SYCL for Xilinx Versal ACAP AIE CGRA
Keryell, Ronan; Gozillon, Andrew; Harnisch, Gauthier; Kwon, Hyun; Chakaravarthy, Ravikumar; Wittig, Ralph

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.

Download date: 08 Jun 2022
SYCL for Xilinx Versal ACAP AIE CGRA
Keryell, Ronan; Gozillon, Andrew; Harnisch, Gauthier; Kwon, Hyun; Chakaravarthy, Ravikumar; Wittig, Ralph

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):
Keryell, R., Gozillon, A., Harnisch, G., Kwon, H., Chakaravarthy, R., & Wittig, R. (Accepted/In press). SYCL for Xilinx Versal ACAP AIE CGRA. Poster session presented at IWOCL SYCLCon 2021,

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 Xilinx Versal ACAP AIE CGRA

Andrew Gozillon\textsuperscript{2} Gauthier Harnisch\textsuperscript{1} Ronan Keryell\textsuperscript{1} Hyun Kwon\textsuperscript{1} Ravikumar Chakaravarthy\textsuperscript{1} Ralph Wittig\textsuperscript{1}

\textsuperscript{1}Xilinx \textsuperscript{2}University 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 backends.

Xilinx Versa ACAP is a new system-on-chip (SoC) device integrating various computing resources like various CPUs, as FPGA, a coarse-grain reconfigurable area (CGRA), etc., interconnected by different network-on-chip (NoC). The AIE CGRA is an array of 400 VLIW DSP operating on 512-bit vectors with their own neighborhood distributed memory (32 KiB data, 16 KiB instructions).

The host runtime runs on the ARM A72 CPU of the ACAP and controls the CGRA through the Xilinx instructions.

We expose architectural details to the programmer through some SYCL extensions and extend SYCL with a distributed memory (32 KiB data, 16 KiB instructions).

Type-safe access to heterogeneous neighbor memories (PDE, stencil...)

<table>
<thead>
<tr>
<th>auto t::own = t::mem();</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (int i = 0; i &lt; image_size; ++i)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>// Compute dx/dt</td>
</tr>
<tr>
<td>auto north = t::mem()[i] + 1 - t::mem()[i];</td>
</tr>
<tr>
<td>// Integrate horizontal speed</td>
</tr>
<tr>
<td>t::mem()[i] = north/alpha;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>t::barrier();</td>
</tr>
<tr>
<td>if constexpr (t::is_memory_module_south())</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>auto below = t::mem()[i];</td>
</tr>
<tr>
<td>for (int i = 0; i &lt; image_size; ++i)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>below_v[image_size - 1 - i]</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Different multi-level implementation/emulation for hardware-software co-design

Full SYCL compiler & runtime implementation

- Run on real hardware or hardware simulator
- Pure SYCL C++ implementation
- No specific compiler required
- Run on (laptop) host CPU at full C++ speed, standard debugging, thread-simulator of hardware features...
- 1 thread per host... thread, 1 thread per AIE tile, 1 thread per GPU work-item, 1 thread per FPGA work-item
- Easy code instrumentation for statistics by adapting SYCL C++ classes
- Use normal debugger
- Gdb is scriptable in Python to expose new features
- Can experiment with Xilinx devices from year 2030
- Mix-and-match
- Run some parts of the hardware remotely or in simulators
- Allow kernels on host CPU while using memory-mapped real hardware (DMA, AXI streams, NoC...)
- Distribute execution across data-center (Celerity SYCL for MPI+SYCL)

Implementation

Conclusion

- Provide direct programming in C/C++ and SYCL for CGRA like Xilinx ACAP VCU102
- Complement existing Xilinx ML environment and Xilinx ADF for Xilinx’s Process Network
- Follow SYCL philosophy: based on plain C++ without extensions
- Implementation based on tSYCL, amAPI DPC++ and Synopsis ASIP Designer chessa-clang
- Single-source pure C++ is incredibly powerful for hardware-software co-design
- Provide CPU emulation for architecture research and co-design
- Can detect hardware concurrency issues with CPU emulation
- Memory-mapped and merged into https://github.com/tziSYCL/tziSYCL
- Retargetable to other CGRA

First approach: expose geographical view of the chip

- Each tile is a sub-device with some neighborhood
- Can launch tasks on each tile
- Can use AIE-specific functions inside kernels
- Neighbor communications
- DMA
- Hardware locks
- Control of the AIE NoC done by host
- Close to usual SYCL programming model otherwise

Second approach: entangled mode by weaving tiles and memories

- Plan SYCL can be cumbersome with 400 tiles to program...
- Provide a cooperative mode allowing meta-programming tiles and memories
- Compile-time specialization of tiles (instructions) and memories (data)
- Allow each tile to access neighbor tile memories in a type-safe way

SYCL with individual tile model: 1 tile \( \equiv \) 1 sub-device

```cpp
#include <iostream>
#include <type_traits>

using namespace std;

int main() {
    // Define an AIE CGRA with all the tiles of a VCU102
    acap::aie::device<acap::aie::layout::full> aie;
    // Distributed work on each tile, which is SYCL sub-device
    d.for_each_tile([&](auto t) { |
        // This will instantiate uniformly the same lambda for all the tiles as the tile device compiler is executed only once, since each tile has the same code
        t.single_task(auto x, t)
            { |
            std::cout << "Hello, I am the AIE tile (" << t.x() << ", " << t.y() << ") << std::endl;
        });
    // Wait for the end of each tile execution
    d.for_each_tile([&](auto t) { t.wait(); });
    d.tile(3,4).single_task([&] { std::cout << "Hello from (3,4)" << std::endl; });
    d.for_each_tile([&](auto t) { t.wait(); });
}```

CGRA architecture model and its SYCL model

<table>
<thead>
<tr>
<th>CGRA</th>
<th>Host Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIE</td>
<td>ARM A72 CPU</td>
</tr>
<tr>
<td>CGRA</td>
<td>Xilinx instructions</td>
</tr>
<tr>
<td>Host</td>
<td>SYCL runtime</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CGRA architecture model and its SYCL model</th>
</tr>
</thead>
<tbody>
<tr>
<td>acap::aie::tile(2,1)</td>
</tr>
<tr>
<td>acap::aie::tile</td>
</tr>
<tr>
<td>acap::aie::tile</td>
</tr>
<tr>
<td>acap::aie::tile</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CGRA architecture model and its SYCL model</th>
</tr>
</thead>
<tbody>
<tr>
<td>acap::aie::device<a href="">acap::aie::layout::full</a> aie;</td>
</tr>
<tr>
<td>aie.run&lt;tile_prog, tile_memory&gt;();</td>
</tr>
<tr>
<td>aie.run&lt;tile_prog, tile_memory&gt;();</td>
</tr>
</tbody>
</table>

Type-safe access to heterogeneous neighbor memories (PDE, stencil...)

```cpp
auto t::own = t::mem();
for (int i = 0; i < image_size; ++i) { |
    // Compute dx/dt
    auto north = t::mem()[i] + 1 - t::mem()[i];
    // Integrate horizontal speed
    t::mem()[i] = north/alpha;
}

// Barrier
if constexpr (t::is_memory_module_south()) {
    auto below = t::mem()[i];
    for (int i = 0; i < image_size; ++i) |
    { |
    below_v[image_size - 1 - i] = t::mem()[i]; |
}
```