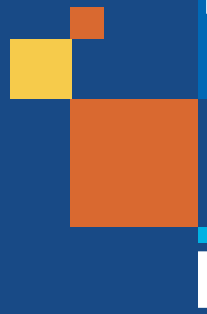
DPC++ and OpenMP* Compilers for Diverse Architectures

Xinmin Tian

Intel Architecture, Graphics and Software – Intel Corporation

November 13th, 2020



AGENDA

- Why oneAPI Toolkits?
- X^e Multi-Tile GPUs
- Software Big Picture
- Recap: OpenMP Subset for GPUs
- MiniWeather Workload using OpenMP Target
- A deep-dive of AI workload performance tuning
- Host and JIT Compilation Flow
- Roadmap and Summary

Why oneAPI Toolkits?

- Modern workloads are incredibly diverse, and so are architectures.
- No single architecture is best for every workload.
- Maximizing performance takes a mix of Scalar, Vector, Matrix and Spatial (SVMS) architectures deployed in CPU, GPU, FPGA and other future architectures.
- oneAPI Toolkits
  - deliver the tools we need to deploy modern applications and solutions across SVMS architectures.
  - unleash the power of achieving **P**roductivity, **P**erformance and **P**ortability.

Xe HP Multi-tile GPU

1-Tile
>10 FP32 TFLOPS

2-Tile
>20 FP32 TFLOPS

4-Tile
>40 FP32 TFLOPS

The Big Picture with HPC/AI frameworks and Applications

HPC/AI C/C++/DPC++ and Fortran Optimized Applications

Portability frameworks: Kokkos, RAJA, etc.

Direct Programming

Porting tools

Data Parallel C++	OpenMP C/C++	OpenMP FORTRAN	Future languages
Unified, LLVM-based backend			
Runtimes			

API-based Programming

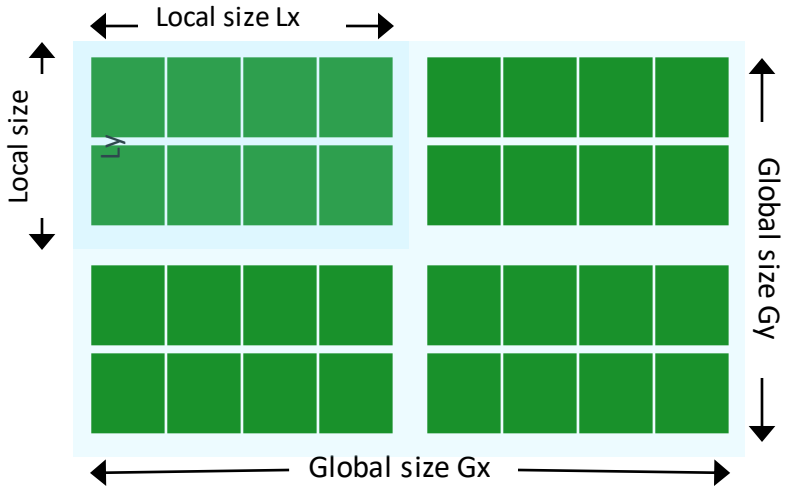
AI/Math	Parallel	Visual
oneDNN	oneTBB	oneVPL
oneMKL	oneDPL	OpenVINO
oneDAL	MPI	IPP
oneCCL		

Analysis tools:
Debuggers
Profilers
Advisors

DPC++ (SYCL) Language Components

Buffers	Unified Shared Memory	Device Management	Built-in Functions
Abstracts memory management and data movement	Pointer-based programming that enables desired level of control over data movement	Manage contexts; Discover and select devices	Rich library of common math operators and atomic operations
Scheduler	**Data Parallelism and Tasks**	**Low Level Interop**	**Data Types**
Graph execution of kernels controlled by dependences	Many ways to express task and data parallel computations	Interoperability with OpenCL objects	Scalar and vector data types, swizzles
		Error Handling	**Extensions**
		Detect and diagnose	Device-specific extensions to enable maximum performance

DPC++ (SYCL) Execution Model: Mapping Work-Items onto an nd_range



- range ←→item (parallel_for/parallel_for_work_item)
- nd_range ←→nd_item (parallel_for with ndrange)
  - Can get work-group id info
  - Barrier synchronization

ND-Range (2D)

Global size: {8,4} → 32 ***work-items***

Local size: {4,2} → 4 ***work-groups***

Work-group size: {2,2}

range(size_t x, size_t y, size_t z)

nd_range(range globalSize, range localSize)

Recap: OpenMP 4.5/5.1 Subset for GPU

- **Offload code to run on a target device**
  - omp target [clause[[,] clause],…]
    - structured-block
  - omp declare target
    - [function-definitions-or-declarations]
- **Map variables to a target device**
  - map ([map-type:] list) // map clause
    - map-type := alloc | tofrom | to | from
  - omp target [enter | exit] data [clause[[,] clause],…]
    - structured-block
  - omp target update [clause[[,] clause],…]
  - omp declare target
    - [variable-definitions-or-declarations]
- **Work-sharing for acceleration**
  - omp teams/single/master [clause[[,] clause],…]
  - omp distribute [for | do] [clause[[,] clause],…]
- for-loops
- **Parallel and simd code to run on GPU**
  - omp parallel for/do [simd] [clause[[,] clause],…]
  - omp simd
  - A set of composite and combined constructs [clause[[,] clause],…]
    - E.g. #pragma omp target teams distribuite parallel for simd
- **Synchronization**
  - omp atomic [clause[[,] clause],…]
    - map-type := alloc | tofrom | to | from
  - omp critical [clause[[,] clause],…]
    - structured-block

OpenMP Runtime API Support

Runtime support routines on CPU Host

```
EXTERN int omp_get_num_devices(void);
EXTERN int omp_get_initial_device(void);
EXTERN void *omp_target_alloc(size_t size, int device_num);
EXTERN void omp_target_free(void *device_ptr, int device_num);
EXTERN int omp_target_is_present(void *ptr, int device_num);
EXTERN int omp_target_memcpy(void *dst, void *src, size_t length, size_t
dst_offset, size_t src_offset, int dst_device, int src_device);
EXTERN int omp_target_memcpy_rect(void *dst, void *src, size_t
element_size,
  int num_dims, const size_t *volume, const size_t *dst_offsets,
  const size_t *src_offsets, const size_t *dst_dimensions,
  const size_t *src_dimensions, int dst_device, int src_device);
EXTERN int omp_target_associate_ptr(void *host_ptr, void *device_ptr,
size_t size, size_t device_offset, int device_num);
EXTERN int omp_target_disassociate_ptr(void *host_ptr, int device_num);
EXTERN int omp_is_initial_device(void);
EXTERN int omp_get_initial_device(void);
EXTERN void kmp_global_barrier_init(void);       // Intel externsion
EXTERN void kmp_global_barrier(void);            // Intel externsion
EXTERN void omp_set_default_device(int dev_num )
EXTERN int omp_get_default_device(void)
```

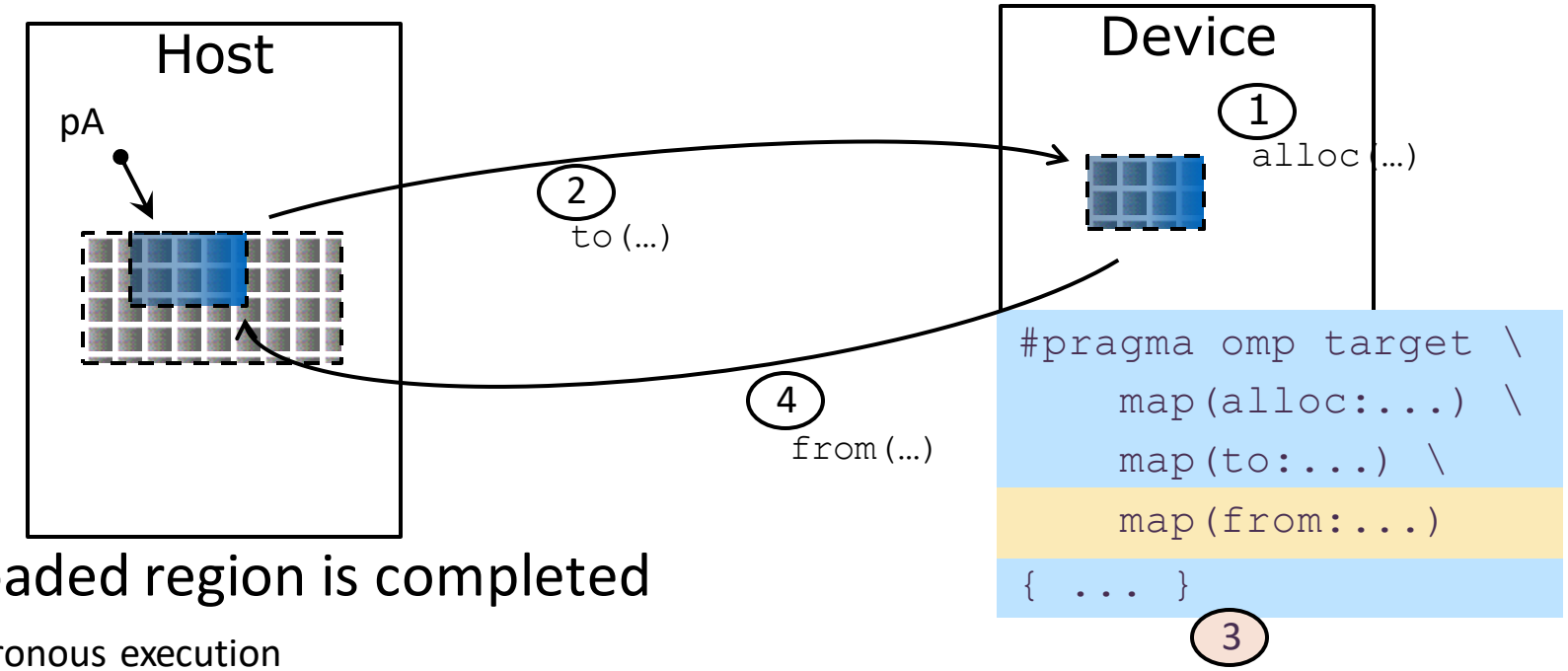
Device Runtime Routines for GPU

```
EXTERN int omp_get_team_num(void);
EXTERN int omp_get_num_teams(void);
EXTERN int omp_get_team_size(int);
EXTERN int omp_get_thread_num(void);
EXTERN int omp_get_num_threads(void);
EXTERN int omp_in_parallel(void);
EXTERN int omp_get_max_threads(void);
EXTERN int omp_get_device_num(void);
EXTERN int omp_get_num_devices(void);
```

- omp_target_host_alloc(…)
- omp_target_shared_alloc(…)
- omp_target_device_alloc(…)

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

OpenMP and DPC++ Code Examples

OpenMP

```
int8_t memA[M][K] = {0}; //or memA[M][strideA]
int8_t memB[K][N] = {0}; //or memB[K][strideB]
int    memC[M][N] = {0}; //or memC[M][strideC]

#ifdef OFFLOAD
#pragma omp target teams map(memC[0: M*N], sub_c[0:sM*sK])
            map(to: memA[0:M*K], memB[0:K*N], sub_a[0:sM*sK],
sub_b[0:sK*sN])
#endif
{ submatrix<int8_t, sM, sK> sub_a;
  submatrix<int8_t, sK, sN> sub_b;
  submatrix<int, sM, sN> sub_c;
#ifdef OFFLOAD
  #pragma omp distribute parallel for collapse(2)
#else
  #pragma omp parallel for collapse(2)
#endif
  for(i=0; i<M; i+=sM) {     //assuming M % sM = 0
    for(j=0; j<N; j+=sN) {   //assuming N % sN = 0
      submatrix_load(sub_c, &memC, i, j, strideC, ROW_MAJOR);

      for (int k = 0; k < K; k += sK) { //assuming K % sK = 0
        submatrix_load(sub_a, &memA, i, k, strideA, ROW_MAJOR);
        submatrix_load(sub_b, &memB, k, j, strideB, COLUMN_MAJOR);
        sub_c = submatrix_mad(sub_a, sub_b, sub_c);
      }
      submatrix_store(sub_c, &memC, i, j, strideC, ROW_MAJOR);
    }
  }
}
```

DPC++

```
queue Q;
range<2> G = {M, N*32};
range<2> L = {1, SG_SIZE};   // ATS: SG_SIZE = 8,  PVC: SG_SIZE = 16;

int8_t *memA = malloc_shared<int8_t>(M*K, Q);
int8_t *memB = malloc_shared<int8_t>(K*N, Q);
int    *memC = malloc_shared<int>(M*N, Q);

Q.parallel_for(nd_range<2>{G, L}, [=](nd_item<2> it)
  [[cl::intel_reqd_sub_group_size(w)]] {

   const auto g_idx = it.get_global_id(0);
   const auto g_idy = it.get_global_id(1);
   const auto sg_startx = g_idx - it.get_local_id(0);
   const auto sg_starty = g_idy - it.get_local_id(1);

   submatrix<int8_t, sM, sK> sub_a; // 2,32
   submatrix<int8_t, sK, sN> sub_b; // 32,16
   submatrix<int, sM, sN> sub_c;    // 2,16

   submatrix_load(sub_c, memC, sg_startx * sM, sg_starty, N, ROW_MAJOR);

   for (int k = 0; k < K; k += sk) {
     submatrix_load(sub_a, memA, sg_startx * sM, k, K, ROW_MAJOR);
     submatrix_load(sub_b, memB, k, sg_starty, N, COLUMN_MAJOR);
     sub_c = submatrix_mad(sub_a, sub_b, sub_c);
   }
   submatrix_store(sub_c, memC, sg_startx * sM, sg_starty, N, ROW_MAJOR);
}).wait();
```

Mini-Weather Code using OpenMP Target

https://github.com/mrnorman/miniWeather

miniWeather is a mini-app developed by Dr. Matthew Norman (OLCF). The code mimics the basic dynamics seen in atmospheric weather and climate:

- Solves 2D Euler equations of compressible flow
- 4th-order central finite-volume scheme
- 3rd-order Runge-Kutta time integration
- 4th-order hyperviscosity
- Enabled for multiple parallel programming models: **MPI**, **OpenMP**, **OpenMP offload**, **OpenACC**, **C++** (w/ **CUDA**, **HIP** in backend)

Example of mapping variables between the host and target device

miniWeather_mpi_openmp45.cpp

```
#pragma omp target data
map(to:state_tmp[:(nz+2*hs)*(nx+2*hs)*NUM_VARS],hy_dens_cell[:nz+2*hs],hy_dens_theta_cell[:nz+2*hs],hy_dens_int[:nz+1],hy_dens_theta_int[:nz+1],hy_pressure_int[:nz+1]) \

map(alloc:flux[:(nz+1)*(nx+1)*NUM_VARS],tend[:nz*nx*NUM_VARS],sendbuf_l[:hs*nz*NUM_VARS],sendbuf_r[:hs*nz*NUM_VARS],recvbuf_l[:hs*nz*NUM_VARS],recvbuf_r[:hs*nz*NUM_VARS]) \

map(tofrom:state[:(nz+2*hs)*(nx+2*hs)*NUM_VARS])
```

Example of the target teams construct

miniWeather_mpi_openmp45.cpp

```
#pragma omp target teams distribute parallel for simd collapse(3)
for (ll=0; ll<NUM_VARS; ll++) {
  for (k=0; k<nz; k++) {
    for (i=0; i<nx; i++) {
      inds = ll*(nz+2*hs)*(nx+2*hs) + (k+hs)*(nx+2*hs) + i+hs;
      indt = ll*nz*nx + k*nx + i;
      state_out[inds] = state_init[inds] + dt * tend[indt];
    }
  }
}
```

MiniWeather: Build and run with Intel oneAPI

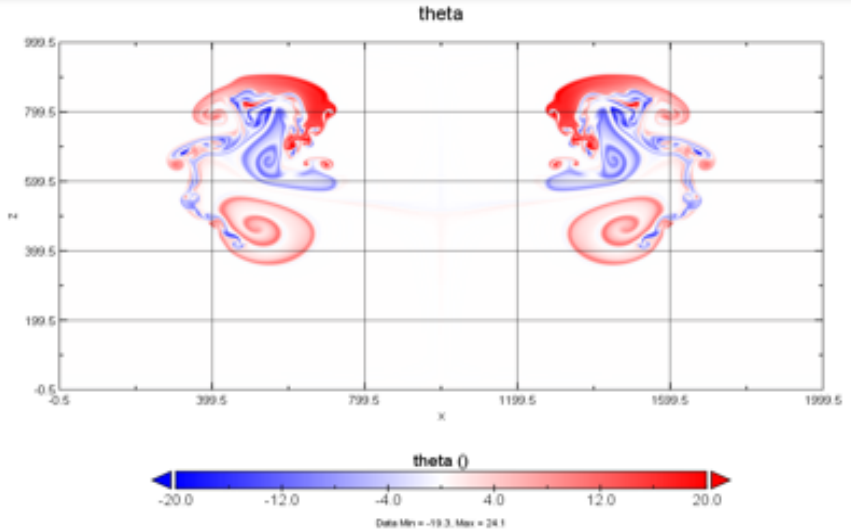
Compile miniWeather with Intel oneAPI (example for cmake)

```
#!/bin/bash
git clone git@github.com:mrnorman/miniWeather.git
cd miniWeather/c/build

source /opt/intel/oneapi/setvars.sh
source /opt/intel/oneapi/Gen9_setup_beta09_20.04LTS.sh

./cmake_clean.sh

export I_MPI_CC=icx
export I_MPI_CXX=icpx
export I_MPI_F90=ifx
cmake -DCMAKE_CXX_COMPILER=mpiicpc \
      -DPNETCDF_PATH=/path/to/netcdf/lib \
      -DOPENMP_FLAGS="-fiopenmp -fopenmp-targets=spir64" \
      -DOPENMP45_FLAGS="-fiopenmp -fopenmp-targets=spir64" \
      -DCXXFLAGS="-I/path/to/netcdf/include -L/path/to-
netcdf/lib" \
      -DDATA_SPEC=DATA_SPEC_COLLISION \
      -DNX=2000 \
      -DNZ=1000 \
      -DSIM_TIME=700 \
      ..
make
export OMP_TARGET_OFFLOAD="MANDATORY"
./openmp45
```

Example of running miniWeather app on Intel Gen9 Graphics

```
$ source /opt/intel/oneapi/setvars.sh
$ clinfo -l
Platform #0: Intel(R) OpenCL
 `-- Device #0: Intel(R) Core(TM) i7-6770HQ CPU @ 2.60GHz
Platform #1: Intel(R) OpenCL HD Graphics
 `-- Device #0: Intel(R) Gen9 HD Graphics NEO
Platform #2: Intel(R) OpenCL HD Graphics
 `-- Device #0: Intel(R) Gen9 HD Graphics NEO
./openmp45
nx_glob, nz_glob: 2000 1000
dx,dz: 10.000000 10.000000
dt: 0.033333
*** OUTPUT ***
Elapsed Time: 0.000000 / 700.000000
Elapsed Time: 0.033333 / 700.000000
…
CPU Time: 22.8183 sec
d_mass: 1.562607e-14
d_te:   -4.914851e-06
```

Output data passed correctness check



Performance Impact of Maintaining Cross-Kernel Thread-Data Affinity

- Rodinia-CFD is very sensitive to cross-kernel thread-data affinity

Kx		Ky	
Core-m	Core-n	Core-m	Core-n
Thread-m	Thread-n	Thread-m	Thread-n
Task-m	Task-n	Task-n	Task-m
Dataset-m	Dataset-n	Dataset-m	Dataset-n

- Each kernel to CPU HW threads/cores mapping may be differently due to kernel grain-size and work-stealing
  - OpenMP uses static scheduling for Kx and Ky, thus Thread-Data affinity is well preserved.
- Also reduce buffer access and data transfer cost.

Rodinia-CFD Tuning via Kernel Fusion

```
cgh.parallel_for <init1>(( … ) { … });
for (int i = 0; I < iterations; i++) {
    cgh.parallel_for<copy>(( … ) { … });
    cgh.parallel_for<comp_step_fac>(( … ) { … });
    for (int j=0; j<RK; j++) {
        cgh.parallel_for<comp_flux>(( … ) { … });
        cgh.parallel_for<comp_timestep>(( … ) { … });
    }
}
```

Performance improvement

~30% with kernel fusion

```
cgh.parallel_for <init1>((…) { … });
for (int i = 0; i < iterations; i++) {
    cgh.parallel_for<copy_and_comp_step_fac>((…) {
      copy code … … …
      step fac code … … …
    });
    for (int j = 0; j<RK; j++) {
        cgh.parallel_for<comp_flux_and_timestep>((…) {
           flux code … … …
           comp_timestep code … … …
        });
    }
}
```

Kernel Fusion + Loop Sinking

```
cgh.parallel_for <init1>((…) { … });
for (int i = 0; I < iterations; i++) {
    cgh.parallel_for<copy>((…) { …});
    cgh.parallel_for<comp_step_fac>((…) { …});
    for (int j=0; j<RK; j++) {
        cgh.parallel_for<comp_flux>((…) { …});
        cgh.parallel_for<comp_timestep>((…) { …});
    }
}
```

Performance gain: ~36%

with loop sinking

Performance gain ~54% with

kernel fusion + loop sinking

```
cgh.parallel_for <init1>((…) { … });
for (int i = 0; i < iterations; i++) {
    cgh.parallel_for<copy_and_comp_step_fac>((…) {
      copy code … … …
      step fac code … … …
    });
    cgh.parallel_for<comp_flux_and_timestep>((…) {
      for (int j = 0; j<RK; j++) {
        flux code … … …
        comp_timestep code … … …
      }
    });
}
```

More Aggressive Kernel Fusion and Loop Sinking

```
cgh.parallel_for<comp_flux_and_timestep>((…) {
    init code …
    for (int i = 0; I < iterations; i++) {
      copy code … … …
      step fac code … … …
      for (int j = 0; j<RK; j++) {
        flux code … … …
        comp_timestep code … … …
      }
    }
});
```

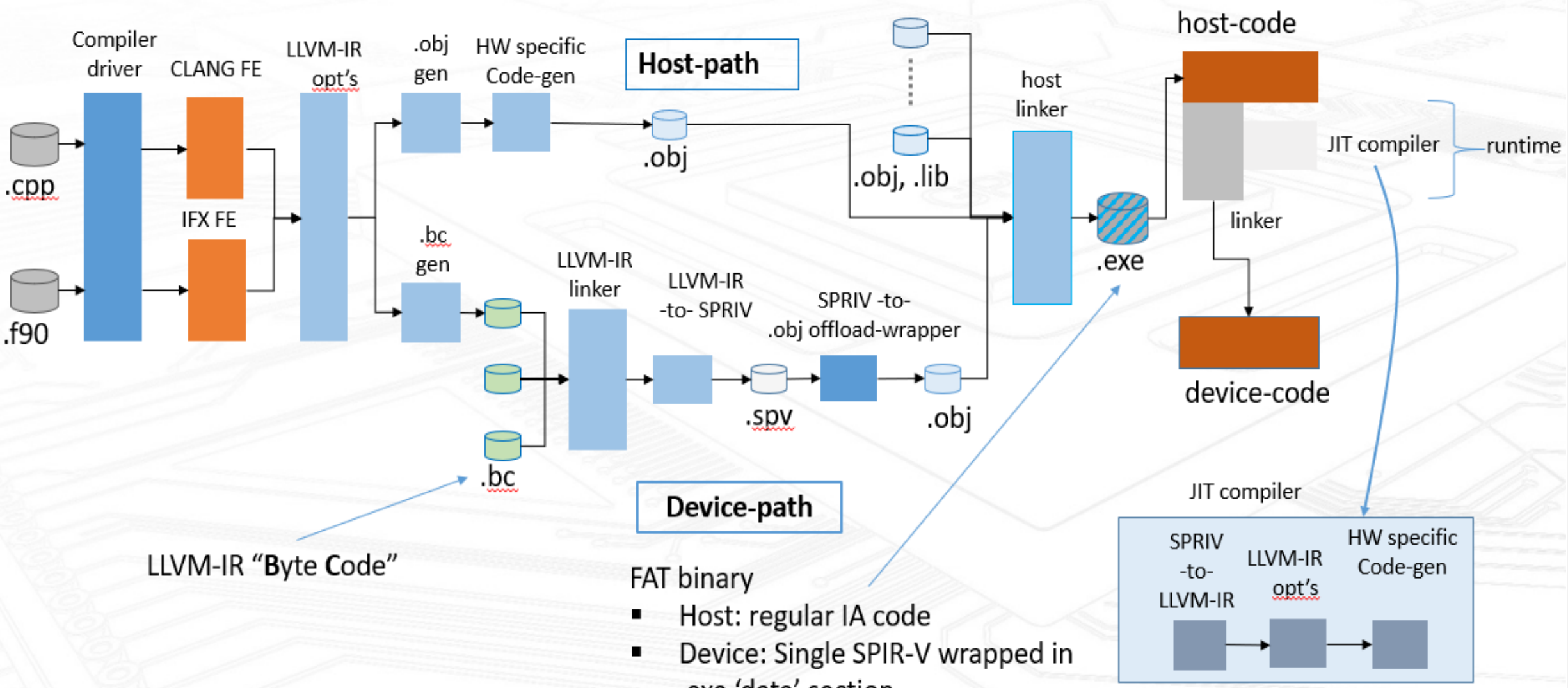
No Transistor Left Behind!

~8x performance improvement on CPU

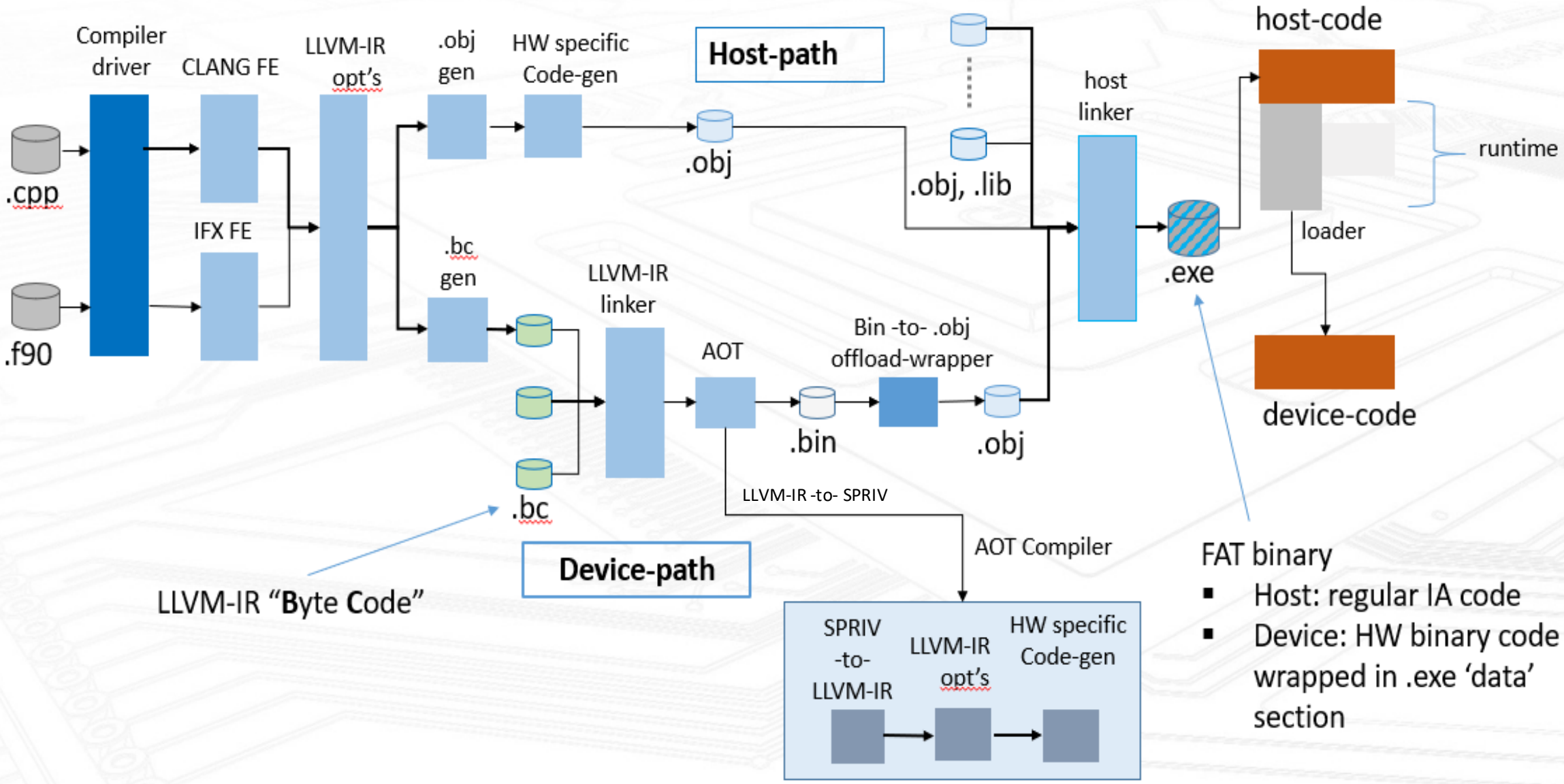
~2x performance improvement on iGPU

~2x performance improvement on dGPU

Just-In-Time (JIT) Compilation Flow



Ahead of Time (AOT) Compilation Flow



OpenMP 4.5/5.0/5.1 and Offloading Roadmap (2020 - Future)

Q3'2020 oneAPI Toolkits Beta08 and Beta09

- Fortran 90 and 95 features
- OpenMP 4.5 and Subset of OpenMP 5.0
- OpenMP and DPC++ composability at USM memory object level
- USM Memory allocation APIs (host, device and shared)
- Device RTL for complex type
- Offloading opt-report support

Q4'2020 oneAPI Toolkits Beta10 and oneAPI Gold

- Early access of LLVM-based Fortran compiler
- Early access of some of OpenMP 5.1 features (e.g. initial declare variant, loop construct)
- X^e Level-Zero plugin in Beta10
- X^e GPU Multi-GPU support
- Performance tuning for HPC/AI applications on CPUs and X^e GPUs

Future Releases

- Add more C/C++ and Fortran language features for CPUs and X^e GPUs.
- Work with community to add more OpenMP 5.0/5.1 features such as metadirective support.
- Add multi-tile support for X^e GPUs
- Add more use-friendly optimization information for opt-report
- Performance tuning for HPC/AI applications on CPUs and X^e GPUs

Get Started with oneAPI Today!

Resources

Intel® DevCloud

Start in the Cloud - No Download, No Installation, No Setup – Sign up here - software.intel.com/devcloud/oneAPI

oneAPI Toolkits

Develop On-Prem – Download & Develop - Get them here - software.intel.com/oneAPI

Community Collaboration

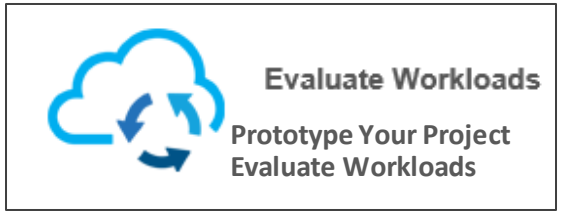
Community – Working with community closely for OpenMP 5.1 support.

OpenMP Specification

Stay with Standard Body - Cross-industry, open, standards-based unified programming model across architectures – Learn more here – **openmp.org**

Learn about Intel® oneAPI Toolkits

Build XPU Applications Intel® Xeon, FPGA, Integrated Graphics GPUs & Xᵉ GPU, DG1**



Learn Data Parallel C++

Interoperable with MPI, OpenMP, Fortran, C/++

Gaining Momentum!
oneAPI 1.0 Now Available!

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.
- Copyright © 2019, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and OpenVINO are trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries.

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