Heterogenous Programming with oneAPI and DPC++(SYCL)

oneAPI – 가속 컴퓨팅을 개발하기 위한 스마트한 방식

MOASYS

2023. 07. 14.

oneAPI 스마트한 방식 시리즈 (2023)

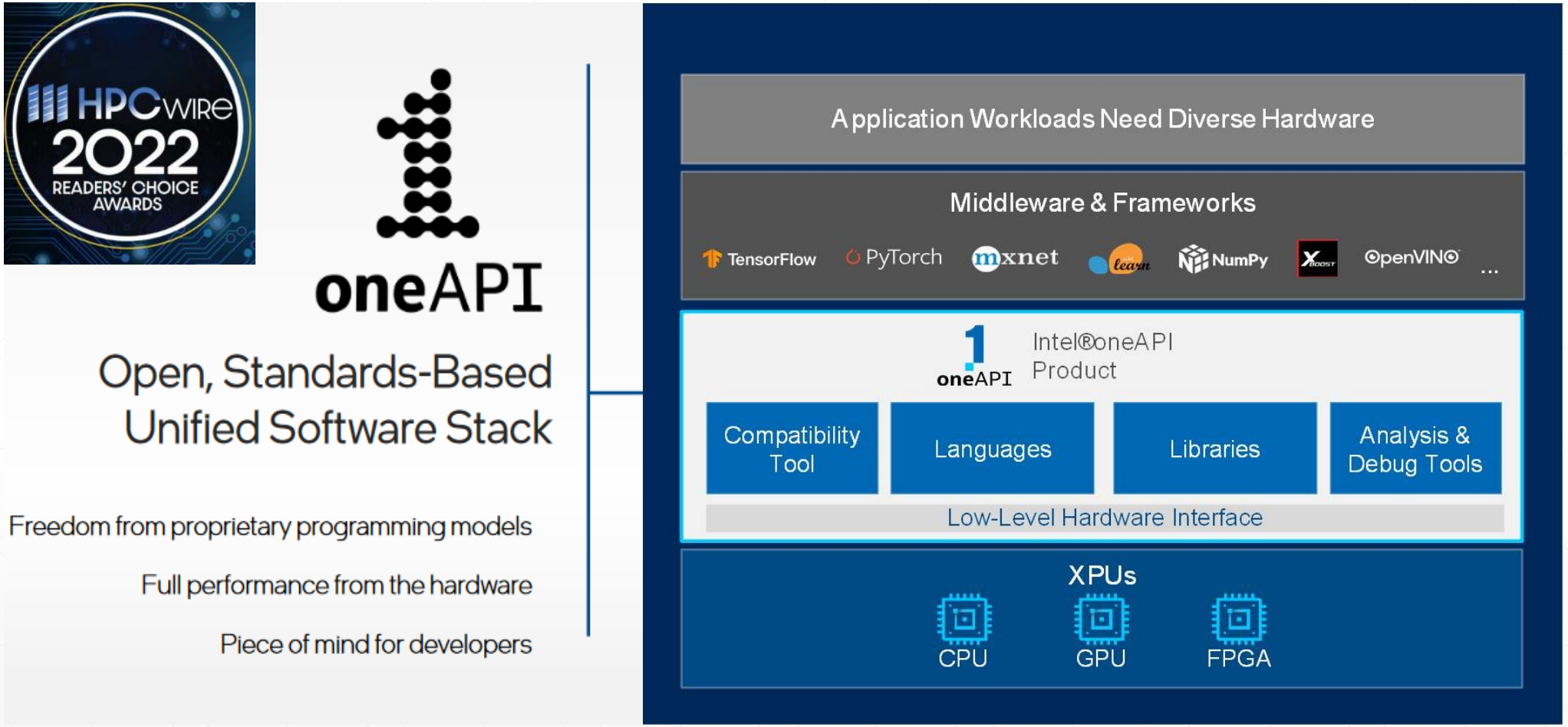
1. **Introducing Intel oneAPI 2023**
   - https://www.allshowtv.com/detail.html?idx=1259

2. **Heterogenous Programming with oneAPI and DPC++(SYCL)**
   - https://www.allshowtv.com/detail.html?idx=1337

목차

- Overview of oneAPI
- DPC++ Basics
- New features of oneAPI 2023
- Hand-on

아키텍처 간 가속 프로그래밍을 위한 스마트한 방식



- CPU, GPU 및 FPGA와 같은 다양한 가속기 (XPU) 지원
- 고성능 컴퓨팅 및 기계 학습을 위한 사양을 지속적으로 발전킴

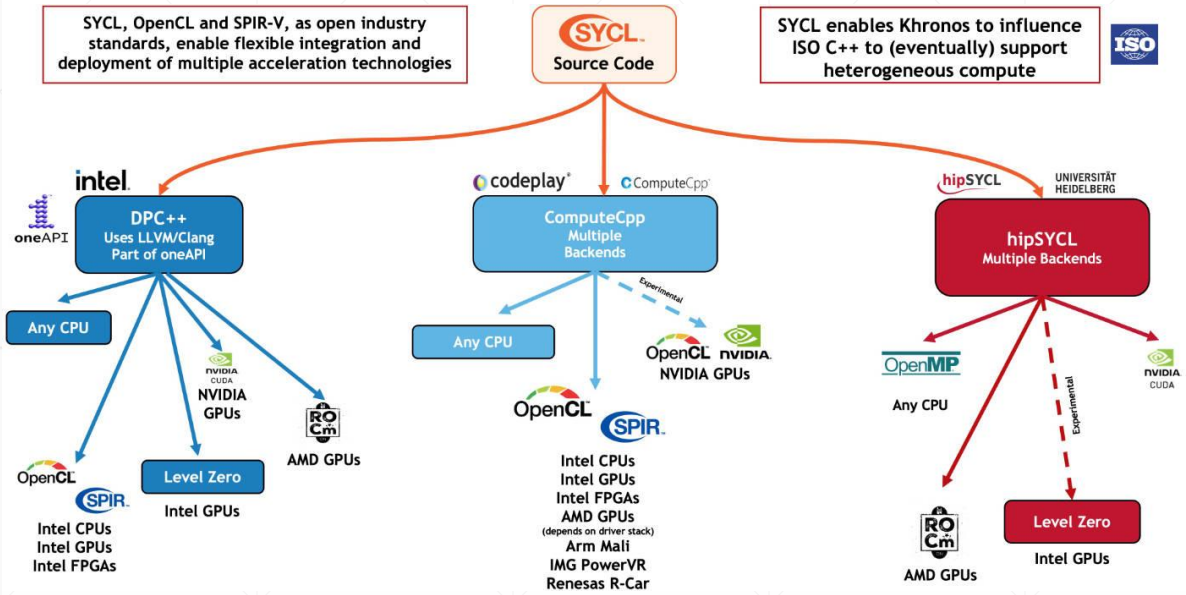
이기종 프로그래밍 모델 지원

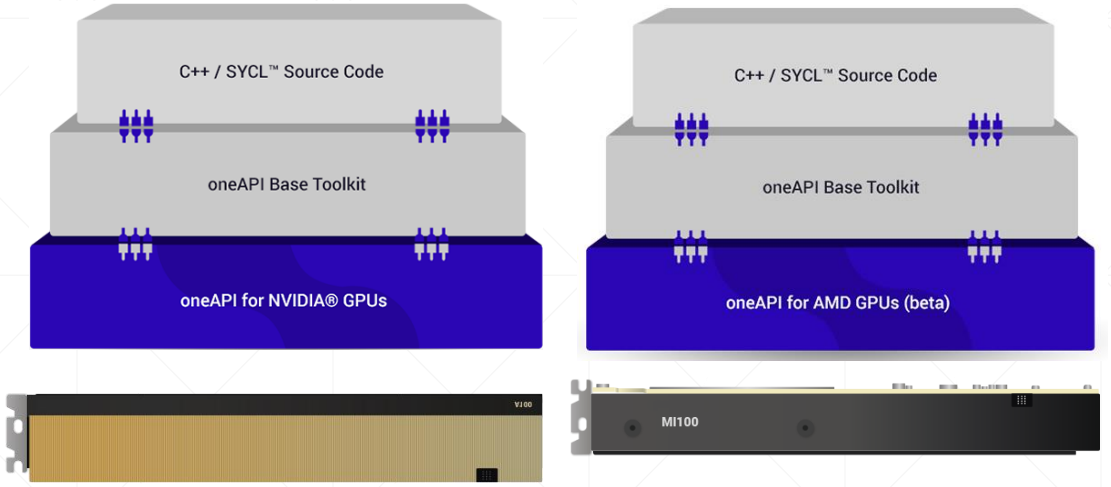
	CUDA	HIP	OpenACC	OpenMP	DPC++/SYCL
Languages	C/C++/Fortran	C/C++/Fortran	C/C++/Fortran	C/C++/Fortran	C++
Abstraction	Low	Low	High	High	Medium
Coding	-	-	Directive-based	Directive-based	C++ lambda
Parallelism	SIMT	SIMT	Fork-join SIMD	Fork-join SIMD	OpenCL
Offload	GPU (NVIDIA)	GPU (NVIDIA/AMD)	GPU (NVIDIA)	CPU/GPU (NVIDIA/AMD/Intel)	CPU/GPU/FPGA (NVIDIA/AMD/Intel)
Compiler	Proprietary	LLVM	PGI/CCE/GCC	PGI/CCE/GCC/LLVM/XL/Intel	LLVM
License	Proprietary	Open-source	Open-source	Open-source	Open-source

- **icc/icpc 와 ifort에서는**
  - GPU용 OpenMP*4.0/4.5의 오프로드 기능은 지원되지 않음
  - OpenMP*5.0/5.1/5.2 도 지원 지원되지 않음

- **dpcpp/icx/icpx/ifx는 새로운 LLVM 기반 컴파일러이며 Intel® oneAPI Toolkit에 포함**
  - GPU용 OpenMP 오프로드 기능은 지원 가능
  - OpenMP*5.0/5.1/5.2도 지원 가능
  - https://www.intel.com/content/www/us/en/developer/articles/technical/openmp-features-and-extensions-supported-in-icx.html

NVIDIA GPU 및 AMD GPU를 위한 컴파일러 플러그인





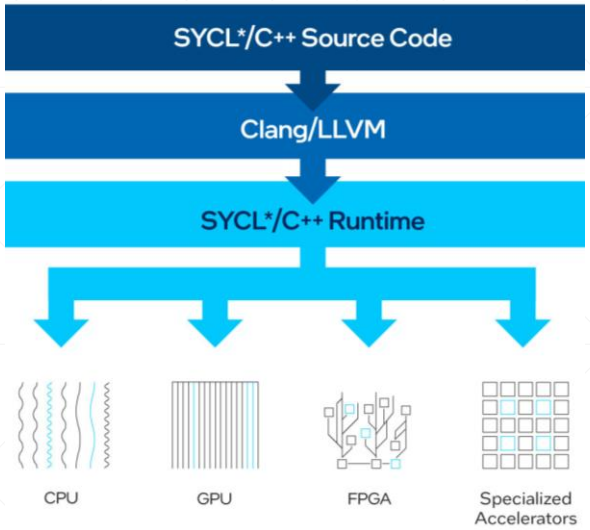
- **Intel DPC++ 컴파일러: oneAPI BASE Toolkit**
  - OpenCL 백엔드: Intel CPU, GPU(Gen9, 11, Xe) 및 FPGA(Stratix, Aria)에 최적화
  - Level Zero 백엔드: Intel GPU를 위한 low level 오프로드 API

- **Codeplay에서 Intel® oneAPI 2023 컴파일러의 최신 바이너리 플러그인을 무료로 다운로드 가능**
  - https://developer.codeplay.com/products/oneapi
  - NVIDIA GPU: A100-PCIe-40GB (sm_80)
  - AMD GPU (베타): AMD Radeon Pro W6600 (gfx1032)

DPC++: vector_add.cpp

```
#include <iostream>
#include <vector>
#include <string>

#include <sycl/sycl.hpp>

#if FPGA_HARDWARE || FPGA_EMULATOR
#include <sycl/ext/intel/fpga_extensions.hpp>
#endif

using namespace sycl;
```

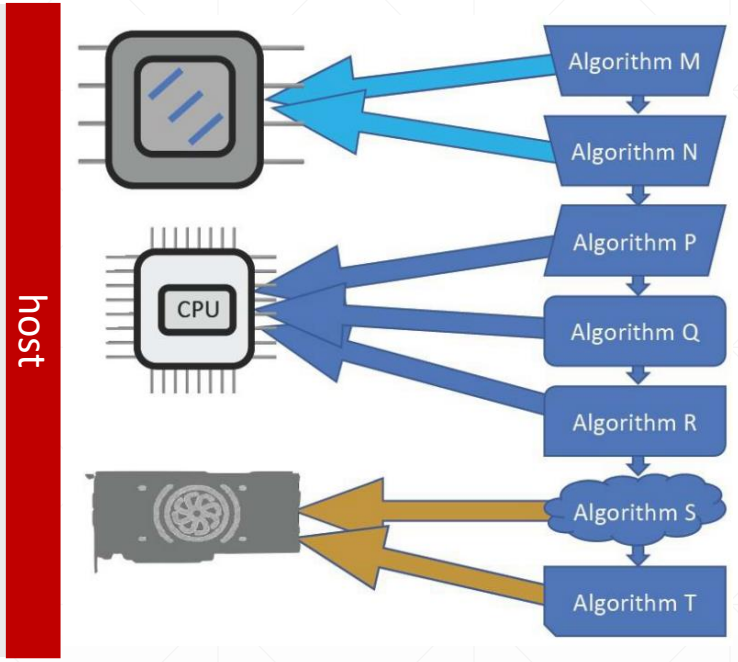
host



- **DPC++ = [C++] + [SYCL] + [확장성]**
  - C++의 생산성: 익숙한 C++ 구조를 사용하여 데이터 병렬 처리와 이기종 프로그래밍을 지원
  - SYCL 표준: OpenCL에 기반으로 이기종 시스템을 위한 프로그램 작성 모델
  - Intel의 확장성: SYCL 간소화, unified shared memory (USM), subgroup, reduction
- **헤더 파일**
  - sycl.hpp: Intel(R) DPC++ 컴파일러가 제공된 SYCL 표준의 헤더 파일
  - fpga_extensions.hpp: FPGA를 지원된 DPC++ 확장성 포함된 헤더 파일

DPC++: vector_add.cpp

```
int main(int argc, char* argv[]) {
#if FPGA_EMULATOR
  // FPGA emulator selector on systems without FPGA card.
  auto selector = sycl::ext::intel::fpga_emulator_selector_v;
#elif FPGA_HARDWARE
  // FPGA selector on systems with FPGA card.
  auto selector = sycl::ext::intel::fpga_selector_v;
#else
  // The default device selects the most performant device.
  auto selector = default_selector_v;
#endif

  // Queue creation
  queue q(selector);
```



- **오프로드 커널용 액셀러레이터를 선택은 간단하다**
  - DPC++는 실행 환경에서 이용 가능한 하드웨어를 검출하여 선택기를 제공
  - default_selector는 모든 이용 가능한 액셀러레이터를 열거하여 가장 성능이 높은 것을 선택
- **FPGA 위한 특수한 선택기**
  - FPGA 액셀러레이터용으로 fpga_selector와 fpga_emulator_selector 클래스를 추가로 제공
  - 해당 fpga_extensions.hpp 헤더 파일에서 정의

DPC++: vector_add.cpp

```
// Vector definition
 auto N = 10000;
 std::vector<int> a(N), b(N), c(N);

 // Vector initialization
 for (auto i = 0; i < a.size(); i++) a.at(i) = i;
 for (auto j = 0; j < b.size(); j++) b.at(j) = 2*j;

 {
   // Buffer
   buffer a_buf(a);
   buffer b_buf(b);
   buffer c_buf(c);

   q.submit([&](handler &h){
     // Create an accessor for each buffer with access permission
     accessor a_acc(a_buf, h, read_only);
     accessor b_acc(b_buf, h, read_only);
     accessor c_acc(c_buf, h, write_only, no_init);

     h.parallel_for(range<1>(N), [=](id<1> i){ c_acc[i] = a_acc[i] + b_acc[i]; });
   })
 }
```

host

device

호스트에 벡터 할당 및 초기화

가속기 메모리에 버퍼 할당 및 초기화

큐에 새로운 명령 제출

어떤 버퍼에 접근할 것인지 정의

계산 작업 병렬로 실행

가속기 메모리에 버퍼 할당 해제

- 호스트 코드와 커널 코드를 포함한 단일 소스로 CPU, GPU, FPGA 디바이스로 오프로드 가능

DPC++: vector_add.cpp

```
  // Result print
  for (int i = 0; i < N; i++) {
    std::cout << "[" << i << "]: "
              << a[i] << " + " << b[i] << " = "
              << c[i] << "\n";
  }

  a.clear();
  b.clear();
  c.clear();

  return 0;
}
```

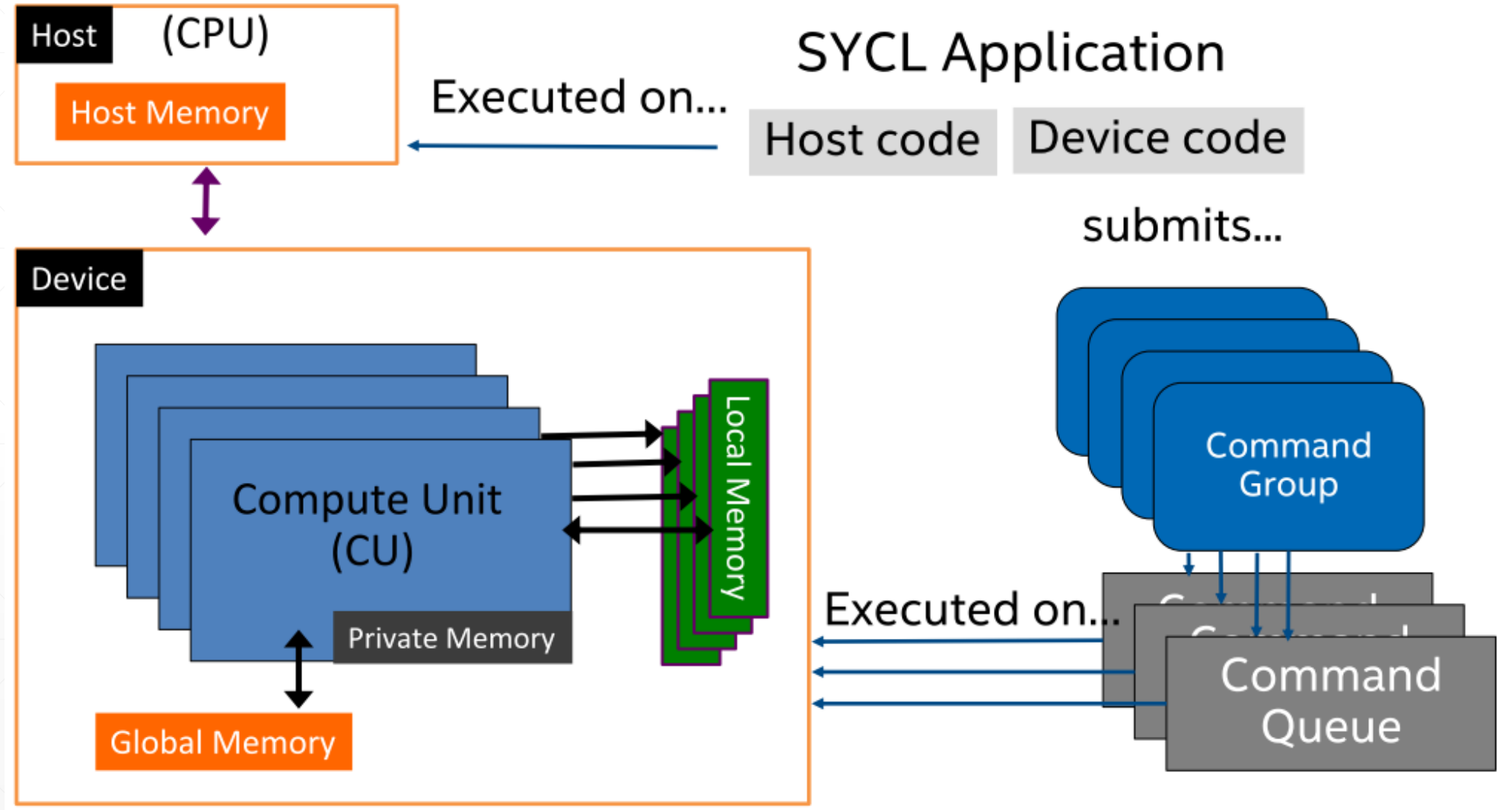
host

호스트에 결과를 출력

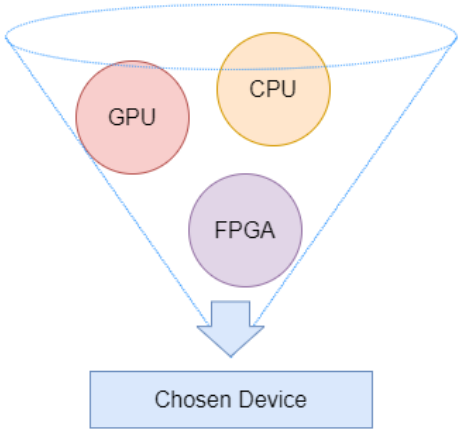
호스트 메모리 할당 해제

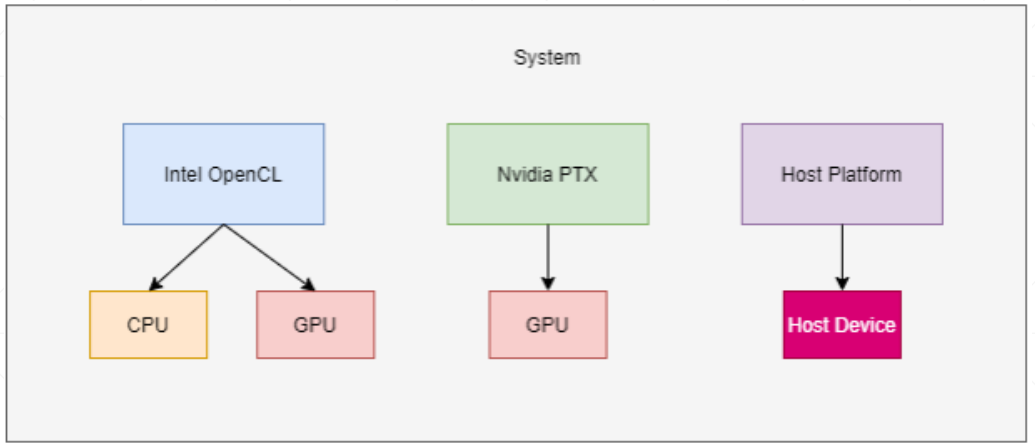
```
$ icpx -fsycl -O2 vector_add.cpp -o vector_add.x

$ ./vector_add.x
[0]: 0 + 0 = 0
[1]: 1 + 2 = 3
[2]: 2 + 4 = 6
[3]: 3 + 6 = 9
...
[9999]: 9999 + 19998 = 29997
```

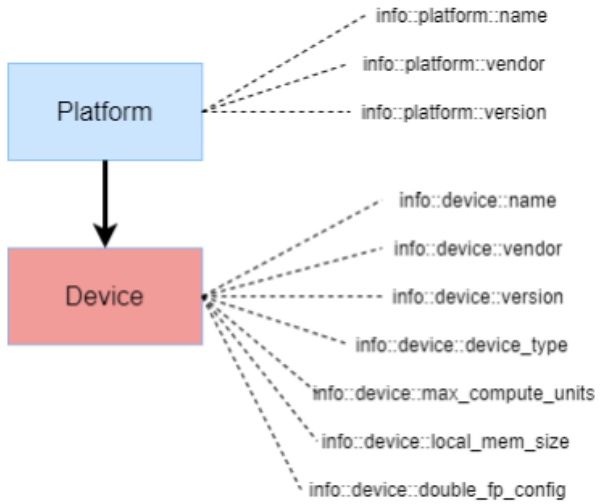
DPC++ 핵심: 실행 모델


Host
(CPU)
Host Memory
Executed on...
SYCL Application
Host code
Device code
submits...
Device
Compute Unit
(CU)
Private Memory
Local Memory
Global Memory
Executed on...
Command
Group
Command
Queue

DPC++ 핵심: 플랫폼



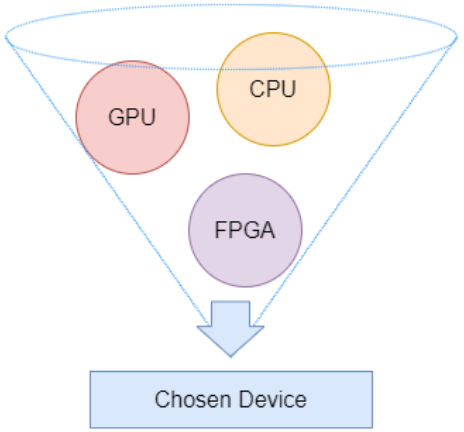


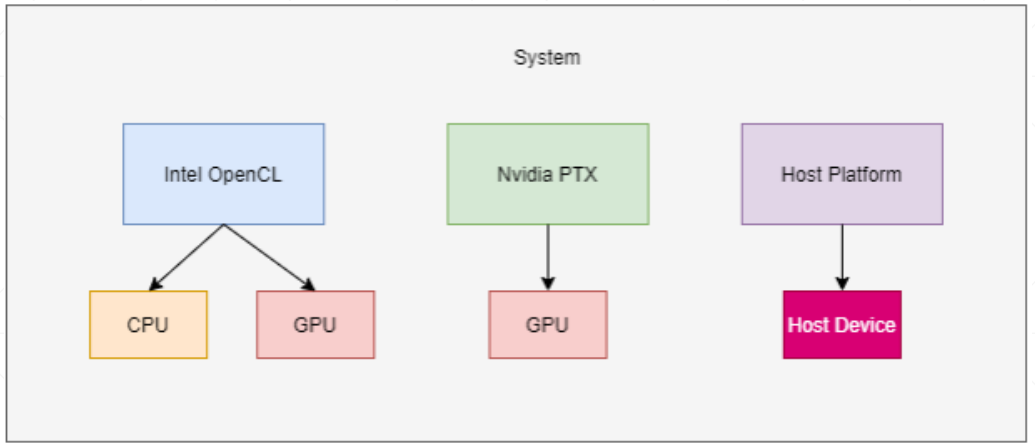


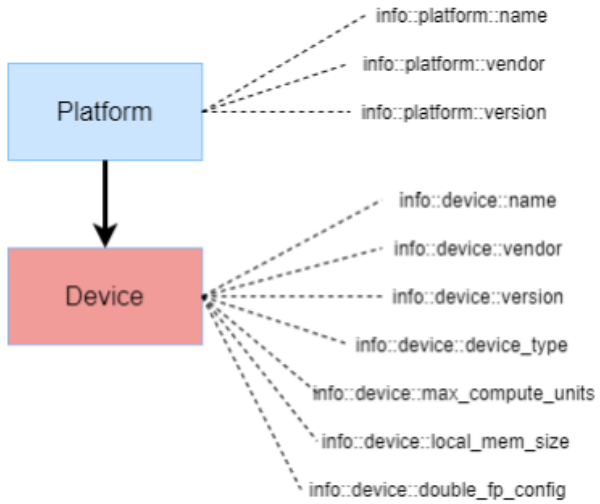
- **플랫폼 클래스**
  - SYCL은 실행때 시스템에서 사용 가능한 플랫폼 집합을 검색함
  - 각 플랫폼은 Intel OpenCL 또는 Nvidia PTX와 같은 백엔드 구현을 나타냄
  - 플랫폼 대한 정보를 문의하기 위한 `sycl::platform::get_info()` 함수를 제공함
    - platform 이름, 공급업체 및 버전

```
for (auto const& this_platform : platform::get_platforms() ) {
      std::cout << "Found platform: "
                << this_platform.get_info<info::platform::name>() << "\n";
```

DPC++ 핵심: 디바이스







- **디바이스 클래스**
  - oneAPI에서 가속기의 기능을 나타냄
  - 플랫폼 대한 정보를 문의하기 위한 `sycl::device::get_info()` 함수를 제공함
    - device 이름, 공급업체 및 버전
    - 메모리의 bandwidth, frequency, cache size, online/offline status
  - debug 위한 SYCL 디바이스의 실행 및 메모리 모델을 에뮬레이트된 호스트 디바이스 제공

```
for (auto const& this_device : this_platform.get_devices() ) {
  std::cout << "  Device: "
            << this_device.get_info<info::device::name>() << "\n";
}
```

DPC++ 핵심: 큐 및 디바이스 선택기

- **디바이스 선택기 클래스**
  - 커널을 실행할 특정 디바이스 선택을 활성화함
    - `sycl::host_selector_v` (항상 사용 가능함)
    - `sycl::default_selector_v` (가장 성능이 높은 가속기)
    - `sycl::accelerator_selector_v` (기타 가속기)
    - `sycl::cpu_selector_v`
    - `sycl::gpu_selector_v`
    - `sycl::ext::intel::fpga_selector_v`
    - `sycl::exit::intel::fpga_emulator_selector_v`

```
auto selector = gpu_selector_v;

queue q(selector);

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

- **큐 클래스**
  - 디바이스에 명령 그룹(command group)을 제출하는 방식 제공함
    - 디바이스에 커널 함수를 제출함
    - 디바이스에 데이터를 전송함
    - 다른 명령이 완료되기를 기다림
  - 각 큐은 정확히 하나의 다바이스(GPU 또는 FPGA)와 연결됨

DPC++ 핵심: 멀티 GPU 선택기

- 선택기 override: 가장 높은 rating 있는 가속기 선택

```
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() & (dev.get_info<info::device::name>().find("AMD")   != std::string::npos)) 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;
}
```

- 실행 시 변수환경으로 선택
  - SYCL_DEVICE_SELECTOR=<backend>:<device>:<id>
  - 예시: export SYCL_DEVICE_SELECTOR=opencl:gpu:1 또는 export SYCL_DEVICE_SELECTOR=hip:gpu:0

DPC++ 핵심: 명령

- **명령 그룹**
  - 큐에서 sycl::queue::submit() 함수를 호출하여 구성됨
  - C++ lambda 함수
    - [&]: lambda 함수 밖의 모두 변수를 안에서 읽고 쓸수 있음
    - `sycl::handler`: 디바이스에 커널를 제출 방식 드러냄
  - 명령 그룹 안에서 실제 디바이스 코드를 정의
    - `sycl::handler::single_task()`
    - `sycl::handler::parallel_for()`
    - `sycl::handler::copy()`
    - `sycl::handler::update_host()`
    - `sycl::handler::host_task()`
  - 비동기적 (asynchronously) 으로 제출됨
    - `sycl::queue::submit()` 명령 완료된 `sycl::event`의 `wait()` 함수으로 동기화 가능함
    - 동기화 큐: `sycl::queue q{selector, sycl::property_list{sycl::property::queue::in_order{}}}`

```
auto selector = gpu_selector_v;

queue q(selector);

auto e = q.submit([&](handler& h){
  // COMMAND GROUP
  h.parallel_for()
});


e.wait()
```

DPC++ 핵심: 커널 함수

- **커널 함수**
  - `sycl::handler` 제공된 함수으로 작성함
  - 명령 그룹에는 SYCL 커널 함수 명령이 단일로 존재
  - 커널 함수의 규칙
    - C++ lambda로 정의함
    - [=]로 captured-by-value
    - 함수의 포인터 안됨
    - 동적 할당 (dynamic allocation) 안됨
    - 동적 다형성 (dynamic polymorphism) 안됨
    - 재귀 (recursion) 안됨
  - SYCL 2020 표준에서 커널 lambda의 이름을 지정하지 않을 수 있음

```
auto selector = gpu_selector_v;

queue q(selector);

auto e = q.submit([&](handler& h){
  // COMMAND GROUP
  h.parallel_for<my kernel>(..., [=](){
    // KERNEL CODE
  });
});
e.wait()
```

- **병렬 커널**
  - 병렬 커널을 사용하면 작업의 여러 인스턴스를 병렬로 실행할 수 있음
  - 디바이스에 각 반복이 순서에 관계없이 완전히 독립적인 기본 for 루프 오프로드

DPC++ 핵심: 기본 병렬 커널

- 기본 병렬 커널의 기능은 range, id 및 item 클래스를 통해 사용할수 있음
  - range 클래스는 병렬 실행의 반복 공간을 나타냄
  - id 클래스는 병렬로 실행되는 커널의 개별 인스턴스에 인덱스를 부여함
  - item 클래스는 커널 함수의 개별 인스턴스를 나타냄
    - `get_id()`, `get_range()`, `get_offset()`, `get_linear_id()`

CPU 애플리케이션의 for 루프

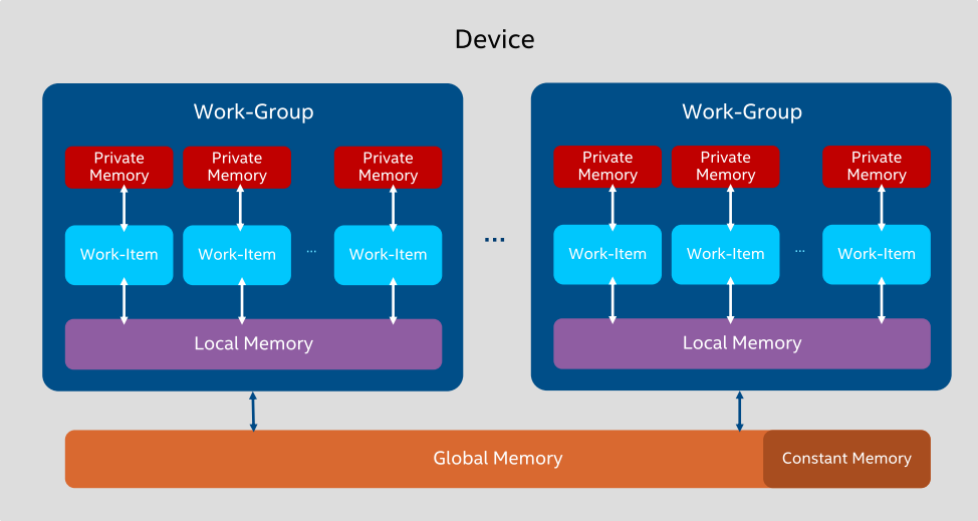
```
for ( int i= 0 ; i < 1024 ; i++){
    c[i] = a[i] + b[i];
});
```

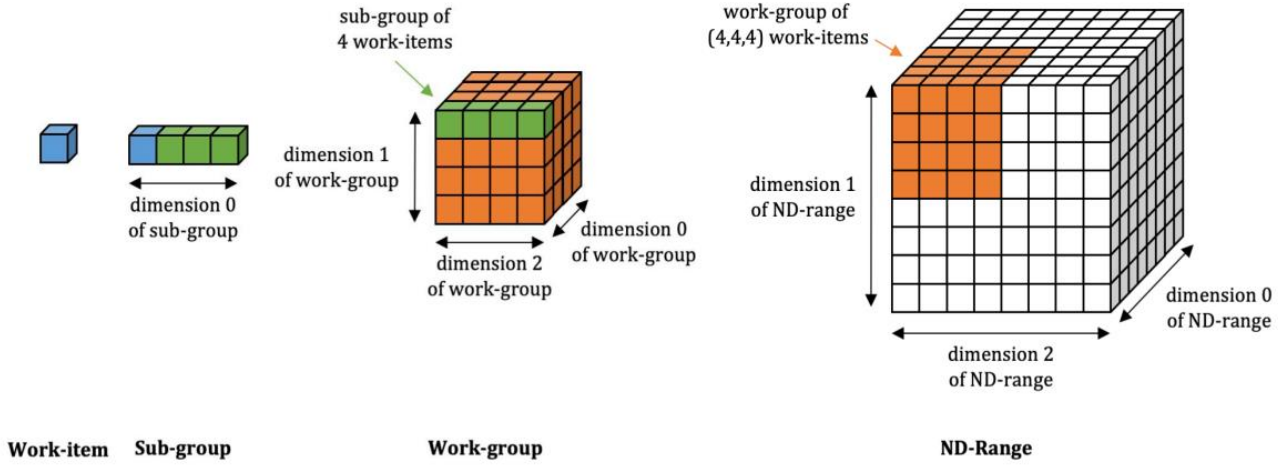
parallel_for를 사용하여 가속기로 오프로드

```
h.parallel_for (range<1>(1024), [=](id<1> idx ){
  c[idx] = a[idx] + b[idx]
});
```

```
h.parallel_for(range<1>(1024), [=](item<1> item){
  auto idx = item.get_id();

  c[idx] = a[idx] + b[idx]
});
```

DPC++ 핵심: ND-Range 커널





- **메모리 최적화**
  - 로컬 (local) 메모리에 대한 접근이 제공하고 하드웨어의 계산 단위에 실행을 매핑함
  - 낮은 수준의 성능 튜닝을 가능한 방식
- **Work-group**
  - 전체 반복 공간은 work-group 이라는 작은 그룹으로 나뉘함
  - 자원 사용 및 부하 분산 작업 배포를 제어할수 있음
- **Work-item**
  - work-group 내의 작업 항목은 하드웨어의 단일 계산 단위에서 예약됨

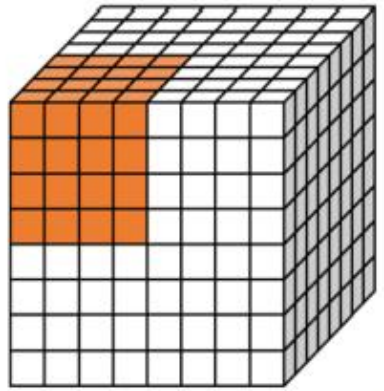
DPC++ 핵심: ND-Range 커널

- **nd_range 커널**
  - nd_range 클래스와 nd_item 클래스를 통해 사용할수 있음
- **nd_range 클래스**
  - 전역 실행 범위와 각 작업 그룹의 로컬 실행 범위를 사용하여 그룹화된 실행 범위를 나타냄
- **nd_item 클래스**
  - 커널 함수의 개별 인스턴스를 나타내며 작업 그룹 범위 및 인덱스를 문의할 수 있음
  - 디바이스 마다 최적한된 work-group 크기 다름
    - `get_global_id()`, `get_local_id()`, `get_global_range()`, `get_local_range()`, `get_nd_range()`, `mem_fence()`

```
h.parallel_for(nd_range<1>(range<1>(1024),range<1>(64)), [=](nd_item<1> item){
                                global range      local range
                                                  (work-group size)
  auto idx = item.get_global_id();
  auto local_id = item.get_local_id();
  // CODE THAT RUNS ON DEVICE
});
```

Intel GPU 구조

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

API 비교: DPC++/SYCL vs. CUDA

Execution model equivalence

CUDA	SYCL	OpenCL
SM	CU	CU
SM core	PE	PE
thread	work-item	work-item
block	work-group	work-group

Memory model equivalence

CUDA	SYCL	OpenCL
register	private memory	private memory
shared memory	local memory	local memory
constant memory	constant memory	constant memory
global memory	global memory	global memory
local memory	N/A(device specific)	N/A(device specific)

Memory management API equivalence

CUDA	SYCL	OpenCL	Description

`cudaMalloc()`
`cudaMallocHost()`
`cudaHostAlloc()`
	`buffer` class	`clCreateBuffer()`
		`clEnqueueMapBuffer()`	

`cudaHostGetDevicePointer()`	`accessor` class	`N/A`	OpenCL does not support a unified memory system.
`cudaMemset()`	`handler::fill()`	`clEnqueueFillBuffer()`	

`cudaMemcpyAsync()`
`cudaMemcpy()`
	`handler::copy()`	`clEnqueueReadBuffer()`
		`clEnqueueWriteBuffer()`
		`clEnqueueCopyBuffer()`
			In SYCL explicit copy is optional. This means that a user can explicitly copy the data between host or device. If a user does not provide an explicit copy, the data transfer is handled implicitly.

`cudaFree()`	`N/A`	`clReleaseMemObject()`	The buffer deletion is handled by the SYCL runtime, when an application exits the end of the SYCL scope {}.

https://developer.codeplay.com/products/computecpp/ce/2.11.0/guides/sycl-for-cuda-developers/migrating-from-cuda-to-sycl

최대 Work Group: Intel vs. NVIDIA

```
Found Platform:
    info::platform::name is 'NVIDIA CUDA BACKEND'
    info::platform::vendor is 'NVIDIA Corporation'
    info::platform::version is 'CUDA 11.2'
    info::platform::profile is 'FULL PROFILE'
  Device: Tesla K40
    is_host(): No
    is_cpu(): No
    is_gpu(): Yes
    is_accelerator(): No
    info::device::vendor is 'NVIDIA Corporation'
    info::device::driver_version is 'CUDA 11.2'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '1024'
    info::device::mem_base_addr_align is '4096'
    info::device::partition_max_sub_devices is '0'
```

```
Found Platform:
    info::platform::name is 'Intel(R) OpenCL HD Graphics'
    info::platform::vendor is 'Intel(R) Corporation'
    info::platform::version is 'OpenCL 3.0 '
    info::platform::profile is 'FULL_PROFILE'
  Device: Intel(R) UHD Graphics P630 [0x3e96]
    is_host(): No
    is_cpu(): No
    is_gpu(): Yes
    is_accelerator(): No
    info::device::vendor is 'Intel(R) Corporation'
    info::device::driver_version is '21.11.19310'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '256'
    info::device::mem_base_addr_align is '1024'
    info::device::partition_max_sub_devices is '0'
```

```
Found Platform:
    info::platform::name is 'SYCL host platform'
    info::platform::vendor is ''
    info::platform::version is '1.2'
    info::platform::profile is 'FULL PROFILE'
  Device: SYCL host device
    is_host(): Yes
    is_cpu(): No
    is_gpu(): No
    is_accelerator(): No
    info::device::vendor is ''
    info::device::driver_version is '1.2'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '1'
    info::device::mem_base_addr_align is '1024'
    info::device::partition_max_sub_devices is '1'
```

```
Found Platform:
    info::platform::name is 'NVIDIA CUDA BACKEND'
    info::platform::vendor is 'NVIDIA Corporation'
    info::platform::version is 'CUDA 11.2'
    info::platform::profile is 'FULL PROFILE'
  Device: Tesla V100
    is_host(): No
    is_cpu(): No
    is_gpu(): Yes
    is_accelerator(): No
    info::device::vendor is 'NVIDIA Corporation'
    info::device::driver_version is 'CUDA 11.2'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '1024'
    info::device::mem_base_addr_align is '4096'
    info::device::partition_max_sub_devices is '0'
```

```
Found Platform:
    info::platform::name is 'Intel(R) OpenCL HD Graphics'
    info::platform::vendor is 'Intel(R) Corporation'
    info::platform::version is 'OpenCL 3.0 '
    info::platform::profile is 'FULL_PROFILE'
  Device: Intel(R) Iris(R) Xe MAX Graphics [0x4905]
    is_host(): No
    is_cpu(): No
    is_gpu(): Yes
    is_accelerator(): No
    info::device::vendor is 'Intel(R) Corporation'
    info::device::driver_version is '21.11.19310'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '512'
    info::device::mem_base_addr_align is '1024'
    info::device::partition_max_sub_devices is '0'
```

```
Found Platform:
    info::platform::name is 'Intel(R) OpenCL'
    info::platform::vendor is 'Intel(R) Corporation'
    info::platform::version is 'OpenCL 2.1 LINUX'
    info::platform::profile is 'FULL_PROFILE'
  Device: Intel(R) Core(TM) i9-10920X CPU @ 3.50GHz
    is_host(): No
    is_cpu(): Yes
    is_gpu(): No
    is_accelerator(): No
    info::device::vendor is 'Intel(R) Corporation'
    info::device::driver_version is '2021.11.3.0.17_160000'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '8192'
    info::device::mem_base_addr_align is '1024'
    info::device::partition_max_sub_devices is '24'
```

- NVIDIA 디바이스의 경우에는 max_work_group_size는 1024 (블록당 최대 스레드 개수)로 고정됨
- Intel 다바이스의 경우에는 max_work_group_size는 디바이스마다 다름: Iris Xe Max (512), Gen 9 (256)
- 호스트 디바이스가 스레드 지원이 없는 OpenCL 디바이스의 에뮬레이션 됨

DPC++ 핵심: Buffer

- **데이터 저장 및 액세스 분리**
  - SYCL의 Buffer는 호스트 및 디바이스에 데이터를 관리함
  - SYCL의 Accessor는 특정 SYCL 커널에 대한 호스트 및 디바이스의 데이터에 대한 잡근 요청함

- **Buffer 클래스**
  - 관리할 데이터에 대한 포인터와 데이터의 요소 개수를 설명하는 범위로 구성할수 있음
  - SYCL2020는 단순화를 위한 Class Template Argument Deduction (CTAD)를 지원 가능함
  - Buffer는 out-of-scope 되어 데이터 동기화가 자동으로 발생함

```
using namespace sycl;

int Arr[64];

buffer<int, 2> Buf(Arr, range<2>(8, 8));
```

CTAD →

```
using namespace sycl;

int Arr[64];

buffer Buf(Arr, range(8, 8));
```

```
using namespace sycl;

std::vector<int> Vector(64)

buffer<int, 1> Buf(Vector, range<1>(64));
```

CTAD →

```
using namespace sycl;

std::vector<int> Vector(64)

buffer Buf(Vector)
```

DPC++ 핵심: Accessor

- **Accessor 클래스**
  - buffer및 명령 그룹의 sycl::handler 통해 생성됨
  - CTAD 통해 요소 유형 및 차원은 버퍼에서 유추 가능함

  - Access 모드:
    - read: 커널 함수는 데이터를 읽음
    - write: 커널 함수는 데이터를 수정, 미리 명령 그룹에 대한 dependencies을 생성함
    - no_init: 커널 함수는 데이터의 초기 값이 필요 없고 이전 명령 그룹에 대한 dependencies을 제거함

  - 호스트 Accessor:
    - 데이터를 호스트에 다시 동기화함
    - Blocking call: 동일한 버퍼를 수정하는 모든 대기열 커널이 실행을 완료한 후 반환됨

```
buffer<int, 1> B(Vector, range<1>(64));

accessor<int, 1, access::mode::read_write> A1(B, h);
accessor<int, 1, access::mode::write> A2(B, h);
accessor<int, 1, access::mode::discard_write> A3(B, h);
```

CTAD →

```
buffer B(Vector);

accessor A1(B, h);
accessor A2(B, h, write_only);
accessor A3(B, h, write_only, noinit);
```

DPC++: scope 통해 데이터 동기화

```
// Vector definition
 auto N = 100;
 std::vector<int> a(N), b(N), c(N);

 // Vector initialization
 for (auto i = 0; i < a.size(); i++) a.at(i) = i;
 for (auto j = 0; j < b.size(); j++) b.at(j) = j;

 {
   // Buffer
   buffer a_buf(a);
   buffer b_buf(b);
   buffer c_buf(c);

   q.submit([&](handler &h){
     // Create an accessor for each buffer with access permission
     accessor a_acc(a_buf, h, read_only);
     accessor b_acc(b_buf, h, read_only);
     accessor c_acc(c_buf, h, write_only, no_init);

     h.parallel_for(range<1>(N), [=](id<1> i){
       c_acc[i] = a_acc[i] + b_acc[i];
     });
   })
 }
```

host

device

호스트에 벡터 할당 및 초기화

scope open

가속기 메모리에 버퍼 할당 및 초기화

큐에 새로운 명령 제출

어떤 버퍼에 접근한 것인지 정의

계산 작업 병렬로 실행

scope close: 데이터 동기화

DPC++: 호스트 Accessor 통해 데이터 동기화

```
// Vector definition
 auto N = 100;
 std::vector<int> a(N), b(N), c(N);

 // Vector initialization
 for (auto i = 0; i < a.size(); i++) a.at(i) = i;
 for (auto j = 0; j < b.size(); j++) b.at(j) = j;

 // Buffer
 buffer a_buf(a);
 buffer b_buf(b);
 buffer c_buf(c);

 q.submit([&](handler &h){
   // Create an accessor for each buffer with access permission
   accessor a_acc(a_buf, h, read_only);
   accessor b_acc(b_buf, h, read_only);
   accessor c_acc(c_buf, h, write_only, no_init);

   h.parallel_for(range<1>(N), [=](id<1> i){
       c_acc[i] = a_acc[i] + b_acc[i];
   });
 })
 host_accessor a_host(a_buf, read_only);
```

host

device

호스트에 벡터 할당 및 초기화

가속기 메모리에 버퍼 할당 및 초기화

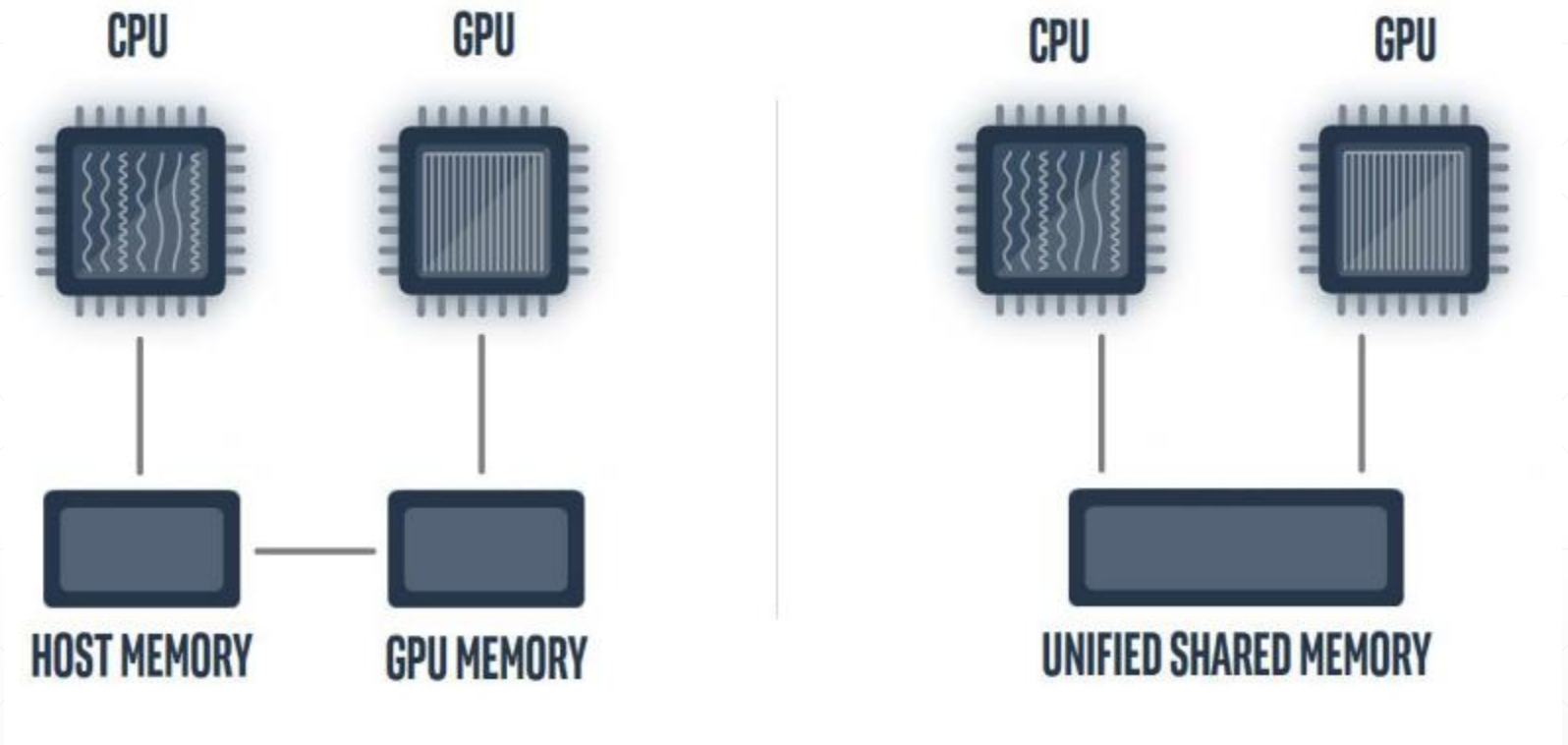
큐에 새로운 명령 제출

어떤 버퍼에 접근한 것인지 정의

계산 작업 병렬로 실행

데이터 동기화 (blocking call)

DPC++ 핵심: Unified Shard Memory (USM)



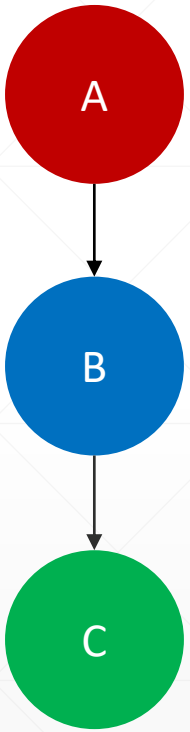
- **SYCL에서 포인터 기반 대안을 제공함**
  - C++ 프로그램에서 모든 포인터를 버퍼로 대체하는 것은 프로그래머에게 부담이 될수 있음
  - 호스트 및 장치 코드에서 동일한 메모리 개체를 참조할수 있음
  - Buffer를 지원하여 가속기에 porting 간소화함

DPC++ 핵심: Unified Shard Memory (USM)

- **USM 할당의 종류**
  - sycl::malloc_host: 호스트의 메모리 공간에 할당하여 가속기에서 접근 가능함
  - sycl::malloc_device: 가속기의 메모리 공간에 할당하며 호스트 접근 불가함
  - sycl::malloc_shared: 가속기와 호스트 모두에서 접근 가능한 공간 할당함

```
// USM
auto N = 1000;

int *a = malloc_shared<int>(N, q);
int *b = malloc_shared<int>(N, q);
int *c = malloc_shared<int>(N, q);

// Vector initialization
for (auto i = 0; i < N; i++) a[i] = i;
for (auto j = 0; j < N; j++) b[j] = j;

// Command group submission
q.parallel_for(range<1>(N), [=](id<1> i) { c[i] = a[i] + b[i]; }).wait();

free(a,q);
free(b,q);
free(c,q);
```

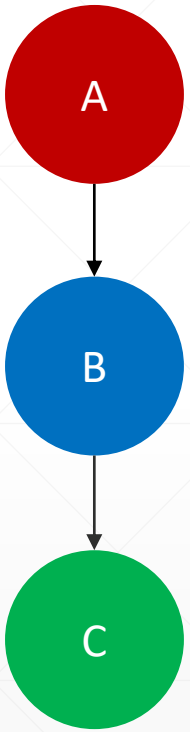
USM: wait()



wait()가 호스트에서 실행을 차단

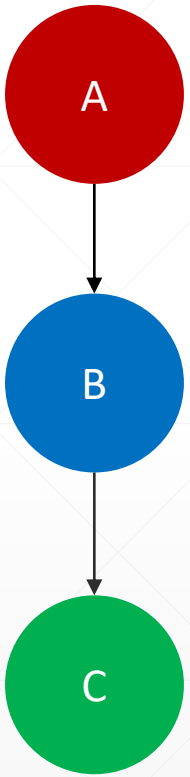
```
queue q;
int* data = malloc_shared<int>(N, q);
for(int i=0; i<N; i++) data[i] = i;

//task A
q.submit([&] (handler &h){
  h.parallel_for(range<1>(N), [=](id<1> i){
    data[i] += 2;
  });
}).wait();

//task B
q.submit([&] (handler &h){
  h.parallel_for(range<1>(N), [=](id<1> i){
    data[i] += 3;
  });
}).wait();

//task C
q.submit([&] (handler &h){
  h.parallel_for(range<1>(N), [=](id<1> i){
    data[i] += 5;
  });
}).wait();

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

USM: in_order 큐




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

//task A
q.submit([&] (handler &h){
  h.parallel_for(range<1>(N), [=](id<1> i){
    data[i] += 2;
  });
});

//task B
q.submit([&] (handler &h){
  h.parallel_for(range<1>(N), [=](id<1> i){
    data[i] += 3;
  });
});

//task C
q.submit([&] (handler &h){
  h.parallel_for(range<1>(N), [=](id<1> i){
    data[i] += 5;
  });
}).wait();

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

USM: depends_on()

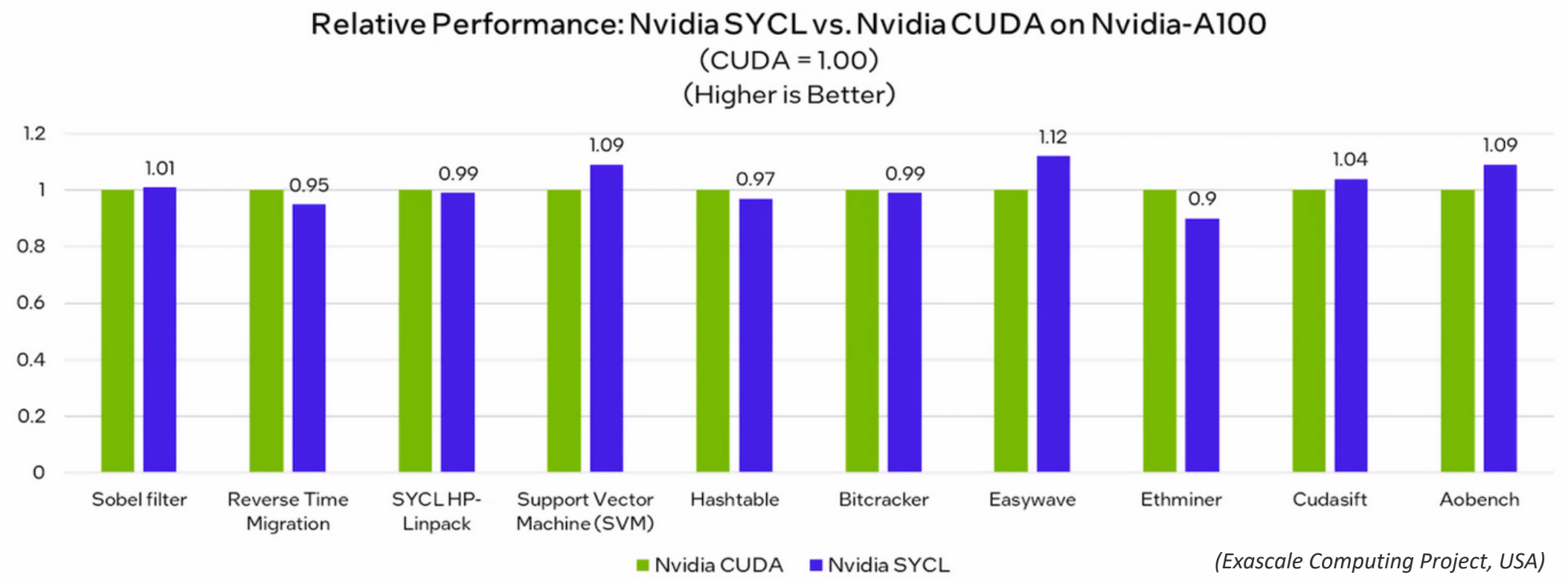



```
queue q;
int* data = malloc_shared<int>(N, q);
for(int i=0; i<N; i++) data[i] = i;

//task A
auto e1 = q.submit([&] (handler &h){
  h.parallel_for(range<1>(N), [=](id<1> i){
    data[i] += 2;
  });
});

//task B
auto e2 = q.submit([&] (handler &h){
  h.depends_on(e1)
  h.parallel_for(range<1>(N), [=](id<1> i){
    data[i] += 3;
  });
});

//task C
q.submit([&] (handler &h){
  h.depends_on(e2)
  h.parallel_for(range<1>(N), [=](id<1> i){
    data[i] += 5;
  });
}).wait();

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

Intel® oneAPI 2023 새로운 기능

- 컴파일러 및 SYCL 지원
  - Intel® oneAPI DPC++/C++ 컴파일러는 BF16을 완벽하게 지원하여 AI 가속을 제공
  - NVIDA 및 AMD GPU 이용하여 CodePlay의 oneAPI 플러그인을 지원
  - SYCLomatic 통해 cuBLAS 및 cuDNN과 같은 CUDA* 라이브러리 호출을 SYCL 및 oneAPI 라이브러리에 변환 가능
- 고성능 컴퓨팅을 위한 최신 Fortran 컴파일러
  - Fortran 2003, Fortran 2008 및 Fortran 2018의 모든 기능과 더 많은 OpenMP 5.x 기능을 지원
  - CPU에서 Co-Array를 지원하며 GPU에서 DO CONCURRENT를 지원
- 성능 라이브러리
  - Intel® oneAPI Math Kernel Library는 데이터 센터 GPU 성능 최적화 포함
  - 새로운 FFT, 1D 및 2D 최적화, 난수 생성기, Sparse BLAS 및 LAPACK 제공
  - Intel ® MPI Library는 GPU 버퍼를 사용하여 Collective 기능의 성능 향상 가능
- 분석 및 디버그
  - Intel® VTune™
    - Intel® Xeon® Max에서 HBM 이용하여 성능 향상 가능
    - Intel® 데이터 센터 Max에서 X^e Link 관련한 CPU/GPU 불균형 및 대역폭 병목 현상 문제 표시

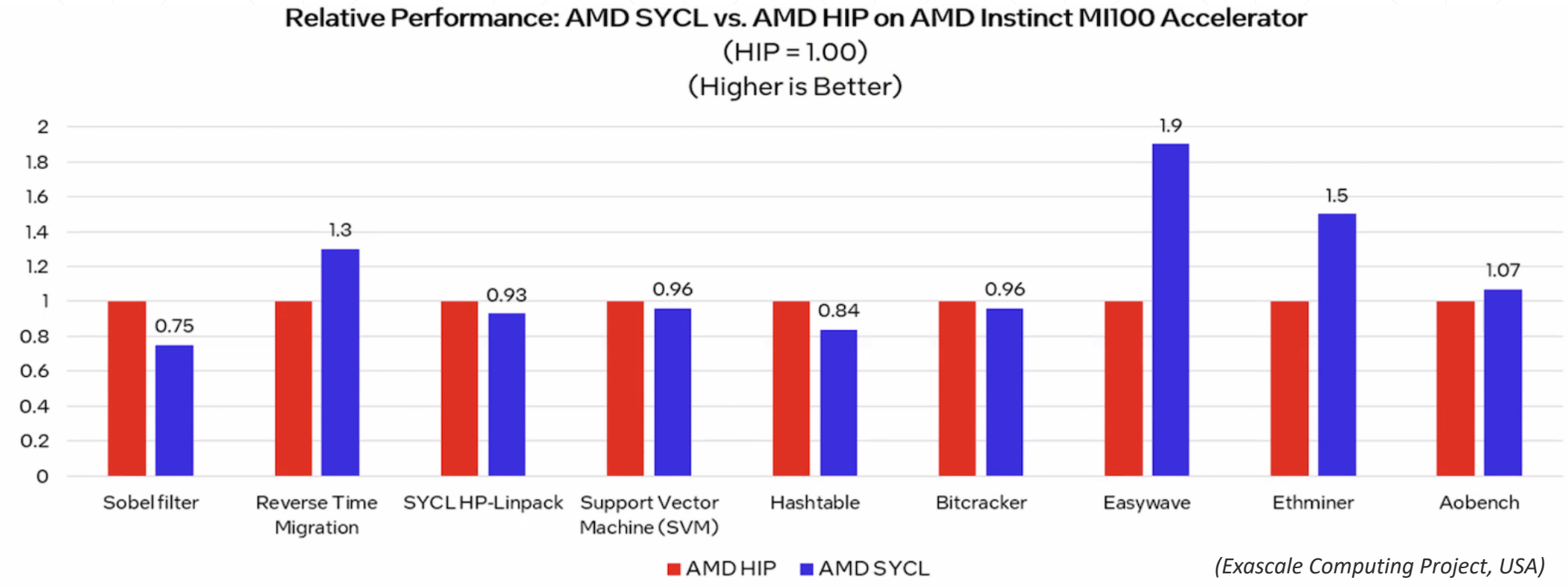
Intel® DPC++ 컴파일러: CUDA 백엔드



(Exascale Computing Project, USA)

- **테스트 환경:**
  - Intel® Xeon® Platinum 8360Y / NVIDIA A100-PCIe-80GB
  - CUDA 11.7
  - NVIDIA GPU의 네이티브 CUDA*에 필적하는 성능

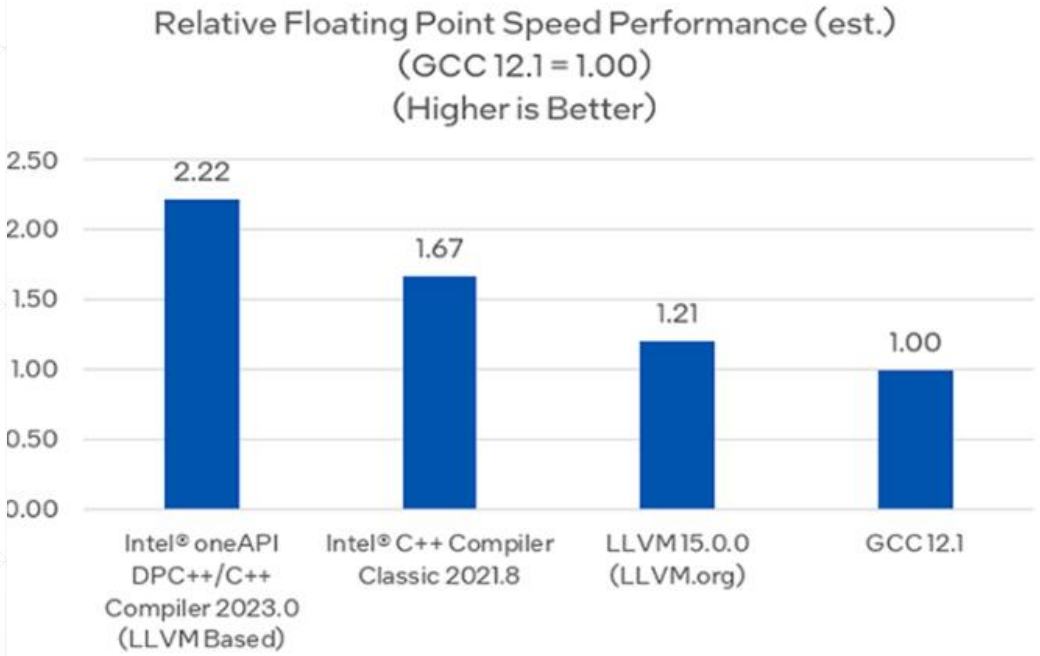
Intel® DPC++ 컴파일러: HIP 백엔드

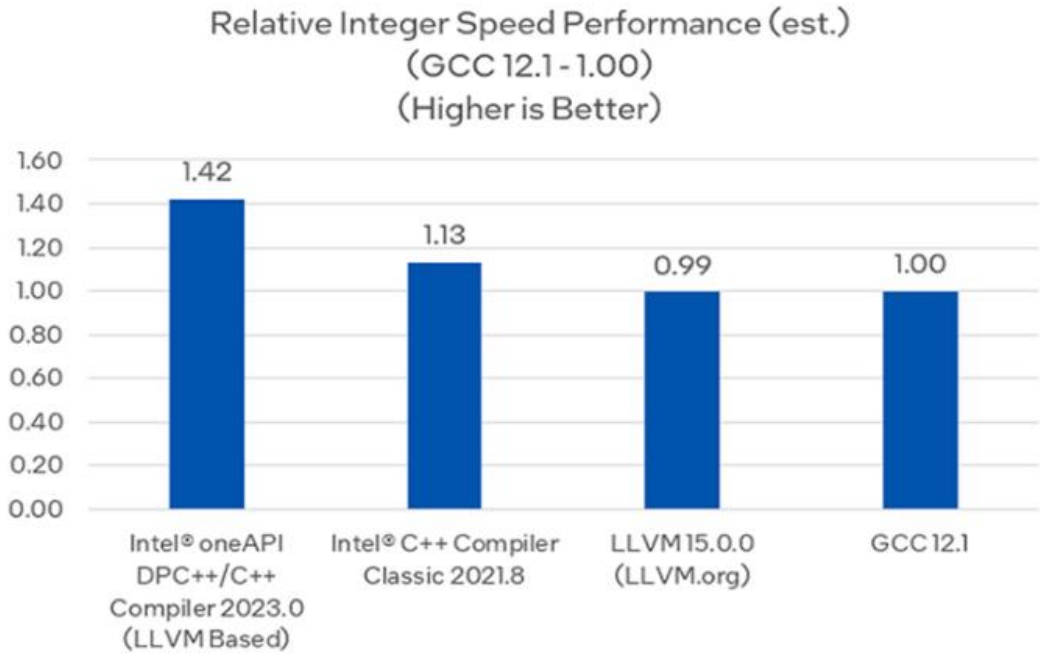


(Exascale Computing Project, USA)

- **테스트 환경:**
  - Intel® Xeon® Gold 6330 / AMD Instinct MI100
  - RoCm 5.2.1
  - SYCL 2020 기능의 50% 이상을 구현하다.

C++ 애플리케이션 성능 향상





- **테스트 환경:**
  - Intel® Xeon® Platinum 8480
  - SPEC 벤치마크: molecular dynamics, biomedical imaging, ray tracing, fluid dynamics etc.

인텔® 맥스 시리즈 성능

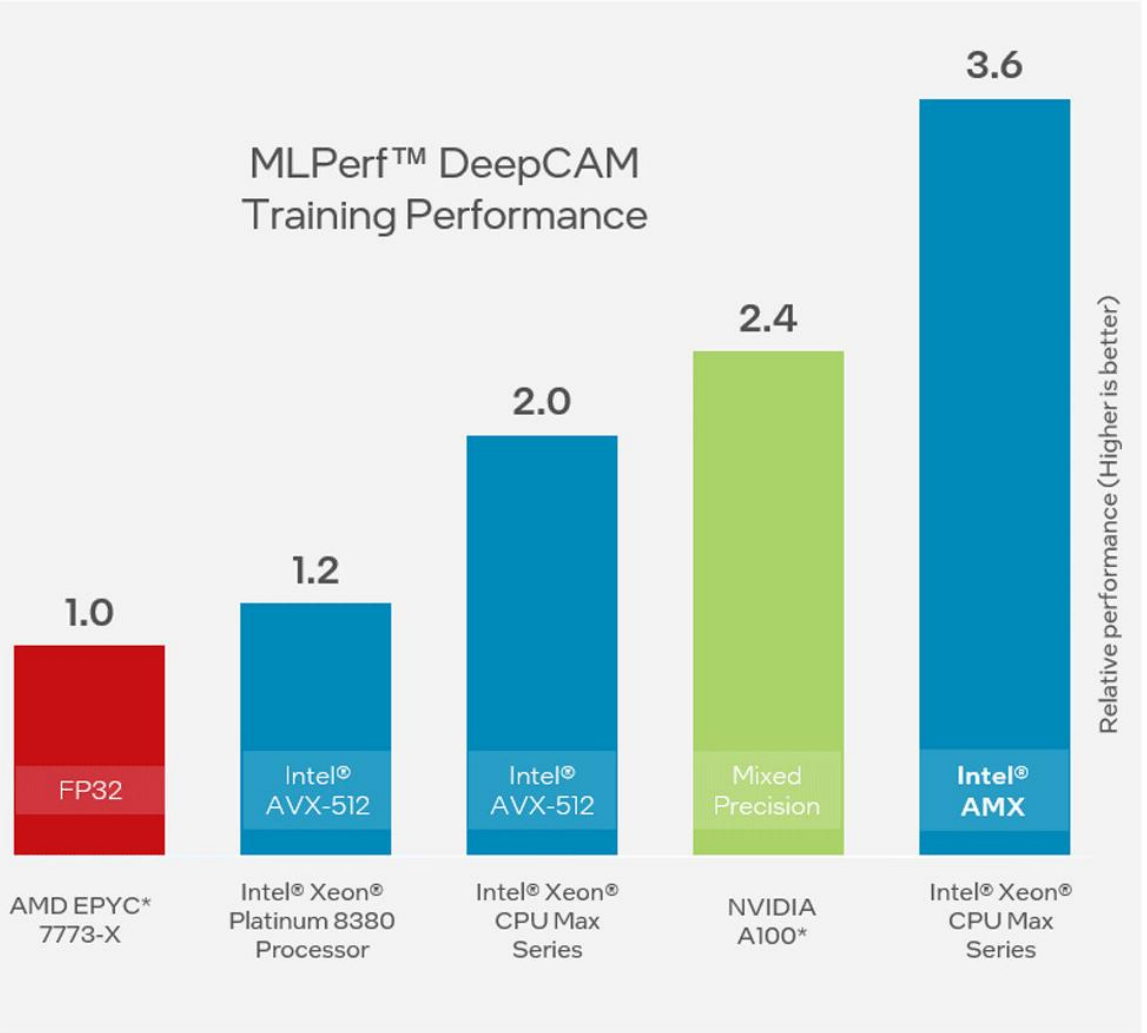
intel

Advanced Matrix Extensions

intel XEON MAX SERIES

Performance Leap in Deep Learning Inference & Training

- INT8 (all sign combinations) with INT32 Accumulation
- Bfloat16 with IEEE SP Accumulation
- Full Intel® Architecture (IA) Programmability
- Very Low Latency



인텔® 맥스 시리즈 성능

Life & Material Science

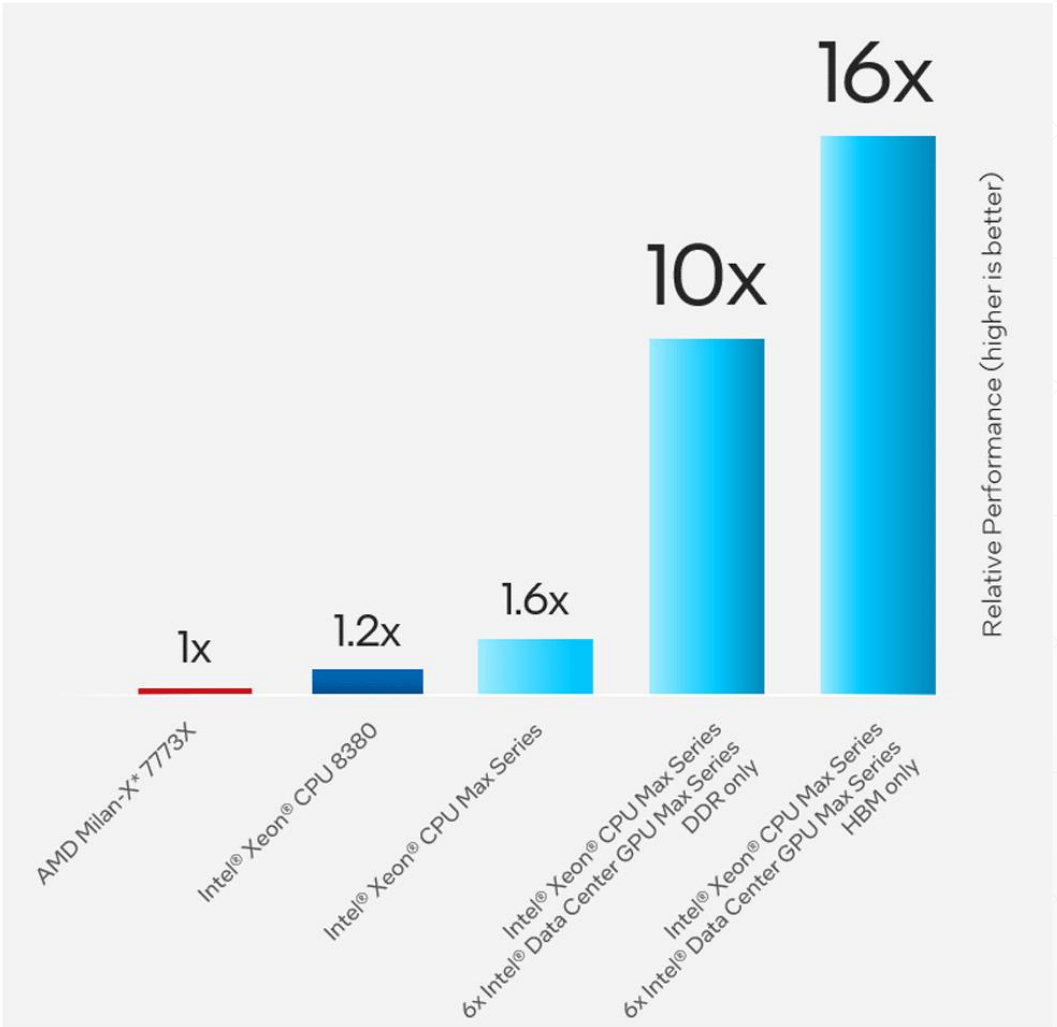
Speeding Exascale Material Discovery with Compute Density and HBM for LAMMPS (Liquid Crystal)

Performance results are **based on testing by Intel as of October 28, 2022** and may not reflect all publicly available security updates.

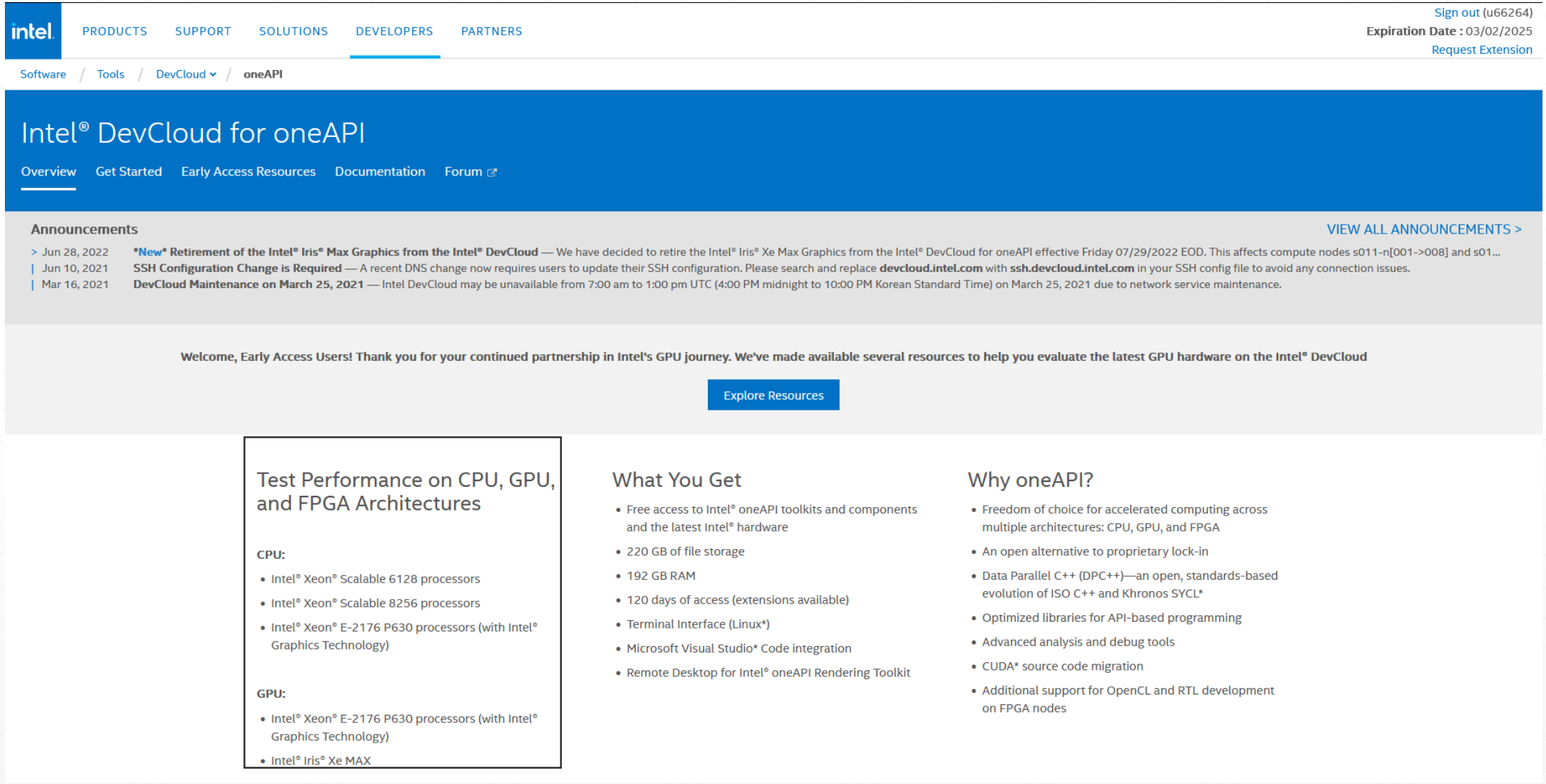
- Intel® Xeon® CPU Max Series: 1-node, 2x Intel® Xeon® CPU Max Series, HT On, Turbo On, Total Memory 128 GB HBM2e, BIOS EGSDCRB1.DWR.0085.D12.2207281916, ucode 0xac000040, SUSE Linux Enterprise Server 15 SP3, Kernel 5.3.18, oneAPI 2022.3.0, LAMMPS built with the Intel package
- Intel® Data Center GPU Max Series with DDR Host: 1-node, 2x Intel® Xeon® CPU Max Series, HT On, Turbo On, Total Memory 1024 GB DDR5-4800 + 128 GB HBM2e, Memory Mode: Flat, HBM2e not used, 6x Intel® Data Center GPU Max Series, BIOS EGSDCRB1.DWR.0085.D12.2207281916, ucode 0xac000040, Agama pvc-prq-54, SUSE Linux Enterprise Server 15 SP3, Kernel 5.3.18, oneAPI 2022.3.0, LAMMPS built with the Intel package
- Intel® Data Center GPU Max Series with HBM Host: 1-node, 2x Intel® Xeon® CPU Max Series, HT On, Turbo On, Total Memory 128 GB HBM2e, 6x Intel® Data Center GPU Max Series, BIOS EGSDCRB1.DWR.0085.D12.2207281916, ucode 0xac000040, Agama pvc-prq-54, SUSE Linux Enterprise Server 15 SP3, Kernel 5.3.18, oneAPI 2022.3.0, LAMMPS built with the Intel package.

See configuration disclosure for details. No product or component can be absolutely secure.

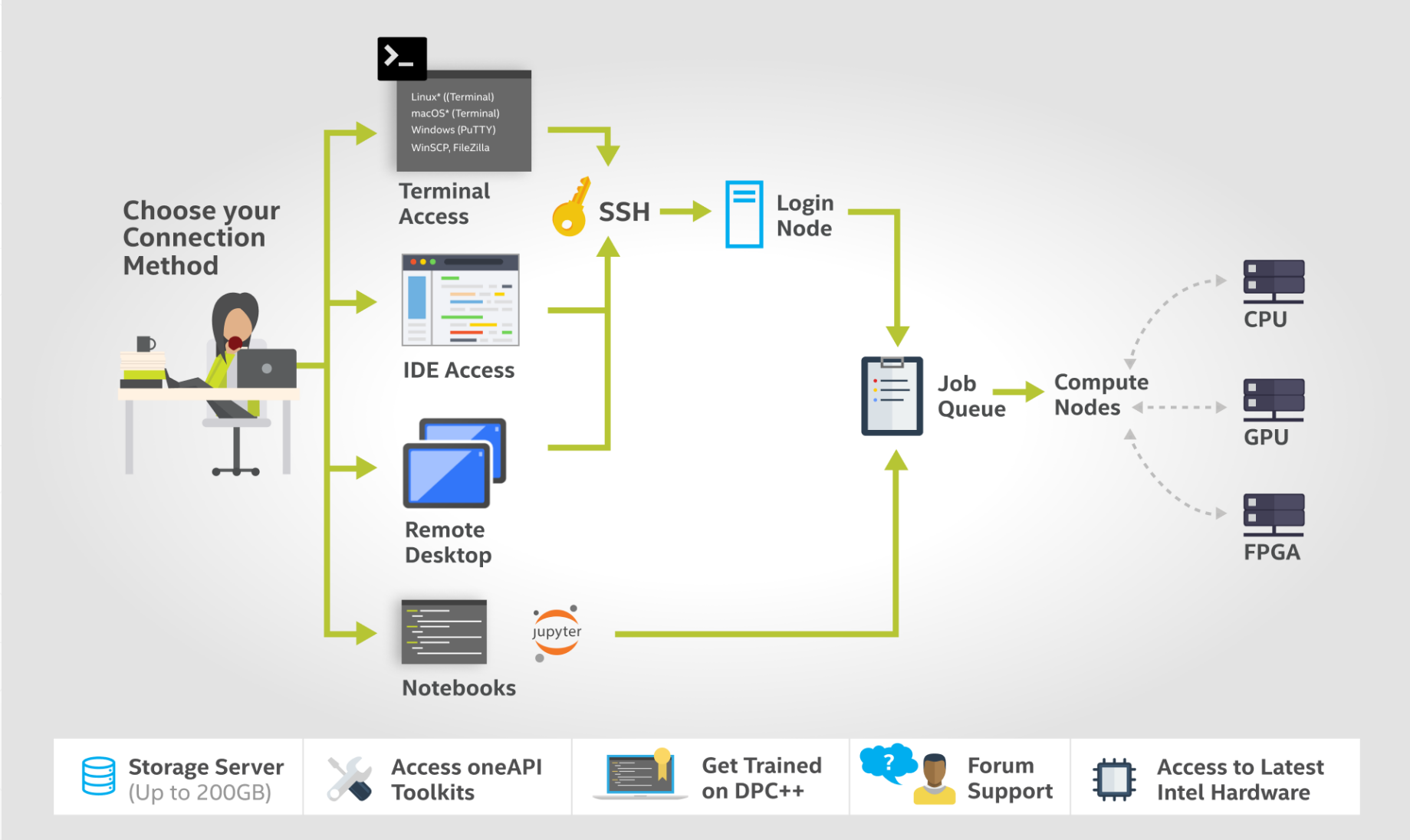
Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex. Your costs and results may vary.



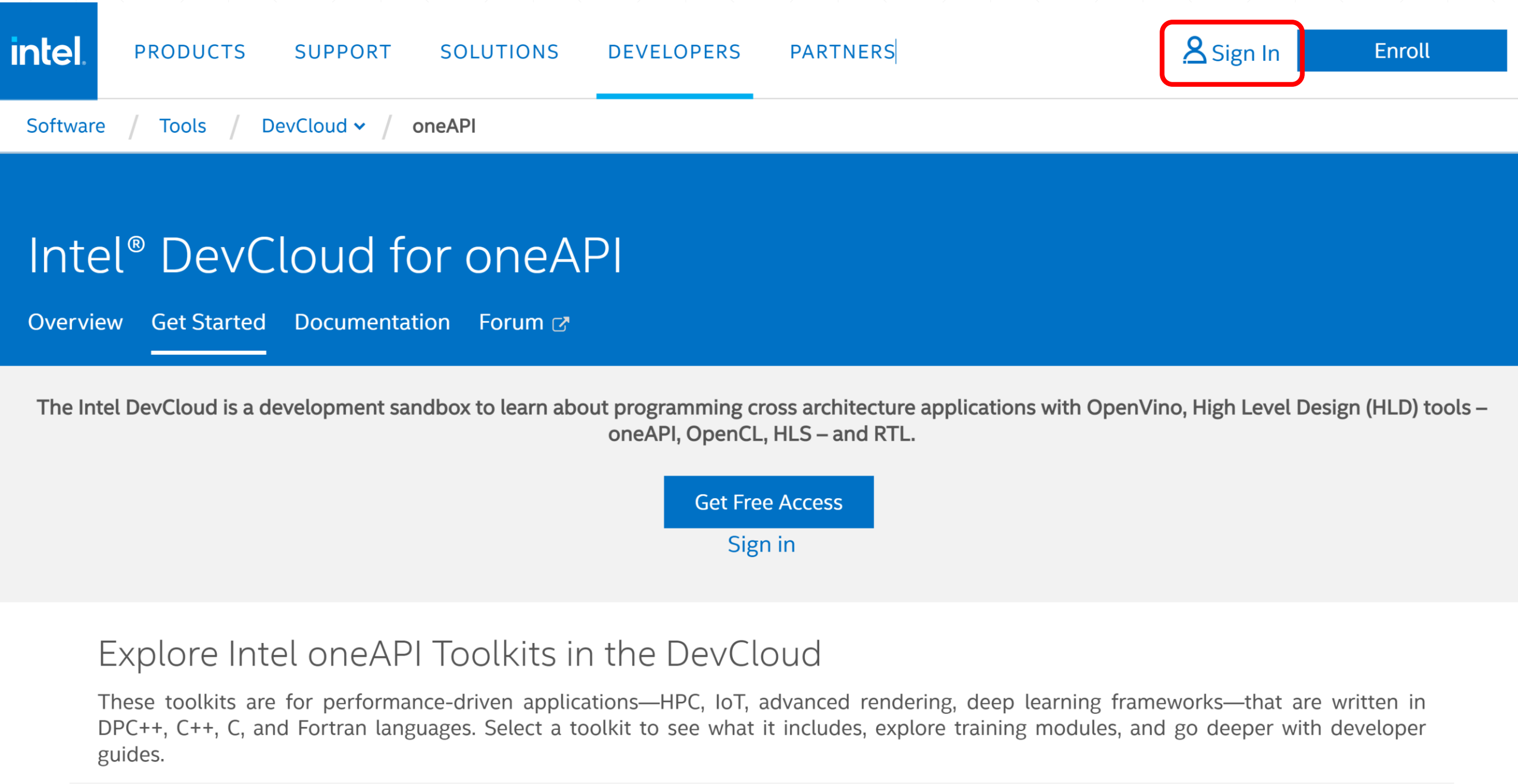
https://devcloud.intel.com/oneapi/



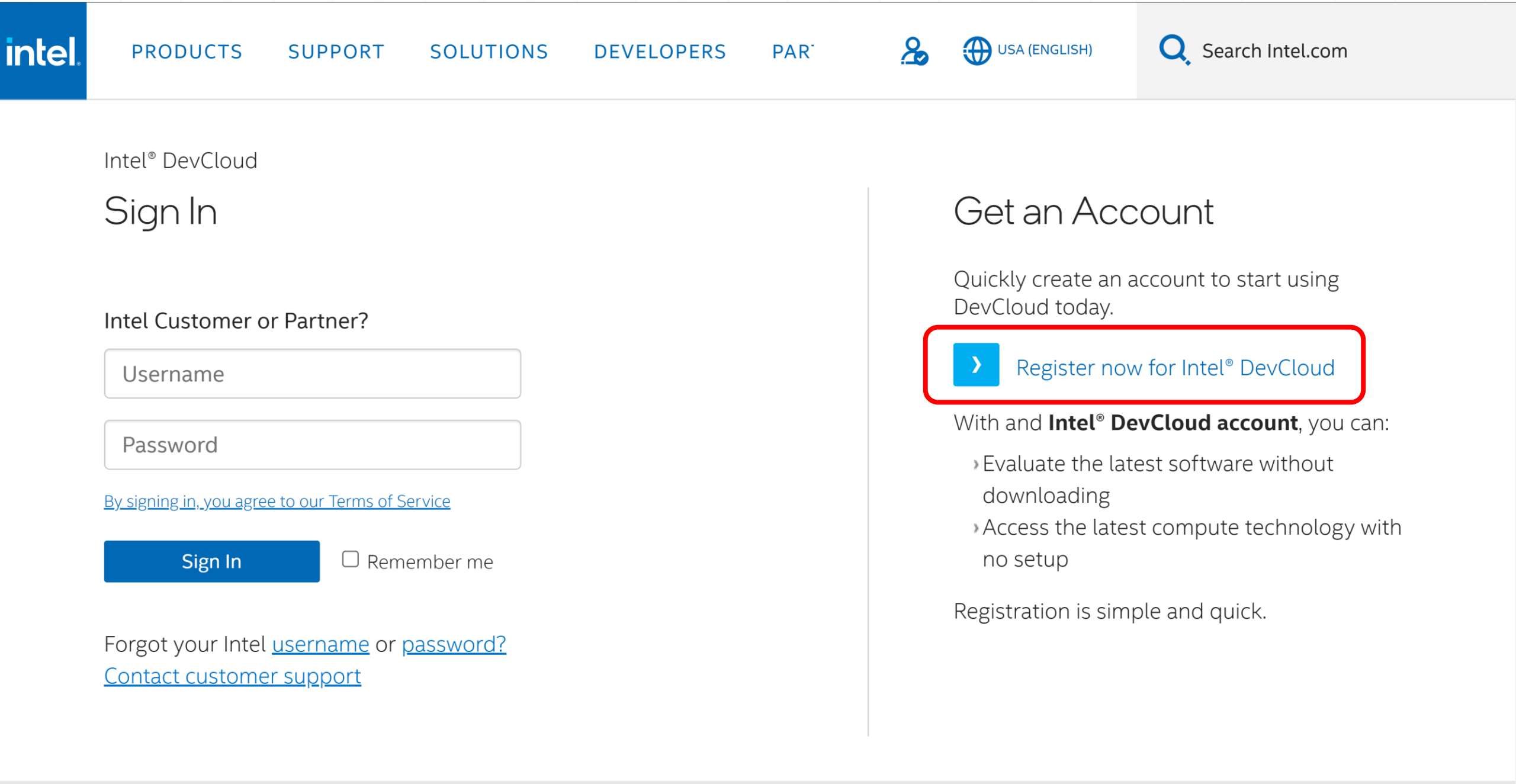
연결 방식


Choose your Connection Method
Linux* ((Terminal)
macOS* (Terminal)
Windows (PuTTY)
WinSCP, FileZilla
Terminal Access
IDE Access
Remote Desktop
Notebooks
jupyter
SSH
Login Node
Job Queue
Compute Nodes
CPU
GPU
FPGA
Storage Server (Up to 200GB)
Access oneAPI Toolkits
Get Trained on DPC++
Forum Support
Access to Latest Intel Hardware

Step 1) Visit https://devcloud.intel.com/oneapi/get_started/



Step 2) Click the “Register now for Intel® DevCloud” link



Step 3) Fill out the "Basic Contact Information" section

Create an Intel® DevCloud Account

Sign up for immediate access to the latest Intel technology without downloads or hardware setup.

Intel Employee? Create account here

All fields are required except any fields specifically marked as optional.

Basic Contact Information

First Name

Last Name

Email Address

Username

Password

Confirm Password

Step 4) Fill out the “More About you” section

Basic Contact Information

Edit

More About you

What is your purpose for using Intel® Devcloud (Select all that apply)

- [ ] HPC Workloads
- [x] AI Training
- [ ] AI Inference

Business or Institution Name

<Enter your business or Institution name here>

What type of user are you?

Teacher/Professor

Step 5) Select what applies and click the “Next Step” button

Subscribe to optional email updates from Intel

- [ ] Select all subscriptions below
- [ ] Developer Zone Newsletter
- [ ] Edge Software Hub Product Communication
- [ ] Programmable Logic Product Announcements
- [ ] Programmable Logic Newsletters
- [ ] Software Developer Product Insights

- [ ] Yes, I would like to subscribe to stay connected to the latest Intel technologies and industry trends by email and telephone. I can unsubscribe at any time.

Next Step

Step 6) Accept the Terms and Conditions and click the “Submit” button

More About you

Edit

Terms and Conditions

I have read and accept the Intel® DevCloud Agreement

By submitting this form, you are confirming you are an adult 18 years or older and you agree to share your personal information with Intel to stay connected to the latest Intel technologies and industry trends by email and telephone. You can unsubscribe at any time. Intel’s web sites and communications are subject to our Privacy Notice and Terms of Use.

This site is protected by reCAPTCHA and the Google Privacy Policy and Terms of Service apply.

Submit

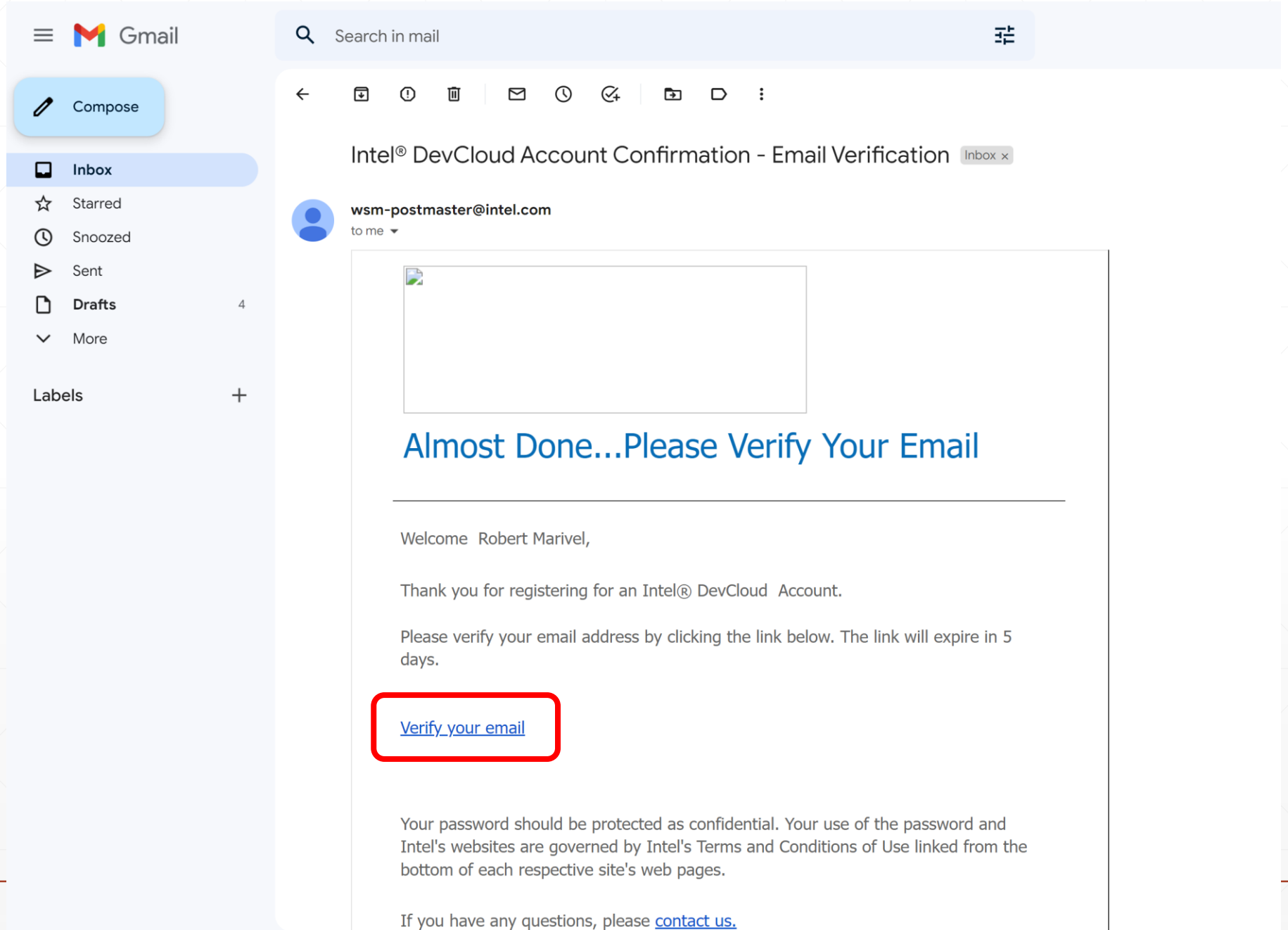
Step 7) Check your email to verify your email address

Almost there!

Check your email for the verification link and **sign in**. The link will expire in 5 days.

Didn't receive the email? Check your spam or junk folder or click on Resend email below. Click Here

Step 8) Click the "Verify your email" link in the email sent by wsm-postmaster@intel.com



https://devcloud.intel.com/oneapi/get_started/

OpenCL for FPGA development

Intel® FPGA SDK for OpenCL™ software technology1 is a development environment that enables software developers to accelerate their applications by targeting heterogeneous platforms with Intel CPUs and FPGAs.

Get Started with your first Sample

- Microsoft* Visual Studio or Eclipse*-based Intel® Code Builder for OpenCL™ API now with FPGA support
- Fast FPGA emulation based on Intel's compiler technology
- Create OpenCL™ project jump-start wizard
- Development Environment for both host (CPU) and accelerator (FPGA)
- Syntax highlighting and code auto-completion features
- FPGA resource and performance analysis
- Fast and incremental FPGA compile

RTL Acceleration Functional Unit

The revolutionary Intel® Quartus® Prime Design Software includes everything you need to design for Intel® FPGAs, SoCs, and complex programmable logic device (CPLD) from design entry and synthesis to optimization, verification, and simulation. Dramatically increased capabilities on devices with multi-million logic elements are providing designers with the ideal platform to meet next-generation design opportunities.

Build and design using standard logic gates. Great for visualization and education.

Get Started with your first Sample

Connect with JupyterLab*

Connect with JupyterLab*

Use JupyterLab* 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.

Launch JupyterLab*

Training Resources

DevCloud Commands

Learn about the features of the compute nodes, data management, and how to submit, query, and delete your jobs.

Introduction to oneAPI and Essentials of Data Parallel C++

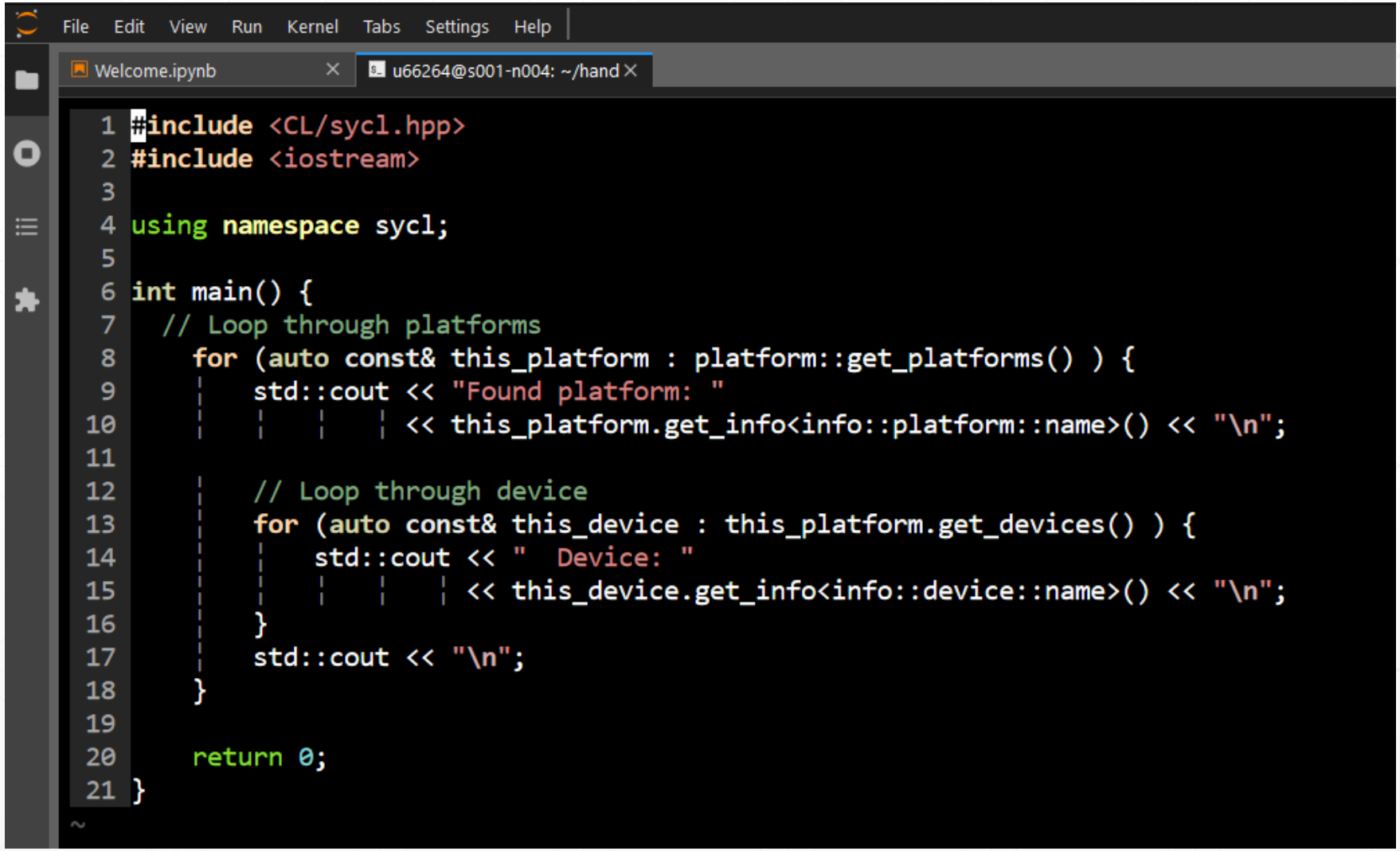
Use JupyterLab* to learn about how oneAPI can solve the challenges of programming in a heterogeneous world and understand the Data Parallel C++ (DPC++) language and programming model.

인터랙티브 작업 제출

```
u66264@s001-n023:~$ qsub -I -l nodes=2:gen9:gpu:ppn=2
qsub: waiting for job 2080043.v-qsvr-1.aidevcloud to start
qsub: job 2080043.v-qsvr-1.aidevcloud ready


################################################################################
#      Date:           Thu 08 Dec 2022 08:22:21 PM PST
#    Job ID:           2080043.v-qsvr-1.aidevcloud
#      User:           u66264
# Resources:           cput=35:00:00,neednodes=2:gen9:gpu:ppn=2,nodes=2:gen9:gpu:ppn=2,walltime=06:00:00
################################################################################
```

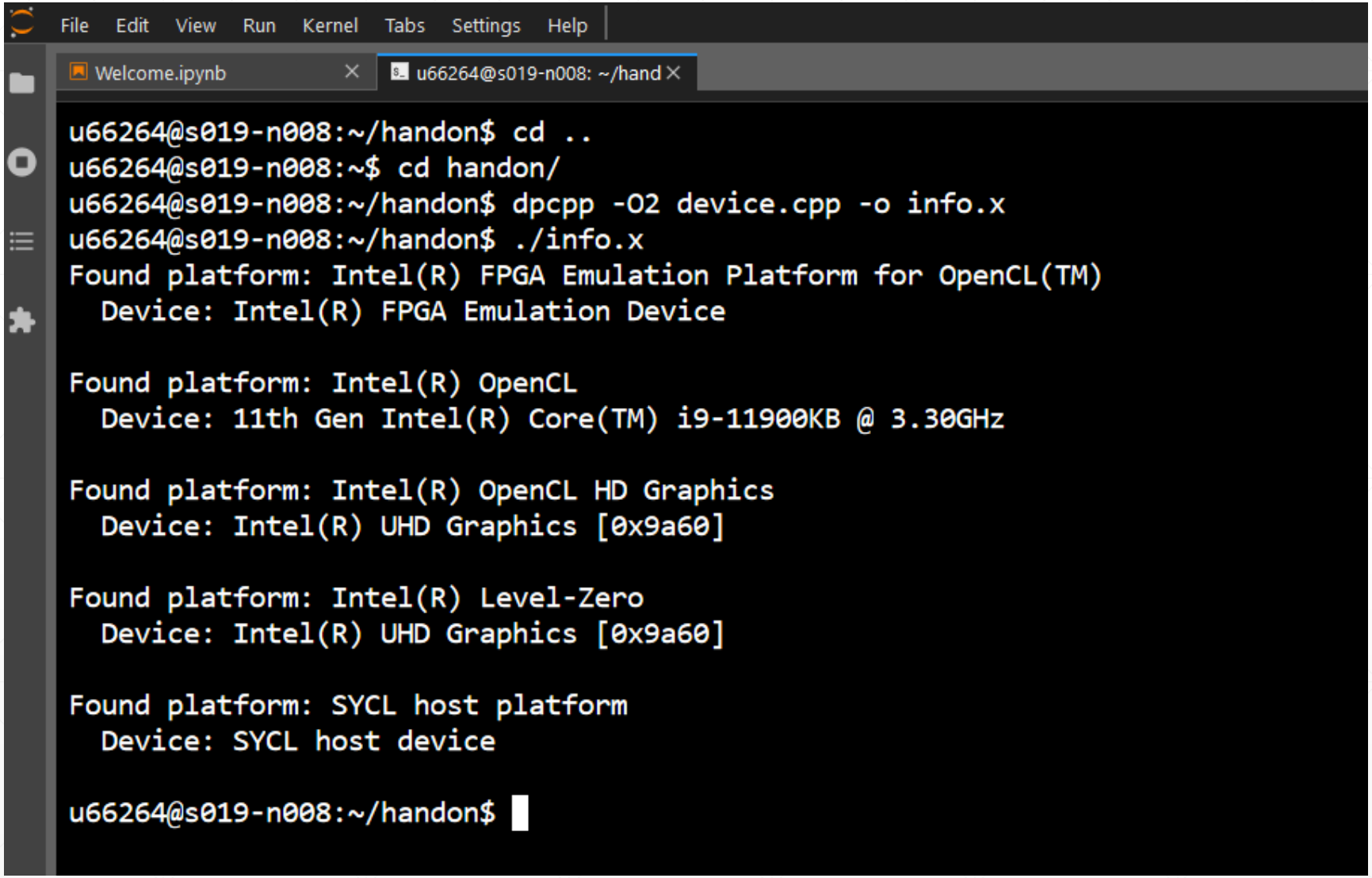
- Request node based on device properties
  - qsub -I -l nodes=[nnodes]:[props]:ppn=[process_per_node]
- Properties describing device class:
  - core / xeon / gpu / fpga
- Properties describing device name:
  - gen9 / gen 11 / aria10 / stratix10 / gold6128 / i9-10920x
- Properties describing purpose:
  - fpga_compile / fpga_runtime / renderkit

디바이스 및 플랫폼 확인



https://github.com/Apress/data-parallel-CPP/tree/main/samples/Ch12_device_information

Intel® Gen 9 그래픽



SYCL Hands-on: 컴파일 및 실행 테스트

```
u66264@s019-n016:~$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2022.15.12.0.01_081451]
[opencl:cpu:1] Intel(R) OpenCL, 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz 3.0 [2022.15.12.0.01_081451]
[opencl:gpu:2] Intel(R) OpenCL HD Graphics, Intel(R) UHD Graphics [0x9a60] 3.0 [22.35.24055]
[ext_oneapi_level_zero:gpu:0] Intel(R) Level-Zero, Intel(R) UHD Graphics [0x9a60] 1.3 [1.3.24055]
```

```
u66264@s019-n016:~$ dpcpp -O2 vector-add-buffers.cpp -o vector-add-buffers.x
icpx: warning: use of 'dpcpp' is deprecated and will be removed in a future release. Use 'icpx -fsycl' [-Wdeprecated]
u66264@s019-n016:~$ icpx -fsycl -O2 vector-add-buffers.cpp -o vector-add-buffers.x
```

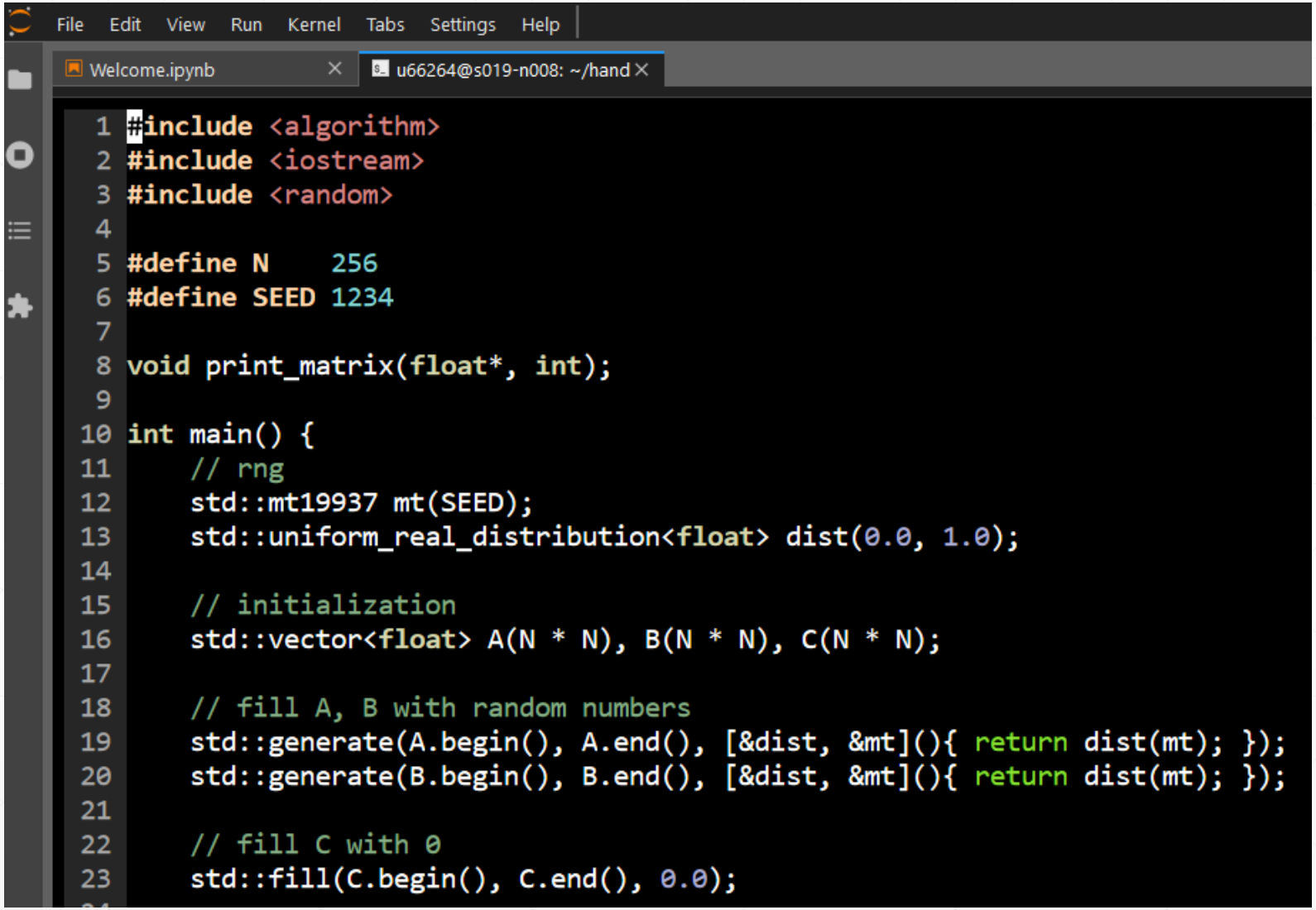
```
u66264@s019-n016:~$ ./vector-add-buffers.x
Running on device: Intel(R) UHD Graphics [0x9a60]
Vector size: 10000
[0]: 0 + 0 = 0
[1]: 1 + 2 = 3
[2]: 2 + 4 = 6
...
[9999]: 9999 + 19998 = 29997
Vector add successfully completed on device.
```

SYCL Hands-on: 실행 시 디바이스 선택

```
u66264@s019-n016:~$ SYCL_DEVICE_FILTER=opencl:cpu SYCL_PI_TRACE=1 ./vector-add-buffers.x
SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_opencl.so [ PluginVersion: 11.15.1 ]
SYCL_PI_TRACE[all]: Selected device: -> final score = 1300
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL
SYCL_PI_TRACE[all]:   device: 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
Running on device: 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
Vector size: 10000
[0]: 0 + 0 = 0
[1]: 1 + 2 = 3
[2]: 2 + 4 = 6
...
[9999]: 9999 + 19998 = 29997
Vector add successfully completed on device.
```

```
u66264@s019-n016:~$ SYCL_DEVICE_FILTER=opencl:gpu SYCL_PI_TRACE=1 ./vector-add-usm.x
SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_opencl.so [ PluginVersion: 11.15.1 ]
SYCL_PI_TRACE[all]: Selected device: -> final score = 1500
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL HD Graphics
SYCL_PI_TRACE[all]:   device: Intel(R) UHD Graphics [0x9a60]
Running on device: Intel(R) UHD Graphics [0x9a60]
Vector size: 10000
[0]: 0 + 0 = 0
[1]: 1 + 2 = 3
[2]: 2 + 4 = 6
...
[9999]: 9999 + 19998 = 29997
Vector add successfully completed on device.
```

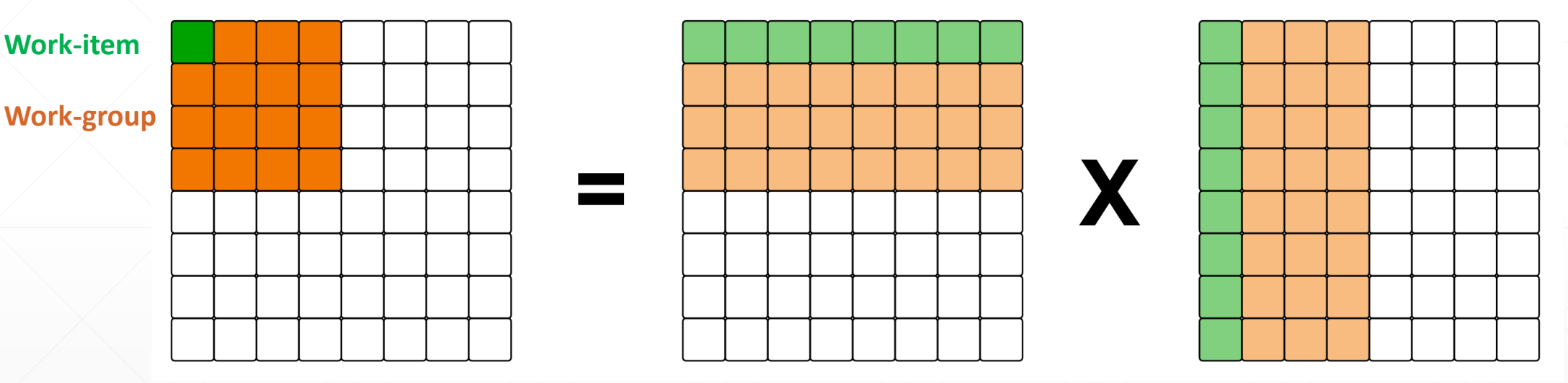
Matrix Multiplication: CPU 버전



Matrix Multiplication: CPU 버전

```
25      // naive matmul
26      for (int i = 0; i < N; i++)
27          for (int j = 0; j < N; j++)
28              for (int k = 0; k < N; k++)
29                  C[i*N+j] += A[i*N+k] * B[k*N+j];
30
31      // print top left 4x4 block of C
32      print_matrix(C.data(), N);
33
34      return 0;
35  }
36
37  void print_matrix(float *matrix, int n) {
38      for (int i = 0; i < fmin(N,4); i++) {
39          for (int j = 0; j < fmin(N,4); j++) {
40              printf ("%12.5f", matrix[i*n+j]);
41          }
42          printf ("\n");
43      }
44  }
```

Matrix Multiplication: ND-Range 커널

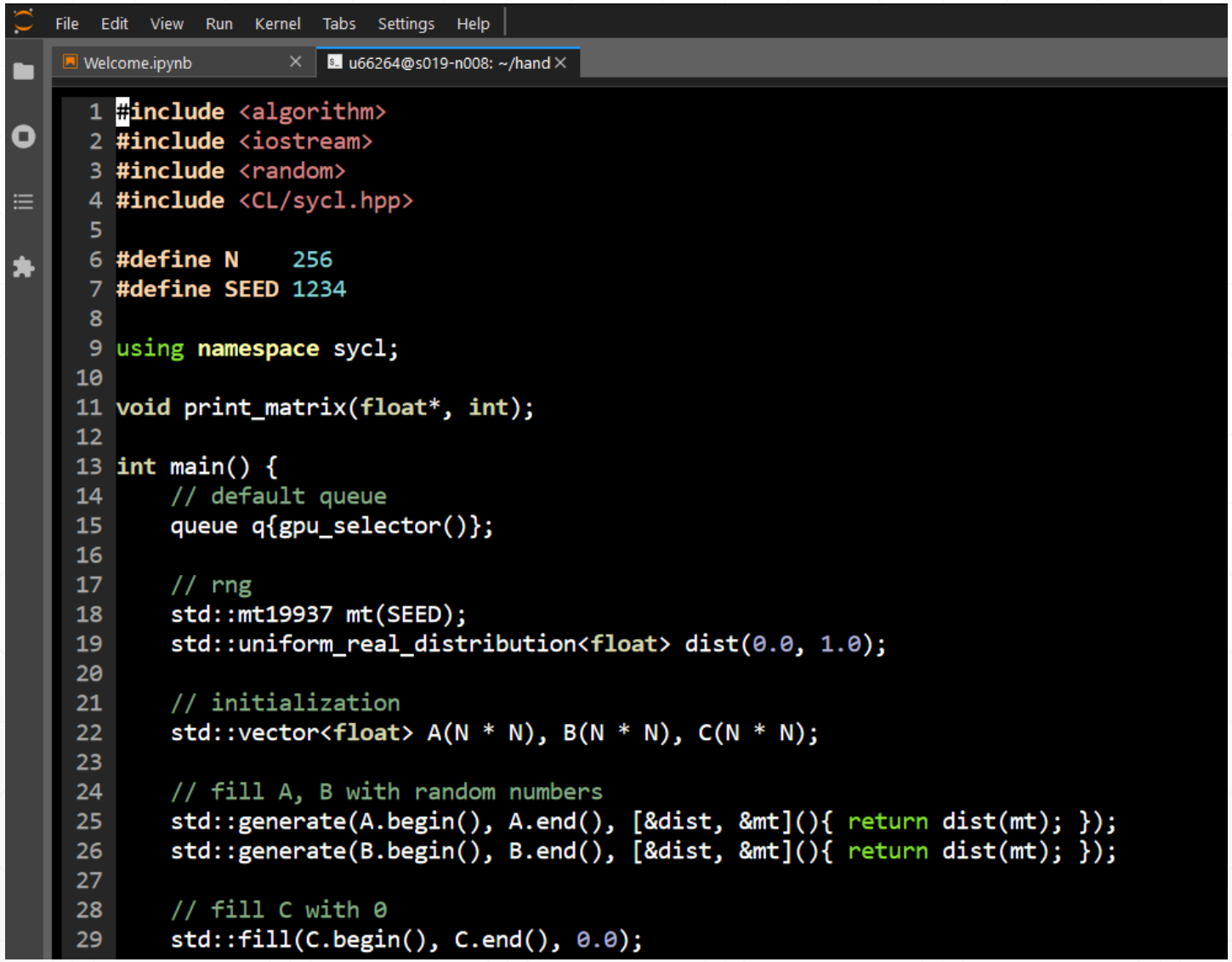


Matrix Multiplication: Buffer 버전

SYCL header

SYCL namespace

Default queue




```cpp
#include <algorithm>
#include <iostream>
#include <random>
#include <CL/sycl.hpp>

#define N    256
#define SEED 1234

using namespace sycl;

void print_matrix(float*, int);

int main() {
    // default queue
    queue q{gpu_selector()};

    // rng
    std::mt19937 mt(SEED);
    std::uniform_real_distribution<float> dist(0.0, 1.0);

    // initialization
    std::vector<float> A(N * N), B(N * N), C(N * N);

    // fill A, B with random numbers
    std::generate(A.begin(), A.end(), [&dist, &mt](){ return dist(mt); });
    std::generate(B.begin(), B.end(), [&dist, &mt](){ return dist(mt); });

    // fill C with 0
    std::fill(C.begin(), C.end(), 0.0);
```

Matrix Multiplication: Buffer 버전

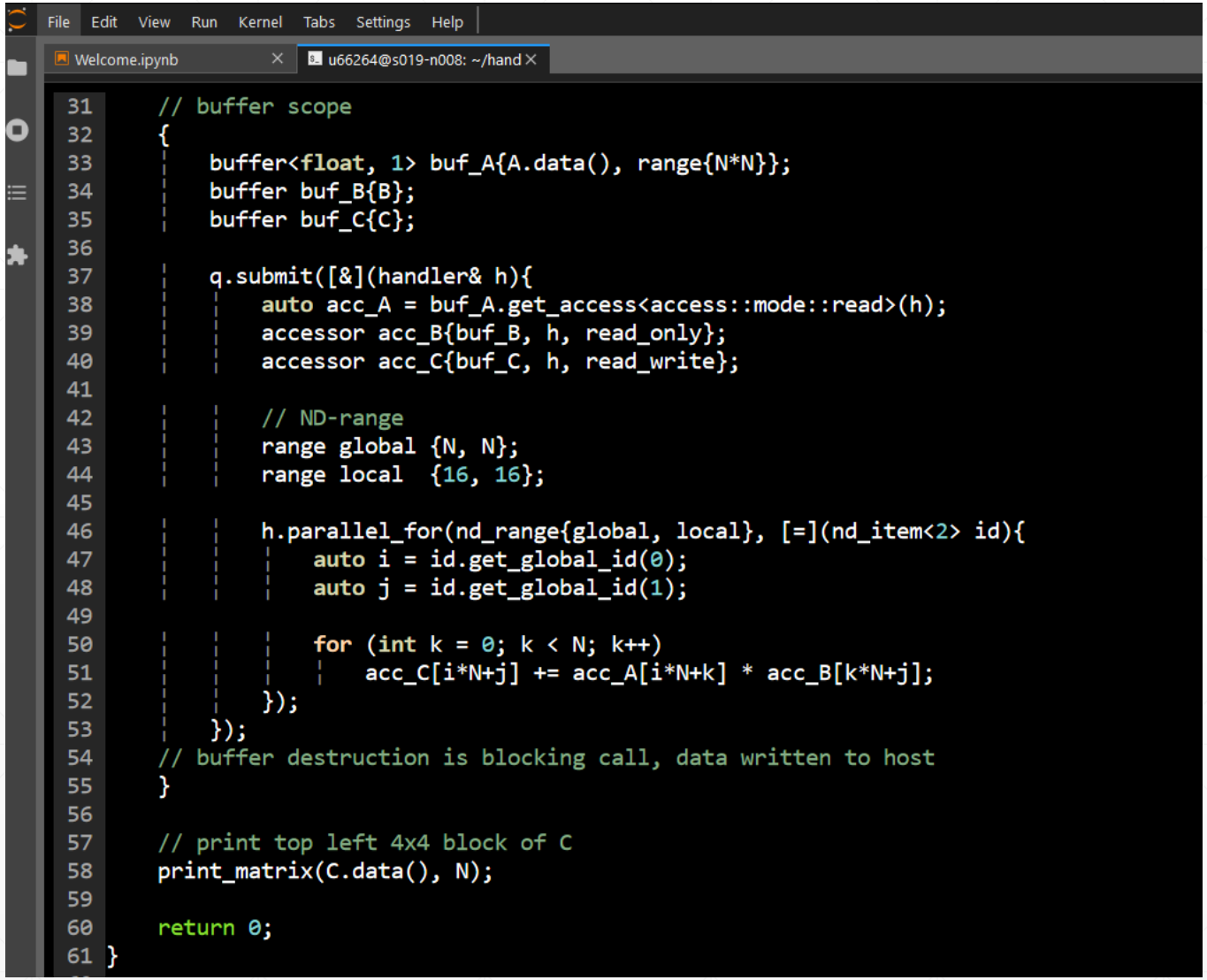
Buffer creation

Accessor creation

ND-range

Parallel for

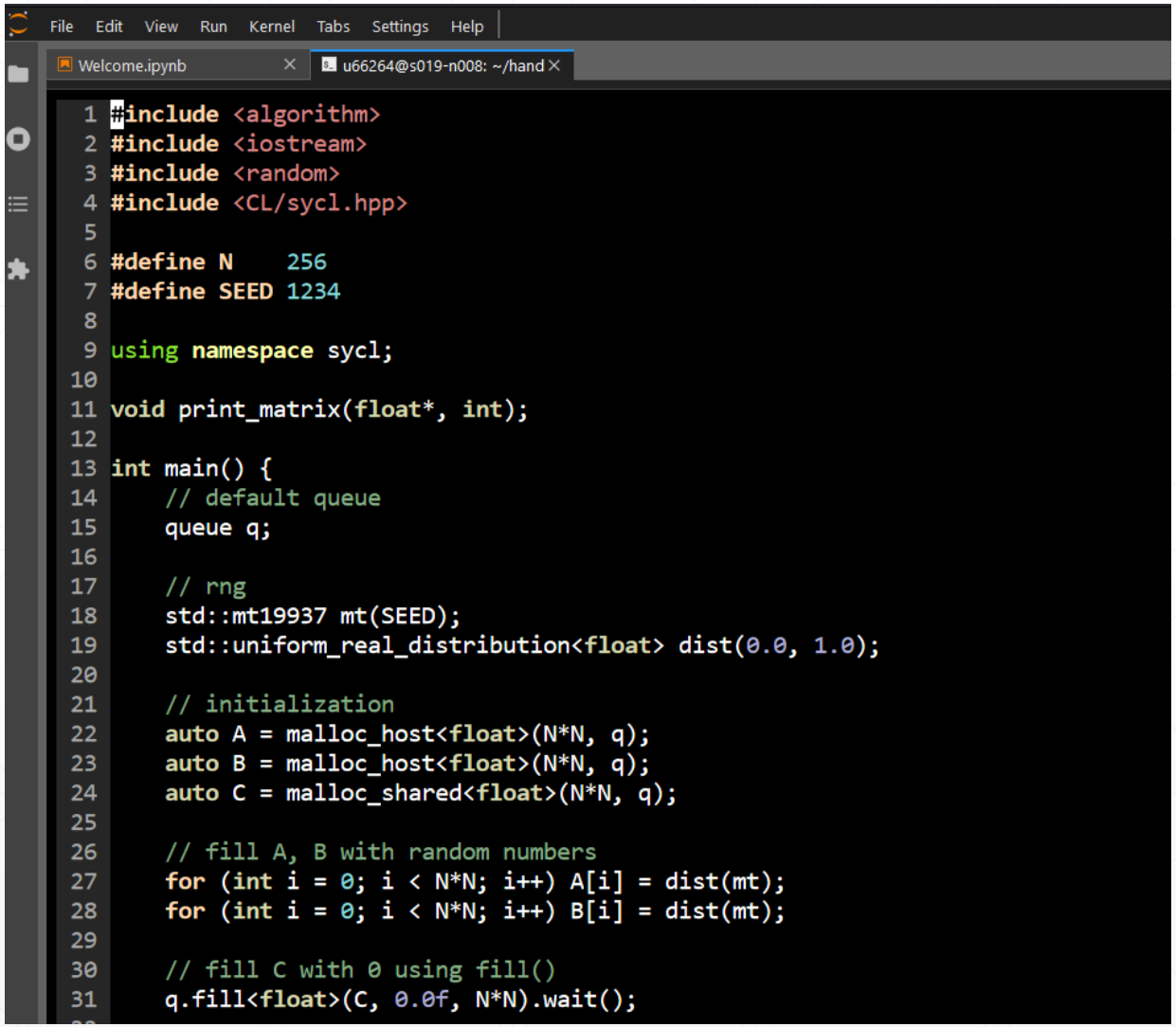
Data written to host




```
31      // buffer scope
32      {
33          buffer<float, 1> buf_A{A.data(), range{N*N}};
34          buffer buf_B{B};
35          buffer buf_C{C};
36
37          q.submit([&](handler& h){
38              auto acc_A = buf_A.get_access<access::mode::read>(h);
39              accessor acc_B{buf_B, h, read_only};
40              accessor acc_C{buf_C, h, read_write};
41
42              // ND-range
43              range global {N, N};
44              range local  {16, 16};
45
46              h.parallel_for(nd_range{global, local}, [=](nd_item<2> id){
47                  auto i = id.get_global_id(0);
48                  auto j = id.get_global_id(1);
49
50                  for (int k = 0; k < N; k++)
51                      acc_C[i*N+j] += acc_A[i*N+k] * acc_B[k*N+j];
52              });
53          });
54      // buffer destruction is blocking call, data written to host
55      }
56
57      // print top left 4x4 block of C
58      print_matrix(C.data(), N);
59
60      return 0;
61  }
```

Matrix Multiplication: USM 버전

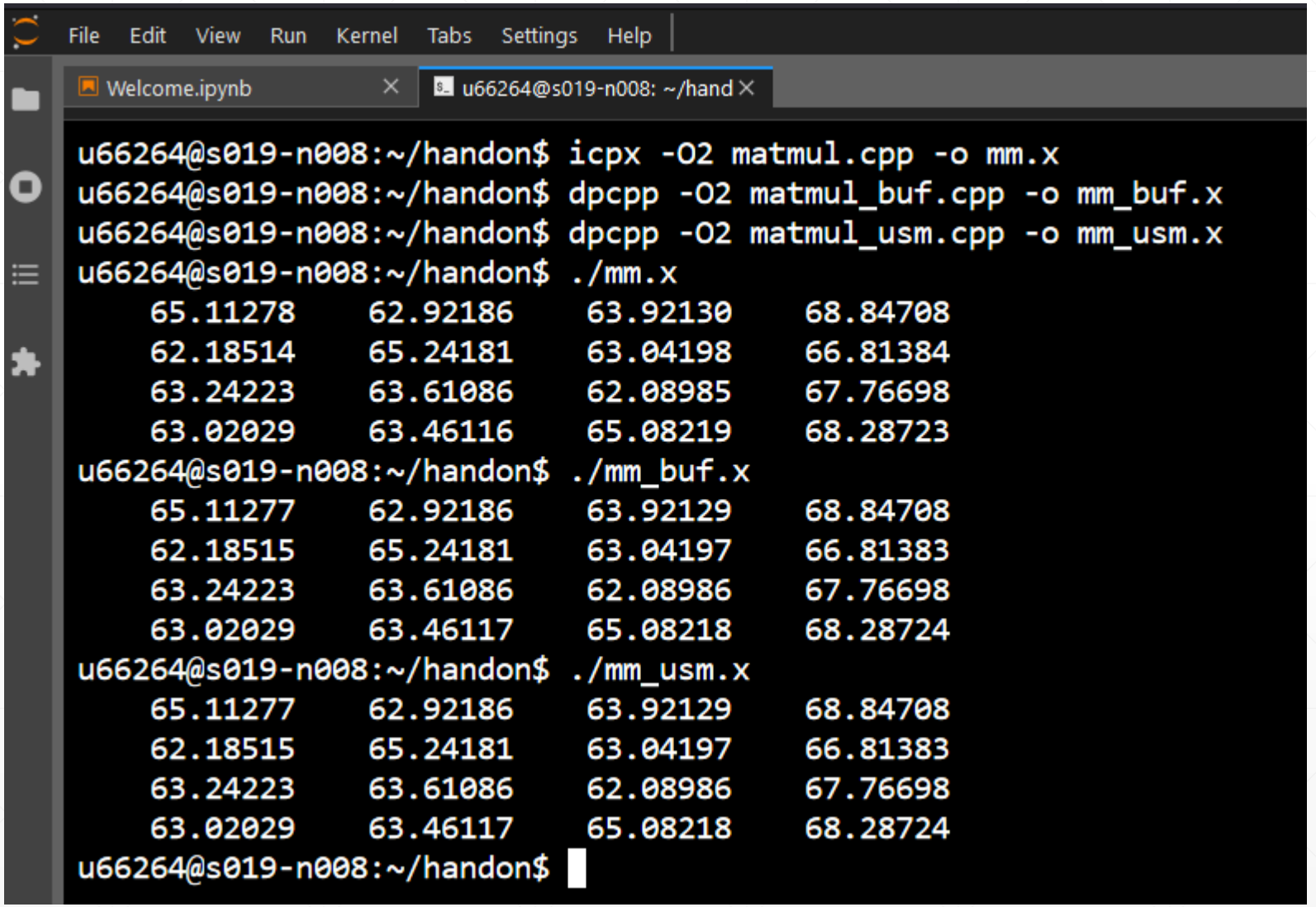
USM pointer allocation



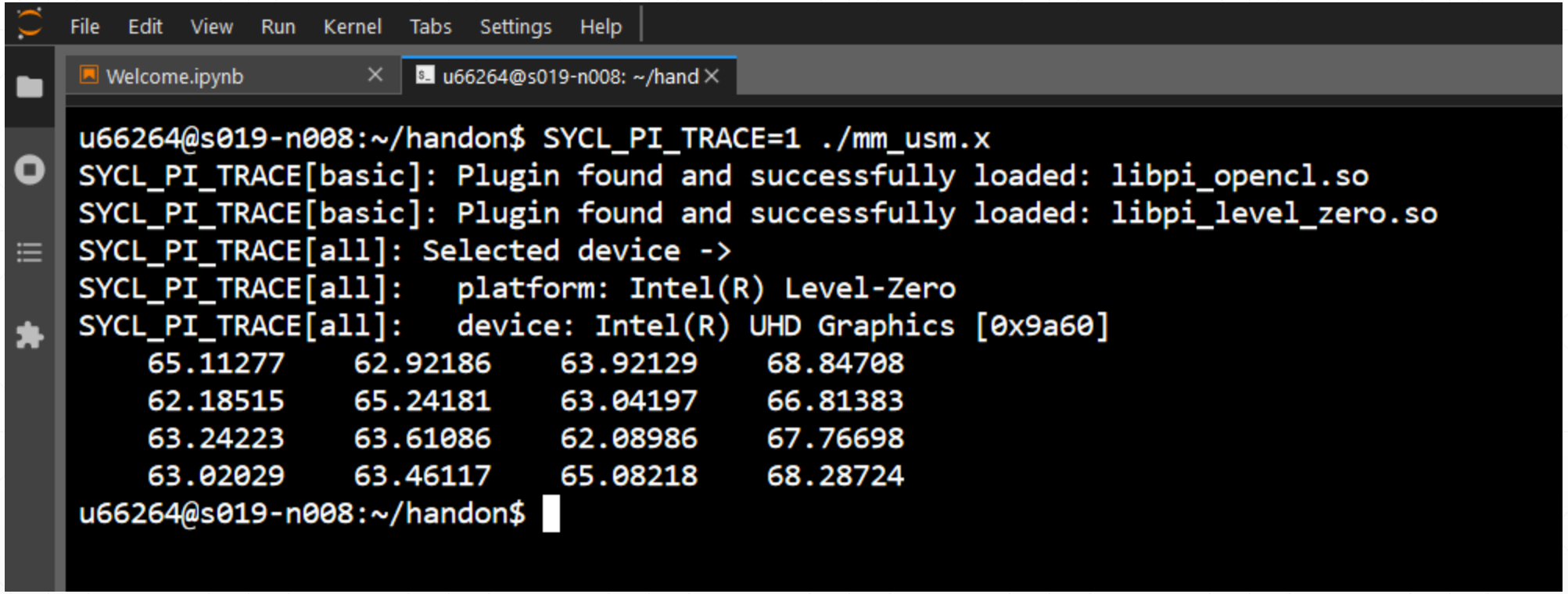
Matrix Multiplication: USM 버전

```
33      // ND-range
34      range global {N, N};
35      range local  {16, 16};
36
37      // queue shortcut
38      q.parallel_for(nd_range{global, local}, [=](nd_item<2> id){
39      ¦   auto i = id.get_global_id(0);
40      ¦   auto j = id.get_global_id(1);
41
42      ¦   for (int k = 0; k < N; k++)
43      ¦   ¦   C[i*N+j] += A[i*N+k] * B[k*N+j];
44      }).wait();
45
46      // print top left 4x4 block of C
47      print_matrix(C, N);
48
49      return 0;
50  }
```

Results


File Edit View Run Kernel Tabs Settings Help
Welcome.ipynb
u66264@s019-n008: ~/hand
u66264@s019-n008:~/handon$ icpx -O2 matmul.cpp -o mm.x
u66264@s019-n008:~/handon$ dpcpp -O2 matmul_buf.cpp -o mm_buf.x
u66264@s019-n008:~/handon$ dpcpp -O2 matmul_usm.cpp -o mm_usm.x
u66264@s019-n008:~/handon$ ./mm.x
65.11278 62.92186 63.92130 68.84708
62.18514 65.24181 63.04198 66.81384
63.24223 63.61086 62.08985 67.76698
63.02029 63.46116 65.08219 68.28723
u66264@s019-n008:~/handon$ ./mm_buf.x
65.11277 62.92186 63.92129 68.84708
62.18515 65.24181 63.04197 66.81383
63.24223 63.61086 62.08986 67.76698
63.02029 63.46117 65.08218 68.28724
u66264@s019-n008:~/handon$ ./mm_usm.x
65.11277 62.92186 63.92129 68.84708
62.18515 65.24181 63.04197 66.81383
63.24223 63.61086 62.08986 67.76698
63.02029 63.46117 65.08218 68.28724
u66264@s019-n008:~/handon$

SYCL Debug with SYCL_PI_TRACE



Device Selection with SYCL_DEVICE_SELECTOR