

C++ and Khronos computing standards

OpenCL, SYCL (and Vulkan, OpenVX?)

Ronan Keryell (ronan.keryell@amd.com)

Fellow Software Development Engineer @ AMD Research & Advanced Development

Khronos SYCL specification editor, Khronos SPIR & OpenCL member & ISO C++ committee member

2023/06/06 @ Khronos OpenCL WG virtual F2F

Remember C++ ?

2-line description by Bjarne Stroustrup

- ▶ Direct mapping to hardware
- ▶ Zero-overhead abstraction

Modern Python/C/Modern C++/Old C++

▶ Python 3.11

```
v = [ 1, 2, 3, 5, 7 ]  
print(v)
```

<https://godbolt.org/z/Kq9vc1jhY>

▶ C99 (also usable in C++)

```
#include <stdio.h>  
int a[] = { 1, 2, 3, 5, 7 };  
for (int i = 0;  
     i < sizeof(a)/sizeof(a[0]);  
     ++i)  
    printf("%d ", a[i]);
```

▶ C++23

```
import std;  
std::vector v { 1, 2, 3, 5, 7 };  
std::println("{} ", v);
```

▶ C++03

```
#include <iostream>  
#include <vector>  
std::vector<int> v;  
v.push_back(1);  
v.push_back(2);  
v.push_back(3);  
v.push_back(5);  
v.push_back(7);  
for (std::vector<int>::iterator i =  
     v.begin(); i != v.end(); ++i)  
    std::cout << *i << std::endl;
```

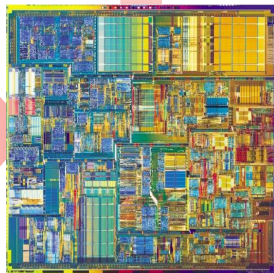
But...

No heterogeneous computing in
C++



3D for the Web

- Real-time apps and games in-browser
- Efficiently delivering runtime 3D assets



Vision and Neural Networks

- Tracking and odometry
- Scene analysis/understanding
- Neural Network inferencing



Real-time 2D/3D

- Virtual and Augmented Reality
- Cross-platform gaming and UI
 - CG Visual Effects
- CAD and Product Design
- Safety-critical displays

Parallel Computation

- Machine Learning acceleration
- Embedded vision processing
- High Performance Computing (HPC)



OpenCL and C++

Non-single source API

- ▶ Host C API to control the device with various C++ wrappers and other languages
 - Old Khronos C++ wrapper v1 (C++03)
 - Less old Khronos C++ wrapper v2 (C++11) <https://github.com/khronos.org/OpenCL-CLHPP>
<https://github.com/KhronosGroup/OpenCL-Headers>
 - Boost.Compute <https://github.com/boostorg/compute>
 - + everybody developing own C++ (or other languages) wrappers
- ▶ Kernel source code or binary (SPIR or target)
 - Legacy OpenCL C source code
 - OpenCL C++ 1.0 for OpenCL 2.2
 - Inspired by SYCL 2.2 modernism ☺ but incompatible with legacy OpenCL C ☹
 - Khronos-friendly-but-independent *C++ for OpenCL* in Clang/LLVM
<https://clang.llvm.org/docs/OpenCLSupport.html>
https://www.khronos.org/opencl/assets/CXX_for_OpenCL.html
- ▶ Split-source + JIT → easy meta-programming with strings
 - Common concept graphics and hardware community

Vector addition with OpenCL C++ host API v2

```
#include <iostream>
#include <iterator>

#define CL_HPP_ENABLE_EXCEPTIONS
#include <CL/cl2.hpp>

constexpr size_t N = 3;
using Vector = float[N];

int main() {
    Vector a = { 1, 2, 3 };
    Vector b = { 5, 6, 8 };
    Vector c;

    // The input read-only buffers for OpenCL on default context
    cl::Buffer buffer_a { std::begin(a), std::end(a), true };
    cl::Buffer buffer_b { std::begin(b), std::end(b), true };

    // The output buffer for OpenCL on default context
    cl::Buffer buffer_c { CL_MEM_WRITE_ONLY, sizeof(c) };

    // Construct an OpenCL program from the source file
    const char kernel_source[] = R"(
__kernel void vector_add(const __global float *a,
```

```
const __global float *b,
__global float *c) {
    c[get_global_id(0)] = a[get_global_id(0)] + b[get_global_id(0)];
}
)";

// Compile and build the program
cl::Program p { kernel_source, true };
// Create the kernel functor taking 3 buffers as parameter
cl::KernelFunctor<cl::Buffer, cl::Buffer, cl::Buffer> k
    { p, "vector_add" };

// Call the kernel with N work-items on default command queue
k(cl::EnqueueArgs(cl::NDRange(N)), buffer_a, buffer_b, buffer_c);

// Get the output data from the accelerator
cl::copy(buffer_c, std::begin(c), std::end(c));

std::cout << std::endl << "Result:" << std::endl;
for(auto e : c)
    std::cout << e << " ";
std::cout << std::endl;
}
```

https://github.com/keryell/heterogeneous_examples/blob/main/vector_add/OpenCL-CLHPP/opencl_vector_add.cpp

SYCL 2020 \equiv heterogeneous simplicity with modern C++

```
#include <iostream>
#include <sycl/sycl.hpp>
constexpr int n = 32;

int main () {
    sycl::buffer<int> buf { n };
    sycl::queue {}.submit([&](auto &h) {
        sycl::accessor a { buf, h, sycl::write_only, sycl::no_init };
        h.parallel_for(n, [=](auto i) { a[i] = i; });
    });
    for (sycl::host_accessor a { buf }; auto e : a)
        std::cout << e << std::endl;
}
```

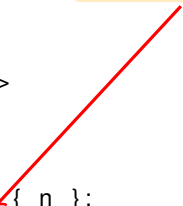

SYCL 2020 \equiv heterogeneous simplicity with modern C++

Abstract storage

► Host or device (remote) memory

```
#include <iostream>
#include <sycl/sycl.hpp>
constexpr int n = 32;

int main () {
    sycl::buffer<int> buf{ n };
    sycl::queue {}.submit([&](auto &h) {
        sycl::accessor a { buf, h, sycl::write_only, sycl::no_init };
        h.parallel_for(n, [=](auto i) { a[i] = i; });
    });
    for (sycl::host_accessor a { buf }; auto e : a)
        std::cout << e << std::endl;
}
```



SYCL 2020 \equiv heterogeneous simplicity with modern C++

Abstract storage

- ▶ Host or device (remote) memory

```
#include <iostream>
#include <sycl/sycl.hpp>
constexpr int n = 32;

int main () {
    sycl::buffer<int> buf{ n };
    sycl::queue {}.submit([&](auto &h) {
        sycl::accessor a { buf, h, sycl::write_only, sycl::no_init };
        h.parallel_for(n, [=](auto i) { a[i] = i; });
    });
    for (sycl::host_accessor a { buf }; auto e : a)
        std::cout << e << std::endl;
}
```

Code executed on device ("kernel")

- ▶ "Single-source"
- ▶ Seamless integration in host code
- ▶ Type-safety
- ▶ Asynchronous execution

SYCL 2020 \equiv heterogeneous simplicity with modern C++

```
#include <iostream>
#include <sycl/sycl.hpp>
constexpr int n = 32;

int main () {
    sycl::buffer<int> buf{ n };
    sycl::queue {}.submit([&](auto &h) {
        sycl::accessor a { buf, h, sycl::write_only, sycl::no_init };
        h.parallel_for(n, [=](auto i) { a[i] = i; });
    });
    for (sycl::host_accessor a { buf }; auto e : a)
        std::cout << e << std::endl;
}
```

Abstract storage

- ▶ Host or device (remote) memory

Code executed on device ("kernel")

- ▶ "Single-source"
- ▶ Seamless integration in host code
- ▶ Type-safety
- ▶ Asynchronous execution

Accessor

- ▶ Express access intention
- ▶ Implicit data flow graph
- ▶ Automatic data transfers across devices
- ▶ Overlap computation & communication

SYCL 2020 \equiv heterogeneous simplicity with modern C++

Queue

- ▶ Direct work to specific accelerator
- ▶ Submission of a command group

Abstract storage

- ▶ Host or device (remote) memory

```
#include <iostream>
#include <sycl/sycl.hpp>
constexpr int n = 32;

int main () {
    sycl::buffer<int> buf{ n };
    sycl::queue {}.submit([&](auto &h) {
        sycl::accessor a { buf, h, sycl::write_only, sycl::no_init };
        h.parallel_for(n, [=](auto i) { a[i] = i; });
    });
    for (sycl::host_accessor a { buf }; auto e : a)
        std::cout << e << std::endl;
}
```

Code executed on device ("kernel")

- ▶ "Single-source"
- ▶ Seamless integration in host code
- ▶ Type-safety
- ▶ Asynchronous execution

Accessor

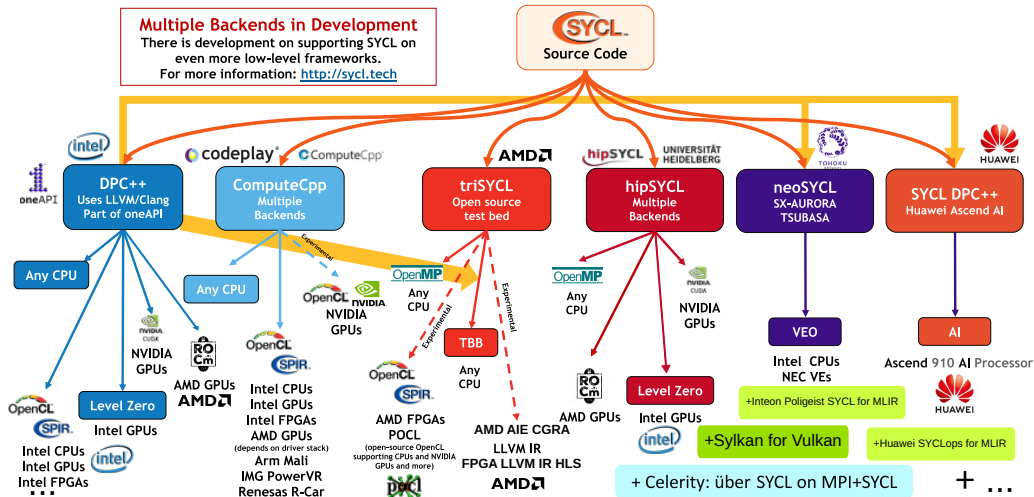
- ▶ Express access intention
- ▶ Implicit data flow graph
- ▶ Automatic data transfers across devices
- ▶ Overlap computation & communication

SYCL grew beyond OpenCL HLM

Multiple Backends in Development

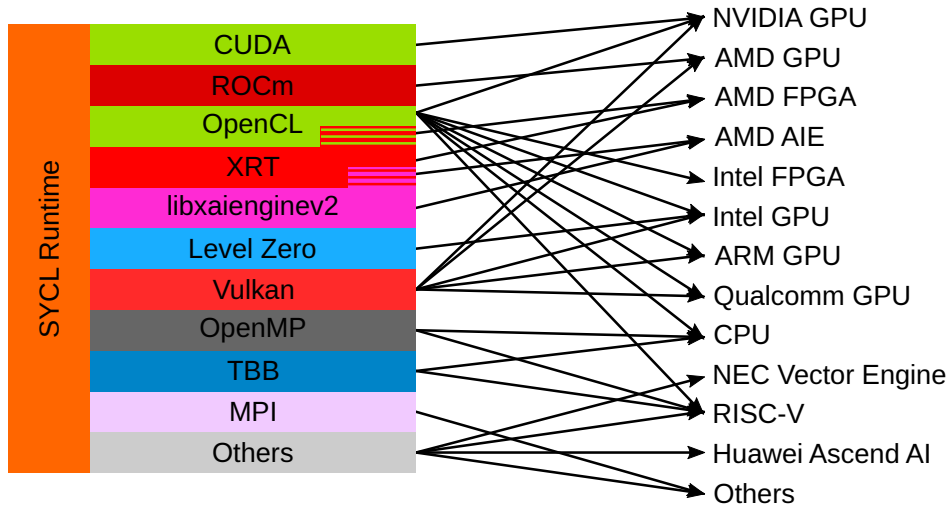
There is development on supporting SYCL on even more low-level frameworks.

For more information: <http://sycl.tech>



<https://www.khronos.org/blog/sycl-2020-what-do-you-need-to-know>

Some of the existing SYCL backends



Unique SYCL feature: interoperability with backends

- ▶ Porting existing code
 - Code already based on OpenCL/CUDA/OpenMP/HIP/... (or whatever backend)
 - Want to change just part of application to use SYCL
- ▶ Incorporating a backend module into a SYCL application
 - Application based on SYCL
 - Want to call some OpenCL/CUDA/OpenMP/HIP/... library (or whatever backend)
- ▶ Take advantage of backend-specific features
- ▶ Disadvantage: reduces portability!
 - Not all implementations may support your backend

➤ Unique feature of SYCL!

Aksel ALPAY, Thomas APPLENCOURT, Gordon BROWN, Ronan KERYELL and Gregory LUECK. "Using interoperability mode in SYCL 2020." In SYCLcon 2022: International Workshop on SYCL. Association for Computing Machinery, May 2022. doi:10.1145/3529538.3529997.

<https://www.iwocl.org/wp-content/uploads/39-presentation-iwocl-syclcon-2022-aksel.pdf>

<https://www.youtube.com/watch?v=XIPhuesdqYE>

SYCL interoperability as OpenCL C++ wrapper

```
#include <cassert>
#include <cstdlib>
#include <sycl/sycl.hpp>
#include <CL/opencl.h>

constexpr int size = 4;

auto check_error(auto&& function) {
    cl_int err;
    auto ret = function(&err);
    if (err != CL_SUCCESS)
        std::exit(err);
    return ret;
};

int main() {
    sycl::buffer<int> a { size };
    sycl::buffer<int> b { size };
    sycl::buffer<int> c { size };

    {
        sycl::host_accessor a_a { a };
        sycl::host_accessor a_b { b };
        for (int i = 0; i < size; ++i) {
            a_a[i] = i;
            a_b[i] = i + 42;
        }
    }

    sycl::queue q;
    std::array kernel_source { R"(
        __kernel void vector_add(const __global float *a,
                                const __global float *b,
                                __global float *c) {
```

```
        c[get_global_id(0)] = a[get_global_id(0)] + b[get_global_id(0)];
    }" };
    cl_context oc = sycl::get_native<sycl::backend::opencl>(q.get_context());
    auto program = check_error([&](auto err) {
        return clCreateProgramWithSource(oc, kernel_source.size(),
                                         kernel_source.data(), nullptr, err);
    });
    check_error([&](auto err) {
        return (*err =
            clBuildProgram(program, 0, nullptr, nullptr, nullptr, nullptr));
    });
    sycl::kernel k = sycl::make_kernel<sycl::backend::opencl>(
        check_error(
            [&](auto err) { return clCreateKernel(program, "vector_add", err);
        }
        q.get_context());
    q.submit([&](sycl::handler& cgh) {
        cgh.set_args(sycl::accessor { a, cgh, sycl::read_only },
                    sycl::accessor { b, cgh, sycl::read_only },
                    sycl::accessor { c, cgh, sycl::write_only, sycl::no_init });
        cgh.parallel_for(size, k);
    });

    {
        sycl::host_accessor a_a { a };
        sycl::host_accessor a_b { b };
        sycl::host_accessor a_c { c };
        for (int i = 0; i < size; ++i)
            assert(a_c[i] == a_a[i] + a_b[i]);
    }
}
```


Vulkan C++ wrapper

- ▶ <https://github.com/KhronosGroup/Vulkan-Hpp>

```
vk::Instance instance = vk::su::createInstance( AppName, EngineName );  
vk::PhysicalDevice physicalDevice = instance.enumeratePhysicalDevices().front();
```

- ▶ Specific namespace for RAI (Resource Acquisition Is Initialization) API version ≈ SYCL objects


```
vk::raii::Context context;  
vk::raii::Instance instance = vk::raii::su::makeInstance( context, AppName, EngineName );  
vk::raii::PhysicalDevices physicalDevices { instance };
```

- ▶ C++20 designated initializers *à la* Python

```
vk::ApplicationInfo applicationInfo{ .pApplicationName = AppName,  
                                     .applicationVersion = 1,  
                                     .pEngineName = EngineName,  
                                     .engineVersion = 1,  
                                     .apiVersion = VK_API_VERSION_1_1 };
```

- ▶ C++20 modules are coming
- ▶ Other C++ wrappers available too (<https://github.com/andy-thomason/Vookoo>)

Culture

- ▶ OpenCL : mostly runtime API  no big C++ culture inside OpenCL WG ☹
 - Designing C++ API requires some cultural knowledge
 - Several vendor implementations are forks from initial Apple contribution
 - Do not benefit from open-source goodies
 - Huge technical debt, no portable IR,...
 - Require other producers to rely on LLVM IR or MLIR to OpenCL C backend
- ▶ SYCL is all about C++
 - Need to be close to Clang/LLVM upstream
 - Otherwise painful technical debt management
- ▶ Vulkan has some C++ culture in the host API

Unconclusion

- ▶ Do we care about C++?
 - What about other languages? Python, Rust...
- ▶ Is this something for independent projects?
- ▶ Who wants to push C++ (with some human power) inside the OpenCL WG?
- ▶ C++ and API: large design space, depend on C++ version...
- ▶ Cultural? Market? Politics? Religion?
- ▶ Just split-source vs single-source independent from C++ aspects?
 - Like CUDA Driver (OpenCL) vs CUDA Runtime (SYCL)?
- ▶ Provide directions to programmers interested by C++ to pick an API (C++ OpenCL C++ vs SYCL)?
- ▶ oneAPI Unified Runtime (\approx SYCL backend abstraction) could have been based on OpenCL API
- ▶ Look also at Vulkan C++ which is split-source too?
 - Interesting: polymorphic API according to C++ standard used

Remember C++ ?	
Modern Python/C/Modern C++/Old C++	
OpenCL and C++	
Vector addition with OpenCL C++ host API v2	
SYCL 2020 ≡ heterogeneous simplicity with modern C++	
SYCL 2020 ≡ heterogeneous simplicity with modern C++	
SYCL 2020 ≡ heterogeneous simplicity with modern C++	
SYCL 2020 ≡ heterogeneous simplicity with modern C++	

2	SYCL 2020 ≡ heterogeneous simplicity with modern C++	12
3	SYCL grew beyond OpenCL HLM	13
6	Some of the existing SYCL backends	14
7	Unique SYCL feature: interoperability with backends	15
8	SYCL interoperability as OpenCL C++ wrapper	16
9	Vulkan C++ wrapper	17
10	Culture	18
11	Unconclusion	19
	You are here !	20