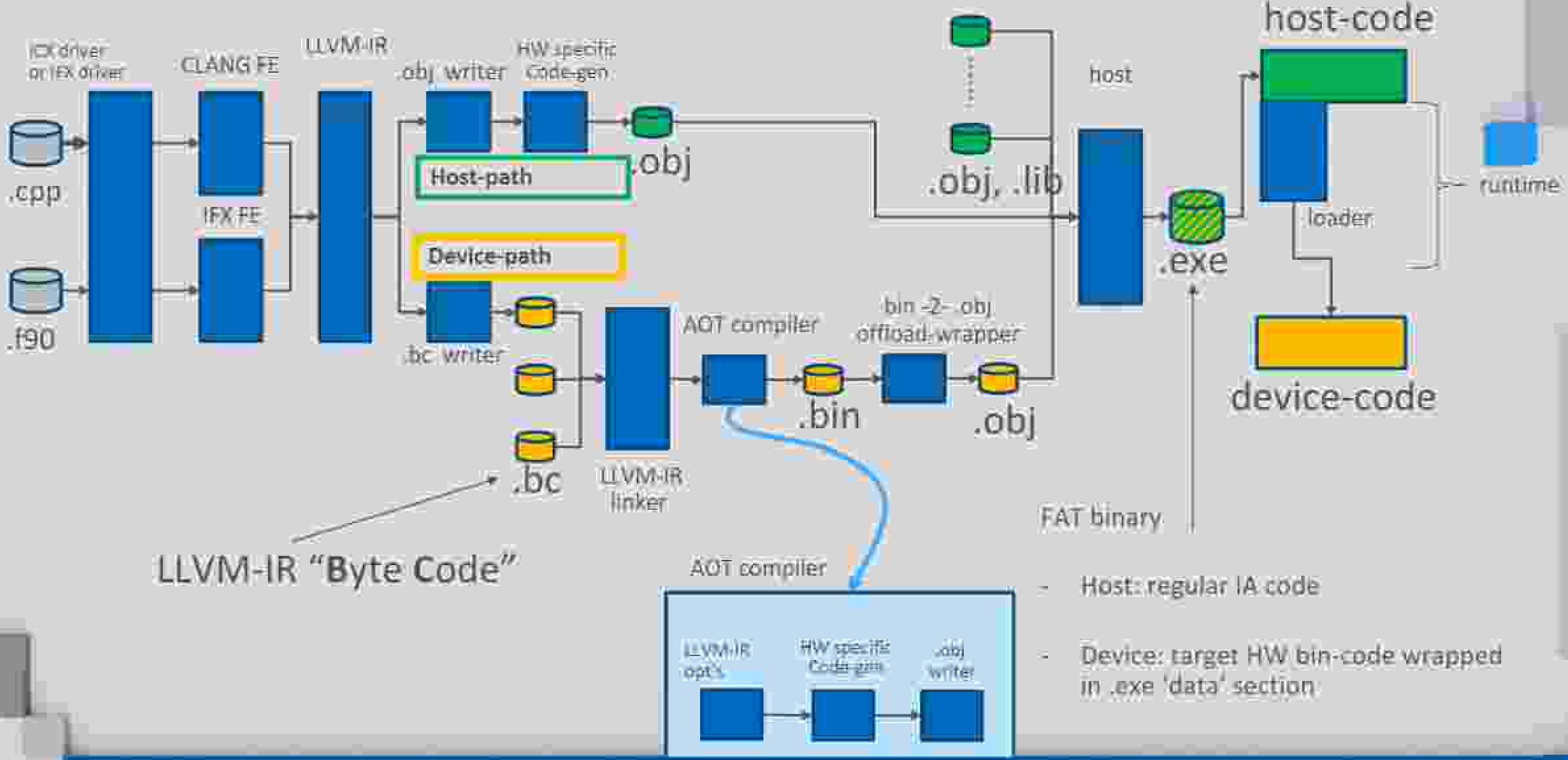
<dev> is your target, use 'ocloc compile --help' for list of targets

https://www.intel.com/content/www/us/en/develop/documentation/oneapi-dpcpp-cpp-compiler-dev-guide-and-reference/top/compilation/ahead-of-time-compilation.html

Just-In-Time (JIT) Compilation Flow



Ahead-Of-Time (AOT) Compilation Flow



intel.

OpenMP Offload with Intel® Compilers

Built-in Support for Intel® X^e

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 for Clang* OMP RT
- -qopenmp maps to –fiopenmp ***today***

-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> will be targets for CPUs and Intel GPUs - look at 2022 release documentation for more information

Example: Simple target offload

Transfer control and data from the host to the device

Syntax (C/C++)

#pragma omp target *[clause[[,] clause],…]*
structured-block

Clauses for TARGET

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

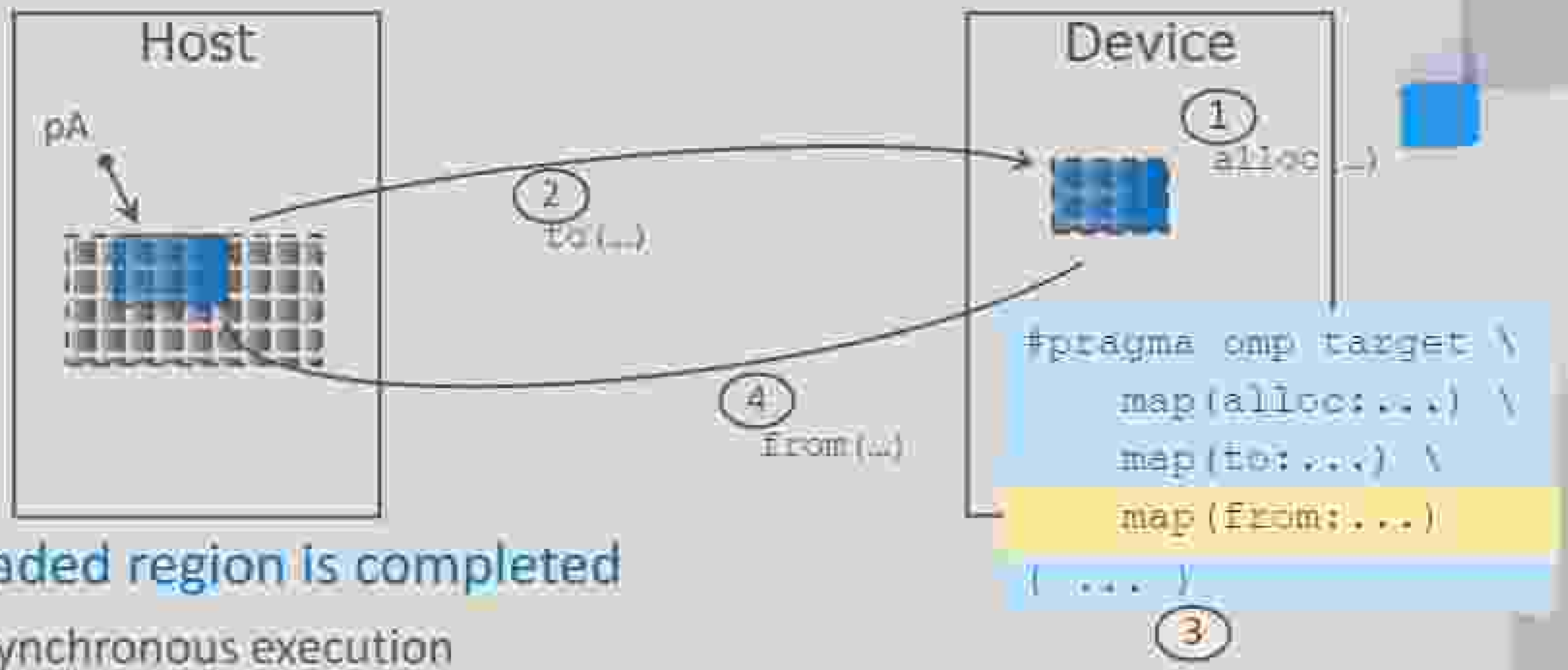
These OMP pragmas cause the loop to execute on a target device (i.e., GPU)

```
#define FLOPS_ARRAY_SIZE ((1024*1024*256)/4)/2  //map fa fb to 256MB
float fa[FLOPS_ARRAY_SIZE] __attribute__((aligned(64)));
float fb[FLOPS_ARRAY_SIZE] __attribute__((aligned(64)));
int main(int argc, char *argv[] )
{
    int i,j,k;
    double tstart, tstop, ttime;
    float a=1.001f;

    #pragma omp parallel for
    for(i=0; i<FLOPS_ARRAY_SIZE; i++)
    {
        fa[i] = (float)i + 1.1f;
        fb[i] = 1.2f;
    }

    #pragma omp target map(fa, fb, a)
    #pragma omp parallel for simd firstprivate(a)
    for(k=0; k<FLOPS_ARRAY_SIZE; k++)
              fa[k] = a * fa[k] + fb[k];
}
```

intel.

Offloading and Device Data Mapping

- Use *target* construct to
  - Transfer control from the host to the 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
- #pragma omp requires unified_shared_memory for managed memory allocation

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

- 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)

- Select Plugin/Driver
  LIBOMPTARGET_PLUGIN= cpu (only works for OpenCL)
  gpu

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

Essential Intel env Var LIBOMPTARGET_PROFILE

- LLVM OpenMP Runtime ENV vars are accepted. Example
- **export LIBOMPTARGET_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
```

intel.

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!

OpenMP Unified Share Memory

Making Data Mapping Easy

Unified Shared Memory

- Single address space for CPU and GPU
- Data migration among CPU and GPUs transparent to the application
  - Explicit mapping of data not required

Type	Location	Accessible From	Allocation Routine
Host	Host	Host or Device	omp_target_alloc_host(size, device_num)
Device	Device	Device	omp_target_alloc_device(size, device_num)
Shared	Host or Device	Host or Device	omp_target_alloc_shared(size, device_num)

- Use Shared or Host memory for implicit data movement to achieve ease of coding
- Use Device memory for explicit data movement to achieve maximum performance

Unified Shared Memory (Implicit) Example

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define SIZE 1024
#pragma omp requires unified_shared_memory
int main() {
  int deviceId = (omp_get_num_devices() > 0) ?
       omp_get_default_device() : omp_get_initial_device();
  int *a = (int *)omp_target_alloc_shared(SIZE * sizeof(int) , deviceId);
  int *b = (int *)omp_target_alloc_shared(SIZE * sizeof(int) , deviceId);
  for (int i = 0; i < SIZE; i++) {
    a[i] = i;     b[i] = SIZE - i;
  }
#pragma omp target teams distribute parallel for
  for (int i = 0; i < SIZE; i++) {
    a[i] += b[i];
  }

  for (int i = 0; i < SIZE; i++) {
    if (a[i] != SIZE) {
      printf("%s failed\n", __func__);
      return EXIT_FAILURE;
    }
  }
  omp_target_free(a, deviceId);
  omp_target_free(b, deviceId);
  printf("%s passed\n", __func__);
  return EXIT_SUCCESS;
}
```

USM support via managed memory allocator

Unified Shared Memory (Explicit) Example

```
int main() {
  int deviceId = (omp_get_num_devices() > 0) ? omp_get_default_device() : omp_get_initial_device();
  int *a = (int *)malloc(SIZE * sizeof(int));  int *b = (int *)malloc(SIZE * sizeof(int));
  for (int i = 0; i < SIZE; i++) {
    a[i] = i;     b[i] = SIZE - i;
  }
 int *a_dev = (int *)omp_target_alloc_device(SIZE * sizeof(int) , deviceId);
 int *b_dev = (int *)omp_target_alloc_device(SIZE * sizeof(int) , deviceId);
 int error=omp_target_memcpy(a_dev, a, SIZE*sizeof(int), 0, 0, deviceId, 0);
 error=omp_target_memcpy(b_dev, b, SIZE*sizeof(int), 0, 0, deviceId, 0);
 #pragma omp target teams distribute parallel for
  for (int i = 0; i < SIZE; i++) {
    a_dev[i] += b_dev[i];
  }
 error=omp_target_memcpy(a, a_dev, SIZE*sizeof(int), 0, 0, 0, deviceId);
 error=omp_target_memcpy(b, b_dev, SIZE*sizeof(int), 0, 0, 0, deviceId);

  for (int i = 0; i < SIZE; i++) {
    if (a[i] != SIZE) { printf("%s failed\n", __func__); return EXIT_FAILURE; }}
 omp_target_free(a_dev, deviceId);
 omp_target_free(b_dev, deviceId);
  free(a);  free(b);
  printf("%s passed\n", __func__);
  return EXIT_SUCCESS;
}
```

Explicit Data Movement from Host to Device

Explicit Data Movement from Device to Host

USM Example (Fortran) – IFX Feature Coming Soon!

```
program main
use omp_lib
integer, parameter :: N=16
integer :: i, dev
integer, allocatable :: x(:)

dev = omp_get_default_device()
!$omp allocate allocator(omp_target_shared_mem_alloc)
allocate(x(N))

do i=1,N
   x(i) = i
end do

!$omp target has_device_addr(x)
!$omp teams distribute parallel do
do i=1,N
   x(i) = x(i) * 2
end do
!$omp end target
…
deallocate(x)
…
end program main
```

`omp_target_host_mem_alloc` and `omp_target_device_mem_alloc` allocation types also available

USM support via managed memory allocator

What Have We Learned?

- Intel® oneAPI LLVM-based compilers support OpenMP offload
- Compiling for OpenMP Offload
- Using Environment Variables to Control the OpenMP Runtime
- Intel's Compiler OpenMP Software Stack, JIT and AOT
- Simplify Data Mapping with Unified Shared Memory
- OpenMP Features Support

Call to Action

- Intel® oneAPI HPC Toolkit
  https://www.intel.com/content/www/us/en/developer/tools/oneapi/hpc-toolkit-download.html
- Intel® oneAPI Base Toolkit
  https://www.intel.com/content/www/us/en/developer/tools/oneapi/base-toolkit-download.html

ICX and IFX OpenMP Features and Support

IFX OpenMP Features Support

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

ICX OpenMP Features Support

[https://www.intel.com/content/www/us/en/developer/articles/technical/openmp-features-and- extensions-supported-in-icx.html](https://www.intel.com/content/www/us/en/developer/articles/technical/openmp-features-and-extensions-supported-in-icx.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

Thank You for Attending!

intel®