



SYCL Tutorial

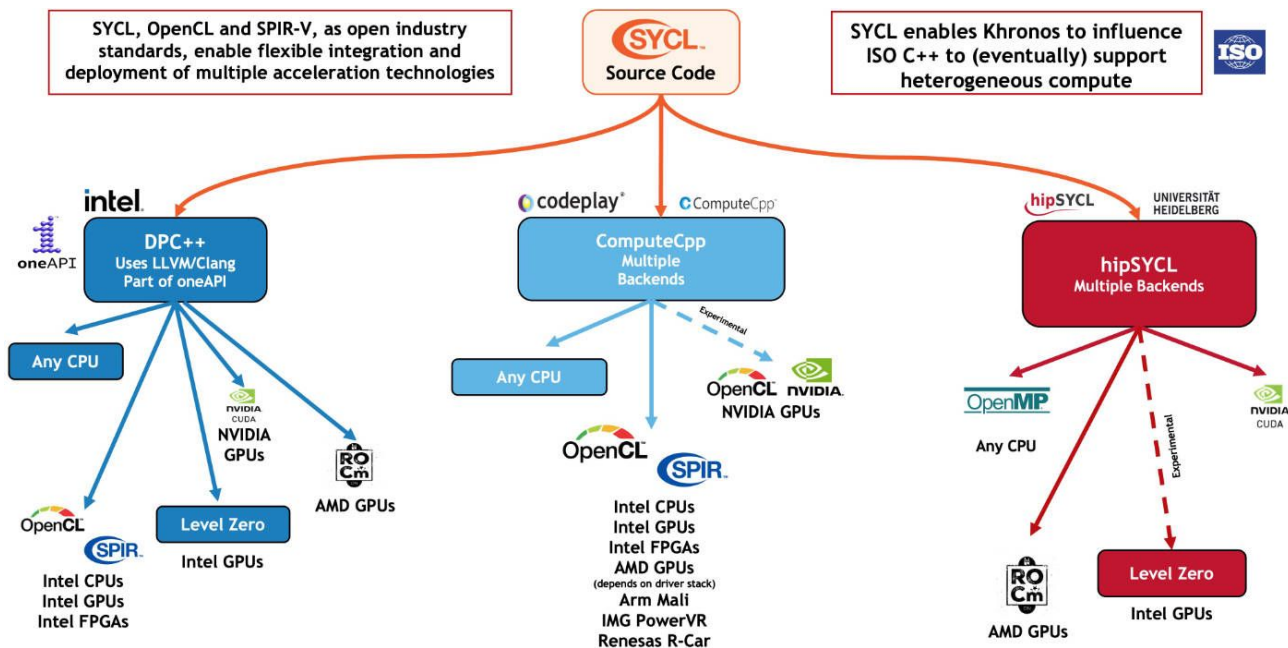
July 20, 2023

What is SYCL

- ❖ Open-standard parallel programming framework that can target a range of hardware accelerators (CPUs, GPUs, FPGA's).
- ❖ SYCL is a C++ library that functions as an OpenCL abstraction layer
 - There is also ongoing development to support CUDA, HIP, and ROCm in the backend

SYCL Implementations

- ❖ SYCL Implementations include
 - ComputeCpp, DPC++, hipSYCL, neoSYCL, and triSYCL



Compiling SYCL Code

- ❖ Compiling SYCL for Different GPUs (Intel support builds)
 - <https://www.intel.com/content/www/us/en/developer/articles/technical/compiling-sycl-with-different-gpus.html>
- ❖ Compile SYCL with Jupyter Notebook on Intel Dev Cloud (Account Required).
 - https://devcloud.intel.com/oneapi/get_started/baseTrainingModules/
- ❖ Compile SYCL with interactive online tool provided by ComputeCPP (deprecated version of SYCL)
 - <https://tech.io/playgrounds/48226/introduction-to-sycl/introduction-to-sycl-2>

Tutorial Module 1: Queues

In SYCL, we refer to the queue as the selection of a device (GPU) that allows the scheduling and execution of kernels (submitted tasks).

```
1 #include <CL/sycl.hpp>
2 #include <iostream>
3
4 // output device information
5 template<typename Queue_type>
6 void output_device_information(Queue_type& Q){
7     std::cout << "DEVICE NAME: "
8               << Q.get_device().template get_info<sycl::info::device::name>()
9               << "\nDEVICE VENDOR: "
10             << Q.get_device().template get_info<sycl::info::device::vendor>()
11             << "\n" << std::endl;
12 }
13
14 int main(){
15     // selecting a default device for queue
16     sycl::queue Q{sycl::default_selector_v};
17     output_device_information(Q);
18 }
```

→ We can select the default devices available on our system.

```
DEVICE NAME: Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz
DEVICE VENDOR: Intel(R) Corporation
```

Tutorial Module 1: Queues

SYCL also has the capability to inquire about available platforms and devices for queue. This allows explicit device selection.

```
27 // selecting a device based on platform and device number
28 sycl::queue get_queue(int platform_index = 0, int device_index = 0){
29     // get the available platforms
30     auto platforms = sycl::platform::get_platforms();
31
32     // select the platform based on the platform index
33     auto selected_platform = platforms[platform_index];
34
35     // get the devices on the selected platform
36     auto devices = selected_platform.get_devices();
37     auto selected_device = devices[device_index];
38
39     // create the queue based on the selected device
40     sycl::queue q(selected_device);
41     return q;
42 }
```

→ Select platform

→ Select device

Tutorial Module 2: Vector Addition (Hello World)

NOTE:
Host = CPU
Device = GPU

- ❖ Vector addition is essentially “Hello World” in the realm of GPGPU programming!

```
40 // copying data from host to device
41 Q.memcpy(A_device, &A_host[0], N*sizeof(double));
42 Q.memcpy(B_device, &B_host[0], N*sizeof(double));
43 Q.wait();
```

→ Copy host to device

```
44
45 // executing the kernel
46 auto event = Q.submit([&](sycl::handler& h){
47     h.parallel_for(sycl::range<1>(N), [=](sycl::id<1> idx){
48         C_device[idx] = A_device[idx] + B_device[idx];
49     });
50 });
```

→ Parallel Vector Add

```
51
52 event.wait();
```

```
53
54 // copying data from device to host
55 Q.memcpy(&C_host[0], C_device, N*sizeof(double)).wait();
```

→ Copy device to host

Tutorial Module 3: Matrix Multiplication

- ❖ Matrix multiplication is another great example when introducing the advantages of GPGPU programming.
 - Embarrassingly parallel and expensive task for a CPU
 - Easy to optimize