
# CUDA to SYCL Migration with Intel(R) oneAPI

## oneAPI – 가속 컴퓨팅을 개발하기 위한 스마트한 방식

MOASYS

2023. 10. 23.

# oneAPI 스마트한 방식 시리즈 (2023)

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

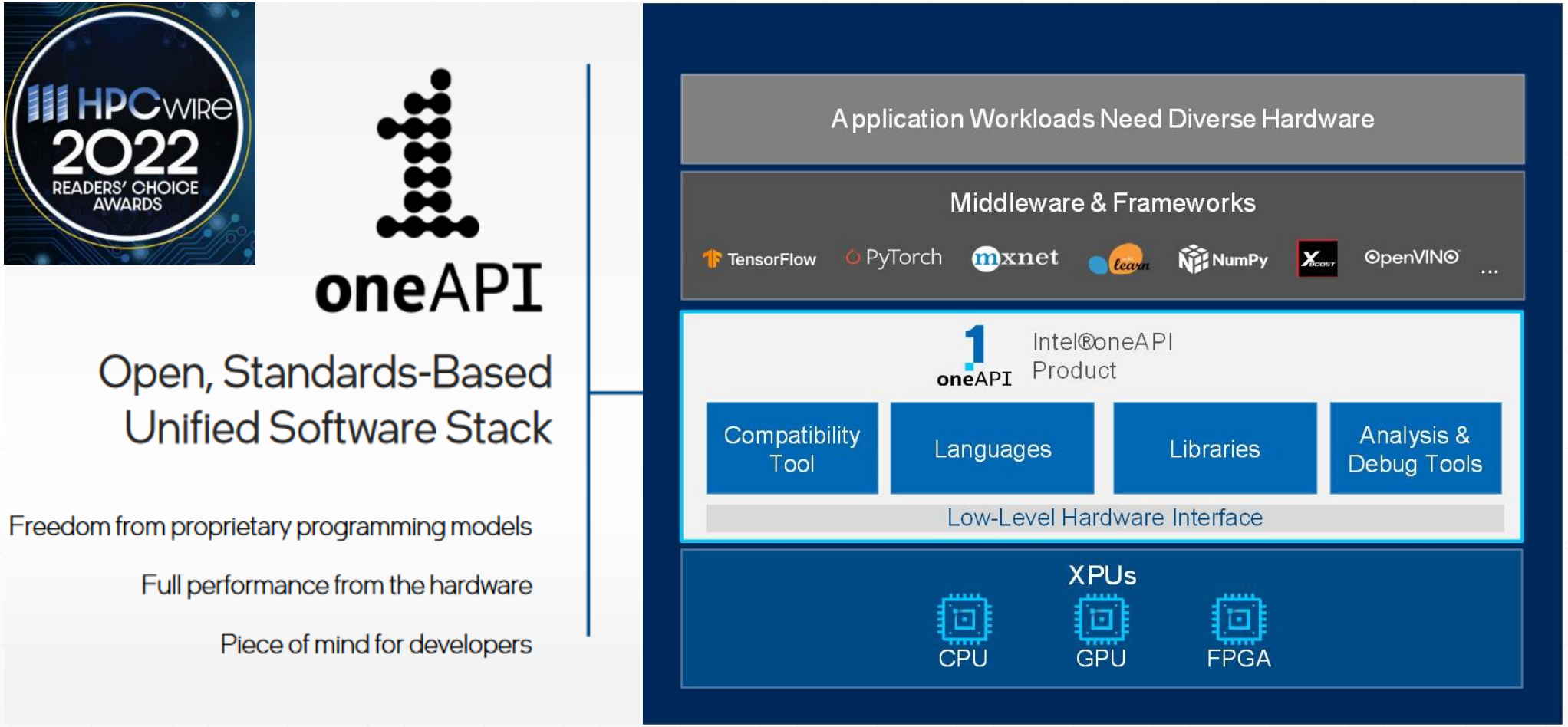
2. **Heterogenous Programming with oneAPI and DPC++(SYCL)**
   - https://www.allshowtv.com/detail.html?idx=1337

3. **CUDA to SYCL Migration with Intel(R) oneAPI**
   - https://www.allshowtv.com/detail.html?idx=1377

# 목차

- Overview of oneAPI
- Syntax and feature comparison-CUDA vs. DPC++
- Migration of CUDA codes with SYCLomatic
- New features of oneAPI 2023.2
- 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

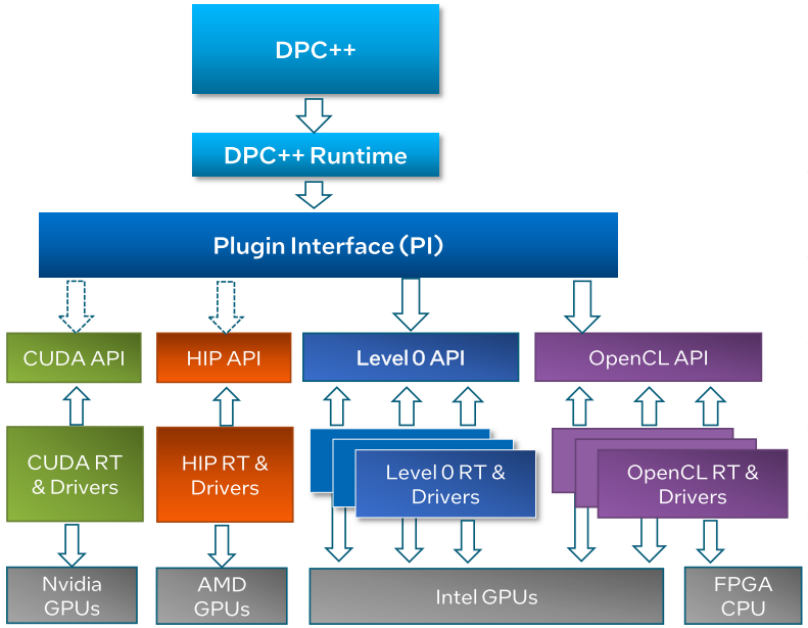
# 이기종 프로그래밍 모델 지원

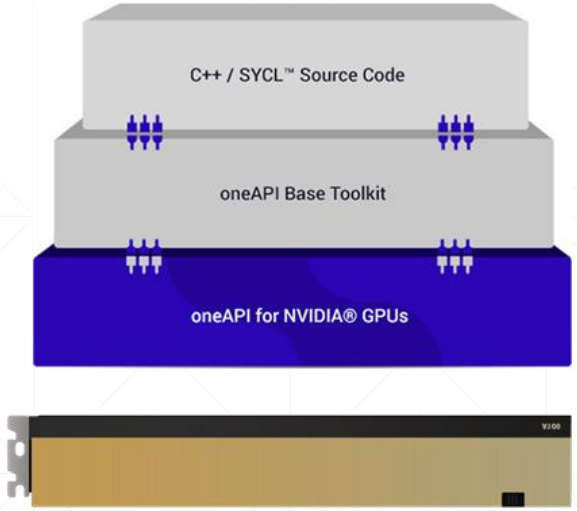
| | CUDA | HIP | OpenACC | OpenMP | DPC++/SYCL |
|---|---|---|---|---|---|
| Languages | C/C++/Fortran | C++/Fortran | C/C++/Fortran | C/C++/Fortran | C++ |
| Abstraction | Low | Low | High | High | Medium |
| Coding | - | - | Directive-based | Directive-based | C++ lambda |
| Parallelism | SIMT | SIMT | Fork-join<br>SIMD | Fork-join<br>SIMD | OpenCL |
| Offload | GPU<br>(NVIDIA) | GPU<br>(NVIDIA/AMD) | GPU<br>(NVIDIA) | CPU/GPU<br>(NVIDIA/AMD/Intel) | CPU/GPU/FPGA<br>(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 |

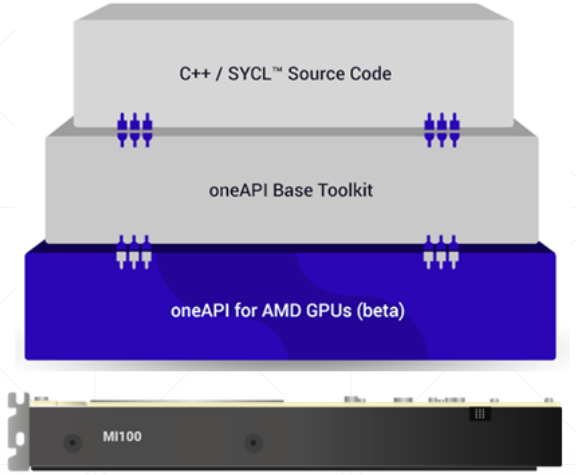
- **icc/icpc 와 ifort에서는**
  - GPU용 OpenMP*4.0/4.5의 오프로드 기능은 지원되지 않음
  - OpenMP*5.0/5.1/5.2 도 지원 지원되지 않음

- **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)

# SYCL 코드의 구조

```
#include <iostream>
#include <sycl/sycl.hpp>

using namespace sycl;

int main() {
    int data[1024];

    queue myQueue;

    {
        buffer<int, 1> resultBuf { data, range<1> { 1024 } };

        myQueue.submit([&](handler& cgh) {
            accessor writeResult { resultBuf, cgh, write_only, no_init };

            cgh.parallel_for(1024, [=](id<1> idx) {
                writeResult[idx] = idx;
            });
        });
    }

    for (int i = 0; i < 1024; i++)
      std::cout << "data[" << i << "] = " << data[i] << std::endl;

    return 0;
}
```

host

- sycl.hpp: Intel 제공된 SYCL 표준의 헤더 파일
- SYCL 이름은 SYCL 네임스페이스에 정의됨
- 호스트에 벡터 할당
- 성능이 가장 높은 디바이스를 암시적으로 선택
- 가속기 메모리에 버퍼 할당

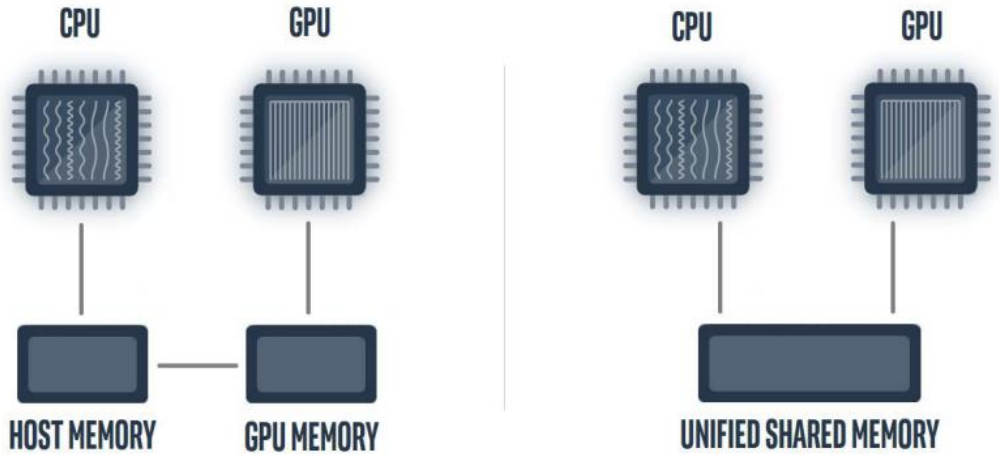
device

- 어떤 버퍼에 접근할 것인지 정의
- 계산 작업 병렬로 실행
- 가속기 메모리에 버퍼 할당 해제

host

- 호스트에 결과를 출력

# SYCL 코드의 구조: 2020 표준



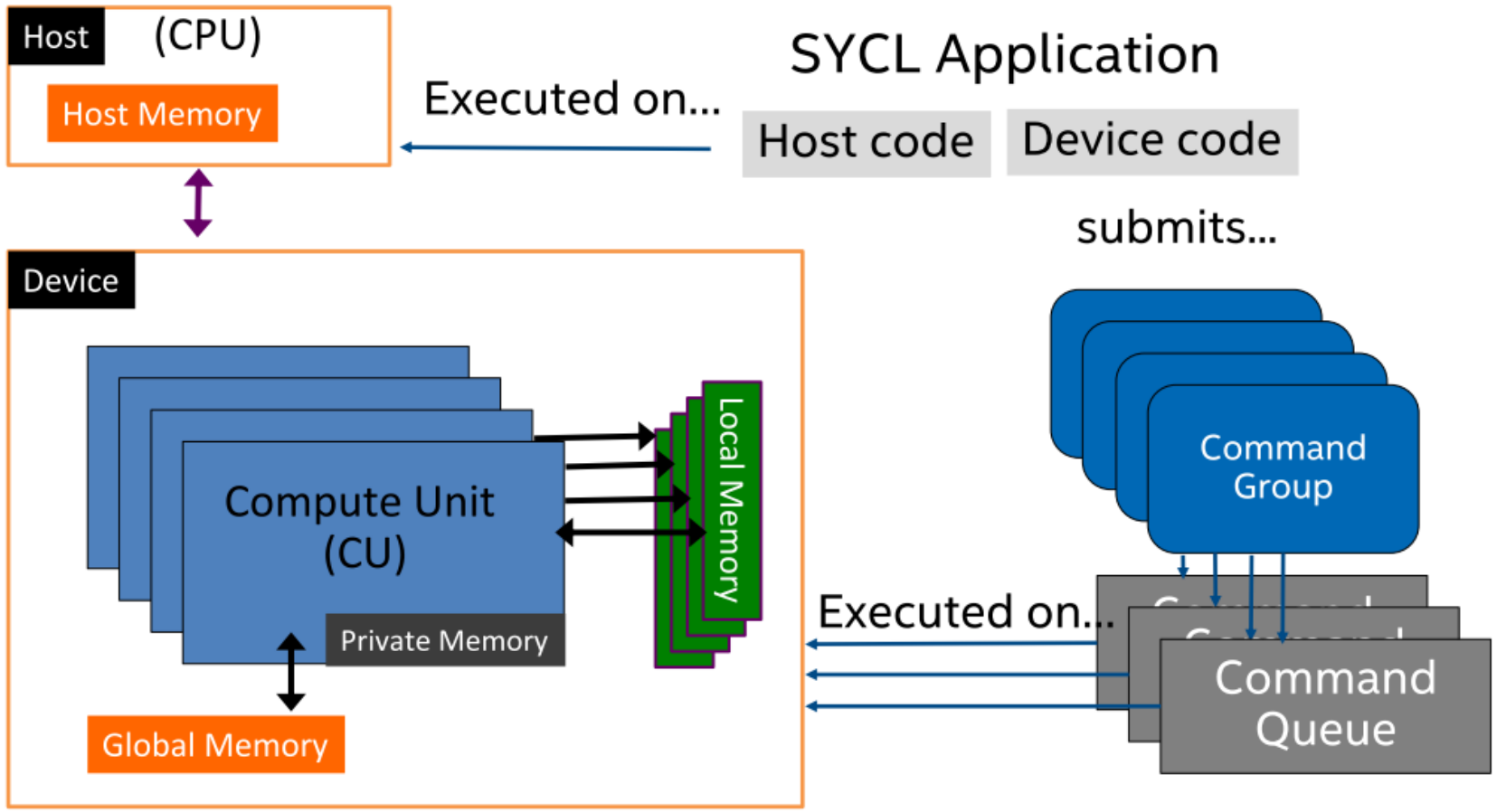

| Type | Description | Accessible on host? | Accessible on device? | Located on |
|---|---|---|---|---|
| device | Allocations in device memory | × | ✓ | device |
| host | Allocations in host memory | ✓ | ✓ | host |
| shared | Allocations shared between host and device | ✓ | ✓ | Can migrate between host and device |

- **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

```
int main() {
    queue myQueue;

    int *data = malloc_shared<int>(1000, myQueue);
```

# SYCL의 실행 모델



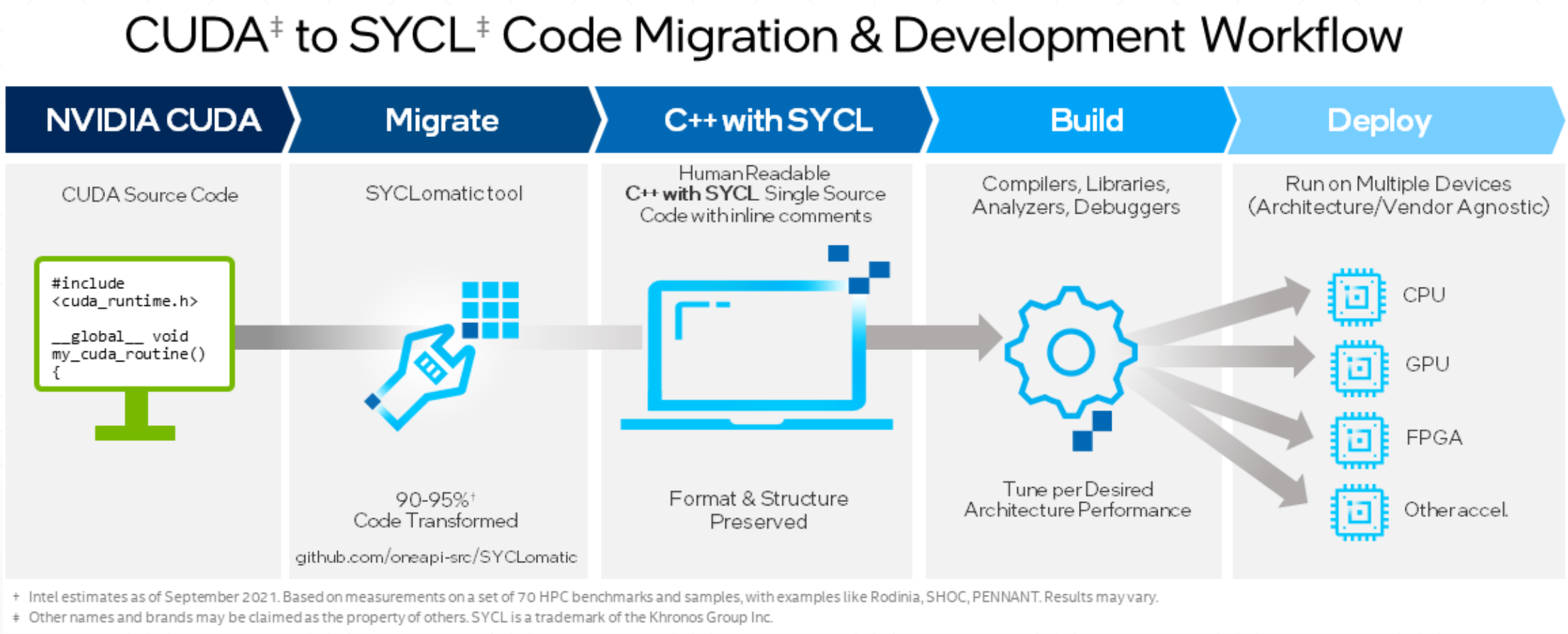

- CUDA 스트림은  in-order으로 진행됨
- SYCL 큐는 기본적으로 out-of-order으로 진행됨

# CUDA와 SYCL의 용어 차이점

| Concept | SYCL | CUDA |
|---|---|---|
| A function executed in parallel on device | Kernel | Kernel |
| Object used to execute kernels on a device | Queue | Stream |
| The N-dimensional parallel index space | ND-Range | Grid |
| A kernel instance in parallel index space | Work-Item | Thread |
| A group of kernel instances | Work-Group | Block |
| A group of kernel instances with additional communication and synchronization | Sub-Group | Warp |
| Memory used to exchange data among instances in a group | Local Memory | Shared Memory |
| Memory which is private to each kernel instance | Private Memory | Register |
| Function used to synchronize instances in a group | group_barrier() | __syncthreads()<br>__syncwarp() |

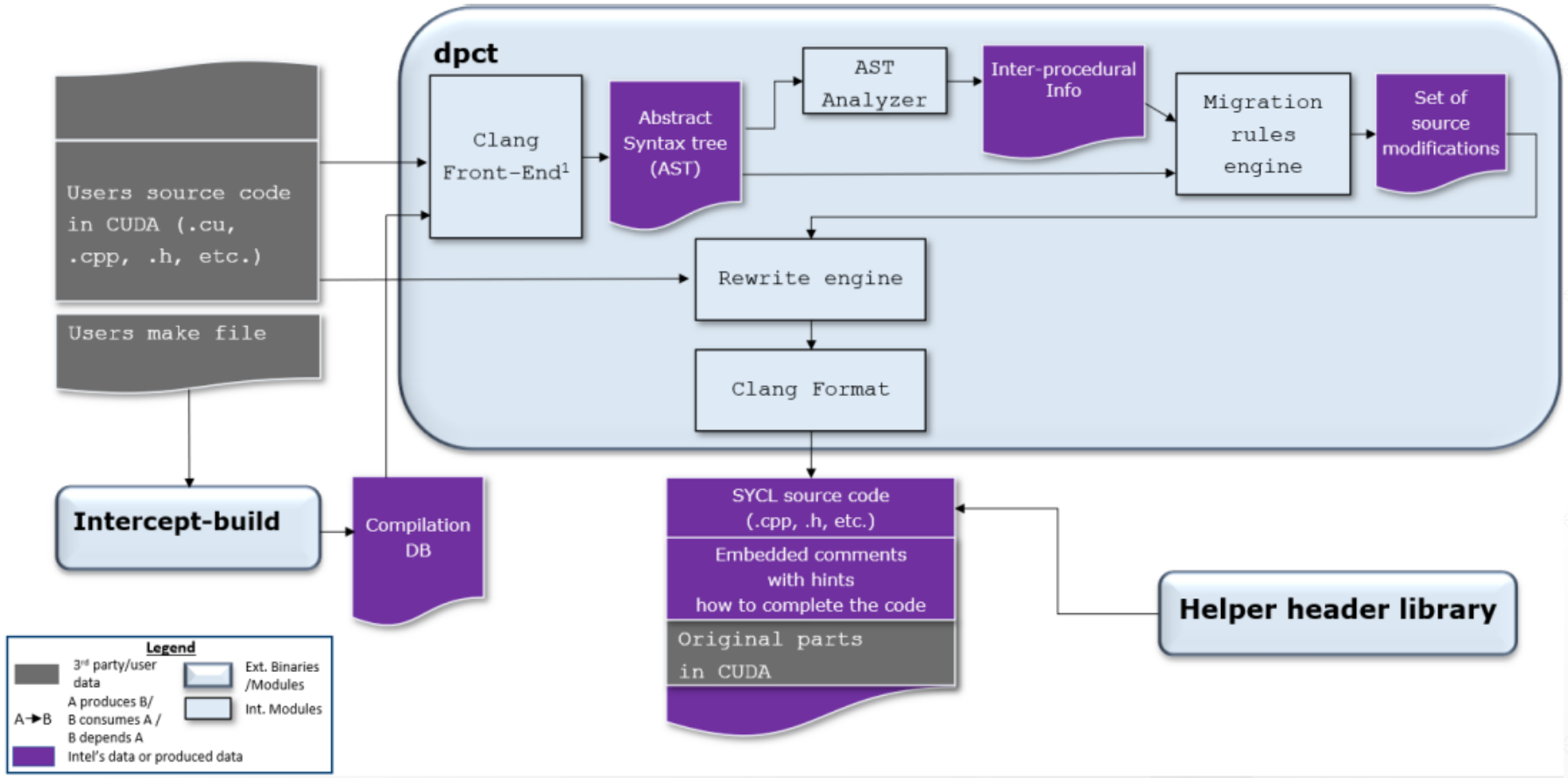
- Sub-group vs. Warp:
  - Warp의 크기는 모든 CUDA 디바이스에서 32로 구성됨.
  - Sub-group의 크기는 디바이스에 따라 다르며 최상의 성능을 위해 직접 최적화가 필요함.

# CUDA에서 SYCL로의 손쉬운 전환



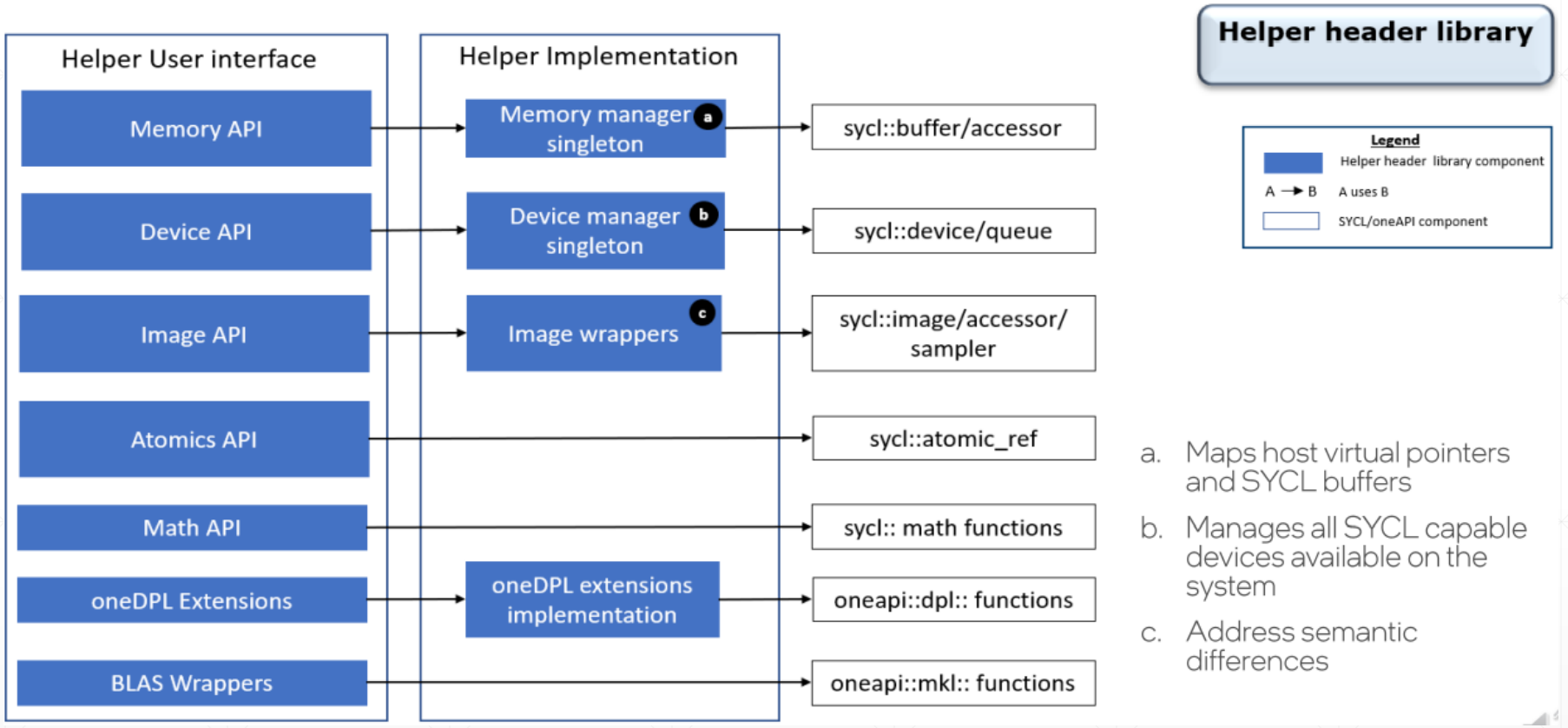

- 이미 CUDA*로 작성된 코드를 DPCPP로 전환하고자 하는 개발자를 지원하며, 사람이 해독할 수 있는 코드를 생성
  - (일반적으로 코드의 90~95%를 자동으로 마이그레이션 가능)
- 개발자가 애플리케이션 마이그레이션을 완료할 수있도록 인라인 코멘트를 제공

# SYCLomatic의 설계




- 단일 소스 프로젝트는 dpct로 마이그레이션 가능됨
- 여러 소스 프로젝트의 경우에는 intercept-build를 사용하여 Compilation DB 생성됨

# SYCLomatic의 Helper Library




- 일부 CUDA 기능은 SYCL에 동등한 구현이 없음
  - 헬퍼 라이브러리는 CUDA 복잡한 기능을 모방한 래퍼 클래스를 제공함

# SYCLomatic 예시

- 단일 소스 마이그레이션
  - Vector addition
  - dpct으로 진행
- 여러 소스 마이그레이션 및 Makefile
  - Jacobi interactive solution
  - 지원되지 않는 기능 해결: CUDA 그래프
  - intercept-build으로 진행
- cuBLAS 마이그레이션
  - Matrix multiplication
  - intercept-build으로 진행
- Hand-on:
  - SYCLomatic 설치
  - CodePlay's NVIDIA plugin 설치

# 단일 소스 마이그레이션: Vector Addition

```
#include <cuda.h>
#include <iostream>
#define N 16

__global__ void VectorAddKernel(float* A, float* B, float* C)
{
    C[threadIdx.x] = A[threadIdx.x] + B[threadIdx.x];
}

int main()
{
    float A[N] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
    float B[N] = {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2};
    float C[N] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    float *d_A, *d_B, *d_C;
    cudaMalloc(&d_A, N*sizeof(float));
    cudaMalloc(&d_B, N*sizeof(float));
    cudaMalloc(&d_C, N*sizeof(float));

    cudaMemcpy(d_A, A, N*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, N*sizeof(float), cudaMemcpyHostToDevice);

    VectorAddKernel<<<1, N>>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, N*sizeof(float), cudaMemcpyDeviceToHost);

    for (int i = 0; i < N; i++) std::cout<< C[i] << " ";
        std::cout << "\n";

    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
    return 0;
}
```

- 기본 사용법

```
$ dpct vector_add.cu

$ tree
.
├── dpct_output
│   ├── MainSourceFiles.yaml
│   └── vectoradd.dp.cpp
└── vectoradd.cu
```

- 사용자 정의된 디렉터리에서 마이그레이션

```
$ dpct --out-root sycl_code vector_add.cu

$ tree
.
├── sycl_code
│   ├── MainSourceFiles.yaml
│   └── vectoradd.dp.cpp
└── vectoradd.cu
```

# 단일 소스 마이그레이션: Vector Addition

```cuda
#include <cuda.h>
#include <iostream>
#define N 16

__global__ void VectorAddKernel(float* A, float* B, float* C)
{
    C[threadIdx.x] = A[threadIdx.x] + B[threadIdx.x];
}

int main()
{

    float A[N] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
    float B[N] = {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2};
    float C[N] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

```sycl
#include <sycl/sycl.hpp>
#include <dpct/dpct.hpp>
#include <iostream>
#define N 16

void VectorAddKernel(float* A, float* B, float* C,
                     const sycl::nd_item<3> &item_ct1)
{
        C[item_ct1.get_local_id(2)] =
            A[item_ct1.get_local_id(2)] + B[item_ct1.get_local_id(2)];
}

int main()
{
    sycl::device dev_ct1;
    sycl::queue q_ct1(
        dev_ct1, sycl::property_list{sycl::property::queue::in_order()});

    float A[N] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
    float B[N] = {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2};
    float C[N] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

- CUDA grids vs. SYCL ND-Range
  - CUDA는 그래픽 표존을 기반하여 첫 번째 차원은 빠르게 움직이는 차원
  - SYCL는 C++ 표존을 기반하여 마지막 차원은 빠르게 움직이는 차원
- CUDA 스트림 vs. SYCL 큐
  - SYCL큐는 in-order으로 생성됨

# 단일 소스 마이그레이션: Vector Addition

```
    float *d_A, *d_B, *d_C;
    cudaMalloc(&d_A, N*sizeof(float));
    cudaMalloc(&d_B, N*sizeof(float));
    cudaMalloc(&d_C, N*sizeof(float));

    cudaMemcpy(d_A, A, N*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, N*sizeof(float), cudaMemcpyHostToDevice);

    VectorAddKernel<<<1, N>>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, N*sizeof(float), cudaMemcpyDeviceToHost);

    for (int i = 0; i < N; i++) std::cout<< C[i] << " ";
        std::cout << "\n";

    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

```
    float *d_A, *d_B, *d_C;
    d_A = sycl::malloc_device<float>(N, q_ct1);
    d_B = sycl::malloc_device<float>(N, q_ct1);
    d_C = sycl::malloc_device<float>(N, q_ct1);

    q_ct1.memcpy(d_A, A, N * sizeof(float));
    q_ct1.memcpy(d_B, B, N * sizeof(float));

    q_ct1.parallel_for(
       sycl::nd_range<3>(sycl::range<3>(1, 1, N), sycl::range<3>(1, 1, N)),
       [=](sycl::nd_item<3> item_ct1) {
           VectorAddKernel(d_A, d_B, d_C, item_ct1);
    });

    q_ct1.memcpy(C, d_C, N * sizeof(float)).wait();

    for (int i = 0; i < N; i++) std::cout<< C[i] << " ";
         std::cout << "\n";

    sycl::free(d_A, q_ct1);
    sycl::free(d_B, q_ct1);
    sycl::free(d_C, q_ct1);

    return 0;
}
```

- USM으로 마이그레이션 됨:
  - 버퍼 API: --usm-level=none
- 커널 실행
  - SYCL은 C++표준의 lambda 기능으로 커널 실행됨

# 여러 소스 마이그레이션 : Jacobi Iterative Solver

- Jacobi Iterative Solver 및 CUDA 그래프
  - https://github.com/NVIDIA/cuda-samples/tree/master/Samples/3_CUDA_Features/jacobiCudaGraphs

- main.cpp:
  - CUDA 스트림
  - 설정GPU의 메모리 할당
  - CPU에서 데이터 초기화
  - 계산을 위해 GPU 메모리로 데이터 복사
  - GPU에서 계산 실행
  - 결과 확인 및 출력

- jacobi.cu:
  - 커널 정의
  - 공유 로컬 메모리 할당
  - 스레드 블록 분할을 위한 협력 그룹
  - 워프 프리미티브 사용
  - 타일의 합계 값 합산을 위한 atomic 사용

# 여러 소스 마이그레이션 : Jacobi Iterative Solver

- Compilation DB 생성

```
$ make clean
$ intercept-buid make
```

- *compile_commands.json*

```
[
    {
        "command": "nvcc -c  -I../../../Common -m64 --std=c++11 -o jacobi.o -D__CUDACC__=1 jacobi.cu",
        "directory": "/scratch/optpar01/work/2023/cuda-samples-11.6/Samples/3_CUDA_Features/jacobiCudaGraphs",
        "file": "/scratch/optpar01/work/2023/cuda-samples-11.6/Samples/3_CUDA_Features/jacobiCudaGraphs/jacobi.cu"
    },
    {
        "command": "nvcc -c  -I../../../Common -m64 --std=c++11 -o main.o -D__CUDACC__=1 main.cpp",
        "directory": "/scratch/optpar01/work/2023/cuda-samples-11.6/Samples/3_CUDA_Features/jacobiCudaGraphs",
        "file": "/scratch/optpar01/work/2023/cuda-samples-11.6/Samples/3_CUDA_Features/jacobiCudaGraphs/main.cpp"
    },
    {
        "command": "nvcc  -m64 -o jacobiCudaGraphs jacobi.o main.o -D__CUDACC__=1",
        "directory": "/scratch/optpar01/work/2023/cuda-samples-11.6/Samples/3_CUDA_Features/jacobiCudaGraphs"
    }
]
```

- 실제 마이그레이션

```
$ dpct -p compile_commands.json --use-experimental-features=logical-group
```

# Jacobi Iterative Solver: API 비교

| Concept | CUDA | SYCL |
|---|---|---|
| header file | #include <cuda_runtime.h> | #include <CL/sycl.hpp><br>#include <dpct/dpct.hpp> |
| Memory allocation on host | cudaMallocHost(&b, N_ROWS * sizeof(double)) | sycl::malloc_host<double>(N_ROWS, q_ct1) |
| Create CUDA Stream / SYCL Queue | cudaStreamCreateWithFlags(&stream1, cudaStreamNonBlocking) | dpct::device_ext &dev_ct1 = dpct::get_current_device();<br>stream1 = dev_ct1.create_queue() |
| Memset on device | cudaMemsetAsync(d_x, 0, sizeof(double) * N_ROWS, stream1) | stream1->memset(d_x_new, 0, sizeof(double) * N_ROWS) |
| Copy memory between host and device | cudaMemcpyAsync(<br>d_A, A, sizeof(float) * N_ROWS * N_ROWS,<br>cudaMemcpyHostToDevice, stream1) | stream1->memcpy(<br>d_A, A, sizeof(float) * N_ROWS * N_ROWS) |
| Free device memory allocation | cudaFree(d_A) | sycl::free(d_A, q) |
| Synchronize host and device | cudaStreamSynchronize(stream) | stream1->wait(); |
| CUDA sync threads / SYCL group_barrier | cg::sync(cta); | item_ct1.barrier(); |
| CUDA Cooperative groups / SYCL sub-groups | cg::thread_block_tile<32> tile32 = cg::tiled_partition<32>(cta); | sycl::sub_group tile32 = item_ct1.get_sub_group(); |
| CUDA Warp function / SYCL group algorithms | tile32.shfl_down(rowThreadSum, offset); | sycl::shift_group_left(tile32,rowThreadSum, offset); |
| CUDA Atomic Add / SYCL Atomic Add | atomicAdd(<br>&b_shared[i % (ROWS_PER_CTA + 1)], rowThreadSum); | dpct::atomic_fetch_add<double, sycl::access::address_space::generic_space>(<br>&b_shared[i % (ROWS_PER_CTA + 1)], rowThreadSum); |

- **dpct::device**는 CUDA 스트림을 모방한 디바이스 선택 및 큐 생성 래퍼 클래스임

# 지원되지 않는 기능 해결

- SYCLomatic의 diagnostic 메시지

```
/*
DPCT1007:16: Migration of cudaGraphCreate is not supported.
*/
checkCudaErrors(cudaGraphCreate(&graph, 0));

/*
DPCT1007:17: Migration of cudaGraphAddMemsetNode is not supported.
*/
checkCudaErrors(
    cudaGraphAddMemsetNode(&memsetNode, graph, NULL, 0, &memsetParams));

/*
DPCT1082:18: Migration of cudaKernelNodeParams type is not supported.
*/
cudaKernelNodeParams NodeParams0, NodeParams1;

/*
DPCT1007:19: Migration of cudaGraphAddKernelNode is not supported.
*/
checkCudaErrors(
    cudaGraphAddKernelNode(&jacobiKernelNode, graph, nodeDependencies.data(),
    nodeDependencies.size(), &NodeParams0));
```

# CUDA API 마이그레이션 지원 현황

## nvGRAPH API

| Function | Migration Support | Diagnostic Message |
|---|---|---|
| nvgraphGetProperty | NO | |
| nvgraphCreate | NO | |
| nvgraphDestroy | NO | |
| nvgraphCreateGraphDescr | NO | |
| nvgraphDestroyGraphDescr | NO | |
| nvgraphSetGraphStructure | NO | |
| nvgraphGetGraphStructure | NO | |
| nvgraphConvertTopology | NO | |
| nvgraphConvertGraph | NO | |
| nvgraphAllocateEdgeData | NO | |
| nvgraphSetEdgeData | NO | |
| nvgraphGetEdgeData | NO | |
| nvgraphAllocateVertexData | NO | |
| nvgraphSetVertexData | NO | |
| nvgraphGetVertexData | NO | |

## cuBLAS API

| Function | Migration Support | Diagnostic Message |
|---|---|---|
| cublasCreate | YES | |
| cublasDestroy | YES | |
| cublasGetVersion | YES | |
| cublasGetProperty | NO | |
| cublasSetStream | YES | |
| cublasGetStream | YES | |
| cublasGetPointerMode | YES | DPCT1026 / DPCT1027 |
| cublasSetPointerMode | YES | DPCT1026 / DPCT1027 |
| cublasSetVector | YES | DPCT1018 / DPCT1020 |
| cublasGetVector | YES | DPCT1018 / DPCT1020 |
| cublasSetMatrix | YES | DPCT1018 / DPCT1020 |
| cublasGetMatrix | YES | DPCT1018 / DPCT1020 |
| cublasSetVectorAsync | YES | DPCT1018 / DPCT1020 |
| cublasGetVectorAsync | YES | DPCT1018 / DPCT1020 |
| cublasSetMatrixAsync | YES | DPCT1018 / DPCT1020 |

https://oneapi-src.github.io/SYCLomatic/dev_guide/api-mapping-status.html#runtime-and-driver-api

# 지원되지 않는 기능 해결

- CUDA 그래프 구현

```
double sumGPU = 0.0;
if (gpumethod == 0) {
    sumGPU = JacobiMethodGpuCudaGraphExecKernelSetParams(d_A, d_b, conv_threshold, max_iter, d_x, d_x_new, stream1);
  } else if (gpumethod == 1) {
    sumGPU = JacobiMethodGpuCudaGraphExecUpdate(d_A, d_b, conv_threshold, max_iter, d_x, d_x_new, stream1);
  } else if (gpumethod == 2) {
    sumGPU = JacobiMethodGpu(d_A, d_b, conv_threshold, max_iter, d_x, d_x_new, stream1);
  }
```

- 해결

```
double sumGPU = 0.0;
sumGPU = JacobiMethodGpu(d_A, d_b, conv_threshold, max_iter, d_x, d_x_new, stream1);
```

- 기능 정의 삭제

```
//double JacobiMethodGpuCudaGraphExecKernelSetParams(
//    const float *A, const double *b, const float conv_threshold,
//    const int max_iter, double *x, double *x_new, dpct::queue_ptr stream) {
//...
```

```
//double JacobiMethodGpuCudaGraphExecUpdate(
//    const float *A, const double *b, onst float conv_threshold,
//    const int max_iter, double *x, double *x_new, dpct::queue_ptr stream) {
//...
```

# Diagnostics Reference

| ID | Message |
|---|---|
| DPCT1000 | Error handling `if-stmt` was detected but could not be rewritten. |
| DPCT1001 | The statement could not be removed. |
| DPCT1002 | Special case error handling `if-stmt` was detected. You may need to rewrite this code. |
| DPCT1003 | Migrated API does not return error code. `(*, 0)` is inserted. You may need to rewrite this code. |
| DPCT1004 | REMOVED.<br>Compatible SYCL code could not be generated. |
| DPCT1005 | The SYCL device version is different from CUDA Compute Compatibility. You may need to rewrite this code. |
| DPCT1006 | SYCL does not provide a standard API to differentiate between integrated and discrete GPU devices. |
| DPCT1007 | Migration of *<API name>* is not supported. |
| DPCT1008 | `<function name>` function is not defined in SYCL. This is a hardware-specific feature. Consult with your hardware vendor to find a replacement. |
| DPCT1009 | SYCL uses exceptions to report errors and does not use the error codes. The original code was commented out and a warning string was inserted. You need to rewrite this code. |
| DPCT1010 | SYCL uses exceptions to report errors and does not use the error codes. The call was replaced with 0. You need to rewrite this code. |
| DPCT1011 | The tool detected overloaded operators for built-in vector types, which may conflict with the SYCL 2020 standard operators (see 4.14.2.1 Vec interface). The tool inserted a namespace to avoid the conflict. Use SYCL 2020 standard operators instead. |

## DPCT1003

### Message

Migrated API does not return error code. `(*, 0)` is inserted. You may need to rewrite this code.

### Detailed Help

Typically, this happens because the CUDA* API returns an error code and then it is consumed by the program logic.

SYCL* uses exceptions to report errors and does not return the error code.

SYCLomatic inserts a `(*, 0)` operator, so that the resulting application can be compiled. This operator returns 0 and is inserted if the return code is expected by the program logic and the new API does not return it. You should review all such places in the code.

### Suggestions to Fix

If in a DPC++ application you:

- Do not need the code that consumes the error code, remove the code and the `(*, 0)` operator.
- Need the code that consumes the error code, try to replace it with an exception handling code and use your logic in an exception handler.

https://oneapi-src.github.io/SYCLomatic/dev_guide/diagnostics-reference.html

# cuBLAS 마이그레이션

- Matrix Multiplication
  - https://github.com/NVIDIA/cuda-samples/tree/master/Samples/4_CUDA_Libraries/matrixMulCUBLAS

- main.cpp:
  - CUDA 스트림 설정
  - GPU의 메모리 할당
  - CPU에서 데이터 초기화
  - 계산을 위해 데이터를 GPU 메모리에 복사
  - GPU에서 계산 실행
  - 결과 확인 및 출력

# cuBLAS Migration

- Compilation DB 생성

```
$ make clean
$ intercept-buid make
```

- *compile_commands.json*

```
[
    {
        "command": "nvcc -c  -I../../../Common -m64 --std=c++11 -o matrixMulCUBLAS.o -D__CUDACC__=1 matrixMulCUBLAS.cpp",
        "directory": "/scratch/optpar01/work/2023/cuda-samples-11.6/Samples/4_CUDA_Libraries/matrixMulCUBLAS",
        "file": "/scratch/optpar01/work/2023/cuda-samples-11.6/Samples/4_CUDA_Libraries/matrixMulCUBLAS/matrixMulCUBLAS.cpp"
    },
    {
        "command": "nvcc  -m64 -o matrixMulCUBLAS matrixMulCUBLAS.o -D__CUDACC__=1",
        "directory": "/scratch/optpar01/work/2023/cuda-samples-11.6/Samples/4_CUDA_Libraries/matrixMulCUBLAS"
    }
]
```

- 실제 마이그레이션

```
$ dpct -p compile_commands.json
```

# matrixMulCUBLAS: API Comparison

| Concept | CUDA | SYCL |
|---|---|---|
| header file | #include <cuda_runtime.h> | #include <CL/sycl.hpp><br>#include <dpct/dpct.hpp> |
| library header file | #include <cublas_v2.h> | #include <dpct/blas_utils.hpp> |
| Memory allocation on device | cudaMalloc((void **)&d_A, mem_size_A) | d_A = (float *)sycl::malloc_device(<br>mem_size_A, dpct::get_default_queue()) |
| Copy memory between host and device | cudaMemcpy(d_A, h_A, mem_size_A, cudaMemcpyHostToDevice) | dpct::get_default_queue().memcpy(<br>d_A, h_A, mem_size_A).wait() |
| Free device memory allocation | free(h_A) | sycl::free(h_A, q) |
| gemm function | cublasSgemm(...) | oneapi::mkl::blas::column_major::gemm(...) |

- cuBLAS gemm

```
checkCudaErrors(cublasSgemm(
        handle, CUBLAS_OP_N, CUBLAS_OP_N, matrix_size.uiWB, matrix_size.uiHA,
        matrix_size.uiWA, &alpha, d_B, matrix_size.uiWB, d_A,
        matrix_size.uiWA, &beta, d_C, matrix_size.uiWB));
```

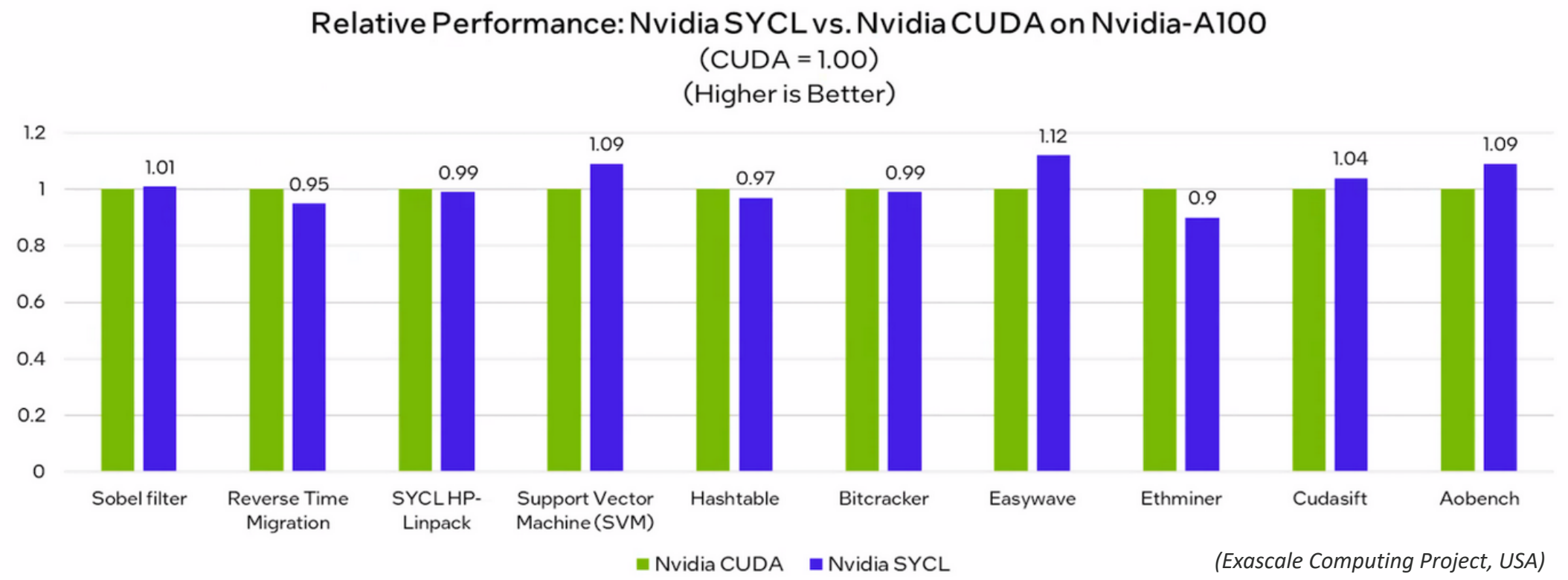
- oneMKL gemm

```
oneapi::mkl::blas::column_major::gemm(
             *handle, oneapi::mkl::transpose::nontrans, oneapi::mkl::transpose::nontrans, matrix_size.uiWB, matrix_size.uiHA,
             matrix_size.uiWA, alpha, d_B, matrix_size.uiWB,
             d_A, matrix_size.uiWA, beta, d_C, matrix_size.uiWB)
```

# 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

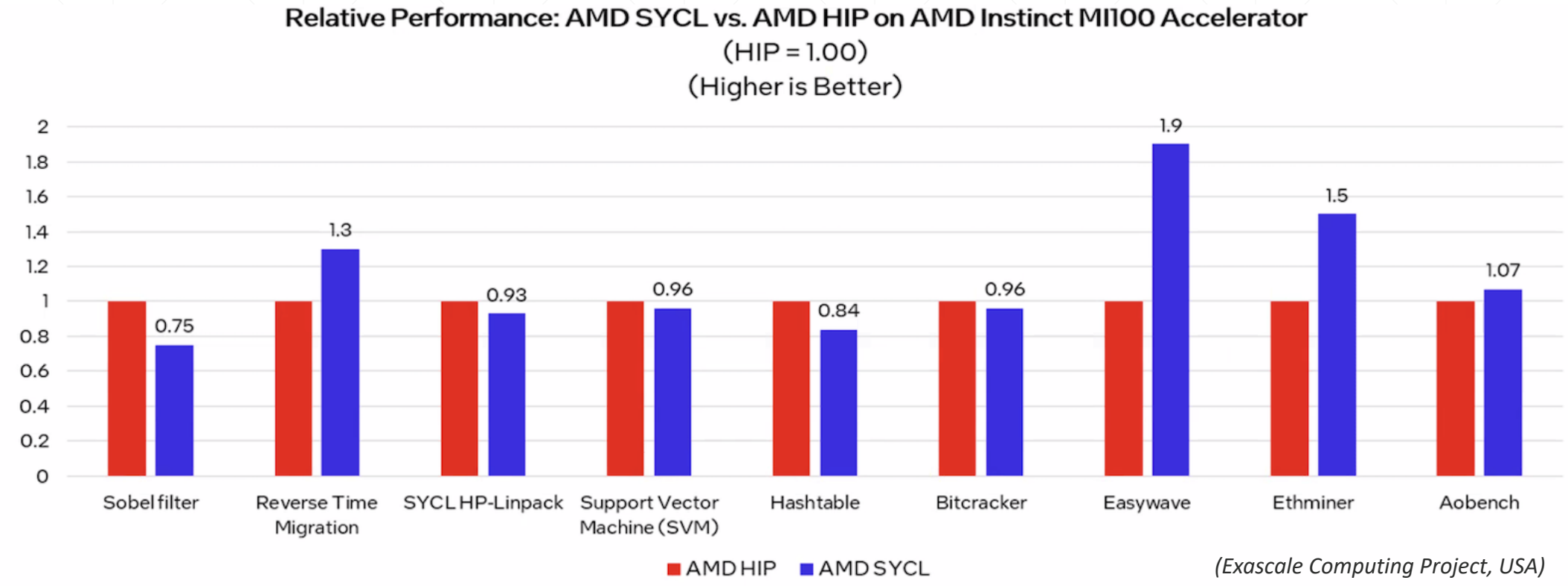
# Intel® DPC++ 컴파일러: CUDA 백엔드




*(Exascale Computing Project, USA)*

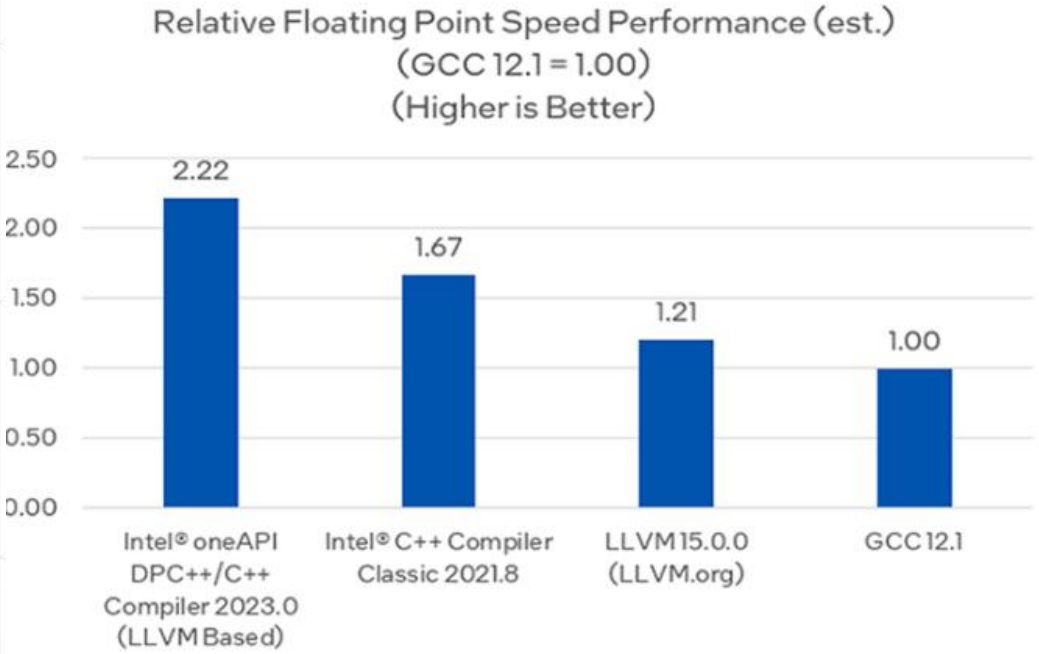
- **테스트 환경:**
  - Intel® Xeon® Platinum 8360Y / NVIDIA A100-PCIe-80GB
  - CUDA 11.7
  - NVIDIA GPU의 네이티브 CUDA*에 필적하는 성능

# Intel® DPC++ 컴파일러: HIP 백엔드

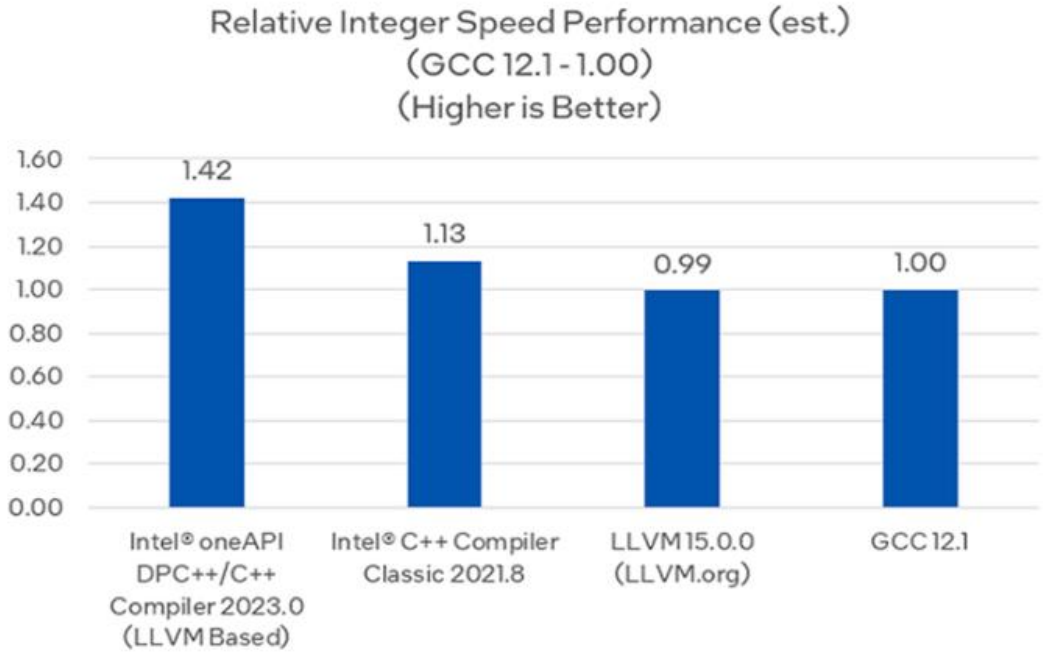



- **테스트 환경:**
  - Intel® Xeon® Gold 6330 / AMD Instinct MI100
  - RoCm 5.2.1
  - SYCL 2020 기능의 50% 이상을 구현

# C++ 애플리케이션 성능 향상



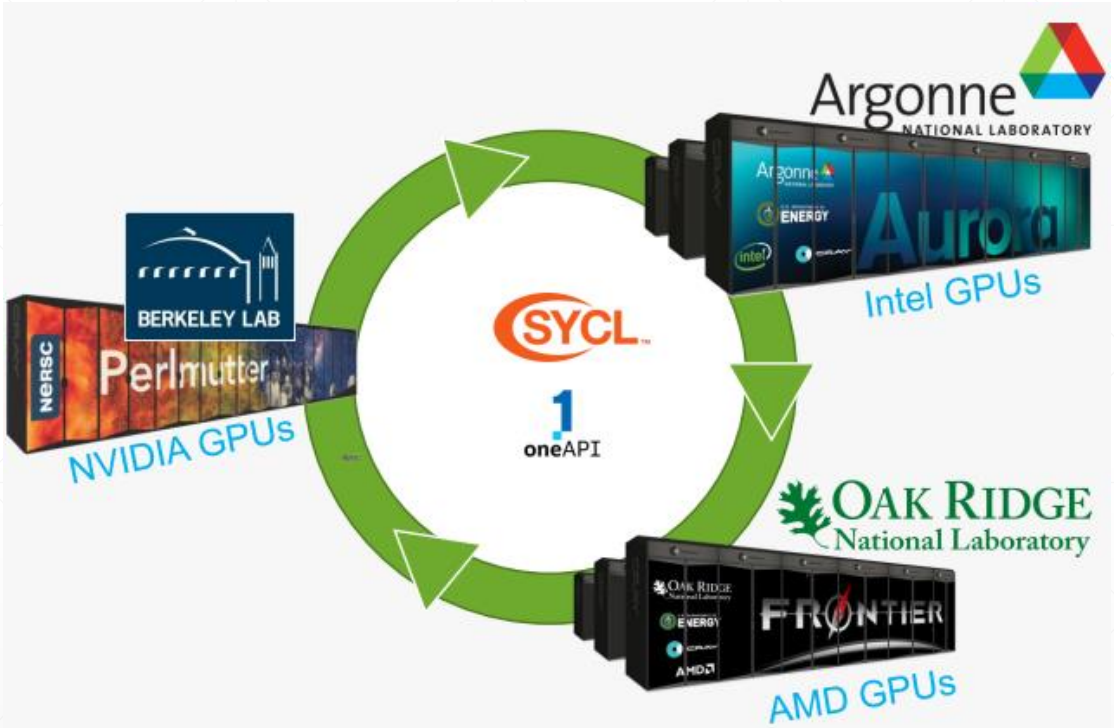

Estimated: internal measurement of the geometric mean of the C/C++ workloads from the SPECspeed* 2017 Floating Point suite (baseline)

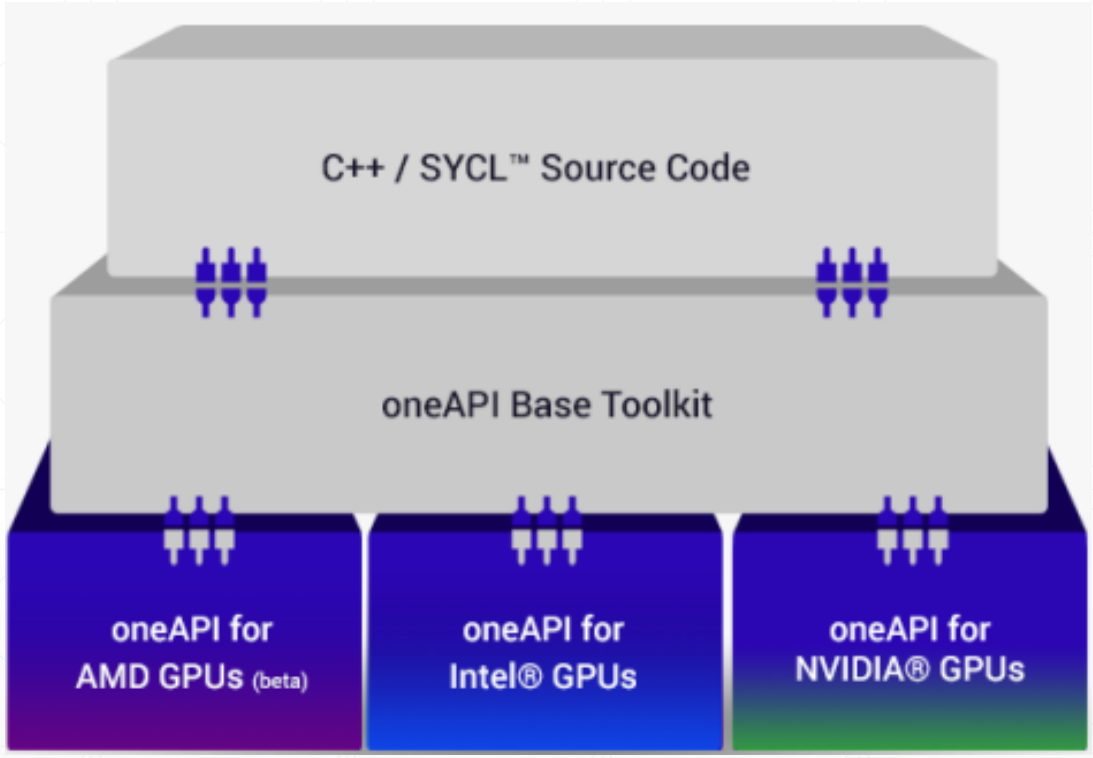



Estimated: internal measurement of the geometric mean of the C/C++ workloads from the SPECspeed* 2017 Integer suite (baseline)

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

# Conclusion



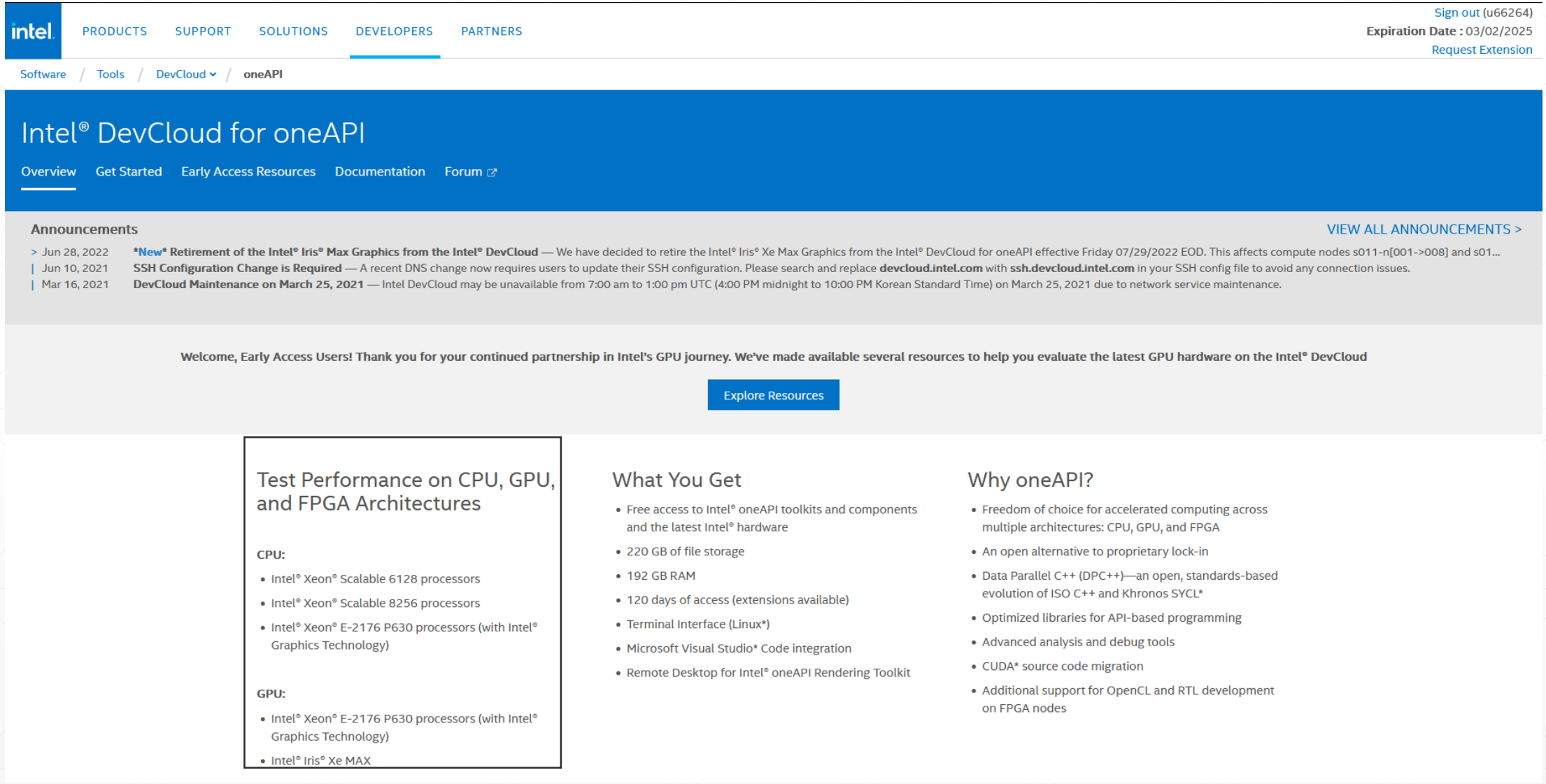

CodePlay Software, *Targeting Multi-Vendor Architectures with oneAPI and SYCL*

# OneAPI 구매 관련

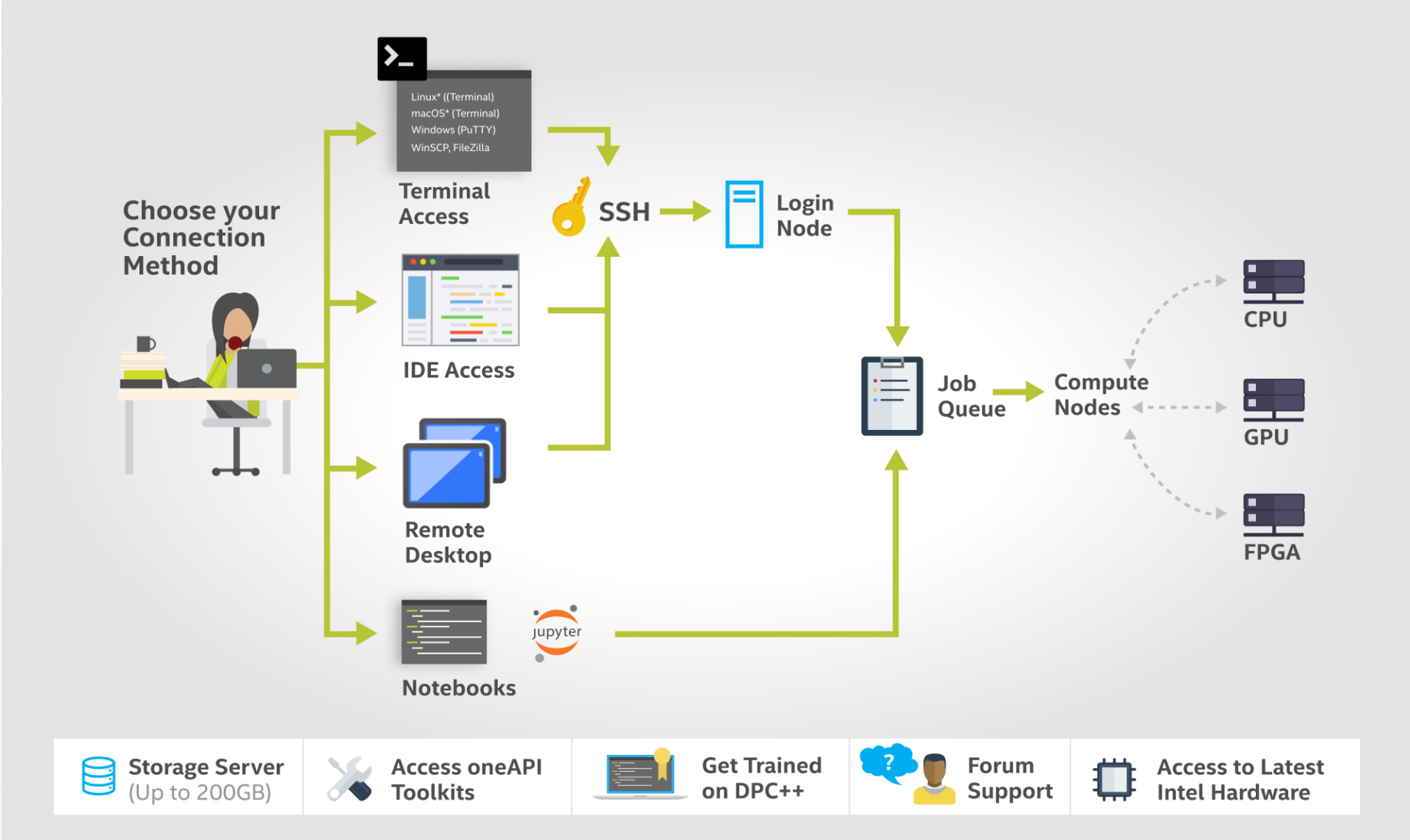
- Purchasing oneAPI software with support provides registered users with many benefits shown on this page https://www.intel.com/content/www/us/en/developer/tools/oneapi/commercial-base-hpc.html and listed here:
  - Direct and private interaction with Intel's support engineers, including the ability to submit confidential support requests
  - Accelerated response time for technical questions and other product needs
  - Free download access to all new product updates and continued access to older versions of the product
  - Priority assistance for escalated defects and feature requests
  - Access to a vast library of self-help documentation built from decades of experience with creating high-performance code

- oneAPI
  - moasys 홈페이지에서 oneAPI 관련 웨비나에 대한 정보를 확인하실 수 있습니다.
  - moasys 홈페이지 : http://www.moasys.com/
  - oneAPI 관련 정보 : http://www.moasys.com/oneapi.html
- oneAPI 구매 문의 : sales@moasys.com

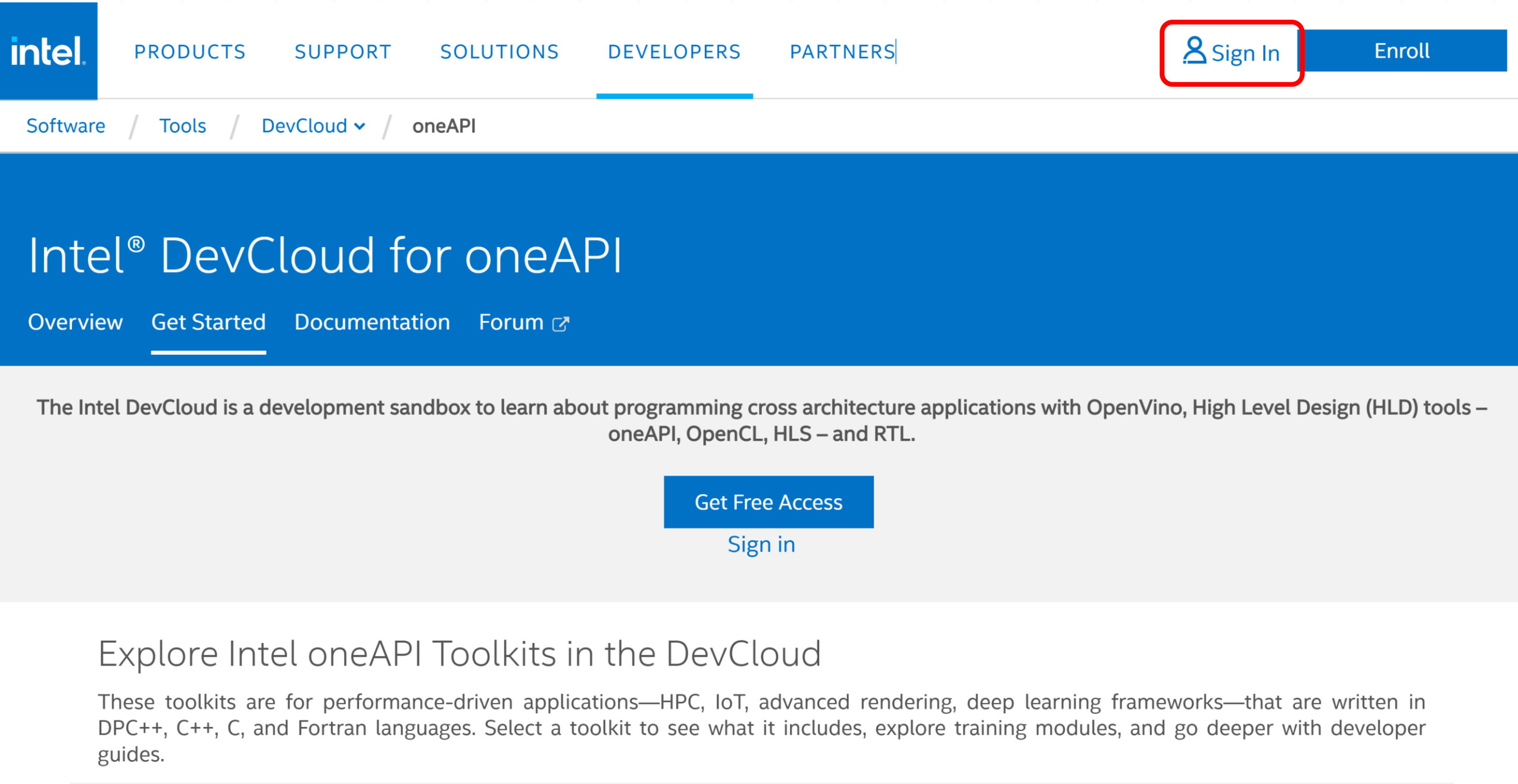
# https://devcloud.intel.com/oneapi/



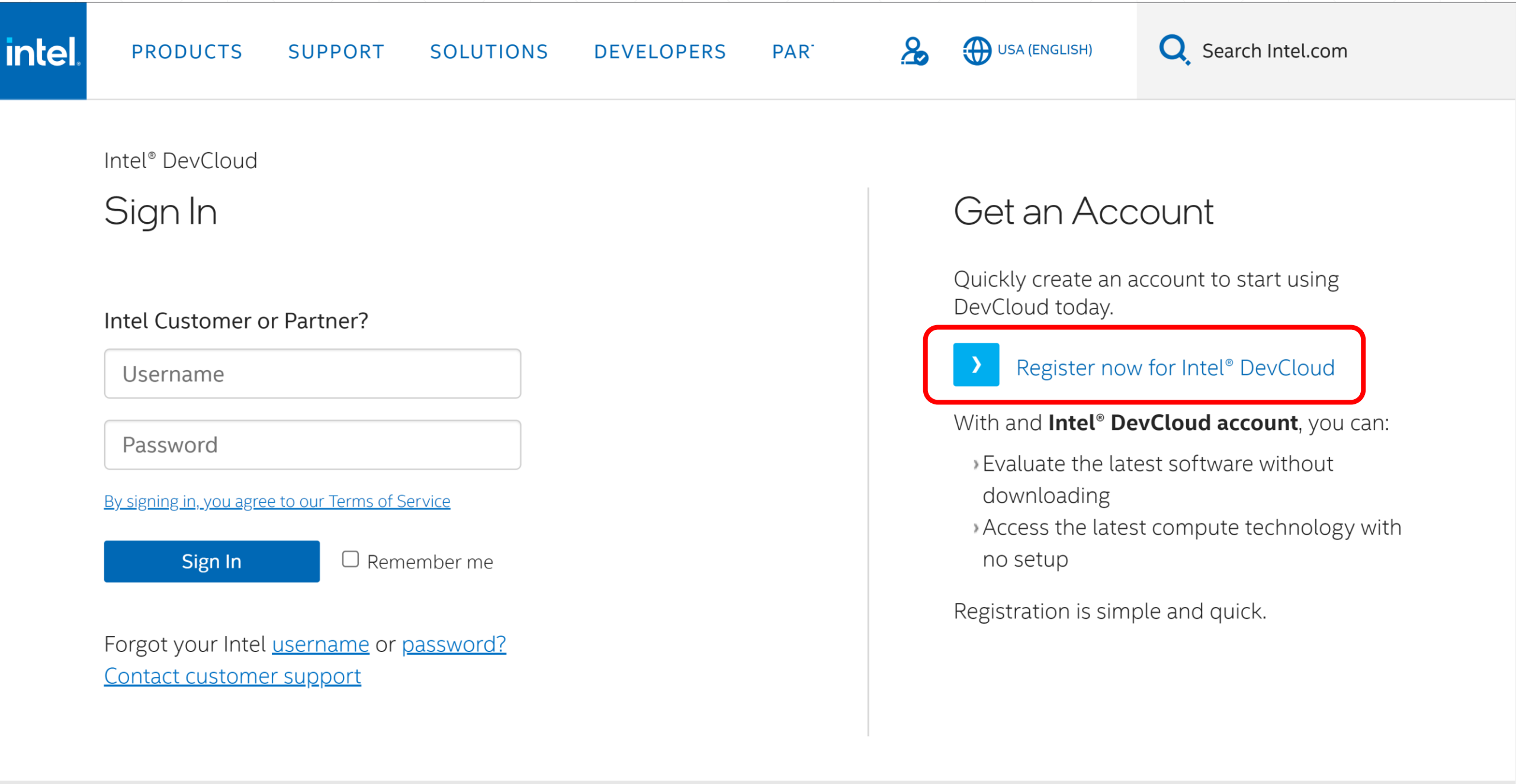
# 연결 방식


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



# 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.

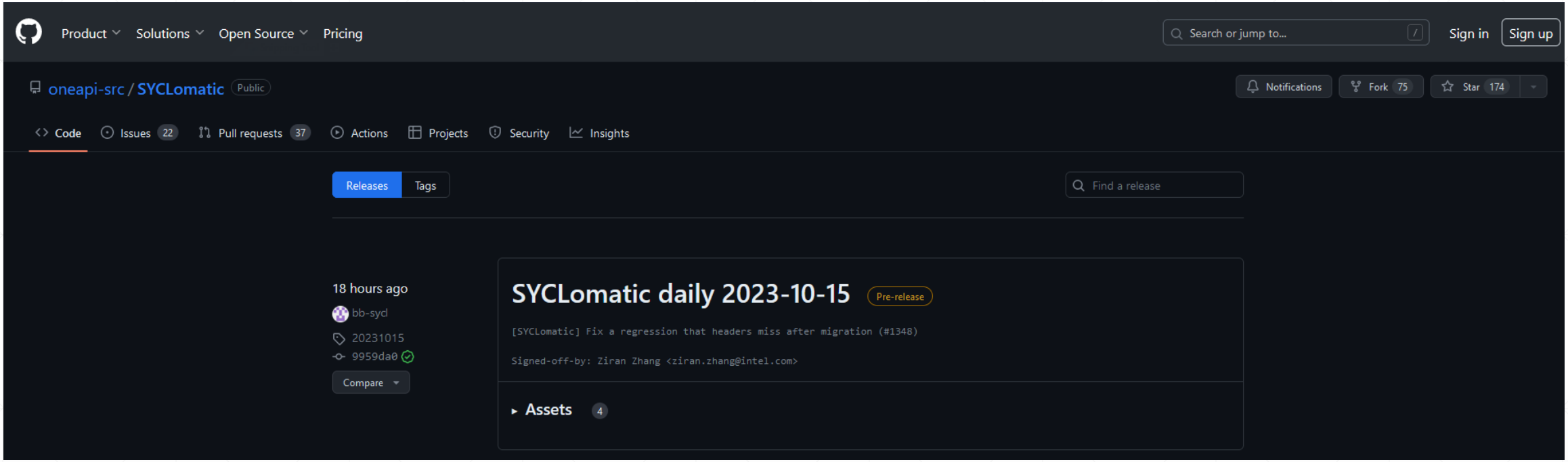
# Interactive Job

```
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
################################################################################
```

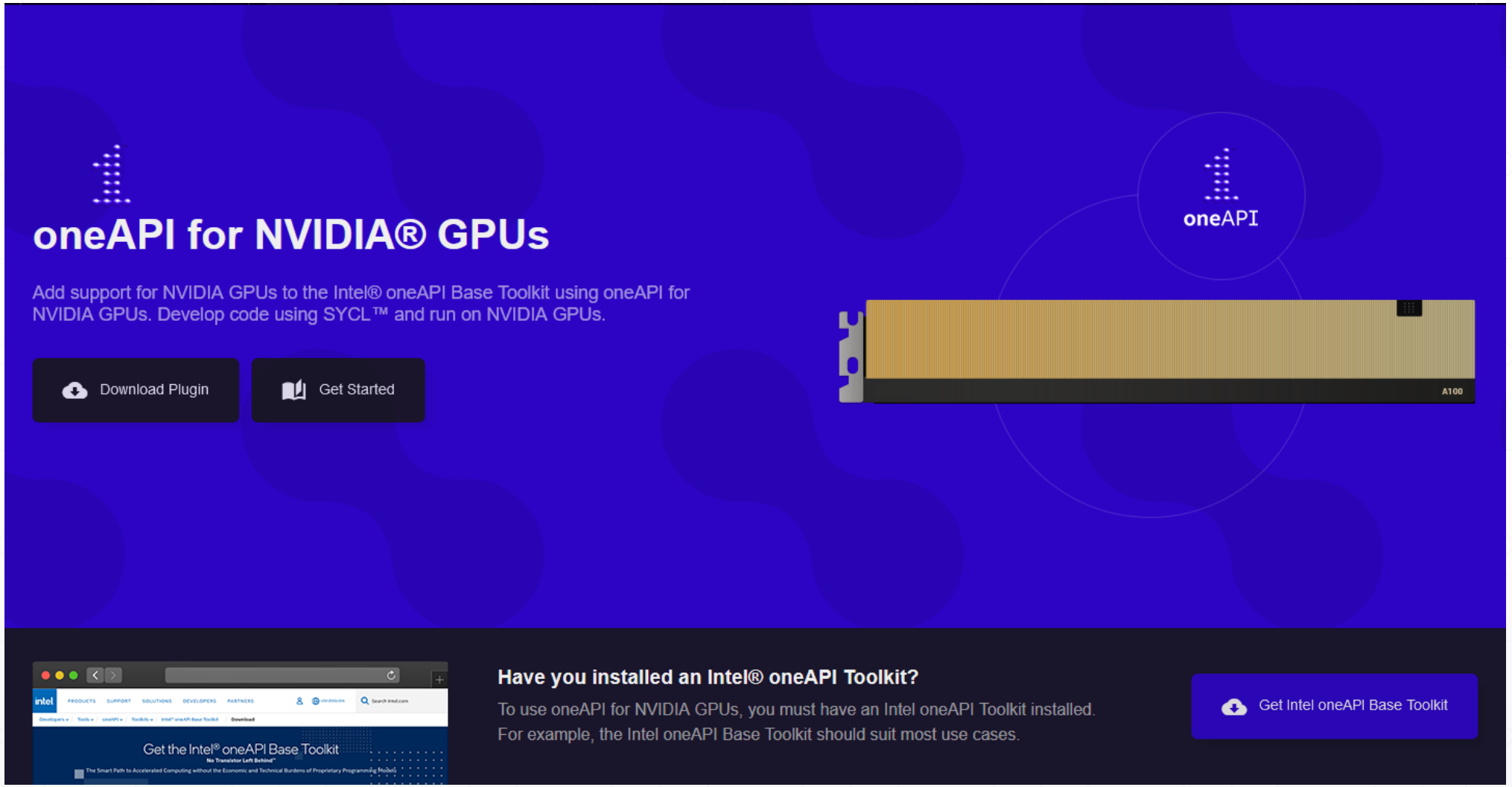
- Request node based on device properties
  - qsub -I -l nodes=[nnodes]:[props]:ppn=[process_per_node]
- Properties describing device class:
  - core / xeon / gpu / fpga
- Properties describing device name:
  - gen9 / iris_xe_max / aria10 / stratix10 / gold6128 / i9-10920x
- Properties describing purpose: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

# SYCLomatic Installtion




```
$ wget https://github.com/oneapi-src/SYCLomatic/archive/refs/tags/20231015.tar.gz
$ tar xf 20231015.tar.gz
$ export PATH=<syclomatic path>/bin:$PATH
$ dpct -h
```

# CodePlay Plugin for oneAPI: Download




https://developer.codeplay.com/products/oneapi/nvidia/home/

# CodePlay Plugin for oneAPI: Installation

- Plugin installation

```
$ sh oneapi-for-nvidia-gpus-2023.2.1-cuda-12.0-linux.sh --install-dir /path/to/intel/oneapi
```

- DPCPP backend enumeration

```
$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2023.16.6.0.22_223734]
[opencl:cpu:1] Intel(R) OpenCL, AMD EPYC 7543 32-Core Processor                3.0 [2023.16.6.0.22_223734]
[ext_oneapi_cuda:gpu:0] NVIDIA CUDA BACKEND, NVIDIA A100-SXM4-80GB 8.8 [CUDA 11.6]
```

- Fat binaries for multiple targets (AMD, NVIDIA, INTEL)

```
$ icpx                                                          \
     -fsycl                                                     \
     -fsycl-targets=amdgcn-amd-amdhsa,nvptx64-nvidia-cuda,spir64    \
     -Xsycl-target-backend=amdgcn-amd-amdhsa --offload-arch=gfx1030 \
     -Xsycl-target-backend=nvptx64-nvidia-cuda --offload-arch=sm_80 \
     -o sycl-app sycl-app.cpp
```

- Backend selection at runtime

```
$ ONEAPI_DEVICE_SELECTOR=cuda:0 SYCL_PI_TRACE=1 ./sycl-app
```

# Vector Addition: CUDA Code

```
$ nvidia-smi
Mon Oct 16 21:04:58 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.47.03    Driver Version: 510.47.03    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-PCIE...  On   | 00000000:3B:00.0 Off |                    0 |
| N/A   28C    P0    23W / 250W |      4MiB / 32768MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+

$ nvcc vectoradd.cu -o vectoradd.x
$ ./vector_add
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

# Vector Addition: SYCL Code

- CUDA development machine:

```
$ syls-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2023.16.6.0.22_223734]
[opencl:cpu:1] Intel(R) OpenCL, Intel(R) Xeon(R) Gold 5217 CPU @ 3.00GHz 3.0 [2021.12.9.0.24_005321]
[ext_oneapi_cuda:gpu:0] NVIDIA CUDA BACKEND, Tesla V100-PCIE-32GB 7.7 [CUDA 11.6]
```

- Compile migrated SYCL Code

```
$ icpx –fsycl vectoradd.dp.cpp –o vectoradd.cpp
./vectoradd.x
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

- Print device information

```
int main()
{
    dpct::device_ext &dev_ct1 = dpct::get_current_device();
    sycl::queue &q_ct1 = dev_ct1.default_queue();
    std::cout << "Device: " << q_ct1.get_device().get_info<sycl::info::device::name>() << "\n";
```

- Recompile:

```
$ icpx –fsycl vectoradd.dp.cpp –o vectoradd.cpp
./vectoradd.x
Device: Intel(R) Xeon(R) Gold 5217 CPU @ 3.00GHz
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

# Vector Addition: SYCL Code

- Compile migrated SYCL Code with NVIDIA backend

```
$ icpx -fsycl vectoradd.dp.cpp -fsycl-targets=nvptx64-nvidia-cuda,spir64 -o vectoradd.x
./vectoradd.x
Device: Tesla V100-PCIE-32GB
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

- Runtime selection of backend

```
$ ONEAPI_DEVICE_SELECTOR=cuda:0 SYCL_PI_TRACE=1 ./vectoradd.x
SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_cuda.so [ PluginVersion: 12.27.1 ]
SYCL_PI_TRACE[all]: Requested device_type: info::device_type::automatic
SYCL_PI_TRACE[all]: Selected device: -> final score = 1500
SYCL_PI_TRACE[all]:   platform: NVIDIA CUDA BACKEND
SYCL_PI_TRACE[all]:   device: Tesla V100-PCIE-32GB
Device: Tesla V100-PCIE-32GB
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

$ ONEAPI_DEVICE_SELECTOR=opencl:1 SYCL_PI_TRACE=1 ./vectoradd.x
SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_opencl.so [ PluginVersion: 12.27.1 ]
SYCL_PI_TRACE[all]: Requested device_type: info::device_type::automatic
SYCL_PI_TRACE[all]: Selected device: -> final score = 1300
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL
SYCL_PI_TRACE[all]:   device: Intel(R) Xeon(R) Gold 5217 CPU @ 3.00GHz
Device: Intel(R) Xeon(R) Gold 5217 CPU @ 3.00GHz
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

# Vector Addition: SYCL Code

- Compile migrated SYCL Code on DevCloud

```
$ icpx -fsycl vectoradd.dp.cpp -o vectoradd.x
./vectoradd.x
Device: Intel(R) UHD Graphics [0x9a60]
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

- Runtime selection of backend

```
$ ONEAPI_DEVICE_SELECTOR=opencl:1 SYCL_PI_TRACE=1 ./vectoradd.x
SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_opencl.so [ PluginVersion: 12.27.1 ]
SYCL_PI_TRACE[all]: Requested device_type: info::device_type::automatic
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
Device: 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

$ ONEAPI_DEVICE_SELECTOR=opencl:2 SYCL_PI_TRACE=1 ./vectoradd.x
SYCL_PI_TRACE[basic]: Plugin found and successfully loaded: libpi_opencl.so [ PluginVersion: 12.27.1 ]
SYCL_PI_TRACE[all]: Requested device_type: info::device_type::automatic
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]
Device: Intel(R) UHD Graphics [0x9a60]
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

# Jacobi Iteraction: CUDA Code

- Download CUDA samples

```
$ wget https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v11.6.tar.gz
$ tar xf v11.6.tar.gz
```

- Compilation of original CUDA codes

```
$ cd cuda-samples-11.6/Samples/3_CUDA_Features/jacobiCudaGraphs
$ make

$ ./jacobiCudaGraphs –help
Command line: jacobiCudaGraphs [-option]
Valid options:
-gpumethod=<0,1 or 2>  : 0 - [Default] JacobiMethodGpuCudaGraphExecKernelSetParams
                       : 1 - JacobiMethodGpuCudaGraphExecUpdate
                       : 2 - JacobiMethodGpu - Non CUDA Graph
-device=device_num     : cuda device id-help         : Output a help message

$ ./jacobiCudaGraphs –gpumethod=2

CPU iterations : 2954
CPU error : 4.988e-03
CPU Processing time: 2095.318115 (ms)
GPU iterations : 2954
GPU error : 4.988e-03
GPU Processing time: 66.236000 (ms)
&&&& jacobiCudaGraphs PASSED
```

# Jacobi Iteraction: CUDA Code

- Compilation of migrated SYCL code with graph

```
$ icpx -fsycl -fsycl-targets=nvptx64-nvidia-cuda,spir64 *.cpp -I ../../../../Common/
jacobi.dp.cpp:211:3: error: unknown type name 'cudaGraph_t'
  cudaGraph_t graph;
  ^
jacobi.dp.cpp:212:3: error: unknown type name 'cudaGraphExec_t'
  cudaGraphExec_t graphExec = NULL;
  ^
jacobi.dp.cpp:216:3: error: use of undeclared identifier 'checkCudaErrors'
  checkCudaErrors(
  ^
jacobi.dp.cpp:219:15: error: use of undeclared identifier 'cudaGraphNode_t'
  std::vector<cudaGraphNode_t> nodeDependencies;
```

- Compilation of migrated SYCL code sans graph features (CUDA machine)

```
$ icpx –fsycl -fsycl-targets=nvptx64-nvidia-cuda,spir64 -I ../../../Common -I ../../../include *.cpp –o jacobi.x
$ ONEAPI_DEVICE_SELECTOR=cuda:0 ./jacobi

CPU iterations : 2954
CPU error : 4.988e-03
CPU Processing time: 283.115997 (ms)
GPU iterations : 2954
GPU error : 4.988e-03
GPU Processing time: 362.898987 (ms)
&&&& jacobiCudaGraphs PASSED
```

# Jacobi Iteraction: CUDA Code

- Compilation of migrated SYCL code sans graph features (DevCloud)

```
$ icpx –fsycl -I ../../../Common -I ../../../include *.cpp –o jacobi.x

$ ONEAPI_DEVICE_SELECTOR=opencl:1 ./jacobi

CPU iterations : 2954
CPU error : 4.988e-03
CPU Processing time: 157.582001 (ms)
GPU iterations : 2954
GPU error : 4.988e-03
GPU Processing time: 1480.723022 (ms)
&&&& jacobiCudaGraphs PASSED

$ ONEAPI_DEVICE_SELECTOR=opencl:2 ./jacobi

CPU iterations : 2954
CPU error : 4.988e-03
CPU Processing time: 152.328003 (ms)
terminate called after throwing an instance of 'sycl::_V1::exception'
  what():  Required aspect fp64 is not supported on the device
Aborted
```

# cuBLAS Migration

- Compilation of original CUDA codes

```
$ cuda-samples-11.6/Samples/4_CUDA_Libraries/matrixMulCUBLAS
$ make

$ ./matrixMulCUBLAS
MatrixA(640,480), MatrixB(480,320), MatrixC(640,320)
Computing result using CUBLAS...done.
Computing result using host CPU...done.
Comparing CUBLAS Matrix Multiply with CPU results: PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
```

- Compilation of migrated code on DevCloud

```
$ icpx \
    -fsycl \
    -I ../../../Common -I ../../../include \
    *.cpp -lmkl_sycl -lmkl_intel_ilp64 -lmkl_sequential -lmkl_core -o matmul.x

$ ./matmul.x
MatrixA(640,480), MatrixB(480,320), MatrixC(640,320)
Computing result using CUBLAS...done.
Computing result using host CPU...done.
Comparing CUBLAS Matrix Multiply with CPU results: PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
```

# THANK YOU

Korea Institute of Science and Technology Information