

Compatibility Tool from CUDA to SYCL (SYCLomatic)

Intel® oneAPI Iberian Tour

CGS

December 14, 2023

- “Intel® oneAPI Programming Guide”,
<https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top.html>
- “Intel® DPC++ Compatibility Tool”, <https://www.intel.com/content/www/us/en/develop/documentation/oneapi-prhttps://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compatibility-tool.html>

Outline

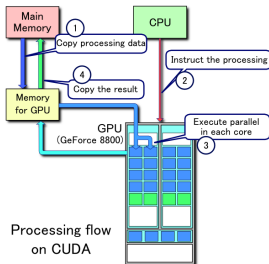
- 1 Introduction
- 2 oneAPI
- 3 Compatibility
- 4 More info



Introduction

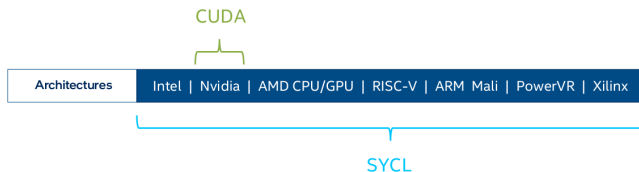
What's CUDA

- CUDA (Compute Unified Device Architecture) is primarily known as a programming model for NVIDIA GPUs
- CUDA helps developers accelerate their applications by harnessing the parallel capabilities of NVIDIA GPU accelerators
- More [info](#)



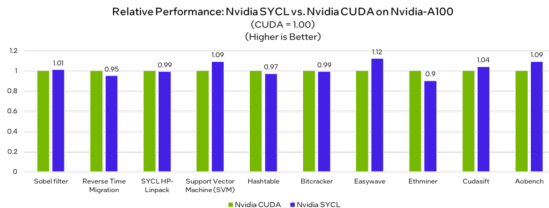
Motivation

- Why migrate code from CUDA?
 - **CUDA**: Supports NVIDIA GPUs
- Why translate it to SYCL?
 - Allows code for heterogeneous processors to be written using modern ISO C++ (at least C++ 17)
 - API for programming both CPUs and GPUs
 - **SYCL**: Supported on Intel, NVIDIA, AMD (CPU & GPUs), RISC-V, ARM-Mali, PowerVR, Xilinx...



Choosing SYCL for Accelerators

- SYCL: Open, based on standards (reminder [UXL Foundation](#))
- Performance across multiple architectures.
- Performance comparable to CUDA on NVIDIA GPUs.
- Code migration possible with SYCLomatic or DPCT



Testing Date: Performance results are based on testing by Intel and August 5, 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, code 0x000303, GPU: Nvidia A100 PCIe 80GB GPU memory. Software: SYCL: open source/CLANG 15.0.0, CUDA 11.7 with NVIDIA NVCC 11.7.64, cuMath 11.7, cuDNN 8.7, Ubuntu 22.04.1, SYCL: open source/CLANG compiler switches: -fexptarget=nvptx64-nvidia-cuda, NVIDIA NVCC compiler switches: -cO3 -gencode arch=compute_86,code=sm_86. Represented workload with Intel optimizations.

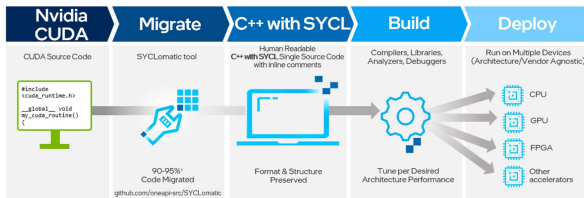
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration-disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/performance/cx300. Your costs and results may vary.

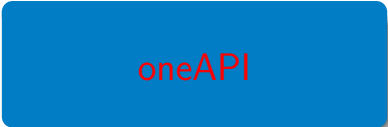
Migration Tool

- **SYCLomatic** helps developers migrate CUDA code to SYCL
 - Automatically converts between 90-95% of CUDA code to SYCL
 - Developers can complete the process with manual coding or by adapting the code
- Intel® DPC++ Compatibility Tool or **DPCT** is Intel's implementation available in the Intel® oneAPI Base Toolkit

CUDA* to SYCL* Code Migration & Development Workflow



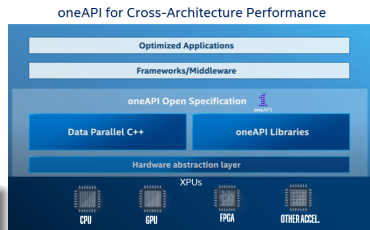
* Intel estimates as of March 2023. Based on measurements on a set of 85 HPC benchmarks and samples, with examples like Rodas, SHOC, PENNANT. Results may vary.
* Other names and brands may be claimed as the property of others. SYCL is a trademark of the Khronos Group Inc.



- Standard-based language: C++ and SYCL
- Powerful APIs for accelerating domain-specific functions

Solution to unique provider

- Open standard to promote community and industry support
- Allows code reuse across different architectures and providers

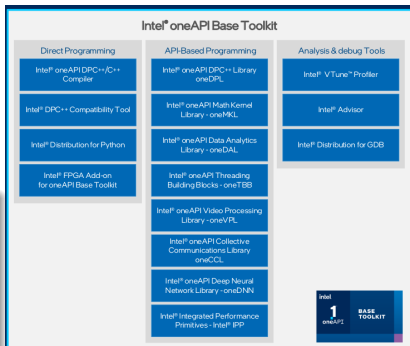


Compilador DPC++

- Basic set of high-performance tools and libraries
- C++ compiler with SYCL support (heterogeneous computing)

Características

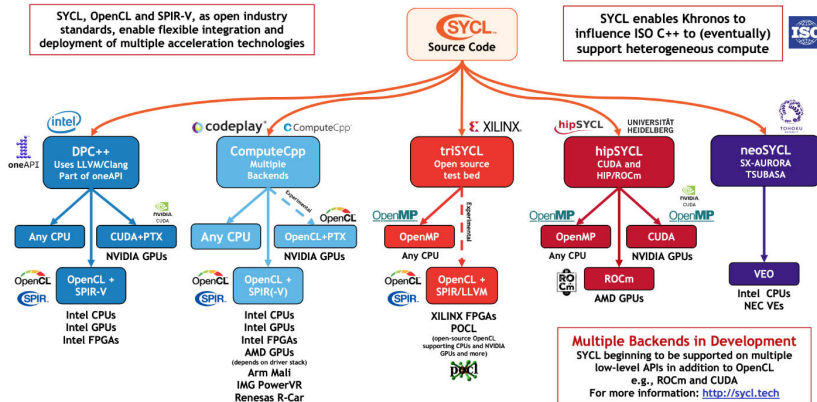
- Data Parallel C++ Compiler
- Portability with SYCLomatic
- Python distribution (optimized libraries scikit-learn, NumPy)



SYCL Implementations

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

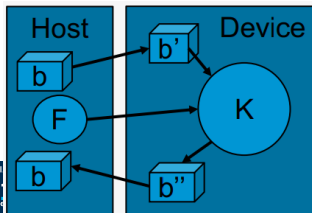
SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute



Pseudo-code

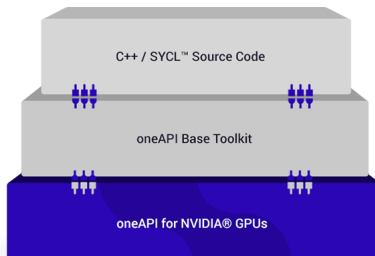
pseudo_code.cpp

```
selector = default_selector()
q = queue(selector)
b = buffer (double, 1000)
q.submit (
    F ( ) {
        a = accessor(b, READ_AND_WRITE)
        K (i) {
            a[i] = a[i] * 2
        }
    }
)
q.wait_and_throw()
a2 = accessor(b, READ_ONLY)
printf("%f", a2[0])
```



Codeplay plugin for NVIDIA GPUs

- Support other vendors such as GPUs from **NVIDIA** or **AMD**
 - Plugins compatibles with the compiler oneAPI DPC++/C++
 - Support plugin for **NVIDIA>sm_50**
 - Support *beta* plugin for **AMD**





Compatibility

CUDA-SYCL Migration

- Some equivalences form CUDA and SYCL API

Action	CUDA	SYCL
Header File	<code>#include<cuda_runtime.h></code>	<code>#include<sycl/sycl.hpp></code>
Create a CUDA stream	<code>cudaStream_t stream1;</code>	<code>sycl::queue q;</code>
Allocate memory on device	<code>cudaMalloc()</code>	<code>sycl::malloc_device()</code>
Memset on device	<code>cudaMemsetAsync()</code>	<code>q.memset()</code>
Memcpy from host to device	<code>cudaMemcpyAsync()</code>	<code>q.memcpy()</code>
Submit kernel to device	<code>kernel_function <<<NUM_BLOCKS, N>>>();</code>	<code>q.parallel_for(sycl::nd_range<1>(N, WG), [=](sycl::nd_item<1> item) { kernel_function(); });</code>
Free device allocation	<code>cudaFree()</code>	<code>sycl::free()</code>
Synchronize host and device	<code>cudaStreamSynchronize(stream)</code>	<code>q.wait()</code>

CUDA-SYCL Migration

- Some equivalences form CUDA and SYCL API

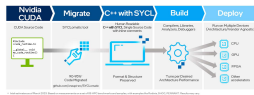
Action	CUDA	SYCL
Shared Local memory	<code>__shared float local_data[N]</code>	<code>sycl::local_accessor<float, 1> local_data(N, h)</code>
Synchronization	<code>__syncthreads()</code>	<code>sycl::group_barrier(group)</code>
Atomic Add	<code>atomicAdd()</code>	<code>auto a = sycl::atomic_ref() a.fetch_add()</code>
Block Thread Index	<code>threadIdx.x</code>	<code>item.get_local_id()</code>

- It is also possible to translate algebra invocation through **cuBLAS** with the oneAPI **oneMKL** library:
 - `cublasSgemv` por `oneapi::mkl::blas::column_major::gemv`

CUDA-SYCL Migration

- Helps developers in migrating code written in CUDA to SYCL¹, generating readable code whenever possible
- 90-95% of the code is automatically migrated²
- Provides comments to assist developers in completing the migration

CUDA¹ to SYCL¹ Code Migration & Development Workflow

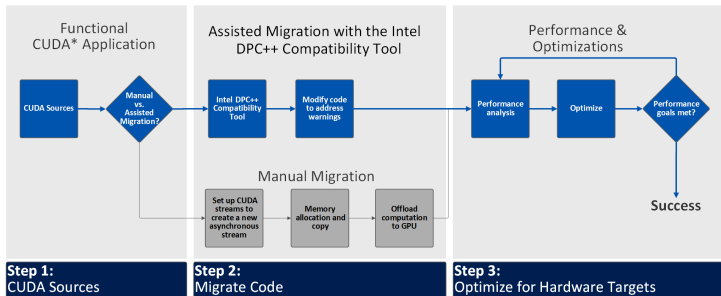


Compatibility Tool (DPCT/SYCLomatic)

- ① Prepare the CUDA source for migration:
 - The DPCT/SYCLomatic compatibility tool searches for CUDA headers
- ② Project migration: **execution of the compatibility tool**
 - For simple projects, migrate file by file
 - For complex projects (Microsoft Visual Studio project or Make/Cmake) is recommendable to build database
- ③ Review converted code:
 - Output files contain annotations to assist in migrating any remaining code that could not be automatically migrated³
- ④ Compilation with Intel® oneAPI DPC++/C++

³Developer Guide and Reference <https://software.intel.com/content/www/us/en/develop/documentation/intel-dpcpp-compatibility-tool-user-guide/top/diagnostics-reference.html>

Compatibility Tool (DPCT/SYCLomatic)



Before starting (DPCT/SYCLomatic)

- Intel® DPC++ Compatibility Tool assist to CUDA code migration to Data Parallel C++ (DPC++)
 - Look at [guide and refence information](#)
 - Visit the [Resease Notes](#) to find more information

Software Requirements

- Installation of [SYCLomatic](#) or use DPCT/SYCLomatic from the [Intel oneAPI Base Toolkit](#)
 - Prepare the environment with source
`/opt/intel/oneapi/setvars.sh`
- CUDA headers are mandatory
 - `/usr/local/cuda/include`
 - `/usr/local/cuda-x.y/include`, where x.y must be some of the following versions: 8.0, 9.x, 10.1, 10.2, 11.0~11.8, 12.0

SYCLomatic instalation

- Download or clone the Github repository from [SYCLomatic](#), download and export the *PATH*

Terminal #1

```
user@comp-$ nvidia-smi
Mon Nov 13 20:07:34 2023

+-----+
| NVIDIA-SMI 470.223.02    Driver Version: 470.223.02    CUDA Version: 11.4    |
+-----+-----+
| 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 K20c        Off      | 00000000:01:00:0  Off  |            Default  |
| 30%   26C    P8      16W / 225W |  0MiB /  4743MiB |      0%              |
+-----+-----+
N/A

+user@comp:~$ mkdir syclomatic
user@comp:~$ cd syclomatic
user@comp:~/syclomatic$ wget https://github.com/oneapi-src/SYCLomatic/releases/download/20231113/linux_release
user@comp:~/syclomatic$ tar -zxvf linux_release.tgz
user@comp:~/syclomatic$ export PATH="/home/$USER/syclomatic/bin:$PATH"
user@comp:~/syclomatic$ c2s --help
USAGE: c2s [options] [<source0> ... <sourceN>]

OPTIONS:
    --always-use-async-handler    - Use async exception handler when creating new sycl::queue with dpct
                                   in addition to default dpct::get_default_queue. Default: off.
    --analysis-scope-path=<dir>   - The directory path for the analysis scope of the source tree that
                                   Default: the value of --in-root.
```

DPCT/SYCLomatic usage

- Migrate a single CUDA source file: `c2s test.cu`
- Migrate a single CUDA source file and copy all SYCLomatic auxiliary header files: `c2s test.cu --gen-helper-function`
- Migrate a single CUDA source file to a directory:

`c2s test.cu --out-root sycl_dir`

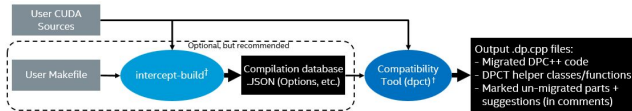
- Migrate a single CUDA source file with a CUDA installation:

`c2s test.cu --cuda-include-path /tmp/cuda/include`

- Migrate a CUDA project with a makefile:

`c2s --project-dir /path/to/project:`

- 1 `intercept-build make`
- 2 `c2s -p compile_command.json`



† Certain CUDA language header files may need to be accessible to the Intel® DPC++ Compatibility Tool

Warnings (DPCT/SYCLomatic)

- The DPCT compatibility tool identifies parts of the code that may require attention from the programmer.
- Comments in the code indicating compatibility or correctness with DPC++
 - Warnings issues as :

warnings.cpp

```
/path/to/file.hpp:26:1: warning: DPCT1003:0: Migrated API does not return error  
      code. (*,0) is inserted. You may need to rewrite this code.  
// source code line for which warning was generated  
~
```

- More details about the meaning of the warnings can be found in the [diagnostic reference guide](#)

Easy project (a single source file)

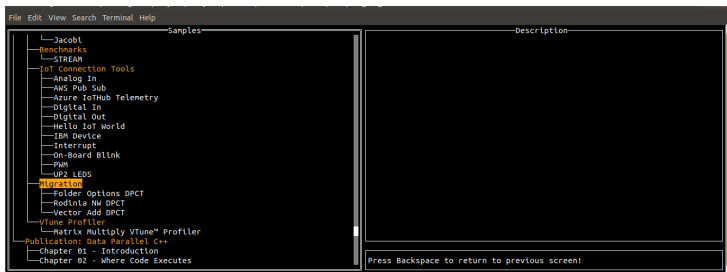
```
1 #include <cuda.h>
2 #include <stdio.h>
3
4 const int vector_size = 256;
5
6 __global__ void SimpleAddKernel(float *A, int offset)
7 {
8     A[threadIdx.x] = threadIdx.x + offset;
9 }
10
11 int main()
12 {
13     float *d_A;
14     int offset = 10000;
15
16     cudaMalloc(&d_A, vector_size * sizeof(float));
17     SimpleAddKernel<<<1, vector_size>>>(d_A, offset);
18
19     float result(vector_size) = { };
20     cudaMemcpy(result, d_A, vector_size*sizeof(float), cudaMemcpyDeviceToHost);
21
22     cudaFree(d_A);
23
24     for (int i = 0; i < vector_size; ++i) {
25         if (i % 8 == 0) printf("n");
26         printf("%4.1f ", result[i]);
27     }
28
29     return 0;
30 }
```

```
1 #include <CL/sycl.hpp>
2 #include <dpct/dpct.hpp>
3 #include <stdio.h>
4
5 const int vector_size = 256;
6
7 void SimpleAddKernel(float *A, int offset, sycl::nd_item<3> item_ct1)
8 {
9     A[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + offset;
10 }
11
12 int main()
13 {
14     dpct::device_ext &dev_ct1 = dpct::get_current_device();
15     sycl::queue &q_ct1 = dev_ct1.default_queue();
16     float *d_A;
17     int offset = 10000;
18
19     d_A = sycl::malloc_device<float>(vector_size, q_ct1);
20     q_ct1.submit([&q_ct1:&handler &agh] {
21         agh.parallel_for(sycl::nd_range(sycl::range(1, 1, vector_size),
22                                         sycl::range(1, 1, vector_size)),
23                         [=](sycl::nd_item<3> item_ct1) {
24                             SimpleAddKernel(d_A, offset, item_ct1);
25                         });
26     });
27
28     float result(vector_size) = { };
29     q_ct1.memcpy(result, d_A, vector_size * sizeof(float)).wait();
30
31     sycl::free(d_A, q_ct1);
32
33     for (int i = 0; i < vector_size; ++i) {
34         if (i % 8 == 0) printf("n");
35         printf("%4.1f ", result[i]);
36     }
37
38     return 0;
39 }
```


- 1 Connect to the Intel Developer Cloud
- 2 In the **Training and Workshops** section
- 3 ... but we will work on **Migrate from CUDA® to C++ with SYCL®**
 - <https://console.cloud.intel.com/training/detail/f82a85e5-acc8-489c-8e1d-0f516c02a662>
- 4 Inside the IDC, open the notebook in the path
Training/HPC/cuda-to-sycl-migration-training/01_SYCL_Migration_Simple_VectorAdd.ipynb

Other examples

- You can find more examples of migration into [GitHub de oneAPI-samples](#)
- ... or using the tool **oneapi-cli**



Multi-source project

- Let's try another example with the **oneapi-cli** tool:
folder-options-dpct
- This example consists of three CUDA files (main.cu, util.cu, and util.h) located in two folders (foo and bar):

```
foo
bar
    util.cu
    util.h
main.cu
```

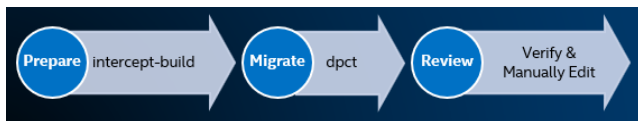
Steps to perform

- 1 Code migration with the tool **c2s**
- 2 Edit and correct the possible issues. It is recommended to check the [Diagnostic Reference](#)
- 3 Compile with DPC++ compile: invoke original *Makefile* in the the target folder but including the **ipcx** as compiler

CUDA Project with Multiple Sources

- Using the DPCT/SYCLomatic tool for multiple sources

- 1 Prepare Intercept-build tool: `intercept-build make`
 - Generates a JSON file with all the source files involved in the project
- 2 DPCT/c2s tool: `c2s -p compile_commands.json`
- 3 Review diagnostic messages⁴ using the reference and manually fix other less obvious issues



⁴Developer Guide and Reference <https://software.intel.com/content/www/us/en/develop/documentation/intel-dpcpp-compatibility-tool-user-guide/top/diagnostics-reference.html>

Hands-on

- Let's try another example with the **oneapi-cli**:
oneAPI-Tools->Migration->Folder Options DPCT
- Now, we will migrate a CUDA project with a makefile:
 - 1 intercept-build make
 - 2 c2s -p compile_command.json

Terminal #1

```
user@comp-$ intercept-build make
user@comp-$ c2s -p compile_commands.json --cuda-include-path=<CUDAPATH_HEADERS>
user@comp-$ cd dpct_output
user@comp-/dpct_output$ make
```

```
dpct_output/foo
    bar
        util.dp.cpp
        util.h
    main.dp.cpp
```

- It is advisable to inspect the migrated source code and identify possible warnings
 - Revise and check the [Diagnostic Reference](#) for more information related to the code **DPCT1015:0**
 - Change `stream_ct1 << "kernel_util,%d\n";` by
`stream_ct1 << "kernel_util," << c << sycl::endl;` in the `util.dp.cpp` file
- Modify the Makefile: `CXX = icpx` and add `-fsycl` to `CPPFLAGS` and `LDFLAGS`
- Finally, go to the *result* folder, compile with `make`, and evaluate the code with *make run*

Terminal #1

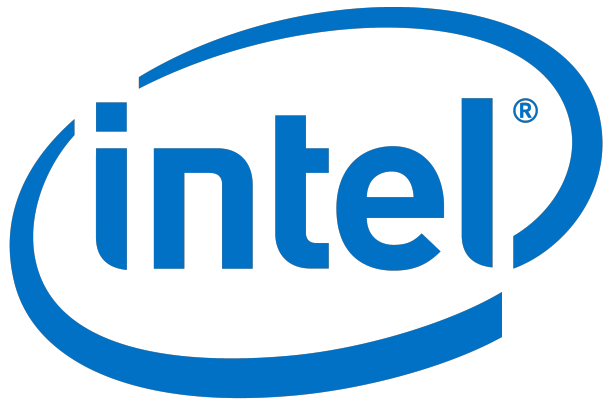
```
user@comp~/.dpct_output$ make
icpx -I./foo -I./foo/bar -fsycl -c -o foo/bar/util.dp.o foo/bar/util.dp.cpp
icpx -I./foo -I./foo/bar -fsycl -c -o foo/main.dp.o foo/main.dp.cpp
icpx ./foo/bar/util.dp.o ./foo/main.dp.o -o foo-bar -fsycl
user@comp~/.dpct_output$ make run
./foo-bar
kernel_main!
kernel_util,2
```



[More info](#)

Available resources

- Intel [oneAPI Base & HPC Toolkit](#)
- Diagnosis [reference for SYCLomatic tool](#)
- Some migration experiences and an application cataloged from [CUDA to SYCL](#)
- Book **Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL** disponible en [el link](#)



Software

Thanks for your attention!!!



Address

Avda. de la industria 4, edif. 1
28108 Alcobendas | Madrid | España



Phone

[+34] 91 663 8683



Email

info@danysoft.com



Website

www.danysoft.com/intel



intel
software extensible