CUDA to SYCL Migration

Rakshith Krishnappa (rakshith.krishnappa@intel.com)

Workshop Agenda

February 15th	Introduction to Using the SYCLomatic Tool and Compiling/Executing SYCL code on Intel Dev Cloud
March 15th	Migrating more complex CUDA source with the SYCLomatic Tool
April 12th	Mini Hackathon and Q&A, Migrating your CUDA Code to SYCL - tips, tricks, and limitations, Migrating CUDA code with libraries.

Session #1 - 02/15/2023, 1:30 – 3:30PM CT

- **Introduction to Using the SYCLomatic Tool and Compiling/Executing SYCL code on Intel Dev Cloud**
  - Installing SYCLomatic tool
  - Understand SYCLomatic tool usage and command line options
  - Migrate a simple CUDA example with just one source file to SYCL
  - Migrate a CUDA example with multiple CUDA source files to SYCL
- In this session we will mainly try to understand how memory allocation and memory copy is accomplished in CUDA versus SYCL, we will also look at how a kernel is offloaded to run on GPU in CUDA versus SYCL.

Session #2 - 03/15/2023, 1:30 – 3:30PM CT

- **Migrating more complex CUDA source with the SYCLomatic Tool**
  - Migrate a CUDA example with multiple CUDA source files to SYCL
  - Optimize Kernel code with SYCL features.
- In this session we will understand how CUDA features like Local Memory, Cooperative groups, warp primitives and atomic operations are migrated to SYCL, we will inspect the CUDA and SYCL source and understand how migration was accomplished using SYCLomatic tool. We will also try to manually optimize the migrated SYCL code for performance using SYCL features.

Session #3 - 04/12/2023, 1:30 – 3:30PM CT

- **Mini Hackathon: Migrating your CUDA Code to SYCL - tips, tricks, and limitations**
  - This session will be a mini hackathon where you can bring your own CUDA source and try to migrate to SYCL, Intel experts will help and answer any questions you may have about the migration process.
  - We will also give an overview of how migration is accomplished when CUDA source use a library like cuBLAS or cuFFT, we will show case other CUDA to SYCL migration projects that are completed and can be used as reference. We will also learn about the current limitation of the SYCLomatic tool, we will learn about some tips and tricks when migrating CUDA to SYCL using SYCLomatic tool.

Pre-requisites

- **CUDA development machine:** Have system ready with CUDA SDK installed, you should be able to compile/run a simple CUDA sample code.

- Sign up for **Intel Developer Cloud** account at devcloud.intel.com/oneapi

Workshop Overview

- These sessions involve **2 steps:**
  1. Migrating the CUDA source on CUDA development machine
  2. Executing migrated SYCL source on Intel CPUs/GPUs on Intel Developer Cloud

- The audience is expected to have a **CUDA development machine** ready for this workshop, we will install SYCLomatic tool on the CUDA development and then migrate the CUDA source to SYCL.
- Once the code migration is complete, we will transfer the migrated SYCL source to **Intel Developer Cloud** to compile, execute and optimize on Intel CPUs/GPUs.

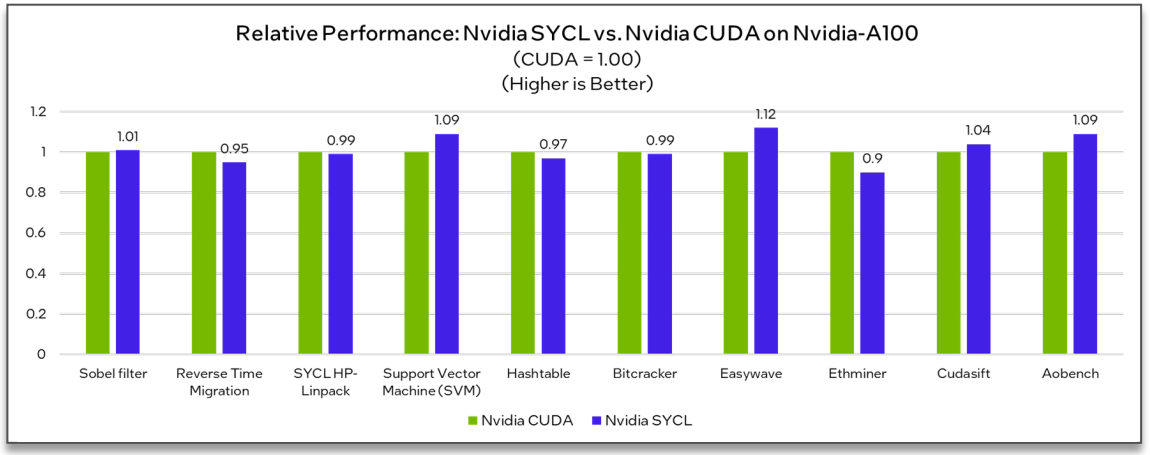
If you do not have a CUDA development machine available, you can just watch the demonstration of step one (CUDA to SYCL migration) and then do the step two on Intel Developer Cloud.

Why Migrate to SYCL?

Accelerating Choice with SYCL

Khronos Group Standard

- Open, standards-based
- Multiarchitecture performance
- Freedom from vendor lock-in
- Comparable performance to native CUDA on Nvidia GPUs
- Extension of widely used C++ language
- Speed code migration via open source SYCLomatic or Intel® DPC++ Compatibility Tool



Testing Date: Performance results are **based on testing by Intel as of August 15, 2022** and may not reflect all publicly available updates.

Configuration Details and Workload Setup: Intel® Xeon® Platinum 8360Y CPU @ 2.4GHz, 2 socket, Hyper Thread On, Turbo On, 256GB Hynix DDR4-3200, ucode 0x000363. GPU: Nvidia A100 PCIe 80GB GPU memory. Software: SYCL open source/CLANG 15.0.0, CUDA SDK 11.7 with NVIDIA-NVCC 11.7.64, cuMath 11.7, cuDNN 11.7, Ubuntu 22.04.1. SYCL open source/CLANG compiler switches: -fsycl-targets=nvptx64-nvidia-cuda, NVIDIA NVCC compiler switches: -O3 --gencode arch=compute_80, code=sm_80. Represented workloads with Intel optimizations.

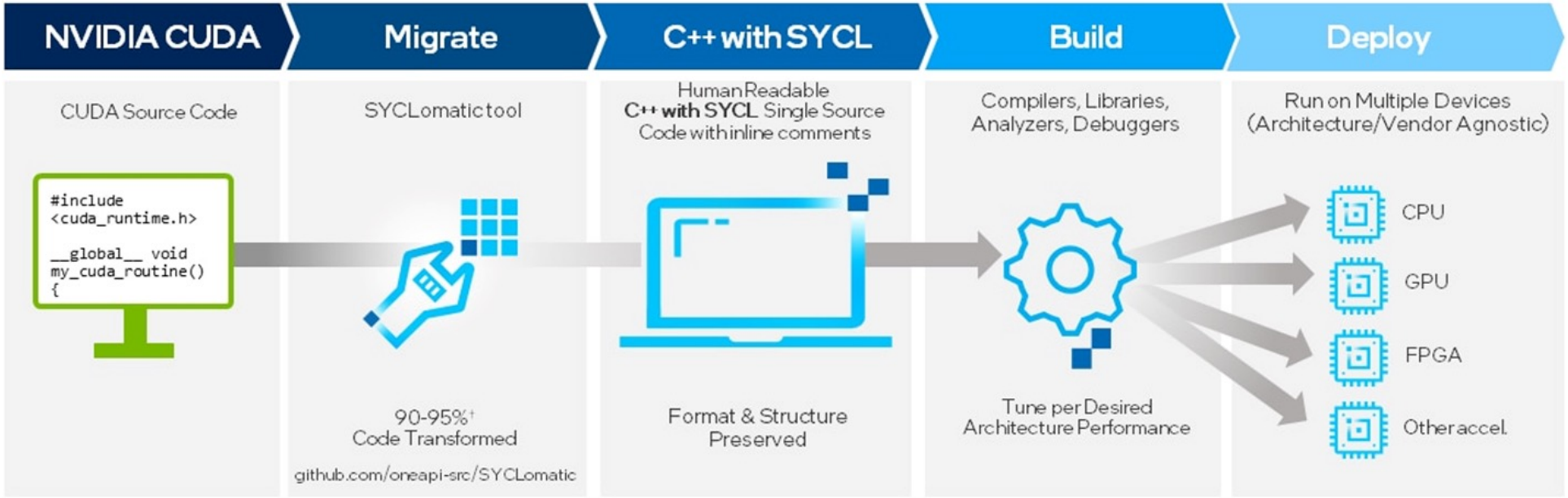
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

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

Architectures	Intel | Nvidia | AMD CPU/GPU | RISC-V | ARM Mali | PowerVR | Xilinx

CUDA to SYCL Migration Made Easy

Open Source SYCLomatic Tool Reduces Code Migration Time



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

~90-95% of code typically migrates automatically[1]

Inline comments are provided to help developers finish porting the application

Intel® DPC++/C++ Compatibility Tool is Intel's implementation, available in the Base Toolkit

[1]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.

Productive and Performant SYCL Compiler

Intel® oneAPI DPC++/C++ Compiler

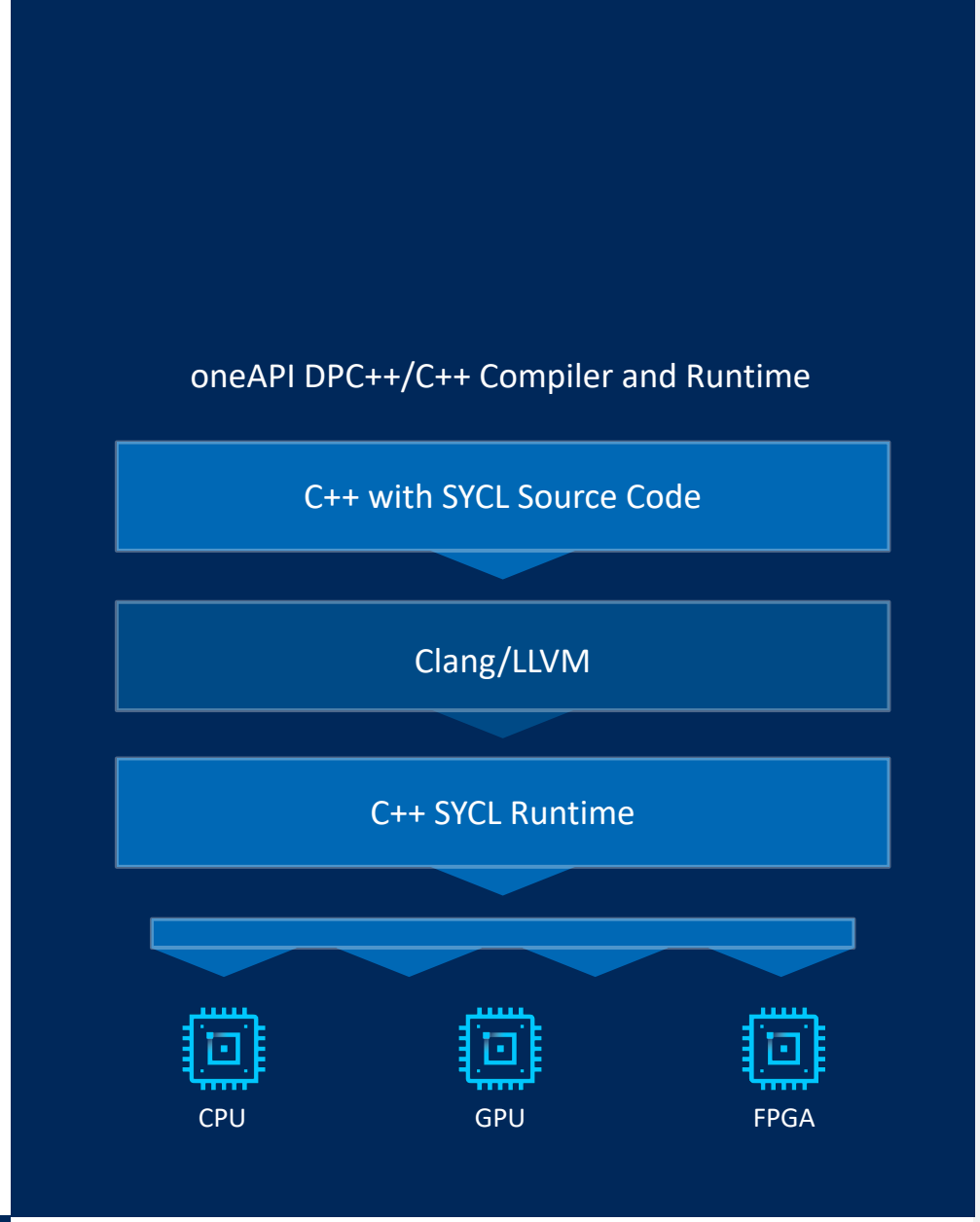
Uncompromised parallel programming productivity and performance across CPUs and accelerators

- Allows code reuse across hardware targets, while permitting custom tuning for a specific accelerator
- Open, cross-industry alternative to single architecture proprietary language

Khronos SYCL Standard

- Delivers C++ productivity benefits, using common and familiar C and C++ constructs
- Created by Khronos Group to support data parallelism and heterogeneous programming

Builds upon Intel's decades of experience in architecture and high-performance compilers



Learn More & Download

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

Codeplay Compiler Plug-ins for Nvidia and AMD GPUs

Adding support for NVIDIA and AMD GPUs to the Intel® oneAPI Base Toolkit

oneAPI for NVIDIA & AMD GPUs

- Free Codeplay download of latest binary plugins to the Intel DPC++/C++ compiler:
  - Nvidia GPU
  - AMD Beta GPU
- Availability at the same time as the Intel oneAPI Base Toolkit
- Plug-ins updated quarterly in-sync with oneAPI

Priority Support

- Sold by Intel and Codeplay and our channel
- Requires Intel Priority support for Intel DPC++ /C++ compiler
- Intel takes first call and Codeplay delivers backend support
- Codeplay access to older versions of plugins

Nvidia GPU plug-in | AMD GPU plug-in | Codeplay blog | Codeplay press release

Vector Addition from CUDA to SYCL - Code Sample

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.
Your costs and results may vary.

Reference Material

CUDA Development Machine – Sanity Check

- Run nvidia-smi in terminal
  - nvidia-smi
  - check that driver version is displayed, Nvidia card is detected and CUDA version is displayed

```
rkris11@holmes:~$ nvidia-smi
Mon Mar  6 19:52:26 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.86.01    Driver Version: 515.86.01    CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| 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 P100-PCIE...  Off  | 00000000:02:00.0 Off |                    0 |
| N/A   29C    P0    24W / 250W |      0MiB / 12288MiB |      3%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

- Check CUDA headers in install path
  - Default installation path: ls /usr/local/cuda/include
  - If CUDA is installed in non-default path, make a note of path, you will need it later
- Check that you are able to compile a simple CUDA code
  - nvcc test.cu

Install SYCLomatic Tool

- Go to https://github.com/oneapi-src/SYCLomatic/releases
  - Under Assets
  - Copy web link to linux_release.tgz
- On you CUDA Development machine:
  - In home directory or anywhere: mkdir syclomatic; cd syclomatic
  - wget <link to linux_release.tgz>
  - tar –xvf linux_release.tgz
  - export PATH="/home/$USER/syclomatic/bin:$PATH"
  - c2s --version

SYCLomatic Tool Usage (c2s --help)

- Migrate a single CUDA source file:
  - `c2s test.cu`
- Migrate a single CUDA source file and copy all syclomatic helper header files:
  - `c2s test.cu --use-custom-helper=all/file/api`
- Migrate a single CUDA source to a specific directory name
  - `c2s test.cu --out-root sycl_code`
- Migrate a single CUDA source with source root tree
  - `c2s test.cu --in-root ../`
- Migrate a single CUDA source with custom CUDA installation
  - `c2s test.cu --cuda-include-path /tmp/cuda/include`
- Migrate a CUDA project with makefile:
  - `intercept-build make`
  - `c2s -p compile_command.json`

test.cu

Compiling SYCL code for Intel and Nvidia targets

- Install oneAPI C++/DPC++ Compiler or Intel oneAPI Base Toolkit
- Install CUDA Plugin for oneAPI from CodePlay
- Link to Installation Instructions

- Compile SYCL for Intel CPUs/GPUs
  - `icpx –fsycl test.cpp`

- Compile SYCL for Nvidia GPUs
  - `clang++ -fsycl -fsycl-targets=nvptx64-nvidia-cuda test.cpp`

Accessing Training Content on Intel DevCloud

- Login to Intel DevCloud: devcloud.intel.com/oneapi
  - Click “Get Started”, scroll down and click on “Launch JupyterLab”

- Copy latest CUDA_To_SYCL_Migration training content:
  - In JupyterLab, File menu -> New -> Terminal
  - `cp -r /data/oneapi_workshop/CUDA_To_SYCL_Migration/ .`

- Access the training content:
  - In Jupyter, on left panel, open `CUDA_To_SYCL_Migration`
  - Open `Welcome.ipynb`

Learn Basics of SYCL Programming

- SYCL 2020 Specification
- Data Parallel C++ Book
- SYCL Academy from CodePlay
- Guided learning path with code samples (Jupyter Notebooks):
  - SYCL Essentials
  - SYCL Performance Portability
- C++ SYCL code samples

Optimizing SYCL code for Intel GPUs

- Refer to Intel GPU Optimization Guide
  - Detailed guide explaining how to optimize SYCL code:
    - Thread Mapping and GPU Occupancy Calculation.
    - Memory allocation and transfer optimization when using Buffers or Unified Shared memory.
    - Kernel code optimization – Local memory, Sub-Groups, Atomics, Reduction and more.
    - Using libraries for offload
    - Debugging and Profiling

CUDA to SYCL Migration Portal

- One Stop Portal for CUDA to SYCL Migration
  - Industry Examples of CUDA Migration to SYCL
  - Learn How to Migrate Your Code
  - Guided flow of migrating from CUDA to SYCL
  - Get all the tools and resources necessary for migrating from CUDA to SYCL