

# Relax and sit tight

The workshop will begin at 9:00 am PST



intel®

# Migrating from CUDA to C++ with SYCL

Rakshith Krishnappa, Developer Evangelist @Intel

[rakshith.krishnappa@intel.com](mailto:rakshith.krishnappa@intel.com)



# About Our Speakers



## Rakshith Krishnappa - Developer Evangelist

Rakshith is a Developer Evangelist at Intel with over 17 years of experience in software development and Intel products. He graduated from Illinois Institute of Technology with master's in Electrical and Computer Engineering. His main focus currently at Intel is in High Performance Computing and oneAPI Products and Solutions.



## Karl Qi - oneAPI Technical Evangelist

As an oneAPI technical evangelist, Karl focuses on enabling HPC and AI customers to create the optimal solution for their needs using the Intel® oneAPI toolkits. He has a particular interest in software that can leverage the capabilities of heterogeneous parallel computing environments. Karl has a bachelor's degree in Electrical Engineering from Cornell University.

# Using the Console

Resize and move windows around

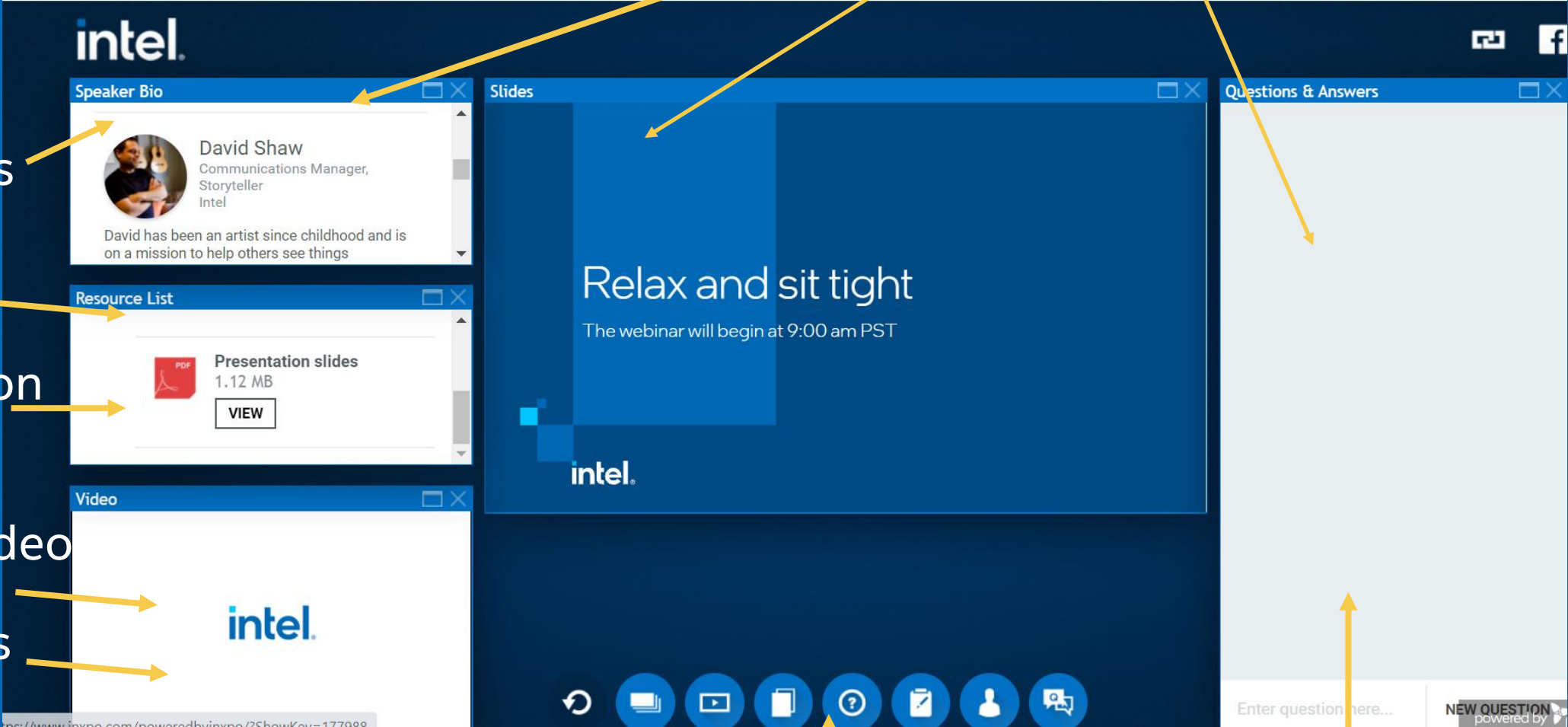
Biographies

Resources

Presentation

Speaker video

Live demos



Quick links

Questions and Answers

# Migrating from CUDA to C++ with SYCL

## Who is this for?

This course is designed for developers who are familiar with Nvidia CUDA development who want to migrate CUDA projects to C++ SYCL and run on heterogenous hardware from different vendors.

# What will you learn?

- Programming Challenges with Multiple Hardware Architecture
- Migration Tools
- CUDA to C++ SYCL migration example
- Compiling SYCL code
- Hands-on workshop for migrating CUDA to C++ SYCL

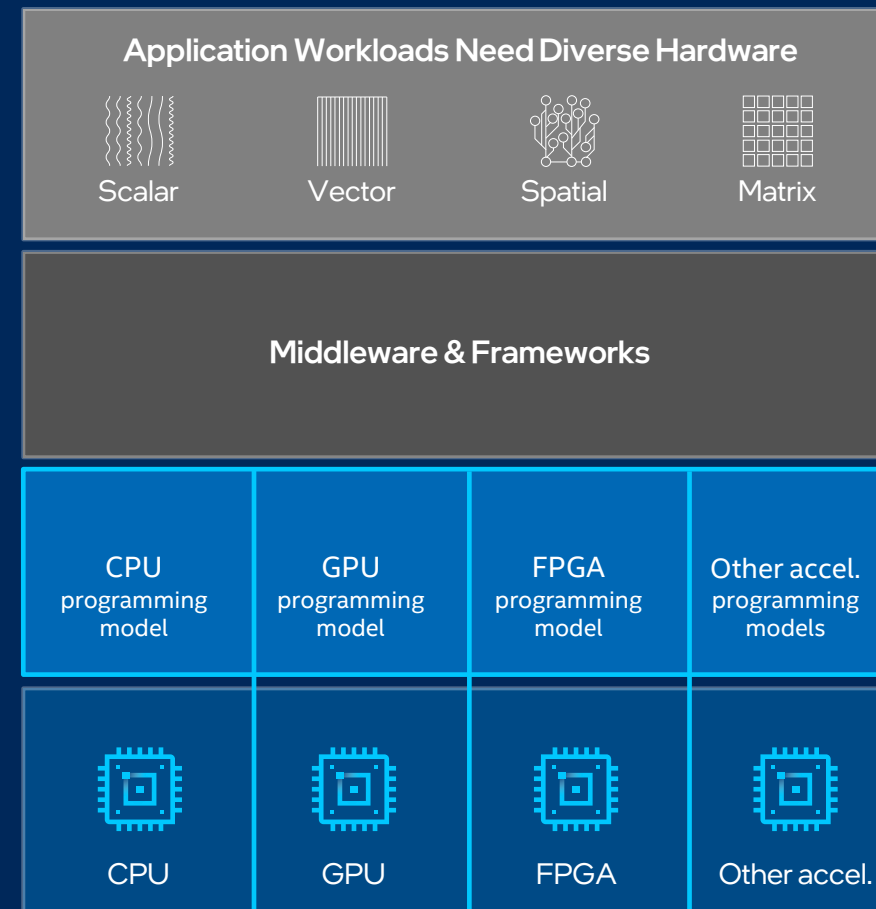
# Programming Challenges for Multiple Architectures

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



# oneAPI

## One Programming Model for Multiple Architectures and Vendors



### Freedom to Make Your Best Choice

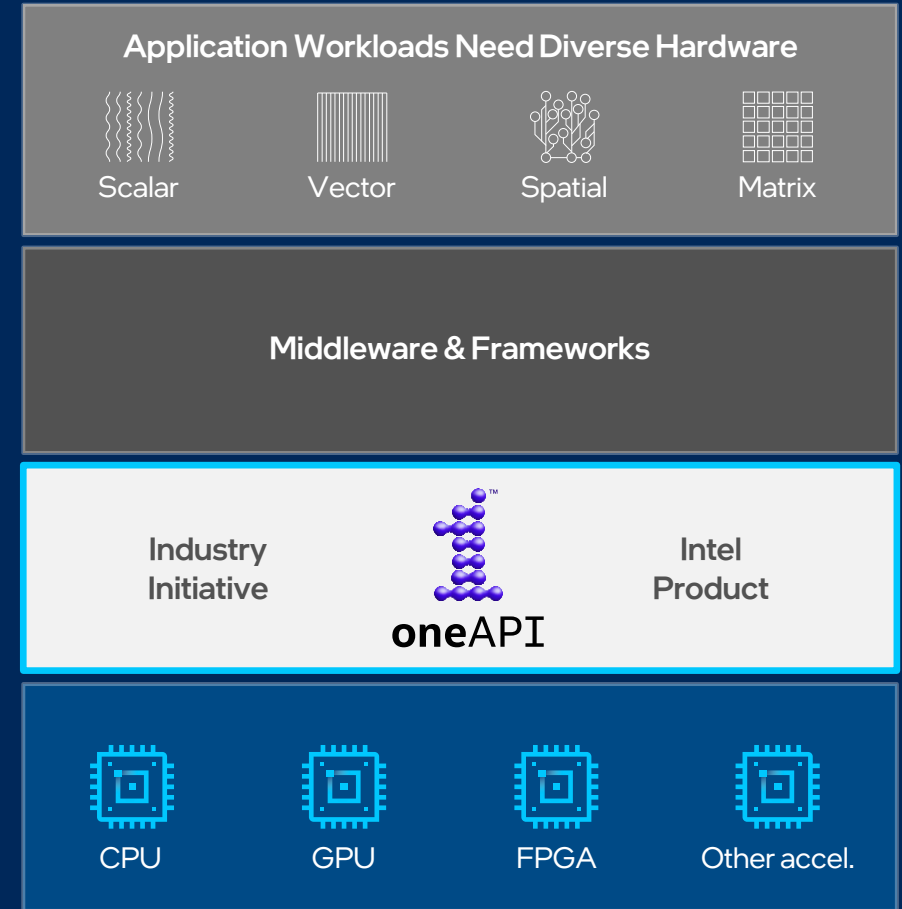
- Choose the best accelerated technology the software doesn't decide for you

### Realize all the Hardware Value

- Performance across CPU, GPUs, FPGAs, and other accelerators

### Develop & Deploy Software with Peace of Mind

- Open industry standards provide a safe, clear path to the future
- Compatible with existing languages and programming models including C++, Python, SYCL, OpenMP, Fortran, and MPI





# Migration Tools

1. Intel® Data Parallel C++ Compatibility Tool  
available in Intel oneAPI Base Toolkit
2. Intel Open-Source SYCLomatic Migration Tool  
(<https://github.com/oneapi-src/SYCLomatic>)

# Intel® DPC++ Compatibility Tool

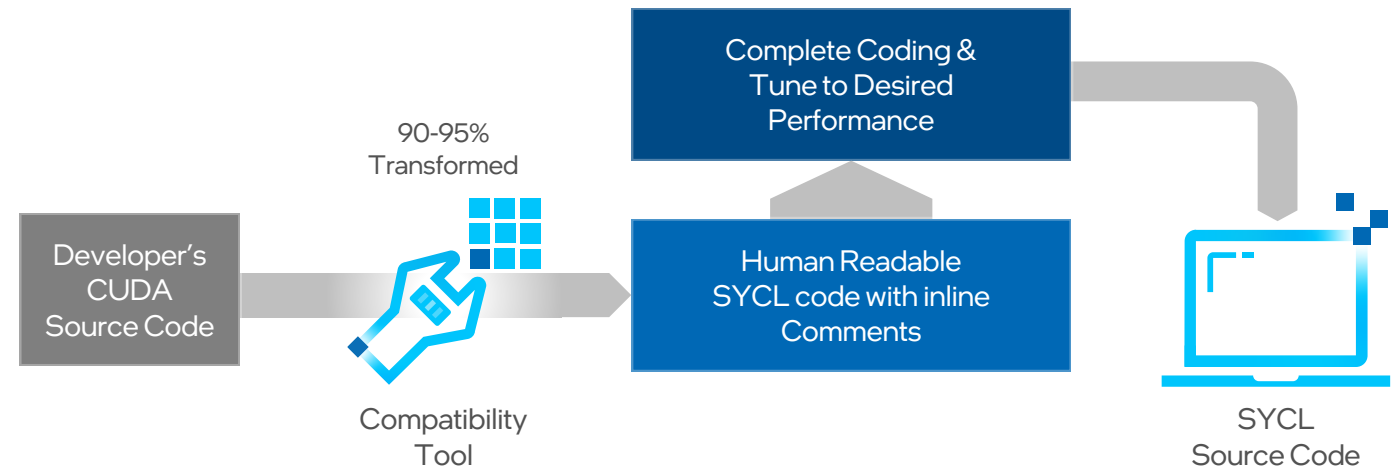
## Minimizes Code Migration Time

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

~90-95% of code typically migrates automatically<sup>1</sup>

Inline comments are provided to help developers finish porting the application

### Intel DPC ++ Compatibility Tool Usage Flow

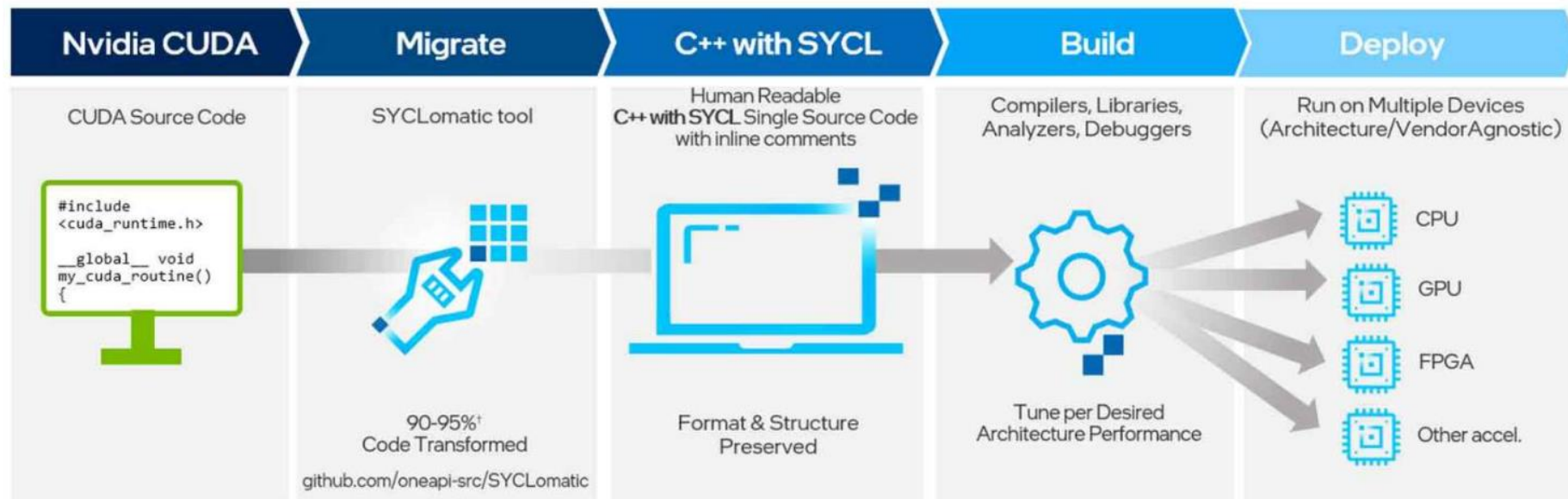


<sup>1</sup>Intel estimates as of September 2021. Based on measurements on a set of 70 HPC benchmarks and samples, with examples like Rodinia, SHOC, PENNANT. Results may vary.

# Intel Open-Sources SYCLomatic Migration Tool to Help Developers Create Heterogeneous Code

This open source project enables community collaboration to advance adoption of the SYCL standard, a key step in freeing developers from a single-vendor proprietary ecosystem.

## CUDA<sup>‡</sup> to SYCL<sup>‡</sup> Code Migration & Development Workflow



<sup>†</sup> Intel estimates as of September 2021. Based on measurements on a set of 70 HPC benchmarks and samples, with examples like Rodinia, SHOC, PENNANT. Results may vary.

<sup>‡</sup> Other names and brands may be claimed as the property of others. SYCL is a trademark of the Khronos Group Inc.

# CUDA to C++ SYCL Migration

The diagram illustrates the migration of CUDA code to SYCL code. It shows two code snippets side-by-side, with yellow boxes highlighting specific parts of the code and yellow arrows pointing from descriptive labels to these boxes.

**Header file** points to the include statements in both code snippets.

**Kernel function** points to the `VectorAddKernel` function in both code snippets.

**Device Memory Allocation** points to the memory allocation code in both code snippets.

**Copy host to device** points to the `memcpy` calls in both code snippets.

**Submit Kernel Task** points to the kernel launch code in both code snippets.

```
#include <cuda.h>
#include <iostream>
#define N 2048

// Computation Offloaded to device
__global__ void VectorAddKernel(float* A, float* B, float* C)
{
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    C[id] = A[id] + B[id];
}

int main()
{
    // Initialize data on host
    float A[N], B[N], C[N];
    for (int i = 0; i < N; i++){
        A[i] = 1;
        B[i] = 2;
        C[i] = 0;
    }

    // Allocate memory on device
    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));

    // Copy data from host to device
    cudaMemcpy(d_A, A, N*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, N*sizeof(float), cudaMemcpyHostToDevice);

    // Offload computation to device
    int nThreads = 256;
    int nBlocks = N / nThreads;
    VectorAddKernel<<<nBlocks, nThreads>>>(d_A, d_B, d_C);

    // Copy result data from device to host
    cudaMemcpy(C, d_C, N*sizeof(float), cudaMemcpyDeviceToHost);

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

    // Free device memory
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
    return 0;
}
```

```
#include <CL/sycl.hpp>
#include <dpct/dpct.hpp>
#include <iostream>
#define N 2048

// Computation Offloaded to device
void VectorAddKernel(float* A, float* B, float* C, sycl::nd_item<3> item_ct1)
{
    int id = item_ct1.get_local_range(2) * item_ct1.get_group(2) +
        item_ct1.get_local_id(2);
    C[id] = A[id] + B[id];
}

int main()
{
    dpct::device_ext &dev_ct1 = dpct::get_current_device();
    sycl::queue &q_ct1 = dev_ct1.default_queue();

    // Initialize data on host
    float A[N], B[N], C[N];
    for (int i = 0; i < N; i++){
        A[i] = 1;
        B[i] = 2;
        C[i] = 0;
    }

    // Allocate memory on device
    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);

    // Copy data from host to device
    q_ct1.memcpy(d_A, A, N * sizeof(float));
    q_ct1.memcpy(d_B, B, N * sizeof(float)).wait();

    // Offload computation to device
    int nThreads = 256;
    int nBlocks = N / nThreads;
    /*
    DPCT1049:0: The work-group size passed to the SYCL kernel may exceed the
    limit. To get the device limit, query info::device::max_work_group_size.
    Adjust the work-group size if needed.
    */
    q_ct1.parallel_for(sycl::nd_range<3>(sycl::range<3>(1, 1, nBlocks) *
        sycl::range<3>(1, 1, nThreads),
        sycl::range<3>(1, 1, nThreads)),
        [=](sycl::nd_item<3> item_ct1) {
            VectorAddKernel(d_A, d_B, d_C, item_ct1);
        });

    // Copy result data from device to host
    q_ct1.memcpy(C, d_C, N * sizeof(float)).wait();

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

    // Free device memory
    sycl::free(d_A, q_ct1);
    sycl::free(d_B, q_ct1);
    sycl::free(d_C, q_ct1);
    return 0;
}
```

# Compiler for C++ SYCL

1. Intel<sup>®</sup> oneAPI DPC++/C++ Compiler available in Intel oneAPI Base Toolkit
2. Intel Open-Source `llvm`  
(<https://github.com/intel/llvm>)

# Hands-on Workshop

## Step #1

Migrate CUDA projects to SYCL on your **CUDA Development Machine**

## Step #2

Analyze migrated SYCL code and implement any warnings or unmigrated code and test on **Intel DevCloud** or on your Development Machine with heterogeneous hardware from different vendors.

# Resources

One stop shop portal

<https://developer.intel.com/cuda2sycl>

Open-source Compiler for SYCL

<https://github.com/intel/llvm>

SYCLomatic Migration Tool

<https://github.com/oneapi-src/SYCLomatic>

# Find more workshops, webinars, and events here!

- One API Events and Training calendar

- <https://software.seek.intel.com/oneapi-training-calendar#gs.8qflef>

- Intel Software Developer Zone events

- [calendar#gs.8qf9ffhttps://www.intel.com/content/www/us/en/events/developer/overview.html](https://www.intel.com/content/www/us/en/events/developer/overview.html)

- Upcoming Webinars

- <https://software.seek.intel.com/techdecoded-webinars>





# Legal Notices and Disclaimers

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](http://www.intel.com/benchmarks).

Performance results are based on testing as of the publication date of the referenced papers and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

**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 and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.