You can Win an Intel® NUC

Create a DevCloud account in the link:

- https://sforms.intel.com/DevCloud/?eventcode=lattice-07302021

Post your “DevCloud user id” in the zoom chat window

Towards the end of the workshop 5 registered users will be selected to receive an Intel® NUC

This is a platform with Intel integrated GPU for oneAPI code development

Optimization Notice

Data Parallel C++ Essentials

oneAPI VIRTUAL WORKSHOP

Praveen Kundurthy

What is oneAPI and Data Parallel C++?

intel.

Introduction to oneAPI

- 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) DPC++ Program Structure
  f) Unified Shared Memory (USM)
  g) Sub-Groups
  h) Reduction
- Hands On
  - Introduction to DPC++ - Simple
  - Complex multiplication
  - USM, Sub-Groups and Reductions

Optimization Notice

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

Use device selection to offload kernel workloads

Understand DPC++ New features (Unified Shared memory, Sub-Groups and Reductions)

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

Optimization Notice

Cross-Architecture Programming for Accelerated Compute, Freedom of Choice for Hardware

oneAPI:
Industry Initiative & Intel Products

One Intel Software & Architecture group
Intel Architecture, Graphics & Software
November 2020

HPC wire
2019 Readers' Choice Awards
Top 5 New Products or Technologies to Watch
Intel OneAPI
Awards Winner for 15 Years

All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Programming Challenges

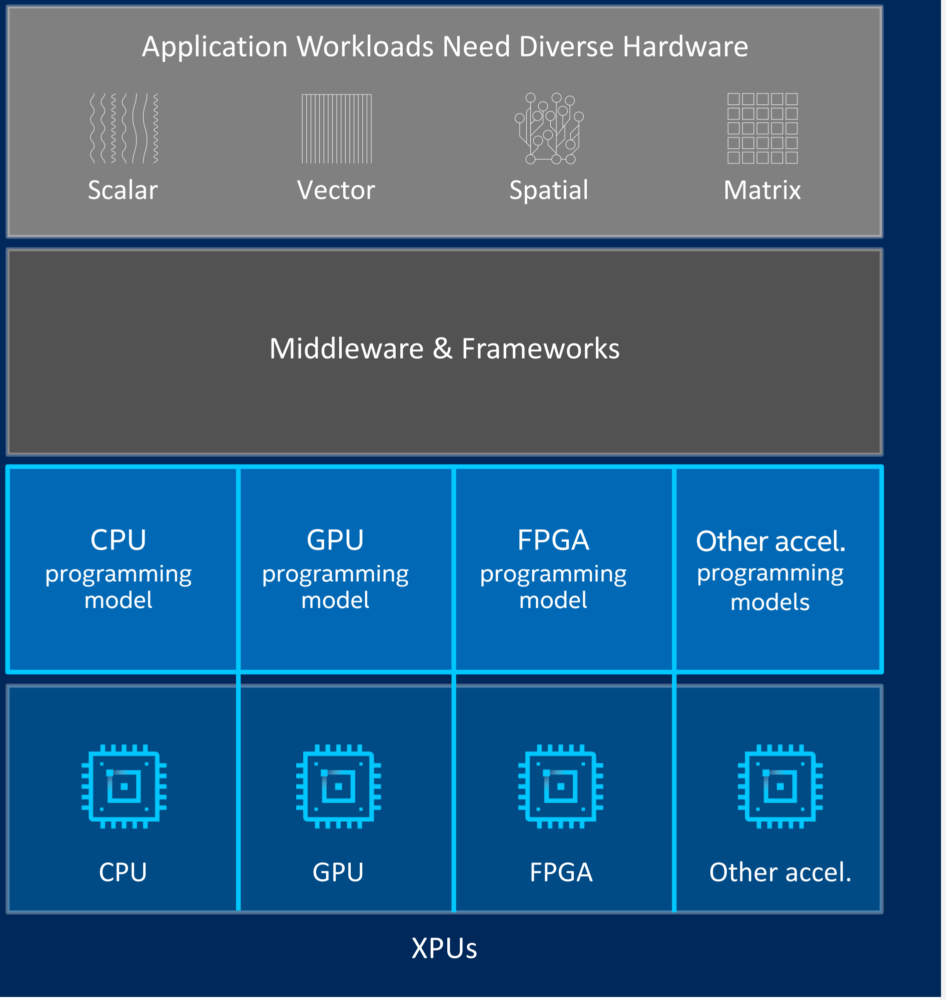
for Multiple Architectures

Growth in specialized workloads

Variety of data-centric hardware required

Separate programming models and toolchains for each architecture are required today

Software development complexity limits freedom of architectural choice



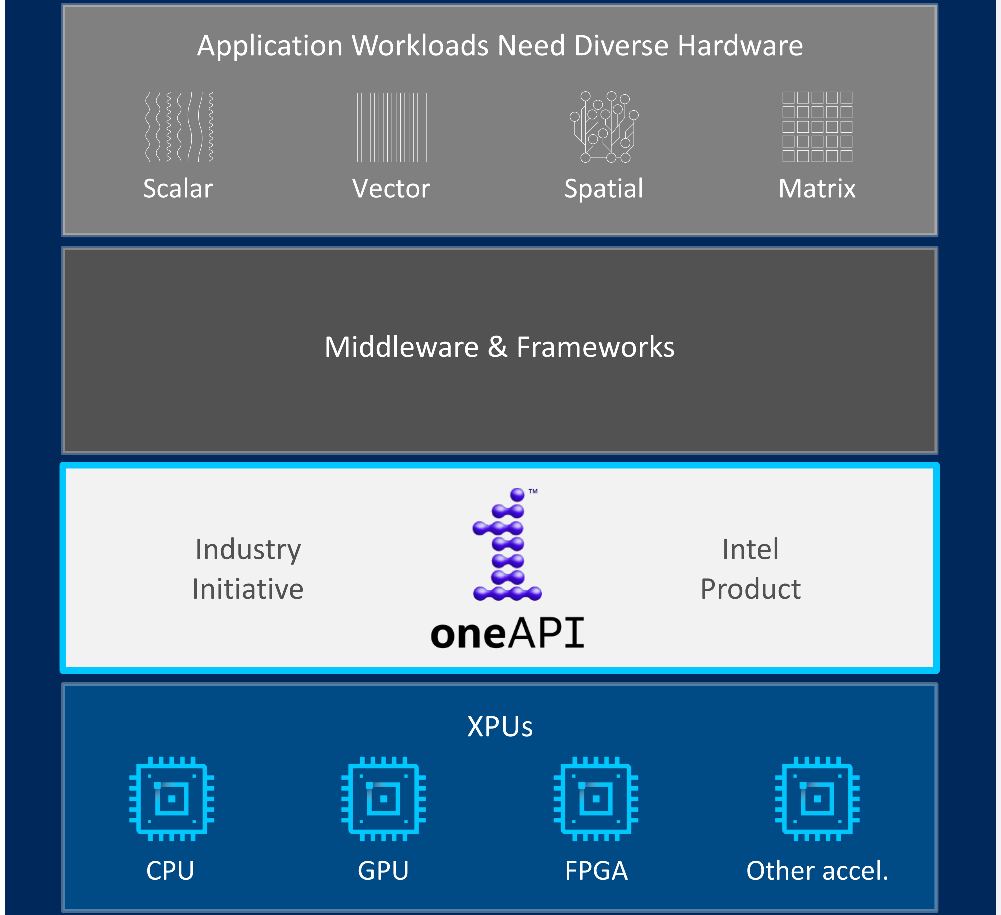
Introducing
oneAPI

Cross-architecture programming that delivers freedom to choose the best hardware

Based on industry standards and open specifications

Exposes cutting-edge performance features of latest hardware

Compatible with existing high-performance languages and programming models including C++, OpenMP, Fortran, and MPI



oneAPI Industry Initiative

Break the Chains of Proprietary Lock-in

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

Powerful libraries designed for acceleration of domain-specific functions

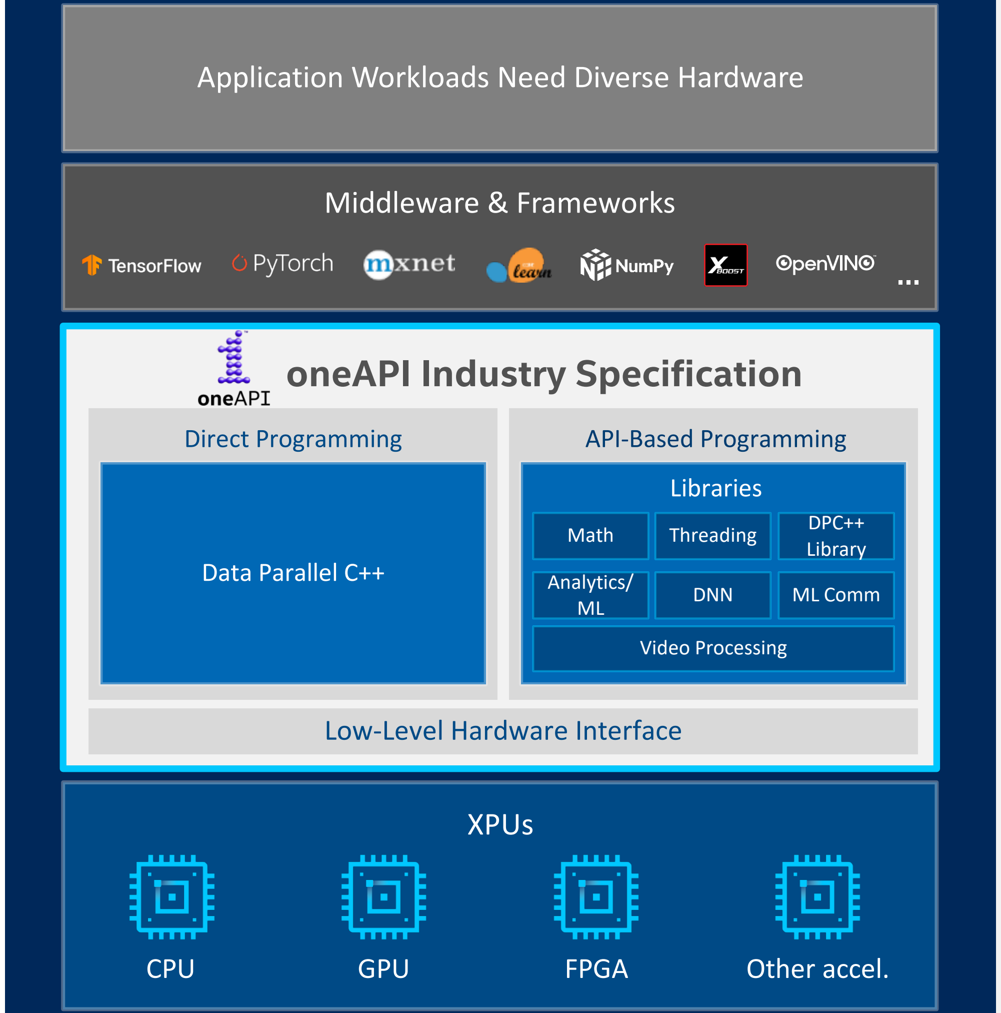
Low-level hardware abstraction layer

Open to promote community and industry collaboration

Enables code reuse across architectures and vendors

oneAPI

The productive, smart path to freedom for accelerated computing from the economic and technical burdens of proprietary programming models



Application Workloads Need Diverse Hardware

Middleware & Frameworks

TensorFlow PyTorch mxnet learn NumPy XBoost OpenVINO ...

oneAPI Industry Specification

Direct Programming

Data Parallel C++

API-Based Programming

Libraries

Math | Threading | DPC++ Library

Analytics/ ML | DNN | ML Comm

Video Processing

Low-Level Hardware Interface

XPUs

CPU GPU FPGA Other accel.

Visit oneapi.com for more details

Intel® oneAPI Toolkits

A complete set of proven developer tools expanded from CPU to XPU

oneAPI

Intel® oneAPI Base Toolkit

Native Code Developers

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

Add-on Domain-specific Toolkits

Specialized Workloads

Intel® oneAPI Tools for HPC

Deliver fast Fortran, OpenMP & MPI applications that scale

Intel® oneAPI Tools for IoT

Build efficient, reliable solutions that run at network's edge

Intel® oneAPI Rendering Toolkit

Create performant, high-fidelity visualization applications

Toolkits powered by oneAPI

Data Scientists & AI Developers

Intel® AI Analytics Toolkit

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

Intel® Distribution of OpenVINO™ Toolkit

Deploy high performance inference & applications from edge to cloud

Optimization Notice

Intel® oneAPI Base Toolkit

Accelerate Data-centric Workloads

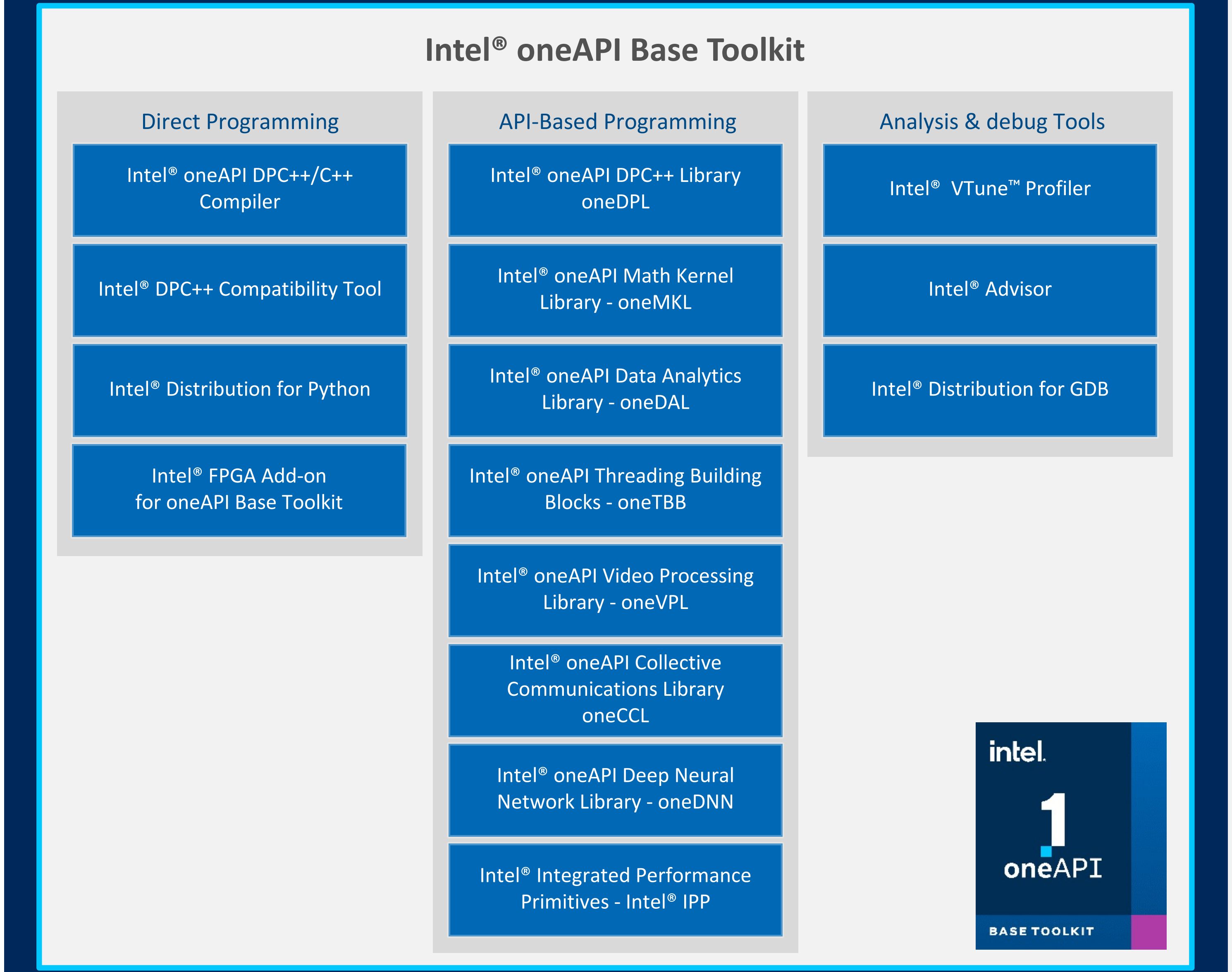
A core set of core tools and libraries for developing high-performance applications on Intel® CPUs, GPUs, and FPGAs.

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++/C++ Compiler
- Intel® DPC++ Compatibility Tool
- Intel® Distribution for Python
- Intel® FPGA Add-on for oneAPI Base Toolkit

API-Based Programming
- Intel® oneAPI DPC++ Library oneDPL
- Intel® oneAPI Math Kernel Library - oneMKL
- Intel® oneAPI Data Analytics Library - oneDAL
- Intel® oneAPI Threading Building Blocks - oneTBB
- Intel® oneAPI Video Processing Library - oneVPL
- Intel® oneAPI Collective Communications Library oneCCL
- Intel® oneAPI Deep Neural Network Library - oneDNN
- Intel® Integrated Performance Primitives - Intel® IPP

Analysis & debug Tools
- Intel® VTune™ Profiler
- Intel® Advisor
- Intel® Distribution for GDB

Learn More: intel.com/oneAPI-BaseKit

Intel® oneAPI Data Parallel C++ Library (oneDPL)

- Three components:

  1. **Standard C++ APIs:** Tested and supported within DPC++ kernels
  2. **Parallel STL:** C++17 algorithms extended with DPC++ execution policies
  3. **STL Extensions:** Additional algorithms, classes and iterators

```
sycl::queue q;
std::vector<int> v(N);
std::sort(oneapi::dpl::execution::make_device_policy(q), v.begin(), v.end());
```

- Recommended for codes using C++17 algorithms, or libraries like Thrust

See https://spec.oneapi.com/versions/latest/elements/oneDPL/source/index.html

Intel® DPC++ Compatibility Tool

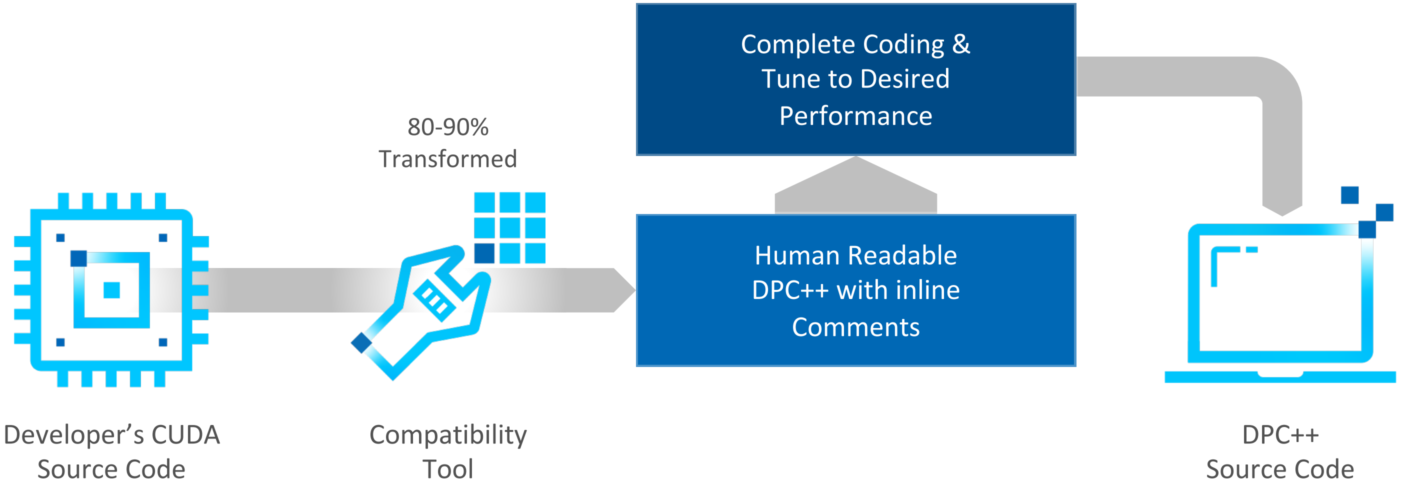
Minimizes Code Migration Time

Assists developers migrating code written in CUDA to DPC++ once, generating **human readable** code wherever possible

~80-90% of code typically migrates automatically

Inline comments are provided to help developers finish porting the application

Intel DPC ++ Compatibility Tool Usage Flow



Complete Coding & Tune to Desired Performance
80-90% Transformed
Human Readable DPC++ with inline Comments
Developer's CUDA Source Code
Compatibility Tool
DPC++ Source Code

Intel® VTune™ Profiler

DPC++ Profiling-Tune for CPU, GPU & FPGA

Analyze Data Parallel C++ (DPC++)

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

Tune for Intel CPUs, GPUs & FPGAs

Optimize for any supported hardware accelerator

Optimize Offload

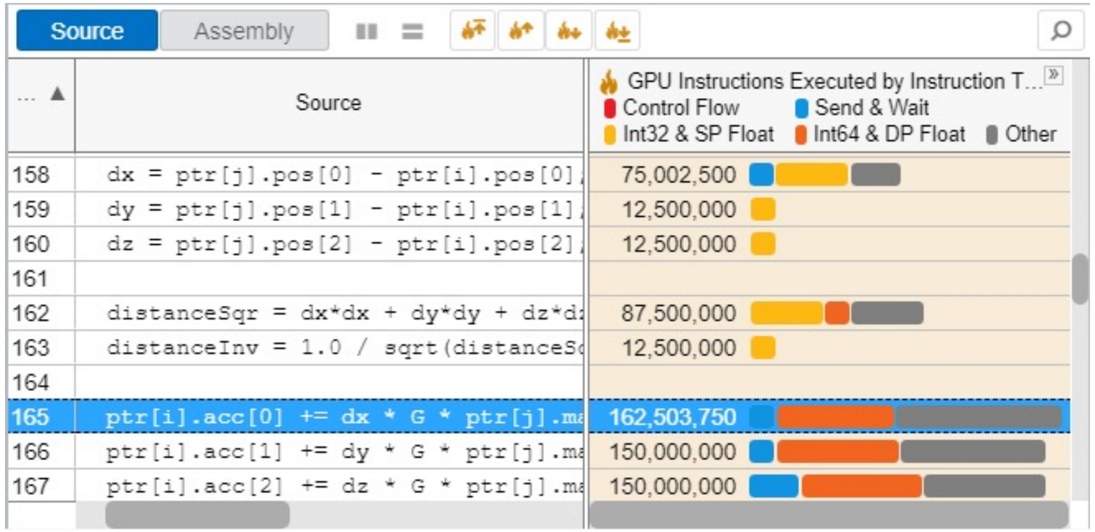
Tune OpenMP offload performance

Wide Range of Performance Profiles

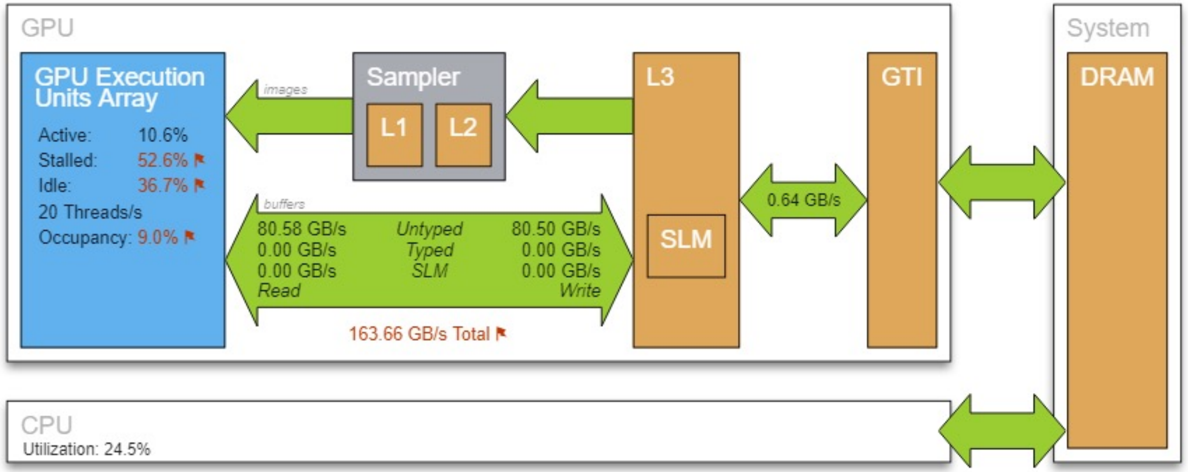
CPU, GPU, FPGA, threading, memory, cache, storage…

Supports Popular Languages

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



… ▲	Source	GPU Instructions Executed by Instruction T… Control Flow · Send & Wait Int32 & SP Float · Int64 & DP Float · Other
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*d`	87,500,000
163	`distanceInv = 1.0 / sqrt(distanceS`	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

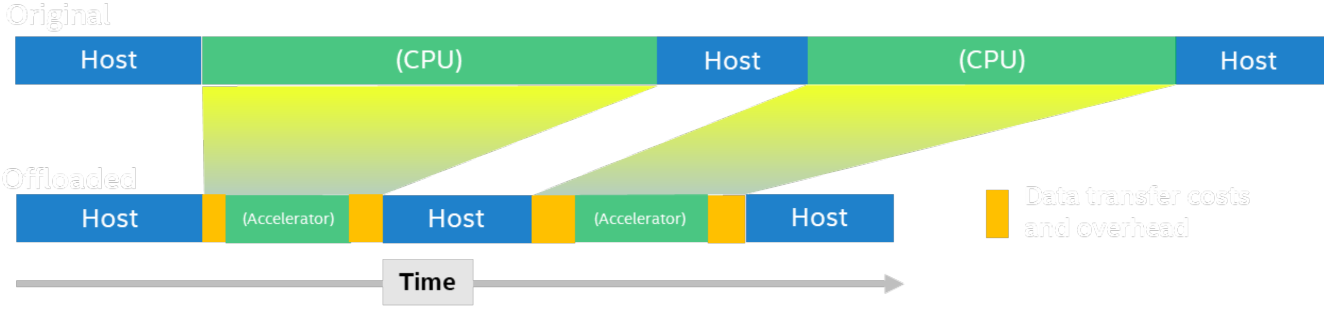


There will still be a need to tune for each architecture.

Optimization Notice

Intel® Advisor

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

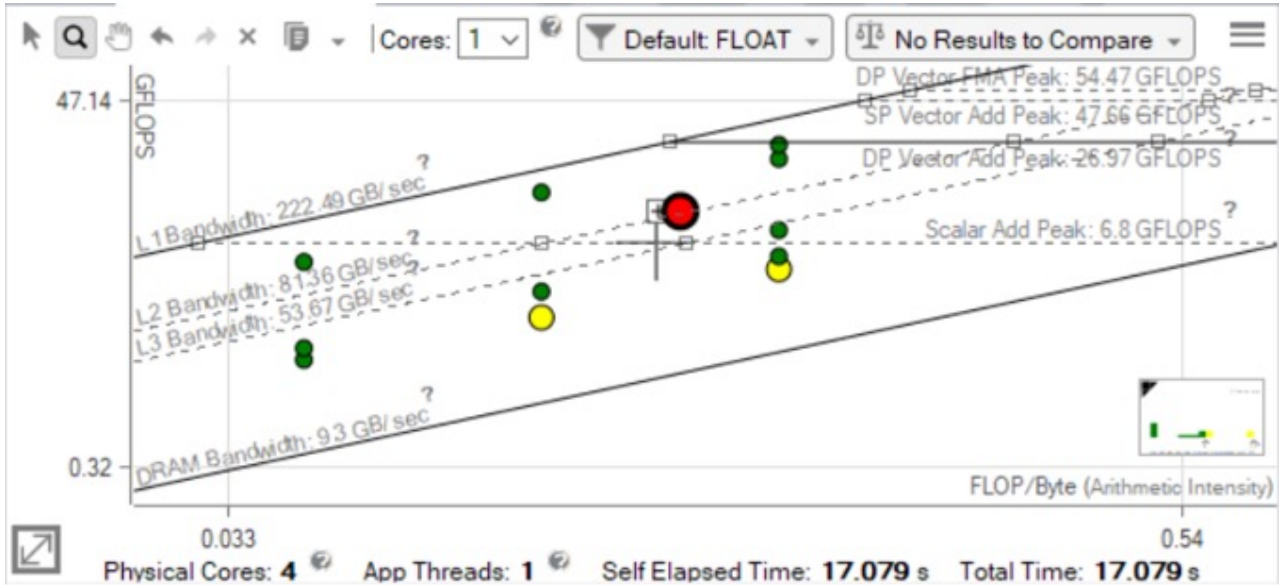
Add and optimize vectorization

Threading Advisor

Add effective threading to unthreaded applications

Flow Graph Analyzer

Create and analyze efficient flow graphs


Cores: 1 | Default: FLOAT | No Results to Compare

Physical Cores: **4** App Threads: **1** Self Elapsed Time: **17.079** s Total Time: **17.079** s

GFLOPS
47.14
0.32
DP Vector FMA Peak: 54.47 GFLOPS
SP Vector Add Peak: 47.66 GFLOPS
DP Vector Add Peak: 26.97 GFLOPS
Scalar Add Peak: 6.8 GFLOPS
L1 Bandwidth: 222.49 GB/sec
L2 Bandwidth: 81.36 GB/sec
L3 Bandwidth: 53.67 GB/sec
DRAM Bandwidth: 9.3 GB/sec
FLOP/Byte (Arithmetic Intensity)
0.033
0.54

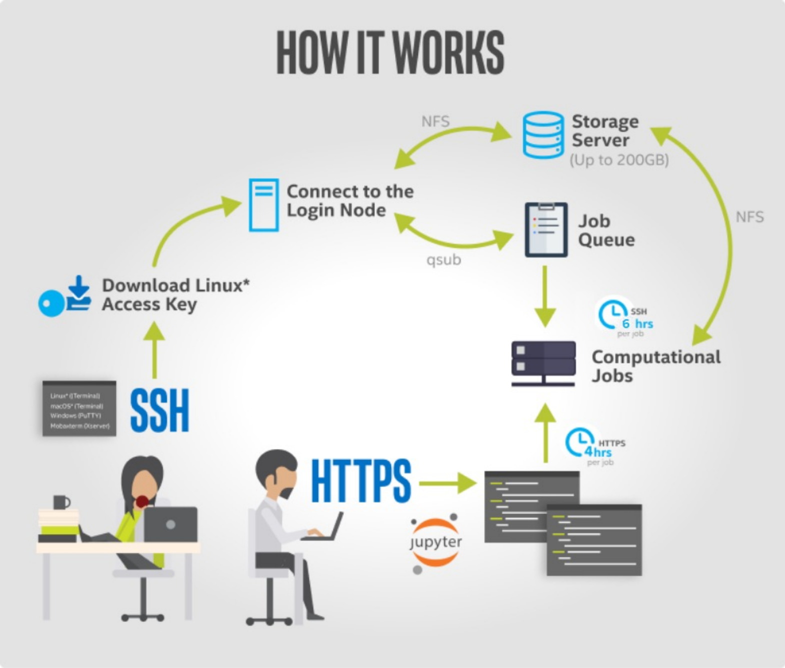
Optimization Notice

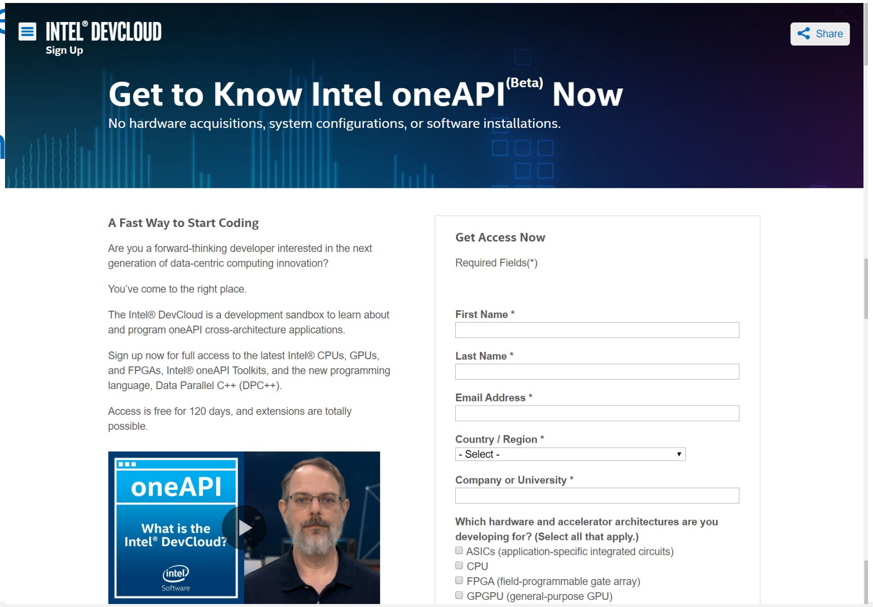
There will still be a need to tune for each architecture.

SETUP INTEL® DEVCLOUD AND JUPYTER ENVIRONMENT

Intel® devcloud for oneAPI

- 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

Share

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.

oneAPI
What is the Intel® DevCloud?
intel Software

Get Access Now

Required Fields(*)

First Name *

Last Name *

Email Address *

Country / Region *
- Select - ▾

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)

Optimization Notice

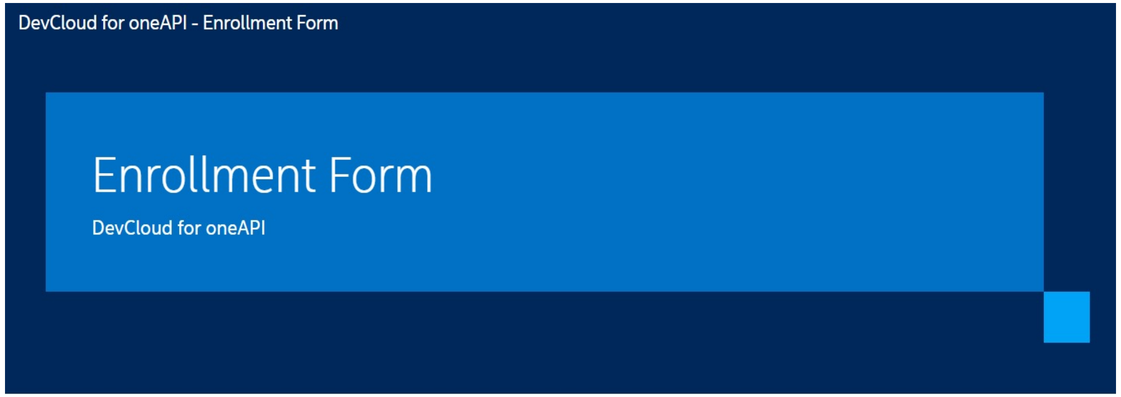
Event code

lattice-07302021

Register to DevCloud using
https://sforms.intel.com/DevCloud/?eventcode=lattice-07302021

Register to Devcloud

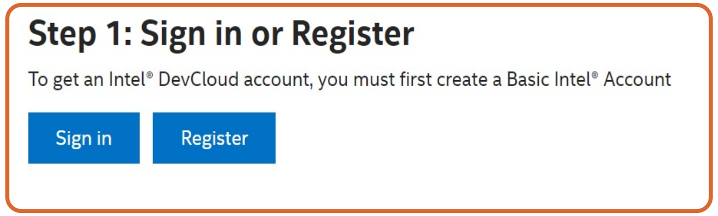
- Step 1: Register or Sign into Intel Developer Zone



DevCloud for oneAPI - Enrollment Form

Enrollment Form

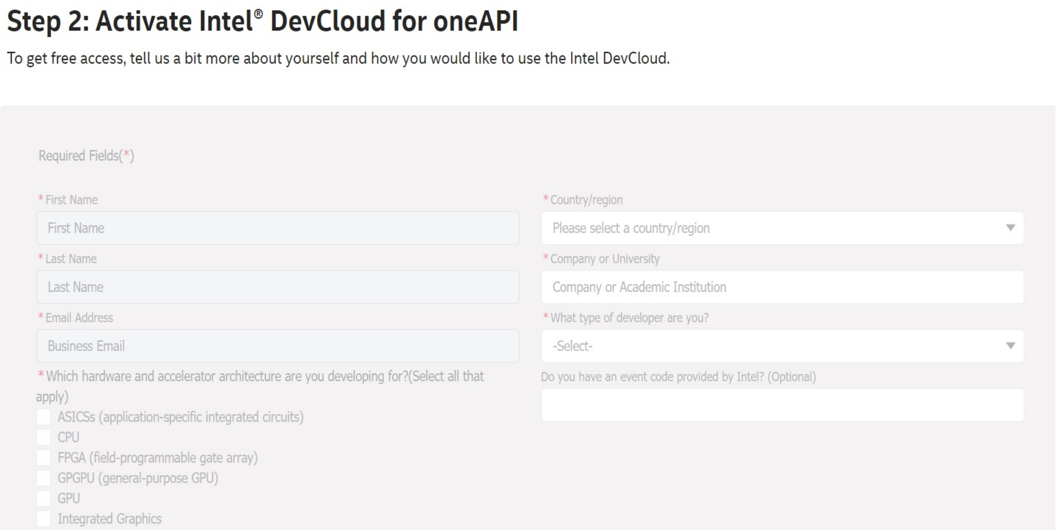
DevCloud for oneAPI



Step 1: Sign in or Register

To get an Intel® DevCloud account, you must first create a Basic Intel® Account

- Step 2: Activate Intel Devcloud Account



Step 2: Activate Intel® DevCloud for oneAPI

To get free access, tell us a bit more about yourself and how you would like to use the Intel DevCloud.

Required Fields(*)

*First Name: First Name

*Last Name: Last Name

*Email Address: Business Email

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

□ ASICSs (application-specific integrated circuits)
□ CPU
□ FPGA (field-programmable gate array)
□ GPGPU (general-purpose GPU)
□ GPU
□ Integrated Graphics

*Country/region: Please select a country/region

*Company or University: Company or Academic Institution

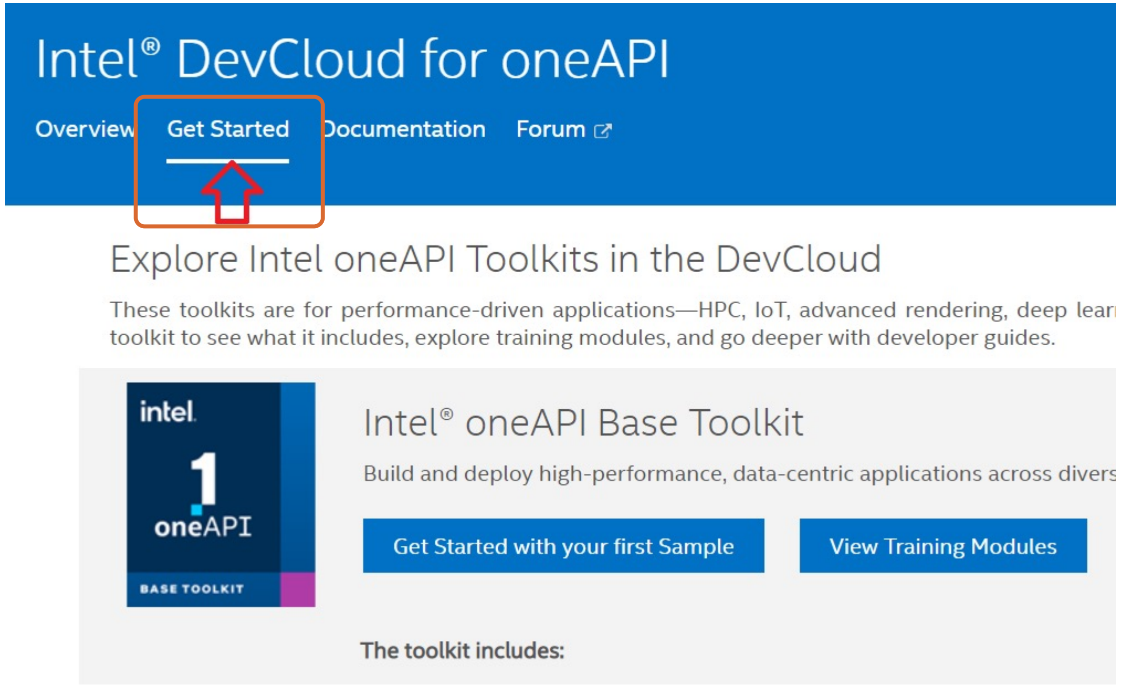
*What type of developer are you?: -Select-

Do you have an event code provided by Intel? (Optional)

Optimization Notice

Get Started with Devcloud

- Step 3: Click on Get Started button



Intel® DevCloud for oneAPI

Explore Intel oneAPI Toolkits in the DevCloud

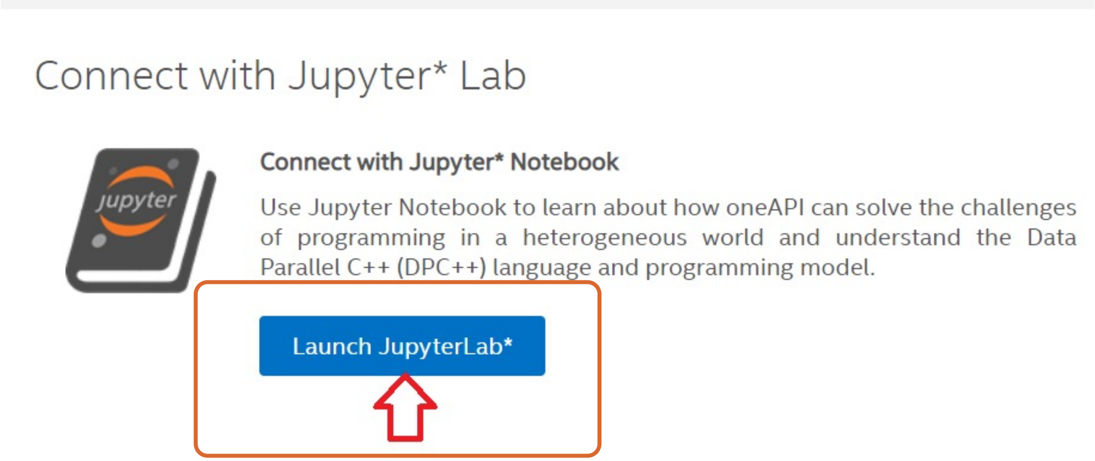
These toolkits are for performance-driven applications—HPC, IoT, advanced rendering, deep lear toolkit to see what it includes, explore training modules, and go deeper with developer guides.

Intel® oneAPI Base Toolkit

Build and deploy high-performance, data-centric applications across divers

The toolkit includes:

- Step 4: Scroll Down to the bottom of the page and click on Launch JupyterLab



Connect with Jupyter* Lab

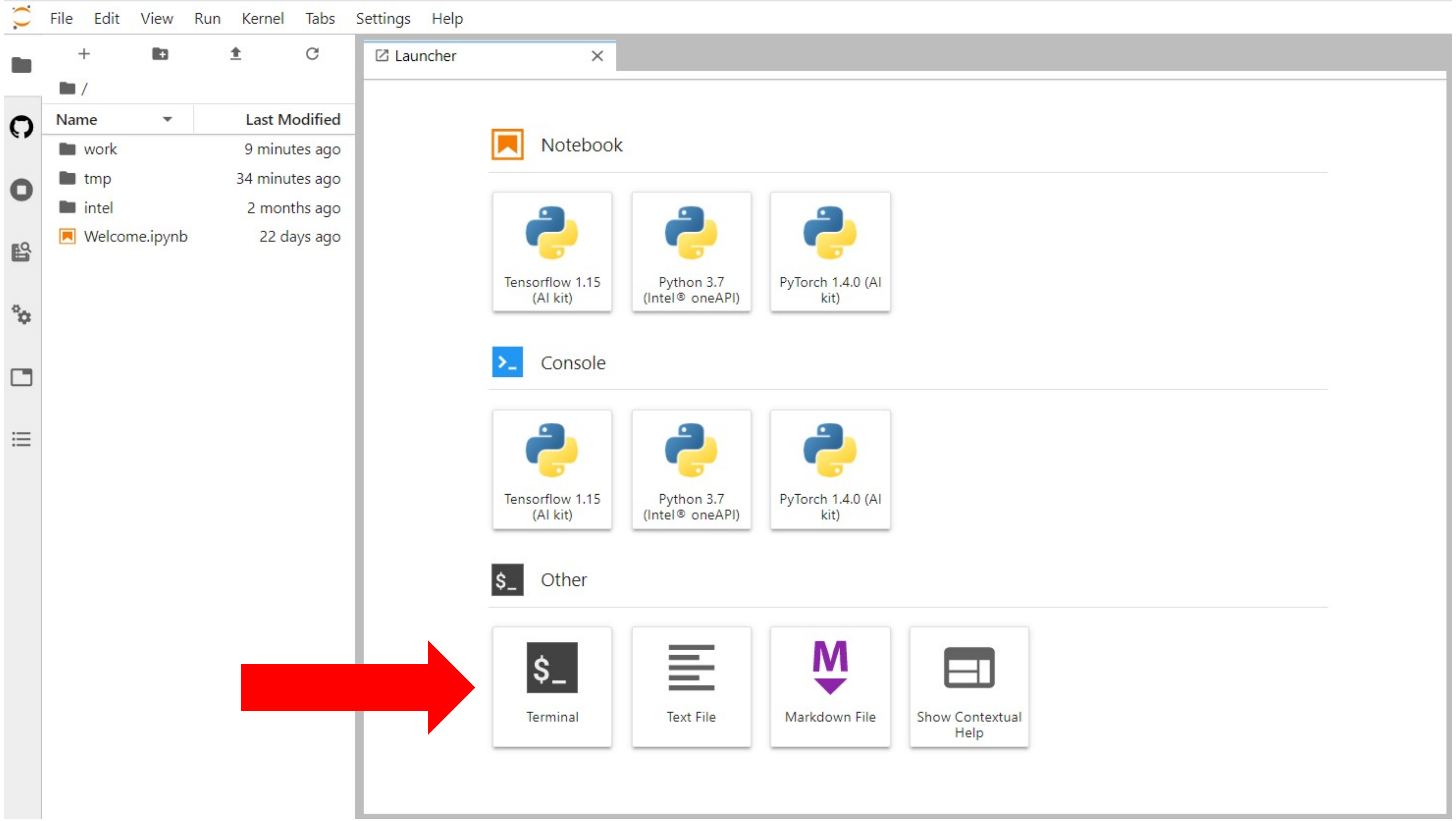
Connect with Jupyter* Notebook

Use Jupyter Notebook to learn about how oneAPI can solve the challenges of programming in a heterogeneous world and understand the Data Parallel C++ (DPC++) language and programming model.

Optimization Notice

Setup Intel® DevCloud and Jupyter Environment

Launch Jupyter and select Terminal



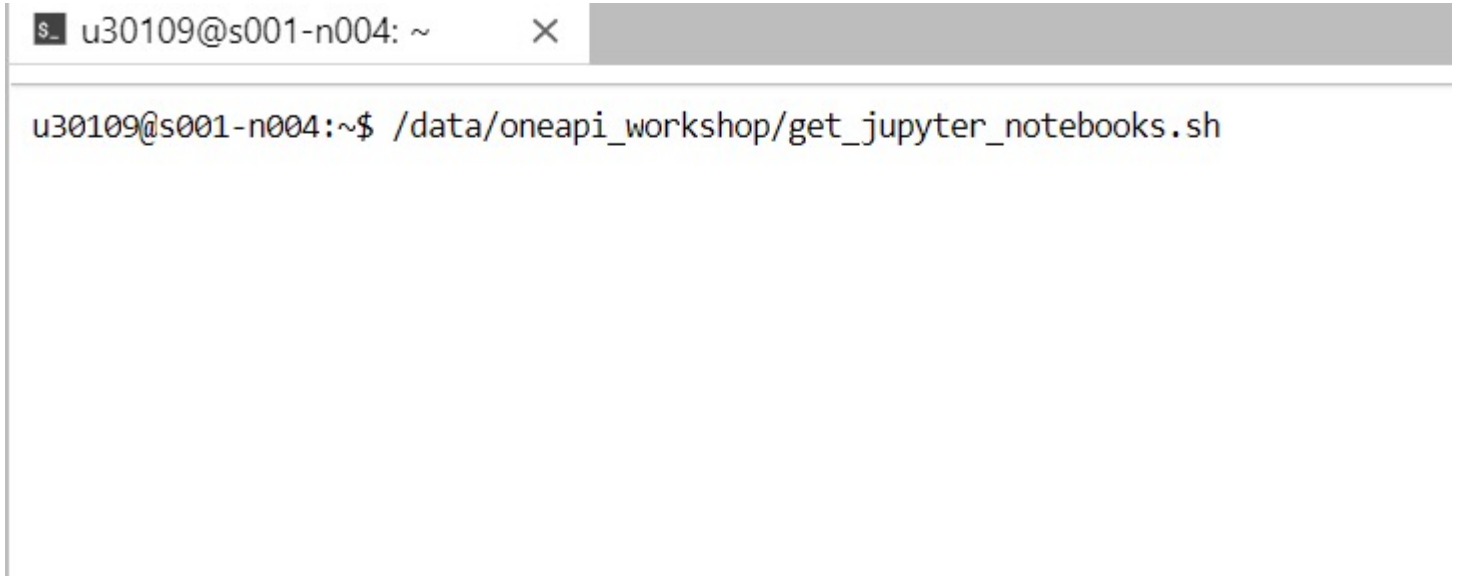
Optimization Notice

Commands to input in terminal

Please execute the following commands in the Jupyter Terminal window

/data/oneapi_workshop/get_jupyter_notebooks.sh

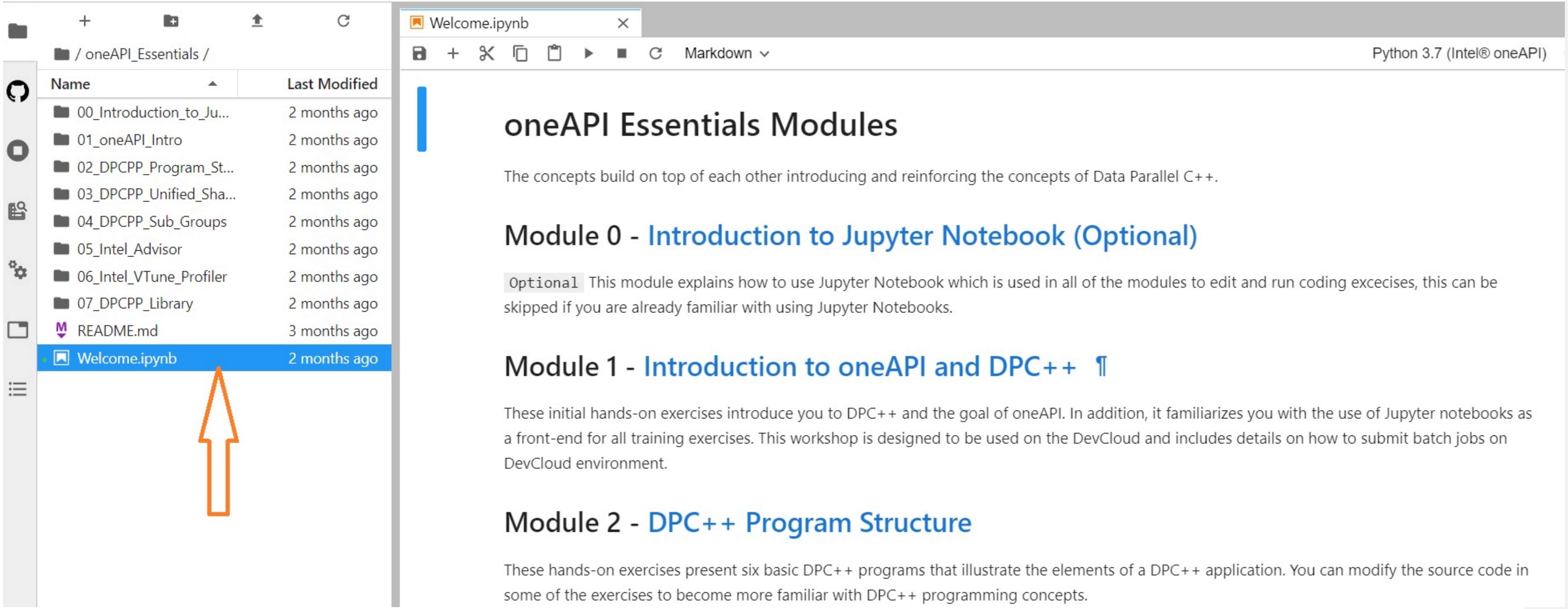
This command copies workshop into the user directory



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

Optimization Notice

Select Welcome.ipynb



/ oneAPI_Essentials /

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 excecises, this can be skipped if you are already familiar with using Jupyter Notebooks.

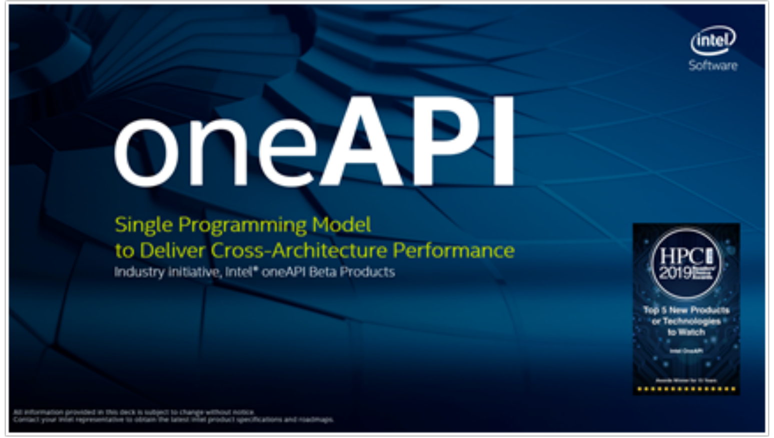
Module 1 - Introduction to oneAPI and DPC++ ¶

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.

DPC++essentials Course

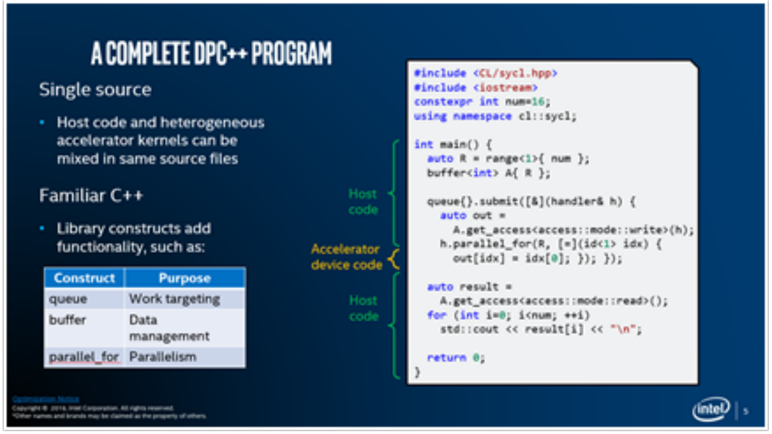


oneAPI

Single Programming Model
to Deliver Cross-Architecture Performance

Industry initiative, Intel® oneAPI Beta Products

All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.



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

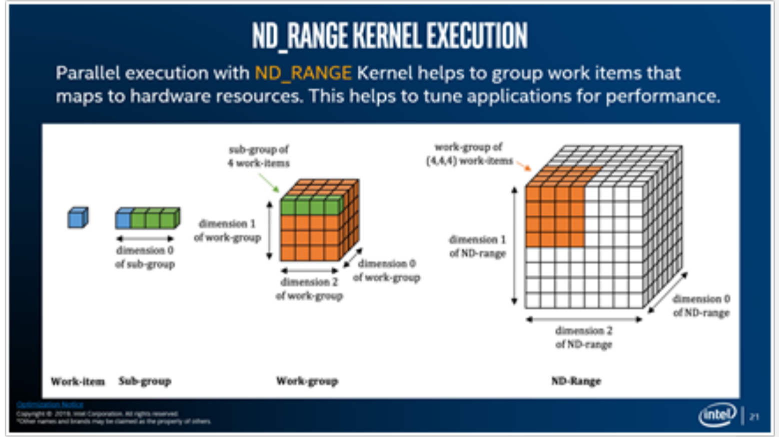
```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
  auto R = range<1>{ 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;
}
```



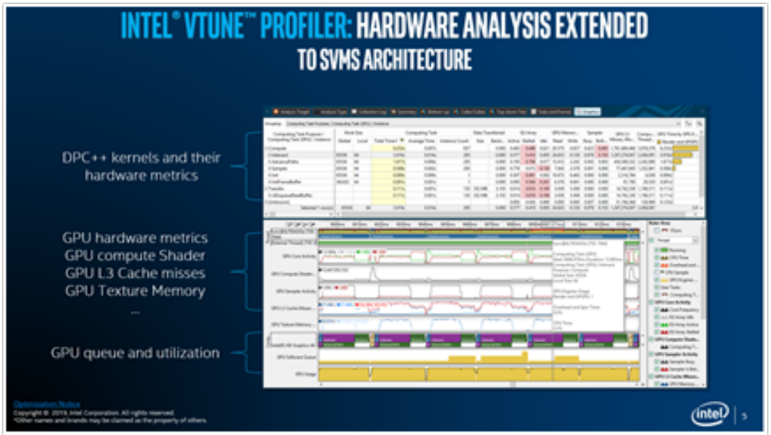
ND_RANGE KERNEL EXECUTION

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

sub-group of 4 work-items
work-group of (4,4,4) work-items
dimension 0 of sub-group
dimension 1 of work-group
dimension 2 of work-group
dimension 0 of work-group
dimension 1 of ND-range
dimension 2 of ND-range
dimension 0 of ND-range
Work-item
Sub-group
Work-group
ND-Range

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)

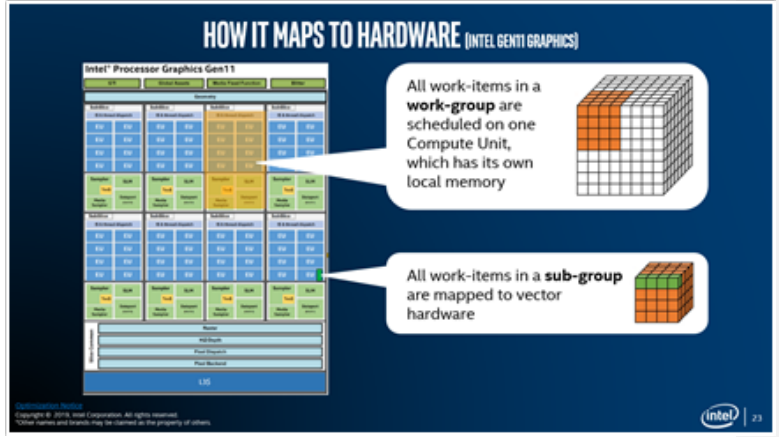


INTEL® VTUNE™ PROFILER: HARDWARE ANALYSIS EXTENDED TO SVMS ARCHITECTURE

DPC++ kernels and their hardware metrics

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

GPU queue and utilization



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

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

Optimization Notice

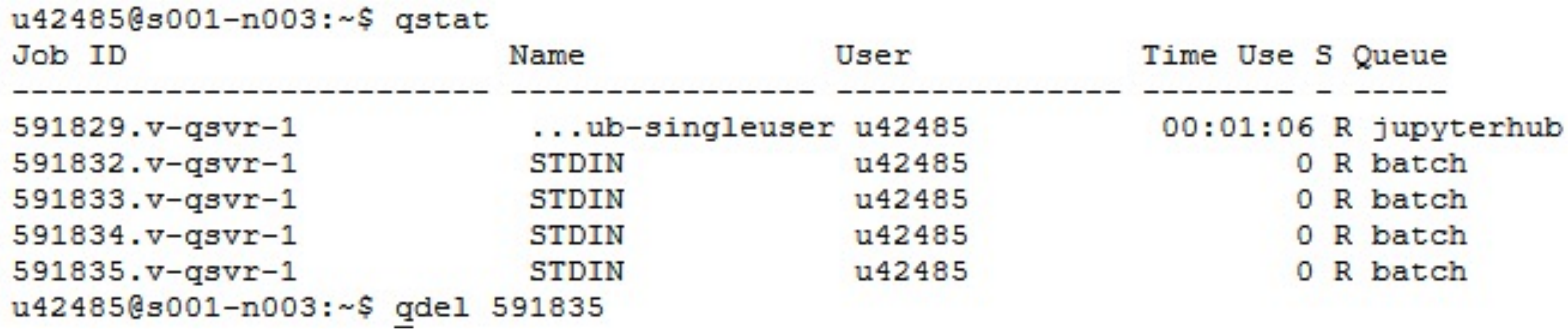
Qsub

- 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 Coding on Intel DevCloud

Run Simple DPC++ Program

Data Parallel C++

Standards-based, Cross-architecture Language
DPC++ = ISO C++ and Khronos SYCL

Parallelism, productivity and performance for CPUs and Accelerators

- Delivers accelerated computing by exposing hardware features
- Allows code reuse across hardware targets, while permitting custom tuning for specific accelerators
- Provides an open, cross-industry solution to single architecture proprietary lock-in

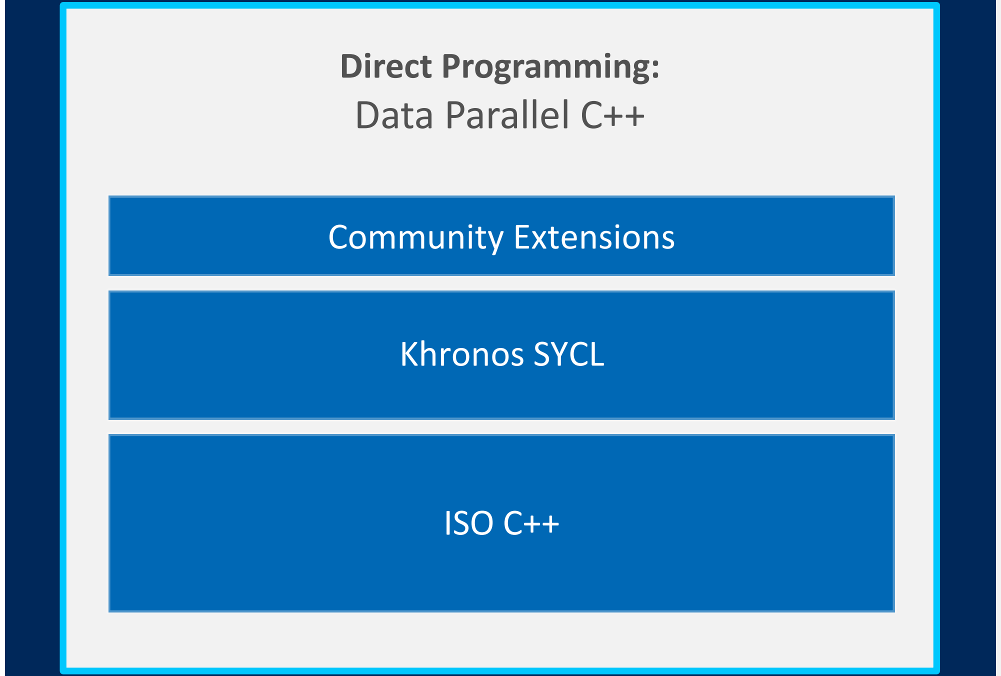
Based on C++ and SYCL

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

Community Project to drive language enhancements

- Provides extensions to simplify data parallel programming
- Continues evolution through open and cooperative development

Apply your skills to the next innovation, not rewriting software for the next hardware platform



The open source and Intel DPC++/C++ compiler supports Intel CPUs, GPUs, and FPGAs.
Codeplay announced a DPC++ compiler that targets Nvidia GPUs.

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* standard

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

Optimization Notice

A Complete DPC++ Program

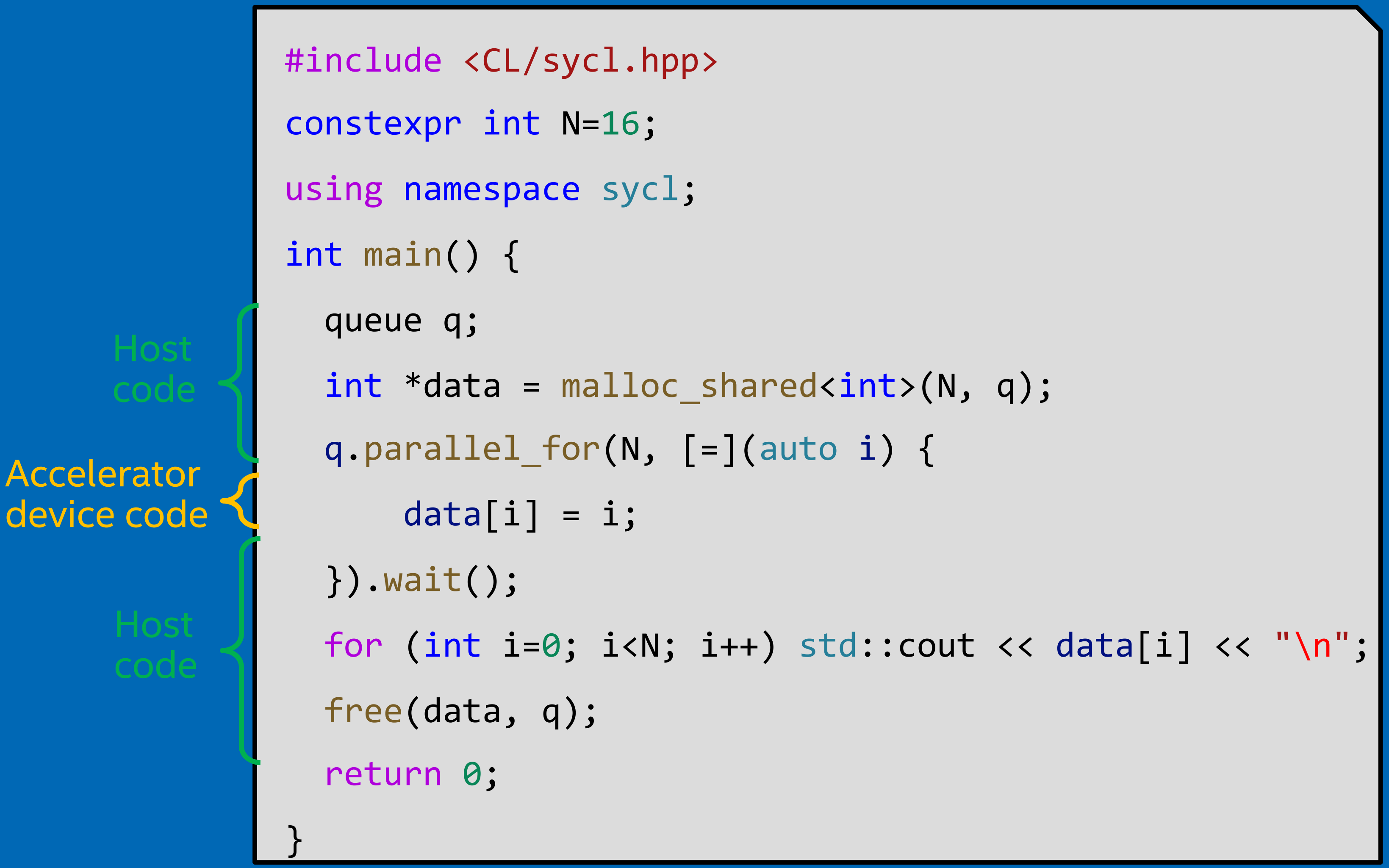
Single source

- 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(N, [=](auto i) {
      data[i] = i;
  }).wait();
  for (int i=0; i<N; i++) std::cout << data[i] << "\n";
  free(data, q);
  return 0;
}
```

DPC++ Program Structure

- Agenda
  - Deciding where code is run
  - Data transfers and synchronization
  - DPC++ execution model and memory model
- Hands On
  - Complex Multiplication

Optimization Notice

Buffer Memory Model

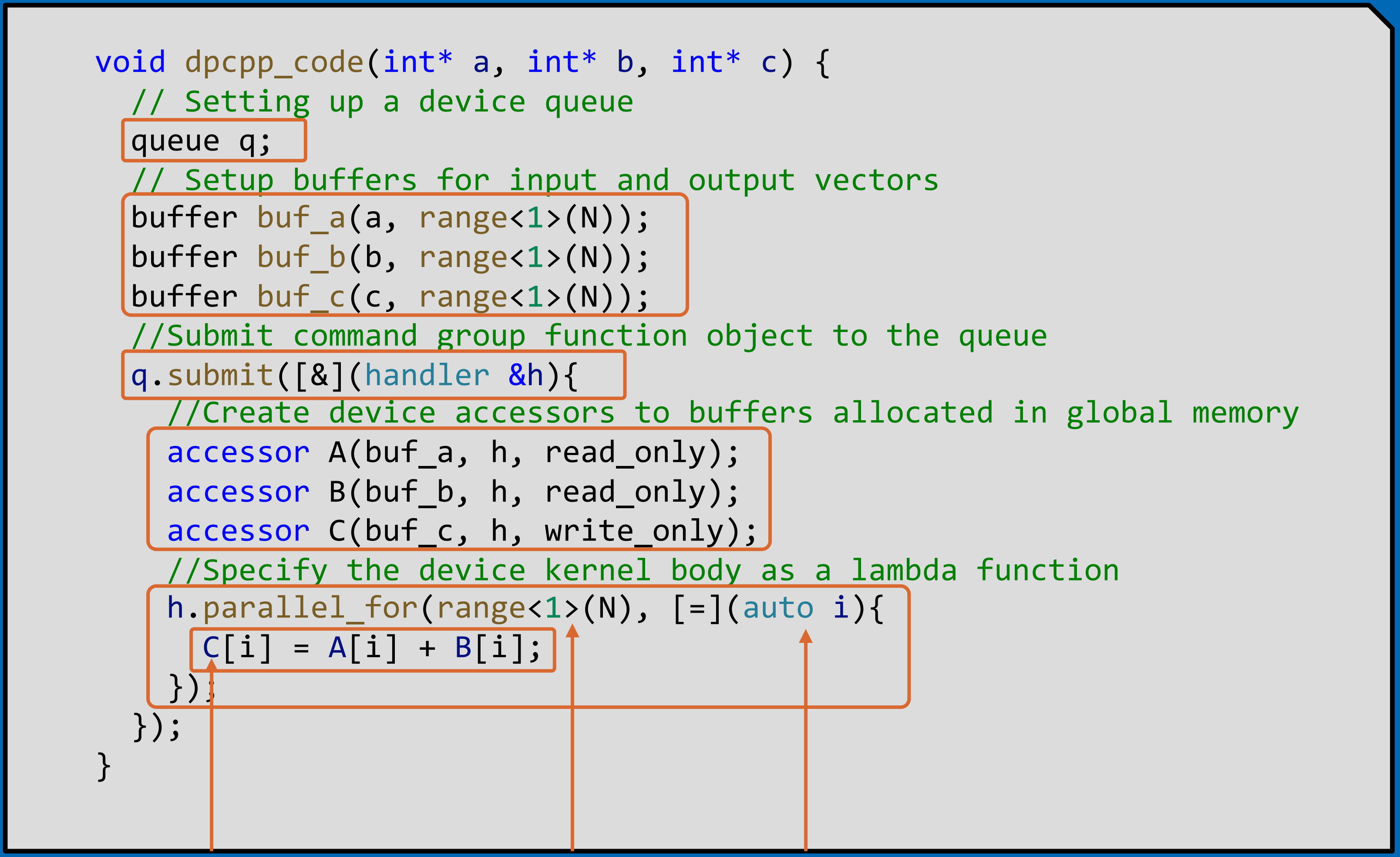
Buffers encapsulate data shared between host and device.

Accessors provide access to data stored in buffers and create data dependences in the graph.

Unified Shared Memory (USM) provides an alternative pointer-based mechanism for managing memory;

```
queue q;
std::vector<int> v(N, 10);
{
  buffer buf(v);
  q.submit([&](handler& h) {
    accessor a(buf, h , write_only);
    h.parallel_for(N, [=](auto i) { a[i] = i; });
  });
}
for (int i = 0; i < N; i++) std::cout << v[i] << " ";
```

DPC++ Code Anatomy



```
void dpcpp_code(int* a, int* b, int* c) {
  // Setting up a 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 group for (asynchronous) execution

Step 4: create accessors describing how buffer is used on the device

Step 5: specify kernel function and launch parameters (e.g. group size)

Step 6: specify code to run on the device

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

Submitting to a Device

- A device represents a specific accelerator in the system.
- Work is not submitted to devices directly, but to a queue associated with the device.
- Creating a queue for a specific device requires a device_selector.

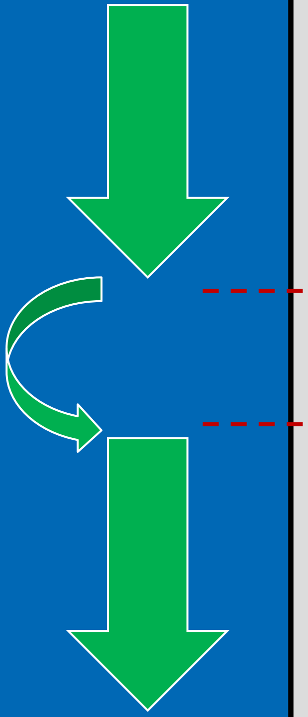
```
default_selector selector;
// host_selector selector;
// cpu_selector selector;
// gpu_selector selector;
queue q(selector);
std::cout << "Device: " << q.get_device().get_info<info::device::name>() << std::endl;
```

Optimization Notice

Asynchronous Execution

Host

Host code execution

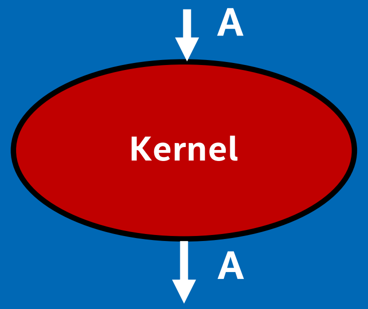


Enqueues kernel to graph, and keeps going

```
#include <CL/sycl.hpp>
constexpr int N=16;
using namespace sycl;
int main() {
  std::vector<int> data(N);
  {
    buffer A(data);
    queue q;
    q.submit([&](handler& h) {
      accessor out(A, h, write_only);
      h.parallel_for(N, [=](auto i) {
        out[i] = i;
      });
    });
  }
  for (int i=0; i<N; ++i) std::cout << data[i];
}
```

Graph

Graph executes asynchronously to host program



Asynchronous Execution

```
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> i) {
      out[i] = i; }); });

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

  q.submit([&](handler& h) {
    accessor out(B, h, write_only);
    h.parallel_for(R, [=](id<1> i) {
      out[i] = i; }); });

  q.submit([&](handler& h) {
    accessor in(A, h, read_only);
    accessor inout(B, h);
    h.parallel_for(R, [=](id<1> i) {
      inout[i] *= in[i]; }); });
}
```

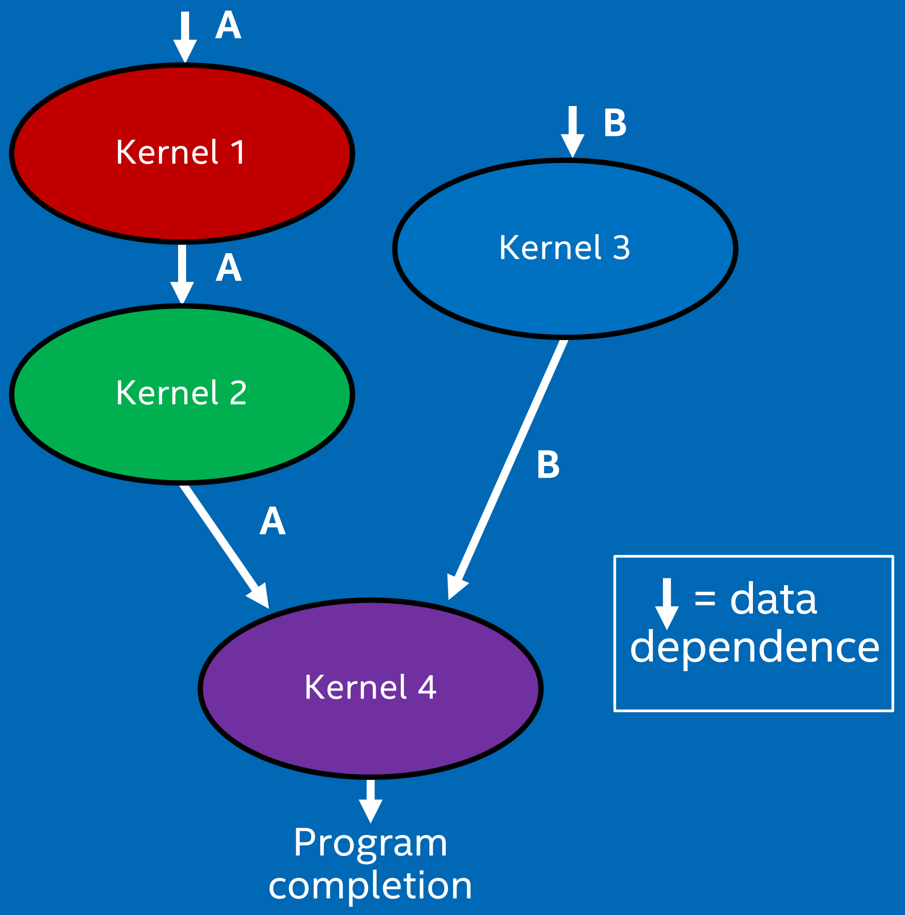
} Kernel 1

} Kernel 2

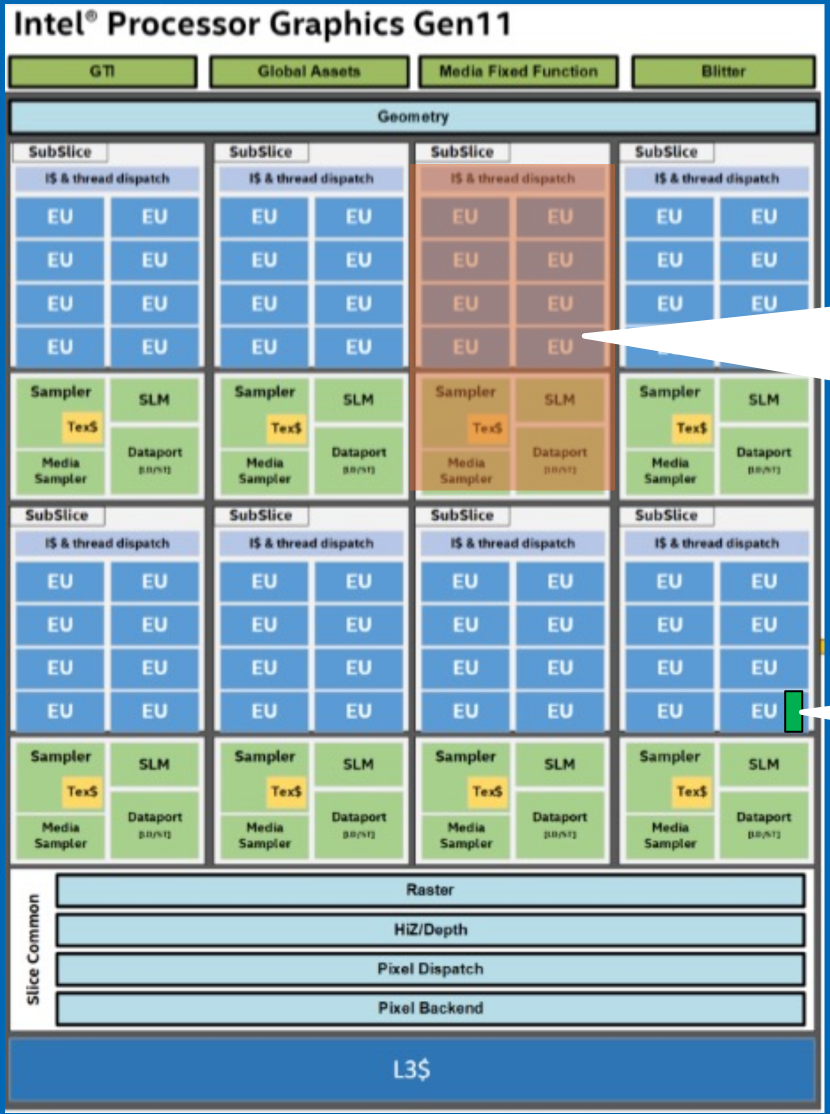
} Kernel 3

} Kernel 4

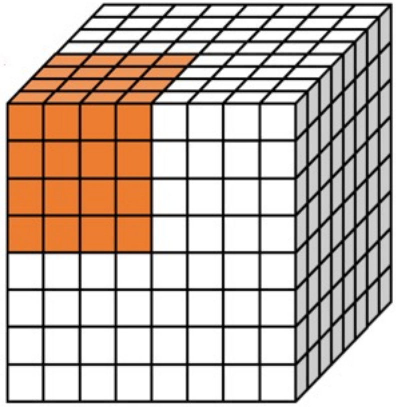
Data and control dependences are resolved by the runtime



Mapping to Hardware (INTEL GEN11 GRAPHICS)

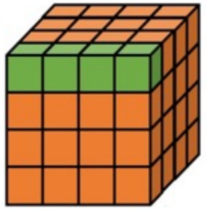


All work-items in a **work-group** are scheduled on one subslice, which has its own local memory.



All work-items in a **sub-group** execute on a single EU thread.

Each work-item in a **sub-group** is mapped to a SIMD lane/channel.



Recap: Important Classes in DPC++

Class	Functionality
`sycl::device`	Represents a specific CPU, GPU, FPGA or other device that can execute SYCL kernels.
`sycl::queue`	Represents a queue to which kernels can be submitted (enqueued). Multiple queues may map to the same `sycl::device`.
`sycl::buffer`	Encapsulates an allocation that the runtime can transfer between host and device.
`sycl::handler`	Used to define a command-group scope that connects buffers to kernels.
`sycl::accessor`	Used to define the access requirements of specific kernels (e.g. read, write, read-write).

`sycl::range, sycl::nd_range`
`sycl::id, sycl::item,`
`sycl::nd_item`
	Representations of execution ranges and individual execution agents in the range.

Optimization Notice

Accessor Modes

Access Mode	**Description**
read_only	Read only Access
write_only	Write-only access. Previous contents not discarded
read_write	Read and Write access

Optimization Notice

Buffer Creation

Buffer Class: Template class with three arguments

- Type of the Object
- Dimensionality of the Buffer
- Optional C++ Allocator

The choice of buffer creation depends on how the buffer needs to be used as well as programmer's coding preferences

Buffer Creation

Lets look at a simple DPC++ code example and see different ways of buffer creation

Buffer for Vectors

Buffer for std::array

Buffer from a host pointer

```
{
  // Create a buffer of ints from an input iterator
  std::vector<int> myVec;
  buffer b1{myVec};
  buffer b2{myVec.begin(), myVec.end()};

  // Create a buffer of ints from std::array
  std::array<int,42> my_data;
  buffer b3{my_data};

  // Create a buffer of 4 doubles and initialize it from a host pointer
  double myDoubles[4] = {1.1, 2.2, 3.3, 4.4};
  buffer b4{myDoubles, range{4}};
}
```

Buffer: use_host_ptr

Use_host_ptr requires the buffer to not allocate any memory on the host

Buffer should use the memory pointed to by a host pointer that is passed to the constructor.

This option can be useful when the program wants full control over all host memory allocations

```
int main() {
    queue q;
    int myInts[42];
    // create a buffer of 42 ints, initialize
    //with a host pointer,
    // and add the use_host_pointer property
    buffer b1(myInts, range(42), property::use_host_ptr{});
}
```

Buffer Properties: use_host_ptr

This property requires the buffer to not allocate any memory on the host, Instead, the buffer should use the memory pointed to by a host pointer that is passed to the constructor.

Initialize vector a and b

Use property::use_host_ptr()

Submit the work

```
queue q;
std::vector<float> a(N, 10.0f);
std::vector<float> b(N, 20.0f);
{
    buffer buf_a(a,{property::buffer::use_host_ptr()});
    buffer buf_b(b,{property::buffer::use_host_ptr()});
    q.submit([&](handler& h) {
        //create Accessors for a and b
        accessor A(buf_a,h);
        accessor B(buf_b,h,read_only);
        h.parallel_for(R, [=](auto i) { A[i] += B[1] ; });
      });
}
```

Buffer: set_final_data

The `set_final_data` method of a buffer is the way to update host memory however the buffer was created.

When the buffer is destroyed, data will be written to the host using the supplied location.

Call the set_final_data to the created shared ptr where the values will be written back when the buffer gets destructed

```
{
    queue q;
    buffer my_buffer(my_data);
    my_buffer.set_final_data(nullptr);
    q.submit([&](handler &h) {
    accessor my_accessor(my_buffer, h);
           h.parallel_for(N, [=](id<1> i) {
            my_accessor[i]*=2;
          });
      });
}
```

Buffer: sub_buffers

A sub-buffer requires three things, a reference to a parent buffer, a base index, and the range of the sub-buffer.

The main advantage of using the sub-buffers is different kernels can operate on different sub buffers concurrently.

Sub Buffer for one dimensional buffer

Sub buffer for a 2-dimensional buffer

```
buffer B(data, range(N));

buffer<int> B1(B, 0, range{ N / 2 });

buffer<int> B2(B, 32, range{ N / 2 });
```

```
buffer<int, 2> b10{range{2, 5}};

buffer b11{b10, id{0, 0}, range{1, 5}};

buffer b12{b10, id{1, 0}, range{1, 5}};
```

Sub Buffers

Buffer for Vectors

Create sub buffers B1 and B2

Submit q1 using B1

Submit q2 using B2

Create Host accessors

```
int main() {
    const int N = 64;    const int num1 = 2;    const int num2 = 3;
    int data[N];
    for (int i = 0; i < N; i++) data[i] = i;  for (int i = 0; i < N; i++) std::cout << data[i] << " ";
    buffer B(data, range(N));
    buffer<int> B1(B, 0, range{ N / 2 });
    buffer<int> B2(B, 32, range{ N / 2 });
    queue q1;
    q1.submit([&](handler& h) {
        accessor a1(B1, h);
        h.parallel_for(N/2, [=](auto i) { a1[i] *= num1; });
    });
    queue q2;
    q2.submit([&](handler& h) {
        accessor a2(B2, h);
        h.parallel_for(N/2, [=](auto i) { a2[i] *= num2; });
    });
    host_accessor b1(B1, read_only);
    host_accessor b2(B2, read_only);
    return 0;
}
```

Synchronization – Host Accessors

```
#include <CL/sycl.hpp>
using namespace sycl;
constexpr int N = 16;

int main() {
  std::vector<double> v(N, 10);
  queue q;

  buffer buf(v);
  q.submit([&](handler& h) {
    accessor a(buf, h)
    h.parallel_for(N, [=](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 kernels that modify the same buffer in any queue completes execution and the data is available to the host via this host accessor.

Optimization Notice

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("Intel") != 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;
}
```

Hands-On: 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

Optimization Notice

Hands-on Coding on Intel DevCloud

Complex Multiplication with DPC++

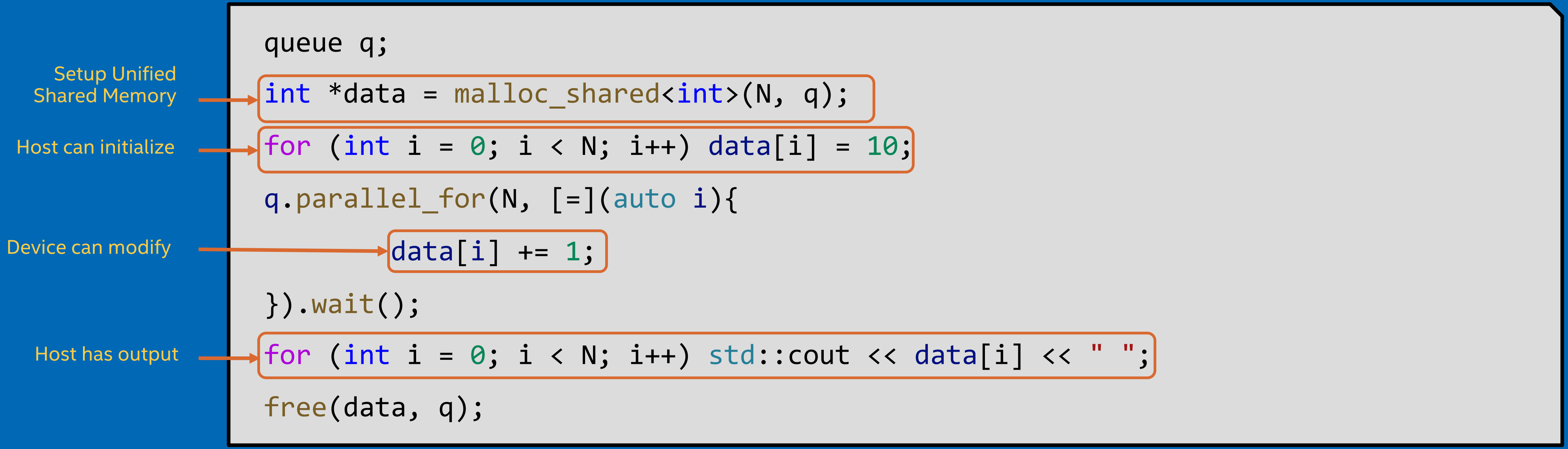
New Features in DPC++

- Agenda
  - Unified Shared Memory (USM)
  - Sub-Groups
  - Reductions
- Hands On
  - USM
  - Sub-group collectives and shuffles
  - Reduction kernels

Optimization Notice

Unified Shared Memory (USM)

USM enables allocations to be identified via pointers, and for the same pointers to be used across the host and device.



```
queue q;
int *data = malloc_shared<int>(N, q);
for (int i = 0; i < N; i++) data[i] = 10;
q.parallel_for(N, [=](auto i){
        data[i] += 1;
}).wait();
for (int i = 0; i < N; i++) std::cout << data[i] << " ";
free(data, q);
```

Unified Shared Memory (USM)

There are three ways to create USM allocations:

Type	Description	Accessible on Host?	Accessible on Device?
`sycl::malloc_device`	Allocations in device memory. Programmer must explicitly transfer data between host and device.	No	Yes
`sycl::malloc_host`	Allocations in host memory. Kernels can access these allocations directly.	Yes	Yes
`sycl::malloc_shared`	Allocations can migrate between host and device memory. Different implementations may provide different guarantees regarding whether allocations can be accessed by host and device concurrently.	Yes	Yes

Optimization Notice

USM – Explicit Data Transfer

`malloc_device()` allocates memory on device; host cannot access directly

Copy memory explicitly from host to device using `q.memcpy()`

Device kernel can use the same (device) pointer

Copy memory explicitly from device to host using `q.memcpy()`

```
queue q(property::queue::in_order{});


int data[N];
for (int i = 0; i < N; i++) data[i] = 10;


int *data_device = malloc_device<int>(N, q);


q.memcpy(data_device, data, sizeof(int) * N);


q.parallel_for(N, [=](auto i) { data_device[i] += 1; });


q.memcpy(data, data_device, sizeof(int) * N).wait();


for (int i = 0; i < N; i++) std::cout << data[i] << std::endl;
free(data_device, q);
```

Optimization Notice

USM – Implicit Data Transfer

`malloc_shared()` allocates memory that can migrate between host and device.

Device kernel can use the same pointer

Host can directly access memory via the same pointer.

```
queue q;

int *data = malloc_shared<int>(N, q);
for (int i = 0; i < N; i++) data[i] = 10;

q.parallel_for(N, [=](auto i) { data[i] += 1; }).wait();

for (int i = 0; i < N; i++) std::cout << data[i] << std::endl;
free(data, q);
```

Optimization Notice

USM – Data Dependencies

- When using buffers, data dependencies between kernels are tracked by the SYCL runtime based on accessor usage.

- When using unified shared memory, data dependencies must be handled by the programmer:
  - Explicit host/device synchronization via `q.wait()` before accessing data
  - Use `sycl::event` objects to specify dependencies between kernels
    OR
    Use in-order queues to add implicit dependencies between kernels

Optimization Notice

Hands-on Coding on Intel DevCloud

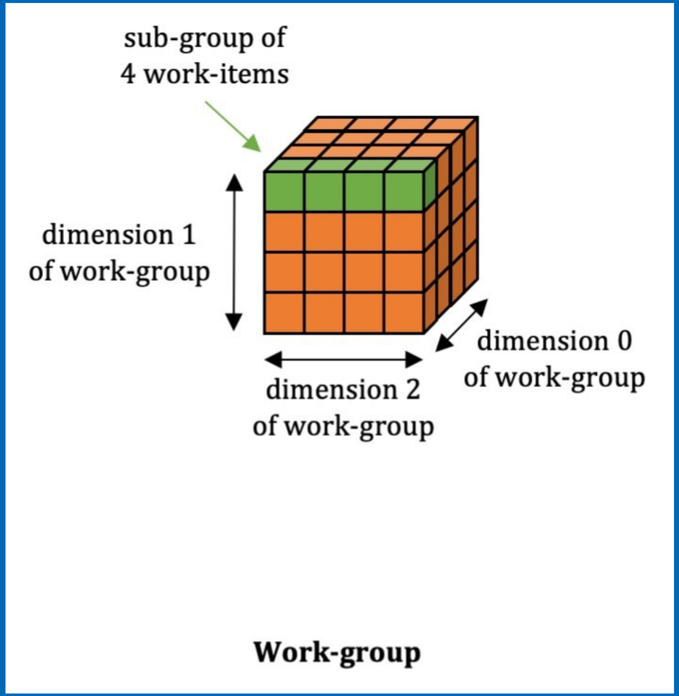
USM Implicit and Explicit Data Movement

Sub-Groups

A subset of work-items within a work-group that execute with additional guarantees and often map to SIMD hardware.

Work-items in a sub-group can communicate directly using shuffle operations.

Sub-groups also provide access to sub-group collectives (e.g. reduction, scan, any/all)



Sub-Groups

sub_group class

A sub-group handle can be obtained from an **nd_item** using get_sub_group()

It exposes functions to:

- Query more information about the sub-group
- Perform shuffle operations or use collective functions.

```
q.parallel_for(nd_range<1>(N,B), [=](nd_item<1> item) {
  auto sg = item.get_sub_group();
  // KERNEL CODE
});
```

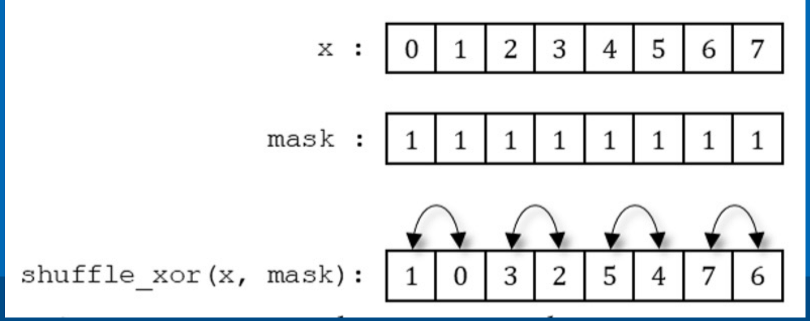
Sub-Groups

Sub-Group Shuffles

- One of the most useful features of sub-groups is the ability to communicate directly between individual work-items without explicit memory operations.

```
h.parallel_for(nd_range<1>(N,B), [=](nd_item<1> item){
        auto sg = item.get_sub_group();
        size_t i = item.get_global_id(0);

        /* Shuffles */
        //data[i] = sg.shuffle(data[i], 2);
        //data[i] = sg.shuffle_up(0, data[i], 1);
        //data[i] = sg.shuffle_down(data[i], 0, 1);
        data[i] = sg.shuffle_xor(data[i], 1);
});
```



x :	0	1	2	3	4	5	6	7
mask :	1	1	1	1	1	1	1	1
shuffle_xor(x, mask):	1	0	3	2	5	4	7	6

Optimization Notice

Sub-Groups

Group Collectives

- The collective functions provide implementations of closely-related common parallel patterns.

- Collectives are available for both work-groups and sub-groups.

```
h.parallel_for(nd_range<1>(N,B), [=](nd_item<1> item){
      auto sg = item.get_sub_group();
      size_t i = item.get_global_id(0);

      /* Collectives */
      data[i] = reduce(sg, data[i], ONEAPI::plus<>());
      //data[i] = reduce(sg, data[i], ONEAPI::maximum<>());
      //data[i] = reduce(sg, data[i], ONEAPI::minimum<>());
});
```

Specifying the Sub-Group Size

The sub-group size can be configured separately for each kernel.
The set of available sub-group sizes is hardware-specific.

```
q.parallel_for(range<1>(N),
               [=](id<1> id) [[intel::reqd_sub_group_size(16)]] {
  // KERNEL CODE
});
```

The sub-group size can be tuned even for kernels that do not use the sub_group class (e.g. to tune for SIMD width and register usage).

Optimization Notice

Hands-on Coding on Intel DevCloud

Sub-Group Shuffles and Collectives

Reductions in a Group

Work-group collectives can be used to compute the sum of all items in each work-group

```
q.parallel_for(nd_range<1>(N, B), [=](nd_item<1> item){
  auto wg = item.get_group();
  size_t i = item.get_global_id(0);

  // Adds all elements in work_group using work_group reduce
  int sum = reduce(wg, data[i], ONEAPI::plus<>());

  // Do something with the reduced value
  ...
});
```

Optimization Notice

Reductions Across Groups (aka Reduction Kernels)

Work-group collectives can be used to compute the sum of all items in each work-group

Partial results can be combined via additional kernel(s)

```
q.parallel_for(nd_range<1>(N, B), [=](nd_item<1> item){
  auto wg = item.get_group();
  size_t i = item.get_global_id(0);

  // Adds all elements in work_group using work_group reduce
  int sum_wg = reduce(wg, data[i], ONEAPI::plus<>());

  // Write work_group sum to first location for each work_group
  if (item.get_local_id(0) == 0) data[i] = sum_wg;
});
```

```
q.single_task([=](){
  int sum = 0;
  for (int i = 0; i < N; i += B) {
      sum += data[i];
  }
  data[0] = sum;
});
```

Optimization Notice

Reductions Across Groups (aka Reduction Kernels)

DPC++ introduces a dedicated abstraction for reduction kernels.

A `reduction` object encapsulates:

1. The reduction variable
2. An optional identity
3. The reduction operator

```
queue q;
int *data = malloc_shared<int>(N, q);
for (int i = 0; i < N; i++) data[i] = i;

int *sum = malloc_shared<int>(1, q);
sum[0] = 0;

q.parallel_for(nd_range<1>{N, B},
                  ONEAPI::reduction(sum, ONEAPI::plus<>()),
                  [=](nd_item<1> it, auto& sum) {
     int i = it.get_global_id(0);
     sum += data[i];
}).wait();

std::cout << "Sum = " << sum[0] << std::endl;
```

Optimization Notice

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
- DPC++ New Features (USM, Sub-Groups, Reductions)
- Hands on activities
  - Introduction to DPC++ - Complex-multiplication
  - USM, Sub-Groups and Reductions)

NOTICES &

- 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.

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®