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.
The FPGA device configuration is generated by Xilinx Vitis 2020.2 fed with LL VM IR SPIR and Xilinx triSYCL. 2 different open-source implementations, Intel’s oneAPI DPC++ with some LL VM passes from We present an experimental SYCL implementation targeting Xilinx Alveo FPGA cards by merging different back-ends.

### Implementation

| 1. Device front-end, will only emit device code  |
| 2. Run optimizations                               |
| 3. Link the device code with the device runtime   |
| 4. Compile each kernel with the backend           |
| 5. Link all kernels                                |
| 6. Package the device image as data for the host   |
| 7. Assemble the packaged device image into a .o    |
| 8. Generate the inclusion header                   |
| 9. Compile the host code                          |
| 10. Link everything together                       |

### Motivation

- 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 LL VM passes from triSYCL.
- The FPGA device configuration is generated by Xilinx Vitis 2020.2 fed with LL VM IR SPIR and Xilinx triSYCL.

### SYCL for Vitis 2020.2: SYCL & C++20 on Xilinx FPGA

### Targets

- We support 3 emulation targets and 1 for real hardware execution
  - Library only
  - 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 it's 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 synthesis
  - Generate reports about expected resource usage and timings
  - Generate reports about expected resource usage and timings
  - Kernel code is executed in 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

- Allow better control on the design and performances
  - Pipeline annotations
    - for (...)  
    - sycl: pipeline([a] [ ... ]); 
    - The loop for will get each stage pipelined in hardware and this will speedup the loop at the cost of using a little bit more hardware and latency.
- DDR bank access property
  - sycl: accessor Accessor
    - Buffer, cgh, sycl: OPENAPI: accessor_property_list(sycl: xilinx: ddr_bank(0));
    - The memory accessed by Accessor will be placed in the DDR memory bank 1
  - Kernel backend compilation option for each kernel
    - cgh: single_task(xilinx: kernel_param("--kernel_frequency 300", arr));
    - The backend compiler invocation for this kernel will receive --kernel_frequency 300 forcing the generation with a kernel clock of 300 MHz

### Future Work

- Focus more on performance
- Expose more hardware details
- Give more control over HLS to the user
- Better adapt the optimizations to FPGA
- Fix more compatibility issues between the SYCL toolchain and Vitis
- Better adapt the optimizations to FPGA
- Give more control over HLS to the user
- More...