



ONEAPI ESSENTIALS WORKSHOP

Praveen Kundurthy, Rakshith Krishnappa



AGENDA

- **Agenda:**
 - a) Introduction & Overview to oneAPI
 - b) Introduction to the Intel® DevCloud
 - c) Introduction to Jupyter notebooks used for training
 - d) Introduction to Data Parallel C++
 - e) Overview of SYCL Fundamental classes
 - f) Device offloading and Custom Device Selector
 - g) Host accessor and Synchronization
- **Hands On:** Introduction to DPC++ - Simple
- **Hands On:** Vector-Add Coding Exercise
- **Hands On:** Complex Multiplication

LEARNING OBJECTIVES

Explain how oneAPI can solve the **challenges of programming** in a heterogeneous world

Use **oneAPI solutions** to enable your workflows

Experiment with **oneAPI tools and libraries** on the Intel® DevCloud

Understand the Data Parallel C++ (DPC++) language and programming model

Explain the SYCL fundamental classes

Use **device selection** to **offload kernel workloads**

Understand various ways to **synchronize** data between host and device

Build a sample DPC++ application through hands-on lab exercises

PROGRAMMING CHALLENGES FOR MULTIPLE ARCHITECTURES

Growth in specialized workloads

No common programming language or APIs

Inconsistent tool support across platforms

Each platform requires unique software investment

Diverse set of data-centric hardware required

Application Workloads Need Diverse Hardware



SCALAR



VECTOR



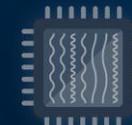
MATRIX



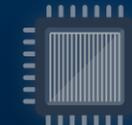
SPATIAL

Middleware / Frameworks

Language & Libraries

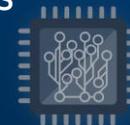


CPU

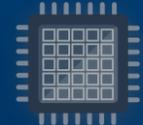


GPU

XPUs



FPGA



OTHER ACCEL.

INTRODUCING ONEAPI

Unified programming model to simplify development across diverse architectures

Unified and simplified language and libraries for expressing parallelism

Uncompromised native high-level language performance

Based on industry standards and open specifications

Interoperable with existing HPC programming models

Application Workloads Need Diverse Hardware



SCALAR



VECTOR



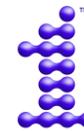
MATRIX



SPATIAL

Middleware / Frameworks

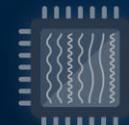
Industry Initiative



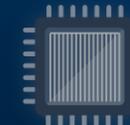
oneAPI

Intel Product

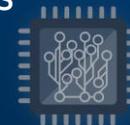
XPUs



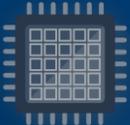
CPU



GPU



FPGA



OTHER ACCEL.

ONEAPI INDUSTRY INITIATIVE

ALTERNATIVE TO SINGLE-VENDOR SOLUTION

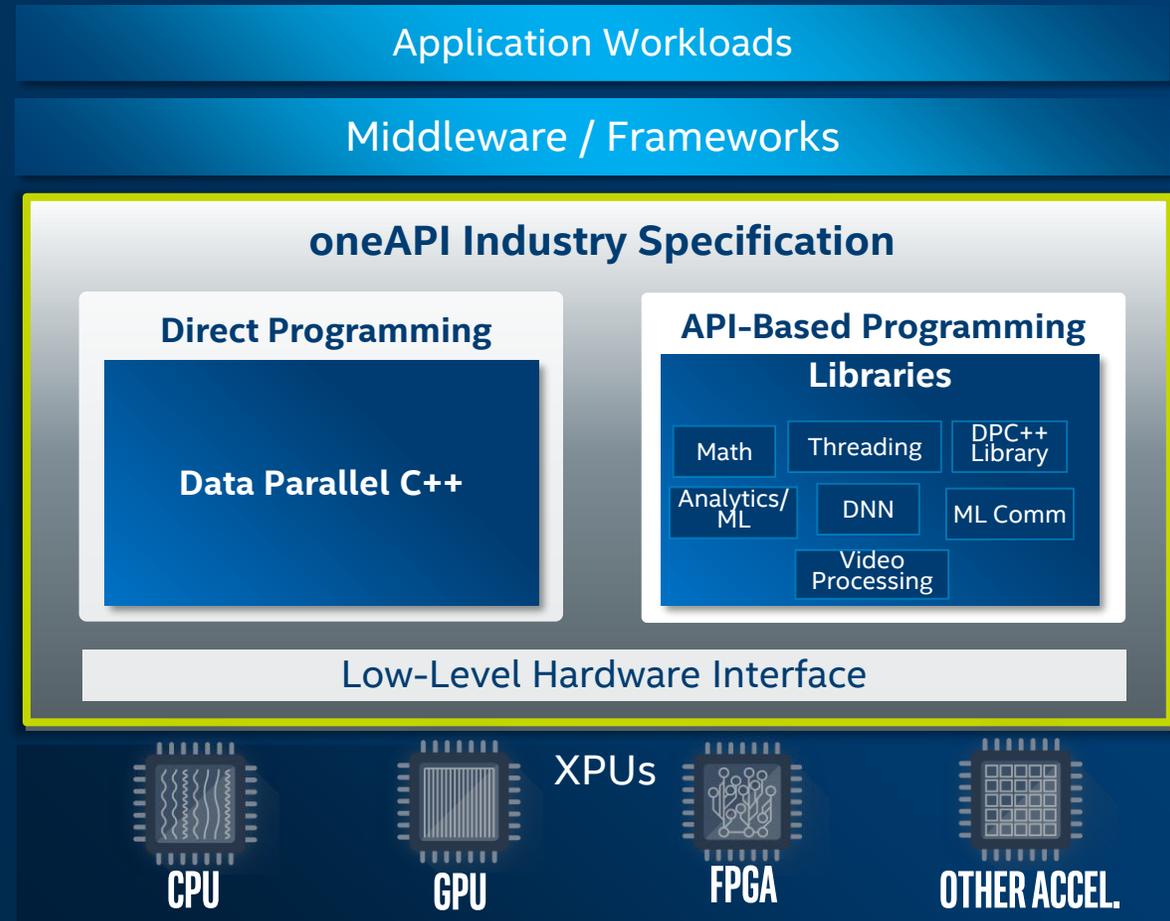
A standards based cross-architecture language, DPC++, based on C++ and SYCL

Powerful APIs designed for acceleration of key domain-specific functions

Low-level hardware interface to provide a hardware abstraction layer to vendors

Open standard to promote community and industry support

Enables code reuse across architectures and vendors



Visit oneapi.com for more details

Some capabilities may differ per architecture and custom-tuning will still be required.

[Optimization Notice](#)

Copyright © 2019, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

INTEL® ONEAPI TOOLKITS^(BETA)

TOOLKITS TAILORED TO YOUR NEEDS: [NATIVE CODE](#) | [DATA SCIENTISTS & AI](#) | [SYSTEMS](#)

Native Code Developers, start with the Intel® oneAPI Base Toolkit.



Intel® oneAPI Base Toolkit

A core set of high-performance tools for building Data Parallel C++ applications and oneAPI library based applications

[Learn More](#)

Add-on Domain-specific Toolkits for Specialized Workloads



Intel® oneAPI HPC Toolkit

Deliver fast C++, Fortran, & OpenMP* applications that scale

[Learn More](#)



Intel® oneAPI IoT Toolkit

Building high-performing, efficient, reliable solutions that run at the network's edge

[Learn More](#)



Intel® oneAPI DL Framework Developer Toolkit

Build deep learning frameworks or customize existing ones so applications run faster

[Learn More](#)



Intel® oneAPI Rendering Toolkit

Create high-performance, high-fidelity visualization applications

[Learn More](#)

Toolkits Powered by oneAPI:

Data Scientists & AI Toolkits

Systems Toolkit

Intel® AI Analytics Toolkit

Accelerate E2E machine learning & data science pipelines with optimized DL frameworks & high-performing Python libraries.

[Learn More](#)

Intel® Distribution of OpenVINO™ Toolkit

Deploy high performance inference & applications from edge to cloud (production-level tool)

[Learn More](#)

Intel® System Bring-Up Toolkit

Debug & tune systems for power & performance

[Learn More](#)

INTEL® ONEAPI BASE TOOLKIT (BETA)

Core set of frequently used tools and libraries for developing high-performance applications across diverse architectures—CPU, GPU, FPGA.

Who Uses It?

A broad range of developers across industries

Add-on toolkit users since this is the base for all toolkits

Top Features/Benefits

Data Parallel C++ compiler, library, and analysis tools

DPC++ Compatibility tool helps migrate existing code written in CUDA*

Python distribution includes accelerated scikit-learn, NumPy, SciPy libraries

Optimized performance libraries for threading, math, data analytics, deep learning, and video/image/signal processing

Intel® oneAPI Base Toolkit

DIRECT PROGRAMMING

Intel® oneAPI DPC++ Compiler

Intel® DPC++ Compatibility Tool

Intel® Distribution for Python*

Intel® FPGA Add-on for oneAPI Base Toolkit

API-BASED PROGRAMMING

Intel® oneAPI DPC++ Library

Intel® oneAPI Math Kernel Library

Intel® oneAPI Data Analytics Library

Intel® oneAPI Threading Building Blocks

Intel® oneAPI Video Processing Library

Intel® oneAPI Collective Comms. Library

Intel® oneAPI Deep Neural Network Library

Intel® Integrated Performance Primitives

ANALYSIS & DEBUG TOOLS

Intel® VTune™ Profiler

Intel® Advisor

GDB*

Optimization Notice

Copyright © 2019, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



INTEL® ONEAPI HPC TOOLKIT (BETA)

A toolkit that makes it easier to build, analyze, optimize & scale HPC applications for Intel® Xeon® Scalable, Intel® Core™ processors & Intel® Accelerators.

Who Uses It?

C/C++, Fortran, OpenMP & MPI application developers

Top Features/Benefits

Optimized compilers & performance libraries for Intel® architectures

Powerful analysis tools to identify optimization opportunities for threading, memory & offloading

Standards-driven to scale forward & preserve development investment

Intel oneAPI Tools for HPC

DIRECT PROGRAMMING

Intel® C++ Compiler with OpenMP*

Intel® Fortran Compiler with OpenMP*

Intel® oneAPI DPC++ Compiler

Intel® DPC++ Compatibility Tool

Intel® Distribution for Python*

Intel® FPGA Add-on for oneAPI Base Toolkit

API-BASED PROGRAMMING

Intel® MPI Library

Intel® oneAPI DPC++ Library

Intel® oneAPI Math Kernel Library

Intel® oneAPI Data Analytics Library

Intel® oneAPI Threading Building Blocks

Intel® oneAPI Video Processing Library

Intel® oneAPI Collective Communications Library

Intel® oneAPI Deep Neural Network Library

Intel® Integrated Performance Primitives

ANALYSIS TOOLS

Intel® Inspector

Intel® Trace Analyzer & Collector

Intel® Cluster Checker

Intel® VTune™ Profiler

Intel® Advisor

GDB*

■ Intel® oneAPI HPC Toolkit +

■ Intel® oneAPI Base Toolkit

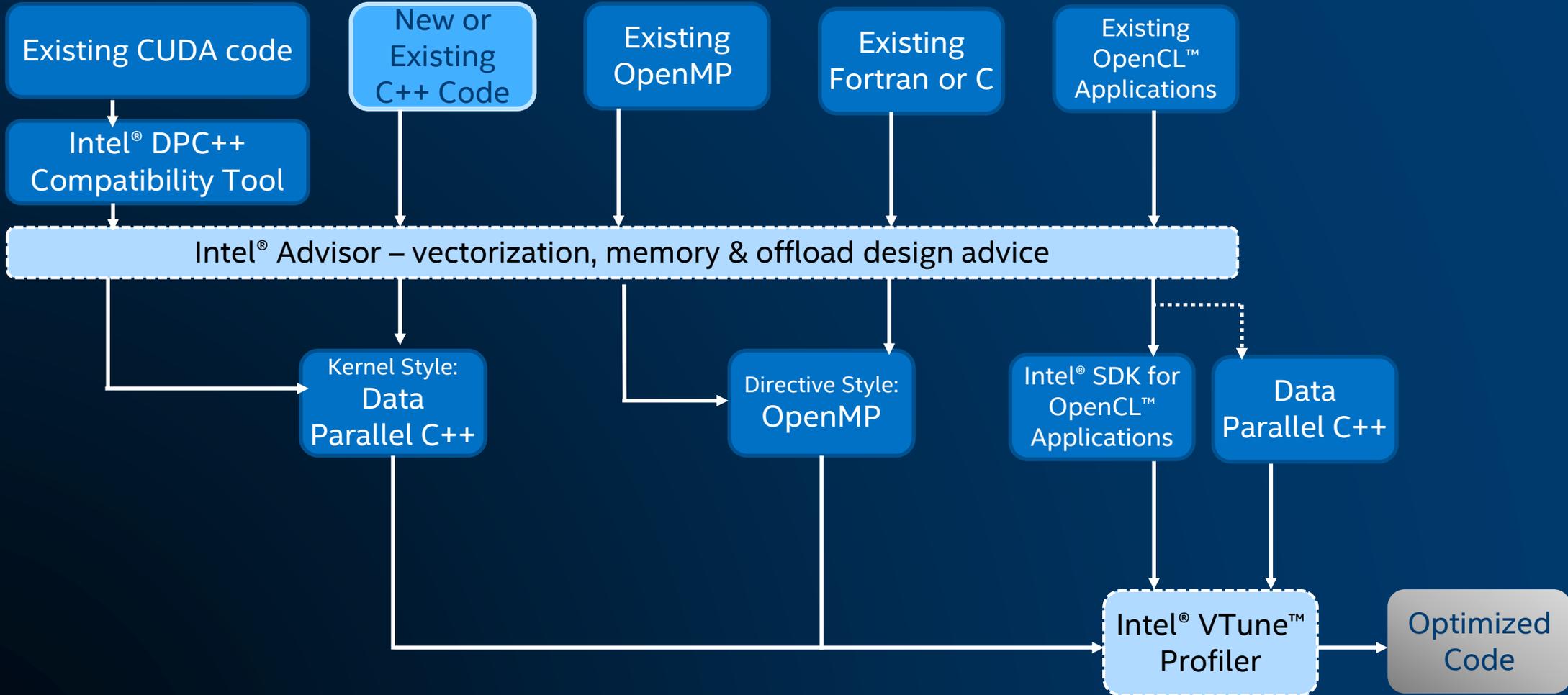
Optimization Notice

Copyright © 2019, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

HPC ONEAPI SINGLE NODE WORKFLOW

I WANT TO ACCELERATE USING DIRECT PROGRAMMING ON A GPU...



Introduce Data Parallel C++, the code structure,
and key concepts to get you writing code quickly!

Get functional quickly. Then analyze and tune.

DATA PARALLEL C++

STANDARDS-BASED, CROSS-ARCHITECTURE LANGUAGE

Parallelism, productivity and performance for CPUs and Accelerators

Allows code reuse across hardware targets, while permitting custom tuning for a specific accelerator

Open, cross-industry alternative to single architecture proprietary language

Based on ISO C++ and Khronos SYCL

Delivers C++ productivity benefits, using common and familiar C and C++ constructs
Incorporates SYCL from the Khronos Group to support data parallelism and heterogeneous programming

Community Project to drive language enhancements

Extensions to simplify data parallel programming

Open and cooperative development for continued evolution

Direct Programming: Data Parallel C++

Community Extensions

Khronos SYCL

ISO C++

The open source and Intel beta DPC++ compiler currently supports hardware including Intel CPUs, GPUs, and FPGAs.

Codeplay announced a [DPC++ compiler that targets Nvidia GPUs](#).

[Optimization Notice](#)

Copyright © 2019, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



WHAT IS DATA PARALLEL C++?

Data Parallel C++

= C++ **and** SYCL* standard **and** extensions

Based on modern C++

- C++ productivity benefits and familiar constructs

Standards-based, cross-architecture

- Incorporates the SYCL standard for data parallelism and heterogeneous programming

DPC++ EXTENDS SYCL 1.2.1

Enhance **Productivity**

- Simple things should be simple to express
- Reduce verbosity and programmer burden

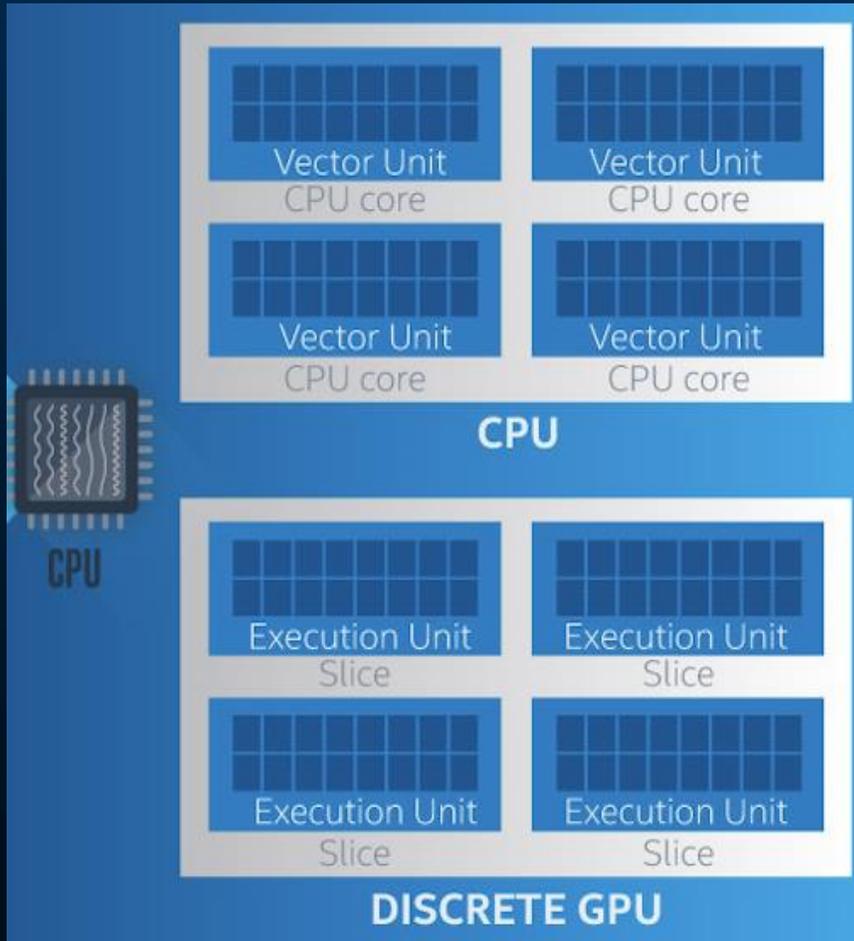
Enhance **Performance**

- Give programmers control over program execution
- Enable hardware-specific features

DPC++: Fast-moving open collaboration feeding into the SYCL* standard

- Open source implementation with goal of upstream LLVM
- DPC++ extensions aim to become core SYCL*, or Khronos* extensions

REQUIREMENTS FOR HETEROGENEOUS COMPUTING



Heterogeneous System

1. What are the entities in the compute system?

Platform Model

Host, Device

2. How are control and execution handled?

Execution Model

Queue, Accessor

3. How is data structured and communicated?

Memory Model

Buffer, Unified Shared Memory

4. What is executed on the device(s)?

Kernel Model

ND-range, work-item

A COMPLETE DPC++ PROGRAM

Single source

- Host code and heterogeneous accelerator kernels can be mixed in same source files

Familiar C++

- Library constructs add functionality, such as:

Construct	Purpose
queue	Work targeting
malloc_shared	Data management
parallel_for	Parallelism

Host code
Accelerator device code

Host code

```
#include <CL/sycl.hpp>
constexpr int N=16;
using namespace sycl;
int main() {
    queue q;
    int *data = malloc_shared<int>(N, q);
    q.parallel_for(range<1>(N), [=](id<1> i) {
        data[i] = i;
    }).wait();
    for (int i=0; i<N; i++) std::cout << data[i] << "\n";
    free(data, q);
    return 0;
}
```

INTEL® VTUNE™ PROFILER (BETA)

DPC++ PROFILING – TUNE FOR CPU, GPU & FPGA

Analyze Data Parallel C++ (DPC++)

See the lines of DPC++ that consume the most time

Tune for CPU, GPU & FPGA

Optimize for any supported hardware accelerator

Optimize Offload

Tune OpenMP* offload performance

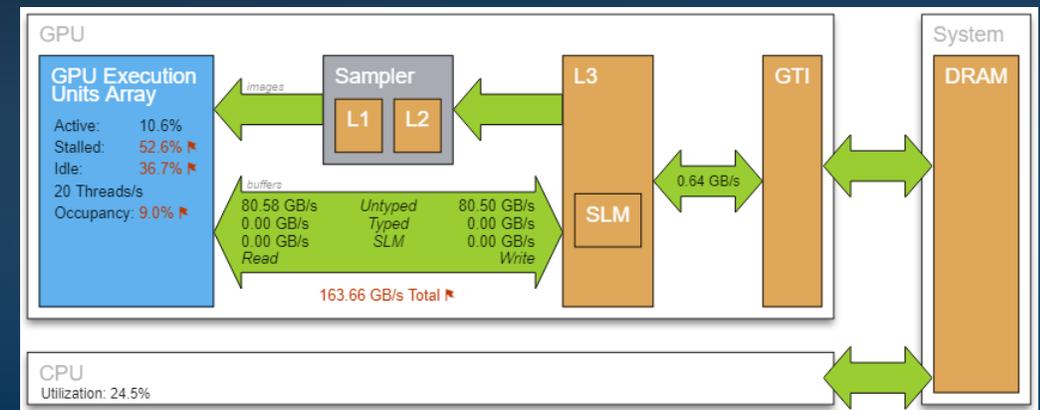
Wide Range of Performance Profiles

CPU, GPU, FPGA, threading, memory, cache, storage...

Supports Popular Languages

DPC++, C, C++, Fortran, Python*, Go*, Java*, or a mix

Source	Assembly	GPU Instructions Executed by Instruction T...
158	dx = ptr[j].pos[0] - ptr[i].pos[0]	75,002,500
159	dy = ptr[j].pos[1] - ptr[i].pos[1]	12,500,000
160	dz = ptr[j].pos[2] - ptr[i].pos[2]	12,500,000
161		
162	distanceSqr = dx*dx + dy*dy + dz*dz	87,500,000
163	distanceInv = 1.0 / sqrt(distanceSqr)	12,500,000
164		
165	ptr[i].acc[0] += dx * G * ptr[j].ma	162,503,750
166	ptr[i].acc[1] += dy * G * ptr[j].ma	150,000,000
167	ptr[i].acc[2] += dz * G * ptr[j].ma	150,000,000



There will still be a need to tune for each architecture.

Optimization Notice

Copyright © 2019, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

INTEL® ADVISOR (BETA)

DESIGN ASSISTANT — DESIGN FOR MODERN HARDWARE

Offload Advisor

Estimate performance of offloading to an accelerator

Roofline Analysis

Optimize CPU/GPU code for memory and compute

Vectorization Advisor

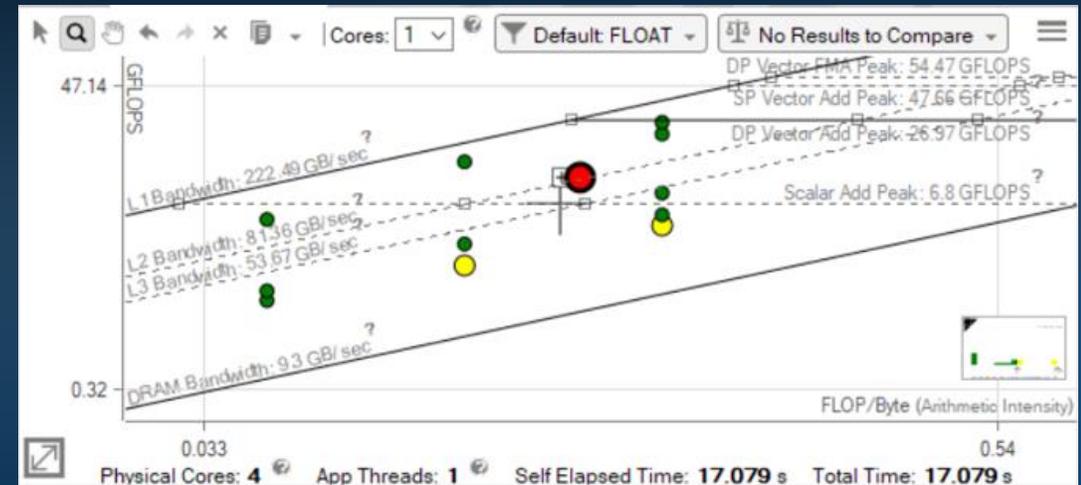
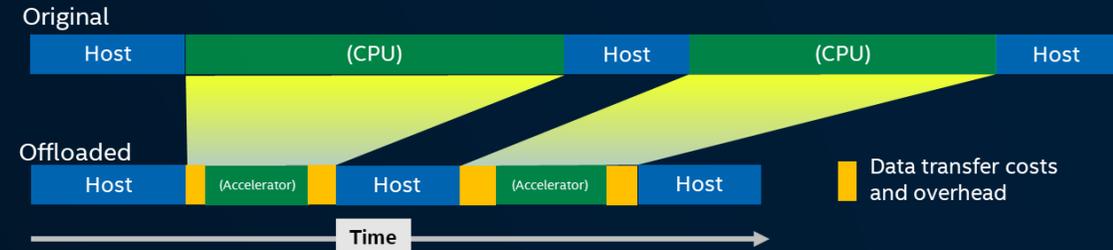
Add and optimize vectorization

Threading Advisor

Add effective threading to unthreaded applications

Flow Graph Analyzer

Create and analyze efficient flow graphs



There will still be a need to tune for each architecture.

[Optimization Notice](#)

Copyright © 2019, Intel Corporation. All rights reserved.

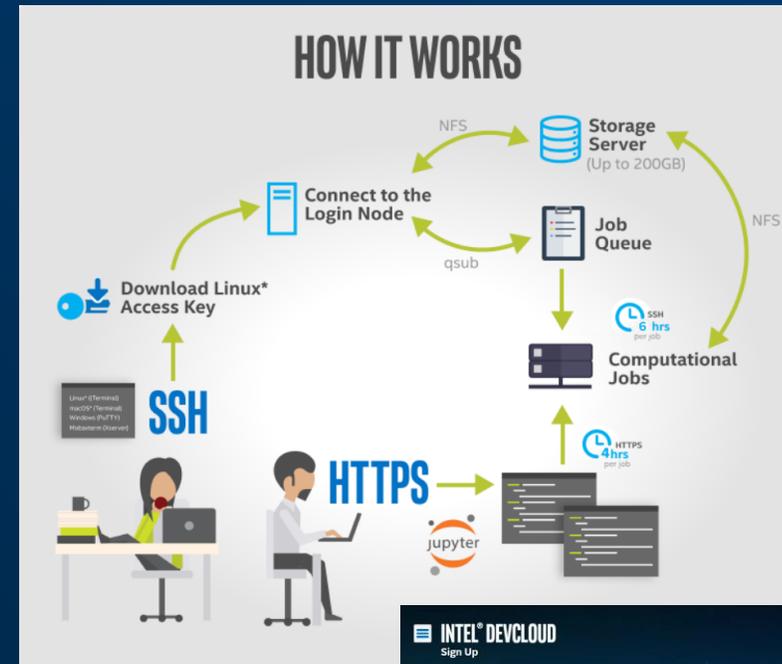
*Other names and brands may be claimed as the property of others.

Setup Intel® DevCloud and Jupyter Environment

INTEL® DEVCLOUD FOR ONEAPI

OVERVIEW

- A development sandbox to develop, test and run workloads across a range of Intel CPUs, GPUs, and FPGAs using Intel® oneAPI beta software
- A fast way to start coding
- Try the oneAPI toolkits, compilers, performance libraries, and tools
- Get 120 days of free access to the latest Intel® hardware and oneAPI software
- No downloads; No hardware acquisition; No installation



INTEL® DEVCLOUD
Sign Up

Get to Know Intel oneAPI (Beta) Now
No hardware acquisitions, system configurations, or software installations.

A Fast Way to Start Coding
Are you a forward-thinking developer interested in the next generation of data-centric computing innovation?
You've come to the right place.
The Intel® DevCloud is a development sandbox to learn about and program oneAPI cross-architecture applications.
Sign up now for full access to the latest Intel® CPUs, GPUs, and FPGAs, Intel® oneAPI Toolkits, and the new programming language, Data Parallel C++ (DPC++).
Access is free for 120 days, and extensions are totally possible.

Get Access Now
Required Fields(*)

First Name *
Last Name *
Email Address *
Country / Region *
Company or University *

Which hardware and accelerator architectures are you developing for? (Select all that apply.)
 ASICs (application-specific integrated circuits)
 CPU
 FPGA (field-programmable gate array)
 GPGPU (general-purpose GPU)

There will still be a need to tune for each architecture.

[Optimization Notice](#)

Copyright © 2019, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

INTEL® DEVCLOUD

ACCESS

- Remote access for professors and students to next-generation deep learning and machine learning development environments.
- Developers may use the cluster for research, coursework, labs, tutorials, and projects
- Hardware Available:
 - Includes a state-of-the-art server cluster powered by the Intel® Xeon® Scalable processor family, Intel® optimized frameworks, and other tools and libraries. Each processor has up to **56 cores** with two-way hyper-threading and up to **192 GB of on-platform DDR4 RAM**.
 - Intel® Arria® 10 FPGAs
 - Intel® Xeon® processors with Intel Graphics Technology
- Each developer is provided **220 GB of file storage** during the access period.
- Each user's home/user directory is not visible to others. Users' home directories on the cluster are deleted after the access period.
- Request access by visiting <http://software.intel.com/devcloud/oneapi>

There will still be a need to tune for each architecture.

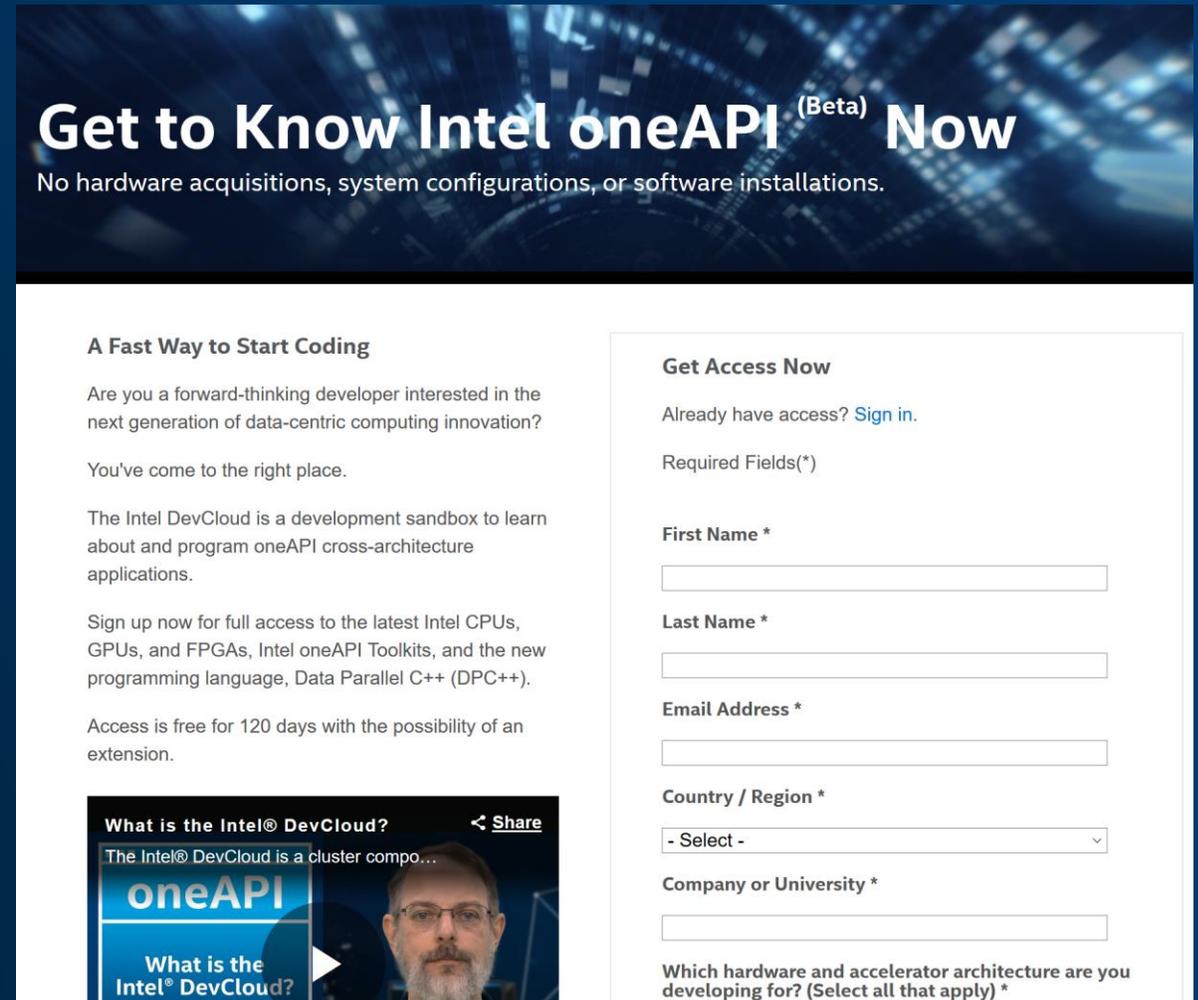
[Optimization Notice](#)

Copyright © 2019, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

STEPS

- Sign up for a DevCloud for oneAPI account here:
<https://intelsoftwaresites.secure.force.com/devcloud/oneapi>
- Open up a Jupyter lab notebook
- Common recipes on the DevCloud



Get to Know Intel oneAPI^(Beta) Now
No hardware acquisitions, system configurations, or software installations.

A Fast Way to Start Coding

Are you a forward-thinking developer interested in the next generation of data-centric computing innovation?

You've come to the right place.

The Intel DevCloud is a development sandbox to learn about and program oneAPI cross-architecture applications.

Sign up now for full access to the latest Intel CPUs, GPUs, and FPGAs, Intel oneAPI Toolkits, and the new programming language, Data Parallel C++ (DPC++).

Access is free for 120 days with the possibility of an extension.

Get Access Now

Already have access? [Sign in.](#)

Required Fields(*)

First Name *

Last Name *

Email Address *

Country / Region *

Company or University *

Which hardware and accelerator architecture are you developing for? (Select all that apply) *

What is the Intel® DevCloud? < Share
The Intel® DevCloud is a cluster compo...
oneAPI
What is the Intel® DevCloud? ▶

LAUNCH JUPYTER AND SELECT TERMINAL

The screenshot displays the Jupyter Launcher application window. The interface is divided into a left sidebar and a main content area. The sidebar contains a file explorer with a table of files and folders:

Name	Last Modified
work	9 minutes ago
tmp	34 minutes ago
intel	2 months ago
Welcome.ipynb	22 days ago

The main content area is titled "Launcher" and is organized into three sections:

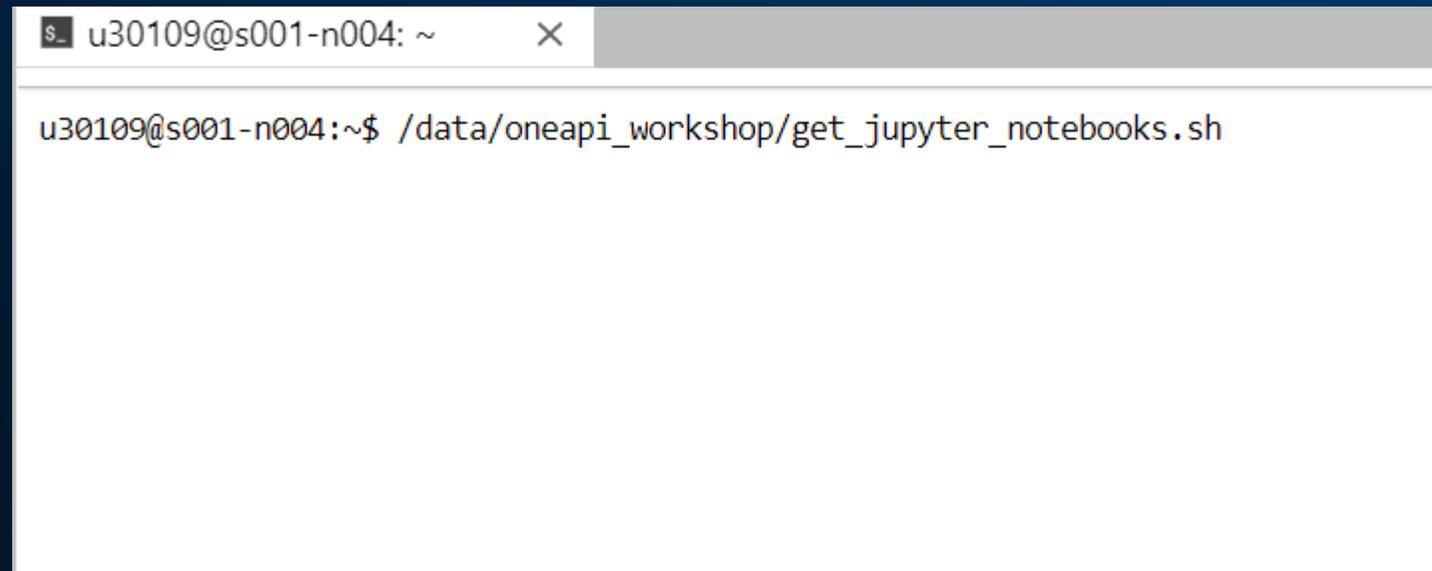
- Notebook**: Contains three options for launching notebooks:
 - Tensorflow 1.15 (AI kit)
 - Python 3.7 (Intel® oneAPI)
 - PyTorch 1.4.0 (AI kit)
- Console**: Contains three options for launching consoles:
 - Tensorflow 1.15 (AI kit)
 - Python 3.7 (Intel® oneAPI)
 - PyTorch 1.4.0 (AI kit)
- Other**: Contains four options for launching other applications:
 - Terminal**: Represented by a terminal icon (\$ _), highlighted with a red arrow.
 - Text File
 - Markdown File
 - Show Contextual Help

COMMANDS TO INPUT IN TERMINAL

Please execute the following commands in the Jupyter Terminal window

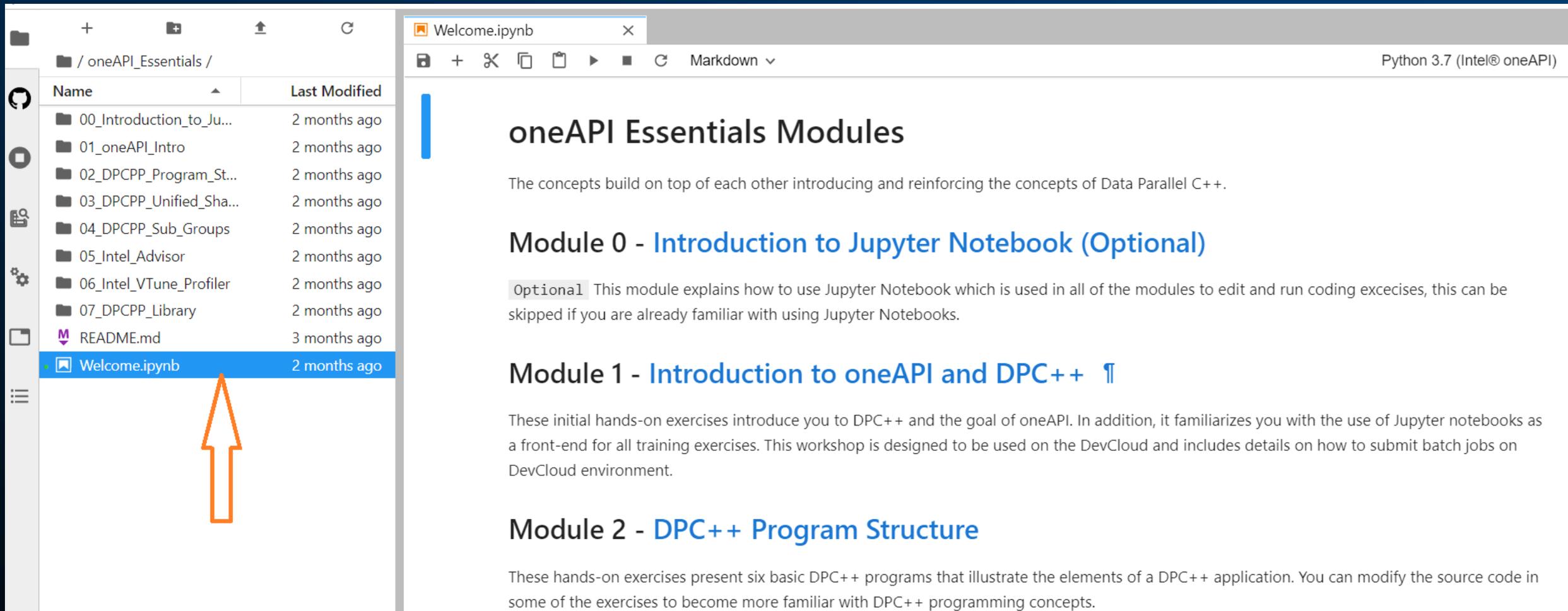
`/data/oneapi_workshop/get_jupyter_notebooks.sh`

This command copies workshop into the user directory

A screenshot of a terminal window. The title bar shows 's. u30109@s001-n004: ~' and a close button. The terminal content shows the command 'u30109@s001-n004:~\$ /data/oneapi_workshop/get_jupyter_notebooks.sh' being entered at the prompt.

```
s. u30109@s001-n004: ~ ×  
u30109@s001-n004:~$ /data/oneapi_workshop/get_jupyter_notebooks.sh
```

SELECT WELCOME.IPYNB



The screenshot displays a Jupyter Notebook interface. On the left, a file explorer shows the directory structure of 'oneAPI_Essentials'. The 'Welcome.ipynb' file is selected and highlighted in blue, with an orange arrow pointing to it. The main notebook editor shows the content of 'Welcome.ipynb', which includes a title 'oneAPI Essentials Modules', a subtitle 'Module 0 - Introduction to Jupyter Notebook (Optional)', and a description of the module. The notebook is running on Python 3.7 (Intel® oneAPI).

Name	Last Modified
00_Introduction_to_Ju...	2 months ago
01_oneAPI_Intro	2 months ago
02_DPCPP_Program_St...	2 months ago
03_DPCPP_Unified_Sha...	2 months ago
04_DPCPP_Sub_Groups	2 months ago
05_Intel_Advisor	2 months ago
06_Intel_VTune_Profiler	2 months ago
07_DPCPP_Library	2 months ago
README.md	3 months ago
Welcome.ipynb	2 months ago

oneAPI Essentials Modules

The concepts build on top of each other introducing and reinforcing the concepts of Data Parallel C++.

Module 0 - Introduction to Jupyter Notebook (Optional)

Optional This module explains how to use Jupyter Notebook which is used in all of the modules to edit and run coding exercises, this can be skipped if you are already familiar with using Jupyter Notebooks.

Module 1 - Introduction to oneAPI and DPC++ ↕

These initial hands-on exercises introduce you to DPC++ and the goal of oneAPI. In addition, it familiarizes you with the use of Jupyter notebooks as a front-end for all training exercises. This workshop is designed to be used on the DevCloud and includes details on how to submit batch jobs on DevCloud environment.

Module 2 - DPC++ Program Structure

These hands-on exercises present six basic DPC++ programs that illustrate the elements of a DPC++ application. You can modify the source code in some of the exercises to become more familiar with DPC++ programming concepts.

INTEL® ONEAPI ACADEMIC WORKSHOP

DPC++ ESSENTIALS



oneAPI

Single Programming Model
to Deliver Cross-Architecture Performance

Industry Initiative, Intel® oneAPI Beta Products

HPC 2019
Top 5 New Products
or Technologies
to Watch

Intel Software

A COMPLETE DPC++ PROGRAM

Single source

- Host code and heterogeneous accelerator kernels can be mixed in same source files

Familiar C++

- Library constructs add functionality, such as:

Construct	Purpose
queue	Work targeting
buffer	Data management
parallel_for	Parallelism

```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
    auto R = range<>{ num };
    buffer<int> A( R );

    queue().submit([&handler& h] {
        auto out =
            A.get_access<access::mode::write>(h);
        h.parallel_for(R, [=](id<1> idx) {
            out[idx] = idx[0]; }); });

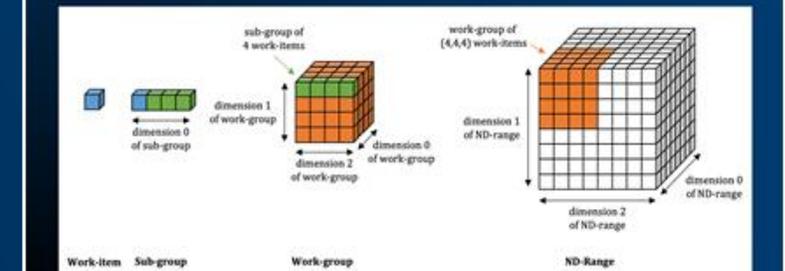
    auto result =
        A.get_access<access::mode::read>();
    for (int i=0; i<num; ++i)
        std::cout << result[i] << "\n";

    return 0;
}
```

Copyright © 2019 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

ND_RANGE KERNEL EXECUTION

Parallel execution with **ND_RANGE** Kernel helps to group work items that maps to hardware resources. This helps to tune applications for performance.

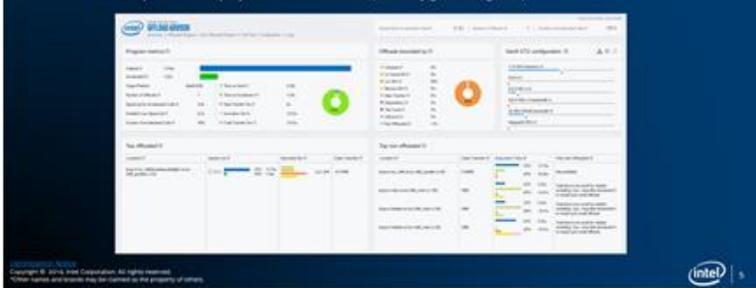


Work-item **Sub-group** **Work-group** **ND-Range**

Copyright © 2019 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

INTEL OFFLOAD ADVISOR (BETA)

- Starting from a baseline binary (running on CPU):
- Helps defining which sections of the code should run on a given accelerator
- Provides performance projection on accelerators (currently gen9 and gen11)



Copyright © 2019 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

INTEL® VTUNE™ PROFILER: HARDWARE ANALYSIS EXTENDED TO SVMs ARCHITECTURE

DPC++ kernels and their hardware metrics

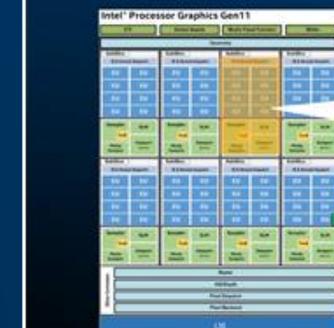
GPU hardware metrics
GPU compute Shader
GPU L3 Cache misses
GPU Texture Memory

GPU queue and utilization



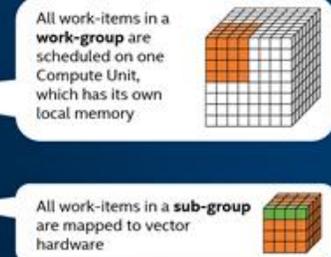
Copyright © 2019 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

HOW IT MAPS TO HARDWARE (INTEL GEN11 GRAPHICS)



All work-items in a **work-group** are scheduled on one Compute Unit, which has its own local memory

All work-items in a **sub-group** are mapped to vector hardware



Copyright © 2019 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

DPC++ Essentials Course Curriculum provides 20 hours of training and exercises using Jupyter Notebooks integrated with Intel® DevCloud

QSUB

- `qsub` can be used to submit jobs to the DevCloud job queue
- Jobs run asynchronously and report status upon completion
- The traditional way to execute `qsub` is to pass it a script:

```
“qsub <script.sh>”
```

- `qsub` requires absolute paths, e.g. `/bin/ls`
- `qsub -w $PWD` – Runs in current folder
- Output file is `<scriptname>.o<jobid>`

QSTAT/QDEL

- `qstat` displays running jobs
- `qdel <jobid>` deletes pending jobs

```
u42485@s001-n003:~$ qstat
Job ID                Name                User                Time Use S Queue
-----
591829.v-qsvr-1      ...ub-singleuser  u42485              00:01:06 R jupyterhub
591832.v-qsvr-1      STDIN              u42485              0 R batch
591833.v-qsvr-1      STDIN              u42485              0 R batch
591834.v-qsvr-1      STDIN              u42485              0 R batch
591835.v-qsvr-1      STDIN              u42485              0 R batch
u42485@s001-n003:~$ qdel 591835
```

INTERACTIVE SHELLS

- Getting an interactive shell
 - `qsub -I`
- Requesting an iGPU/FPGA node
 - `qsub -I -l nodes=1:gpu:ppn=2`
 - `clinfo - lists iGPU info`

HANDS ON EXERCISE – SIMPLE

- Objectives for the hands-on:
 - Presentation of the DPC++ training environment via Jupyter notebooks
 - Compile and Run Simple DPC++ code

TRANSITION TO JUPYTER NOTEBOOK - SIMPLE

Welcome.ipynb

Select link **Introduction to oneAPI and DPC++**

TRANSITION TO JUPYTER NOTEBOOK – VECTOR ADD

Return to Welcome.ipynb

Select link **Introduction to DPC++**

SYCL CLASSES

DEVICE

- The **device** class represents the capabilities of the accelerators in a oneAPI system.
- The device class contains member functions for querying information about the device, which is useful for DPC++ programs where multiple devices are created.
- The function **get_info** gives information about the device:
 - Name, vendor, and version of the device
 - The local and global work item IDs
 - Width for built in types, clock frequency, cache width and sizes, online or offline

```
queue q;  
device my_device = q.get_device();  
std::cout << "Device: " << my_device.get_info<info::device::name>() << std::endl;
```

DEVICE SELECTOR

- The **device_selector** class enables the runtime selection of a particular device to execute kernels based upon user-provided heuristics.
- The following code sample shows use of the standard device selectors (**default_selector**, **cpu_selector**, **gpu_selector**...) and a derived **device_selector**

```
default_selector selector;  
// host_selector selector;  
// cpu_selector selector;  
// gpu_selector selector;  
queue q(selector);  
std::cout << "Device: " << q.get_device().get_info<info::device::name>() << std::endl;
```

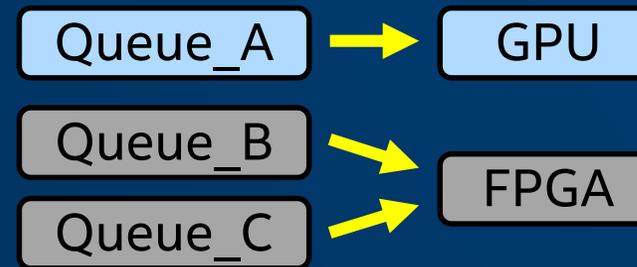
QUEUE

- A queue **submits command groups** to be executed by the SYCL runtime
- Queue is a mechanism where work is submitted to a device.
- A Queue map to one device and multiple queues can be mapped to the same device.

```
queue q;
```

```
q.submit([&](handler& h) {  
    // COMMAND GROUP CODE  
});
```

CHOOSING WHERE DEVICE KERNELS RUN



Work is submitted to queues

- Each queue is associated with exactly one device (e.g. a specific GPU or FPGA)
- You can:
 - Decide which device a queue is associated with (if you want)
 - Have as many queues as desired for dispatching work in heterogeneous systems

Create queue targeting any device:	<code>queue();</code>
Create queue targeting a pre-configured classes of devices:	<code>queue(cpu_selector{});</code> <code>queue(gpu_selector{});</code> <code>queue(intel::fpga_selector{});</code> <code>queue(accelerator_selector{});</code> <code>queue(host_selector{});</code>
Create queue targeting specific device (custom criteria):	<pre> class custom_selector : public device_selector { int operator()(..... // Any logic you want! ... queue(custom_selector{}); </pre>

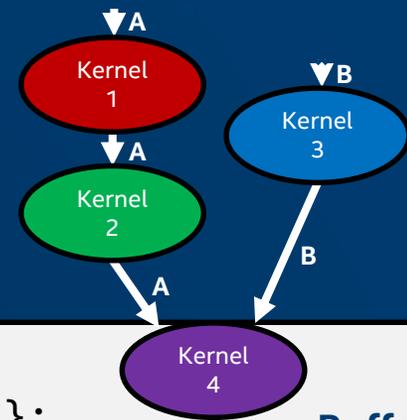
Always available

KERNEL

- The kernel class encapsulates methods and data for executing code on the device when a command group is instantiated
- Kernel object is not explicitly constructed by the user
- Kernel object is constructed when a kernel dispatch function, such as **parallel_for**, is called

```
q.submit([&](handler& h) {  
    h.parallel_for(range<1>(N), [=](id<1> i) {  
        A[i] = B[i] + C[i]);  
    });  
});
```

THE BUFFER MODEL



Buffers: Encapsulate data in a SYCL application

- Across both devices and host!

Accessors: Mechanism to access buffer data

- Create data dependencies in the SYCL graph that order kernel executions

```
int main() {  
    auto R = range<1>{ num };  
    buffer<int> A{ R }, B{ R };  
    queue Q;  
  
    Q.submit([&](handler& h) {  
        accessor out(A, h, write_only);  
  
        h.parallel_for(R, [=](auto idx) {  
            out[idx] = idx[0]; }); });  
  
    Q.submit([&](handler& h) {  
        accessor out(A, h, write_only);  
        h.parallel_for(R, [=](auto idx) {  
            out[idx] = idx[0]; }); });  
    ...  
}
```

DPC++ CODE ANATOMY

- oneAPI programs require the include of `cl/sycl.hpp`.
- It is recommended to employ the namespace statement to save typing repeated references into the `sycl` namespace

```
#include <CL/sycl.hpp>  
using namespace sycl;
```

DPC++ CODE ANATOMY

```
void dpcpp_code(int* a, int* b, int* c) {  
    // Setting up a DPC++ device queue  
    queue q;  
    // Setup buffers for input and output vectors  
    buffer buf_a(a, range<1>(N));  
    buffer buf_b(b, range<1>(N));  
    buffer buf_c(c, range<1>(N));  
    //Submit Command group function object to the queue  
    q.submit([&](handler &h){  
        //Create device accessors to buffers allocated in global memory  
        accessor A(buf_a, h, read_only);  
        accessor B(buf_b, h, read_only);  
        accessor C(buf_c, h, write_only);  
        //Specify the device kernel body as a lambda function  
        h.parallel_for(range<1>(N), [=](auto i){  
            C[i] = A[i] + B[i];  
        });  
    });  
}
```

Step 1: create a device queue
(developer can specify a device type via device selector or use default selector)

Step 2: create buffers
(represent both host and device memory)

Step 3: submit a command for
(asynchronous) execution

Step 4: create buffer accessors
to access buffer data on the
device

Step 5: send a kernel (lambda) for
execution

Step 6: write a kernel

Kernel invocations are
executed in parallel

Kernel is invoked for each
element of the range

Kernel invocation has
access to the invocation id

Done!

The results are copied to vector `c` at `buf_c` buffer destruction

CUSTOM DEVICE SELECTOR

The following code shows derived `device_selector` that employs a device selector heuristic. The selected device prioritizes a GPU device because the integer rating returned is higher than for CPU or other accelerator.

```
#include <CL/sycl.hpp>
using namespace cl::sycl;
class my_device_selector : public device_selector {
public:
    int operator()(const device& dev) const override {
        int rating = 0;
        if (dev.is_gpu() & (dev.get_info<info::device::name>().find("vendor") != std::string::npos))
            rating = 3;
        else if (dev.is_gpu()) rating = 2;
        else if (dev.is_cpu()) rating = 1;
        return rating;
    };
};
int main() {
    my_device_selector selector;
    queue q(selector);
    std::cout << "Device: "
    << q.get_device().get_info<info::device::name>() << std::endl;
    return 0;
}
```

ASYNCHRONOUS EXECUTION

Think of a SYCL application as two parts:

1. Host code
2. The graph of kernel executions

These **execute independently**, except at synchronizing operations

- The host code submits work to build the graph (and can do compute work itself)
- The graph of kernel executions and data movements **executes asynchronously from host code**, managed by the SYCL runtime

ASYNCHRONOUS EXECUTION (CONT'D)

Host

Host code execution

Enqueues kernel to graph, and keeps going



```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
    auto R = range<1>{ num };
    buffer<int> A{ R };

    queue{}.submit([&](handler& h) {
        accessor out(A, h, write_only);
        h.parallel_for(R, [=](id<1> idx) {
            out[idx] = idx[0]; }); });

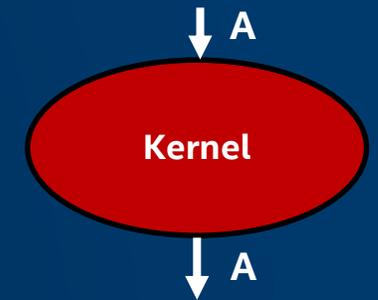
    host_accessor result(A, read_only);
    for (int i=0; i<num; ++i)
        std::cout << result[i] << "\n";

    return 0;
}
```



Graph

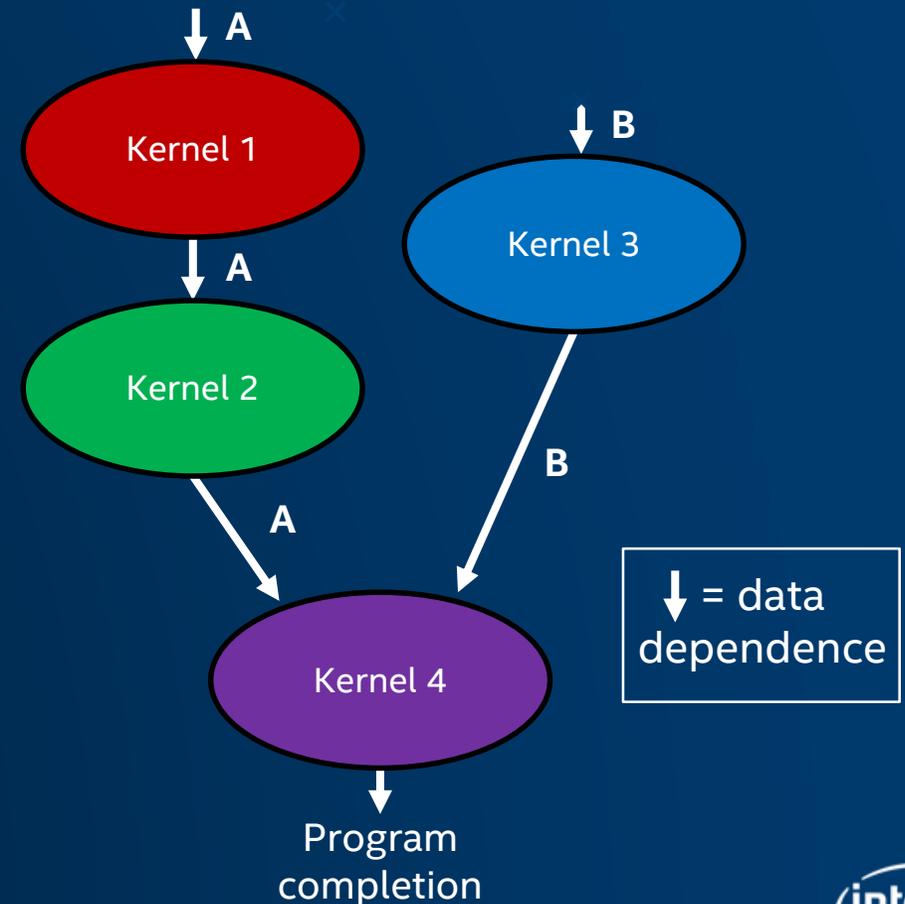
Graph executes asynchronously to host program



GRAPH OF KERNEL EXECUTIONS

```
int main() {  
  auto R = range<1>{ num };  
  buffer<int> A{ R }, B{ R };  
  queue Q;  
  
  Q submit([&](handler& h) {  
    accessor out(A, h, write_only);  
    h.parallel_for(R, [=](id<1> idx) {  
      out[idx] = idx[0]; }); }); } Kernel 1  
  
  Q submit([&](handler& h) {  
    accessor out(A, h, write_only);  
    h.parallel_for(R, [=](id<1> idx) {  
      out[idx] = idx[0]; }); }); } Kernel 2  
  
  Q submit([&](handler& h) {  
    accessor out(B, h, write_only);  
    h.parallel_for(R, [=](id<1> idx) {  
      out[idx] = idx[0]; }); }); } Kernel 3  
  
  Q submit([&](handler& h) {  
    accessor in(A, h, read_only);  
    accessor inout(B, h);  
    h.parallel_for(R, [=](id<1> idx) {  
      inout[idx] *= in[idx]; }); }); } Kernel 4  
}
```

Automatic data and control dependence resolution!



SYNCHRONIZATION – HOST ACCESSOR

```
int main() {
    constexpr int N = 100;
    auto R = range<1>(N);
    std::vector<double> v(N, 10);
    queue q;

    buffer buf(v)
    q.submit([&](handler& h) {
        accessor a(buf, h)
        h.parallel_for(R, [=](auto i) {
            a[i] -= 2;
        });
    });

    host_accessor b(buf, read_only);
    for (int i = 0; i < N; i++)
        std::cout << b[i] << "\n";
    return 0;
}
```

Buffer takes **ownership of the data** stored in vector.

Creating host accessor is a **blocking call** and will only return after all enqueued DPC++ kernels that modify the same buffer in any queue completes execution and the **data is available to the host** via this host accessor.

SYNCHRONIZATION – BUFFER DESTRUCTION – A BETTER WAY

```
#include <CL/sycl.hpp>
constexpr int N=100;
using namespace cl::sycl;

void dpcpp_code(std::vector<double> &v, queue &q){
    auto R = range<1>(N);
    buffer buf(v);
    q.submit([&](handler& h) {
        accessor a(buf, h);
        h.parallel_for(R, [=](auto i) {
            a[i] -= 2;
        });
    });
}

int main() {
    std::vector<double> v(N, 10);
    queue q;
    dpcpp_code(v,q);
    for (int i = 0; i < N; i++)
        std::cout << v[i] << "\n";
    return 0;
}
```

Buffer creation happens within a **separate function scope**.

When execution advances beyond this function scope, **buffer destructor is invoked** which relinquishes the ownership of data and **copies back the data to the host** memory.

HANDS-ON LAB – COMPLEX NUMBER MULTIPLICATION

- In this lab we provide with the source code that **computes multiplication of two complex numbers** where Complex class is the definition of a custom type that represents complex numbers
- In this example the student will learn how to create a **custom device selector** and to target GPU or CPU of a specific vendor. The student will also learn how to pass in a vector of **custom Complex class objects** in parallel and needs to modify the source code to setup a write accessor and call the Complex class member function as kernel to compute the multiplication

TRANSITION TO JUPYTER NOTEBOOK – COMPLEX NUMBER MULTIPLICATION

Welcome.ipynb

Select link **DPC++ Program Structure**

RECAP

oneAPI solves the **challenges of programming** in a heterogeneous world

Take advantage of **oneAPI solutions** to enable your workflows

Use the **Intel® DevCloud** to test-drive oneAPI tools and libraries

Introduced to **DPC++** language and programming model

Important Classes for DPC++ application

Device selection and offloading kernel workloads

DPC++ Buffers, Accessors, Command Group handler, lambda code as kernel

Hands on activities

- Vector-Increment/Vector-add exercise – demonstrate coding ease by modifying the sample source code.
- Complex number multiplication

NOTICES & DISCLAIMERS

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request. No product or component can be absolutely secure. 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 ©, 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