SYCL for Vitis 2020.2
Harnisch, Gauthier; Gozillon, Andrew; Keryell, Ronan; Yu, Lin-Ya; Wittig, Ralph; Forget, Luc

Accepted/In press: 27/04/2021

Document Version
Peer reviewed version

Link to publication on the UWS Academic Portal

Citation for published version (APA):
SYCL for Vitis 2020.2: SYCL & C++20 on Xilinx FPGA

Luc Forget1 Andrew Gozilloni1 Gauthier Harnisch1 Ronan Keryell1 Lin-Ya Yu1 Ralph Wittig1
1Xilinx 2University of the West of Scotland

Abstract

SYCL is a single-source C++ DSL targeting a large variety of accelerators in a unified way by using different back-ends.
We present an experimental SYCL implementation targeting Xilinx Alveo FPGA cards by merging 2 different open-source implementations, Intel's oneAPI DPC++ with some LLVM passes from SyCL.
The FPGA device configuration is generated by Xilinx Vitis 2020.2 fed with LLVM IR SPIR and Xilinx XRT is used as a host OpenCL API top control the device.

Implementation

SYCL is a single-source C++ DSL targeting a large variety of accelerators in a unified way by using different back-ends.
We support 3 emulation targets and 1 for real hardware execution
Allow better control on the design and performances
Allow better control on the design and performances
Hardware emulation
Because of the high-level nature of the SYCL API, it is easy to use.
Converting SPIR-V to SYCL
Converting SPIR-V to SYCL
Converting SPIR-V to SYCL
Converting SPIR-V to SYCL
Converting SPIR-V to SYCL
Converting SPIR-V to SYCL
Converting SPIR-V to SYCL
Converting SPIR-V to SYCL

Usage
Using multiple devices in the same single-source application

Future Work

Focus more on performance
Expose more hardware details
Give more control over HLS to the user
Better adapt the optimizations to FPGA

Notes

1. Device front-end, will only emit device code
2. Run optimizations
3. Convert SPIR-V builtin to "SPIR-df (de facto)"
4. Downgrade the IR to LLVM 6.x
5. Compile kernel
6. Package the device image as data for the host
7. Assemble the packed device image into a .o
8. Generate the inclusion header
9. Compile the host code
10. Link everything together

```cpp
main() {
#include <sycl/sycl.hpp>
#include <iostream>

auto run = [&](sycl::kernel_selector {}, 
    sycl::accessor Accessor) {

    // Implement a generic heterogeneous "executor"
    auto run = [](sycl::host_selector {}, 
        auto a) { a[i] = i; }; // CPU
    run(sycl::accelerator_selector {}, 
        auto a) { a[i] = i; }); // GPU
    run(sycl::gpu_selector {}, 
        auto a) { a[i] = i; }); // GPU

    sycl::accessor acc (v);
    for (int i = 0; i < v.get_count(); ++i)
        std::cout << acc[i] << "\n";
    std::cout << std::endl;
}
```

```
Future Work

Focus more on performance
Expose more hardware details
Give more control over HLS to the user
Better adapt the optimizations to FPGA
```

```
Notes

1. Device front-end, will only emit device code
2. Run optimizations
3. Convert SPIR-V builtin to "SPIR-df (de facto)"
4. Downgrade the IR to LLVM 6.x
5. Compile kernel
6. Package the device image as data for the host
7. Assemble the packed device image into a .o
8. Generate the inclusion header
9. Compile the host code
10. Link everything together

```cpp
main() {
#include <sycl/sycl.hpp>
#include <iostream>

auto run = [&](sycl::kernel_selector {}, 
    sycl::accessor Accessor) {

    // Implement a generic heterogeneous "executor"
    auto run = [](sycl::host_selector {}, 
        auto a) { a[i] = i; }; // CPU
    run(sycl::accelerator_selector {}, 
        auto a) { a[i] = i; }); // GPU
    run(sycl::gpu_selector {}, 
        auto a) { a[i] = i; }); // GPU

    sycl::accessor acc (v);
    for (int i = 0; i < v.get_count(); ++i)
        std::cout << acc[i] << "\n";
    std::cout << std::endl;
}