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):

General rights
Copyright and moral rights for the publications made accessible in the UWS Academic Portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
If you believe that this document breaches copyright please contact pure@uws.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
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
Early version, also known as pre-print

Link to publication on the UWS Academic Portal

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the UWS Academic Portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
If you believe that this document breaches copyright please contact pure@uws.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
SYCL for Vitis 2020.2: SYCL & C++20 on Xilinx FPGA

Luc Forget1 Andrew Gozillon2 Gautier Harnisch1 Ronan Kereyll1 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 IrisC.

The FPGA device configuration is generated by Xilinx Vitis 2020.2 fed with LLVM IR SPIR and Xilinx XRT used as a host OpenCL API top control the device.

We present an experimental SYCL implementation targeting Xilinx Alveo FPGA cards by merging 10. Link everything together

The compilation flow required changes ▶

SYCL for FPGA is trying to make it simpler

FPGA are hard to program


Motivation

FPGA are hard to program

- HLS (High-Level Synthesis) has made it better
- Tools use lots of non-standard language extensions
- Tools are usually split source
- SYCL for FPGA is trying to make it simpler
- Single-source
- Uses the usual compiling process of C++
- Pure modern C++, can be implemented as a normal library for host only execution

Implementation

- Based on Intel’s oneAPI DPC++ because:
  - Open-source  
  - Using OpenCL  
  - Based on LLVM latest ToT
- We use Xilinx’s OpenCL runtime from XRT
- The compilation flow required changes
  - Using Vitis’s v++ as backend compiler
  - Needs downgrading from LLVM ToT to LLVM 6.x
  - Needs converting SPIR-V builtins to “SPIR-df (de facto)"
  - We tweaked the optimization pipeline

SYCL runtime is used as normal C++ library and does not use Vitis at all

Any C++ compiler can be used to compile

Fastest to compile and execute on CPU

Kernel code will be executed natively on the host

Can use runtime checkers like: sanitizers, valgrind... and usual debuggers

But its the furthest from hardware

Software emulation

- Needs to use our custom compiler and use Vitis compiler
- Kernel code is run natively on the host with the Vitis software emulator
- Kernel code is isolated from host code

Hardware emulation

- Needs to use our custom compiler and Vitis RTL simulator
- Kernel code is isolated from host code

Hardware execution on FPGA

- Generates reports about the real resource usage and timings
- Kernel code is executed on the FPGA
- Slowest to compile including Vitis RTL synthesis and FPGA place & route

FPGA-specific extensions to the SYCL standard

- Allows better control on the design and performances
  - Pipeline annotations
  - For loops
  - Partition array
  - Kernel backend compilation option for each kernel
  - Partition array
  - Kernel code is run natively on the host with the Vitis software emulator
  - Kernel code is isolated from host code

Using multiple devices in the same single-source application

```
#include <sycl/sycl.hpp>
#include <iostream>

int main() {
    sycl::buffer<int, 1> arr{ ... };
    sycl::host_accessor acc { v };  
    for (int i = 0; i != v.get_count(); ++i)
        acc[i] = 2*acc[i]; });
    return 0;
}
```

```
auto run = sycl::parallel_for<>
    (sycl::range<1>(), 
     [=] (int i) {
         a[i] = i * i;  
     });
```

```
auto do_host_or_device(sycl::accessor_<T, 0>& a) {
    if (a.get_device().is_device_type<sycl::device::host>())
        // CPU
        return a.read();
    else
        // FPGA
        return a.read();
}
```

```
// Partition array
// ... partition_array<int, 10, sycl::partition::complete<int>> arr(...);  
```

```
// CPU
auto run = sycl::parallel_for<>
    (sycl::range<1>(), 
     [=] (int i) {
         a[i] = i * i;  
     });
```

```
// FPGA
auto run = sycl::parallel_for<>
    (sycl::range<1>(), 
     [=] (int i) {
         a[i] = i * i;  
     });
```

```
// 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
- Usability
- Fix more compatibility issues between the SYCL toolchain and Vitis
- Better adapt the optimizations to FPGA
- Expose more hardware details
- Tools use lots of non-standard language extensions
- Tools are usually split source
- SYCL for FPGA is trying to make it simpler
- Single-source
- Uses the usual compiling process of C++
- Pure modern C++, can be implemented as a normal library for host only execution

```
```
```
```