Enabling SYCL Support for embedded Xilinx FPGAs

DHPCC++ Workshop at EuroPar 2022

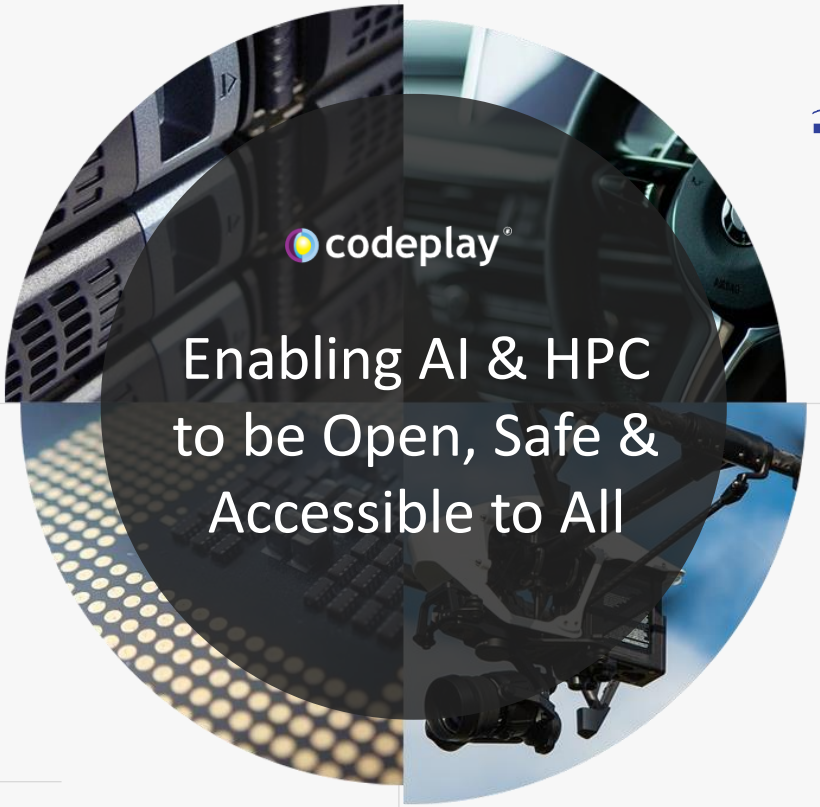
23rd August 2022

Company

Leaders in enabling high-performance software solutions for new AI processing systems

Enabling the toughest processors with tools and middleware based on open standards

Established 2002 in Scotland, acquired by Intel in 2022 and now ~90 employees.



Partners

intel, BROADCOM, SYNOPSYS, CEVA, Imagination, RENESAS, KUC Kyoto Microcomputer Co., Ltd., NSI-TEXE, Berkeley Lab, OAK RIDGE National Laboratory, Argonne National Laboratory

And many more!

Supported Solutions

oneAPI

An open, cross-industry, SYCL based, unified, multiarchitecture, multi-vendor programming model that delivers a common developer experience across accelerator architectures

ComputeCpp™

C++ platform via the SYCL™ open standard, enabling vision & machine learning e.g. TensorFlow™

ComputeAorta™

The heart of Codeplay's compute technology enabling OpenCL™, SPIR-V™, HSA™ and Vulkan™

Markets

High Performance Compute (HPC)
Automotive ADAS, IoT, Cloud Compute
Smartphones & Tablets
Medical & Industrial

Technologies: Artificial Intelligence
Vision Processing
Machine Learning
Big Data Compute

Agenda

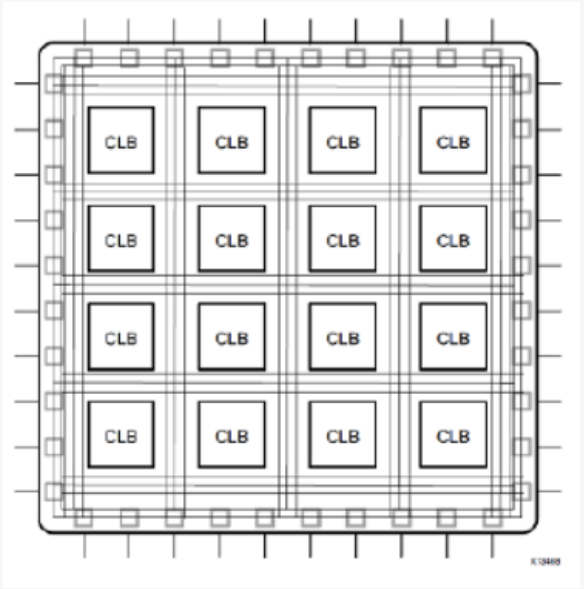
- Introduction:
  - Why bring SYCL to embedded FPGAs?
  - A background on FPGAs.
  - About HLS.
  - SYCL and FPGAs.
- ComputeCpp FPGA integration.
- FPGA-specific program transformations.
- Conclusions and future work.

Why bring SYCL to embedded FPGAs?

- To use FPGAs to accelerate complex modern C++ libraries via SYCL.
  - Especially Machine Learning and Artificial Intelligence, but also Computer Vision and HPC.
- To enable embedded/automotive/medical/etc platforms with FPGAs to run or accelerate software they were not able to support before.

FPGAs

- Field Programmable Gate Arrays.
- Matrix of Configurable Logic Blocks (CLB). (Look up tables, Flip Flops, DSP, BRAM) connected via programmable links.
- Programmed using Hardware Description Languages (HDL), such as VHDL or Verilog.



HDL vs Programming Languages

- "Software" programming languages are inherently sequential: one instruction is executed after the other.
- HDLs describe the structure of a digital circuit. Sequential behaviour is obtained across clock cycles, often by means of state machines.
- Learning an HDL is not like learning a new programming language, it requires a "paradigm shift".

HDL vs Programming Languages

RTL Verilog

```
module dut(rst, clk, q);
input rst;
input clk;
output q;
reg [7:0] c;
always @ (posedge clk)
begin
if (rst == 1b’1) begin
  c <= 8'b00000000;
end
else begin
  c <= c + 1;
end
assign q = c;
endmodule
```

C/C++ HLS

```
void dut() {
  static uint8 c;
  c+=1;
}
```

High Level Synthesis

- "Transformation of the behavioral description of hardware into an RTL model".
- Allows to obtain the VHDL/Verilog description of a circuit, starting from code written in a programming language.
- Tools exist for C, C++, OpenCL-C.

SYCL + FPGAs

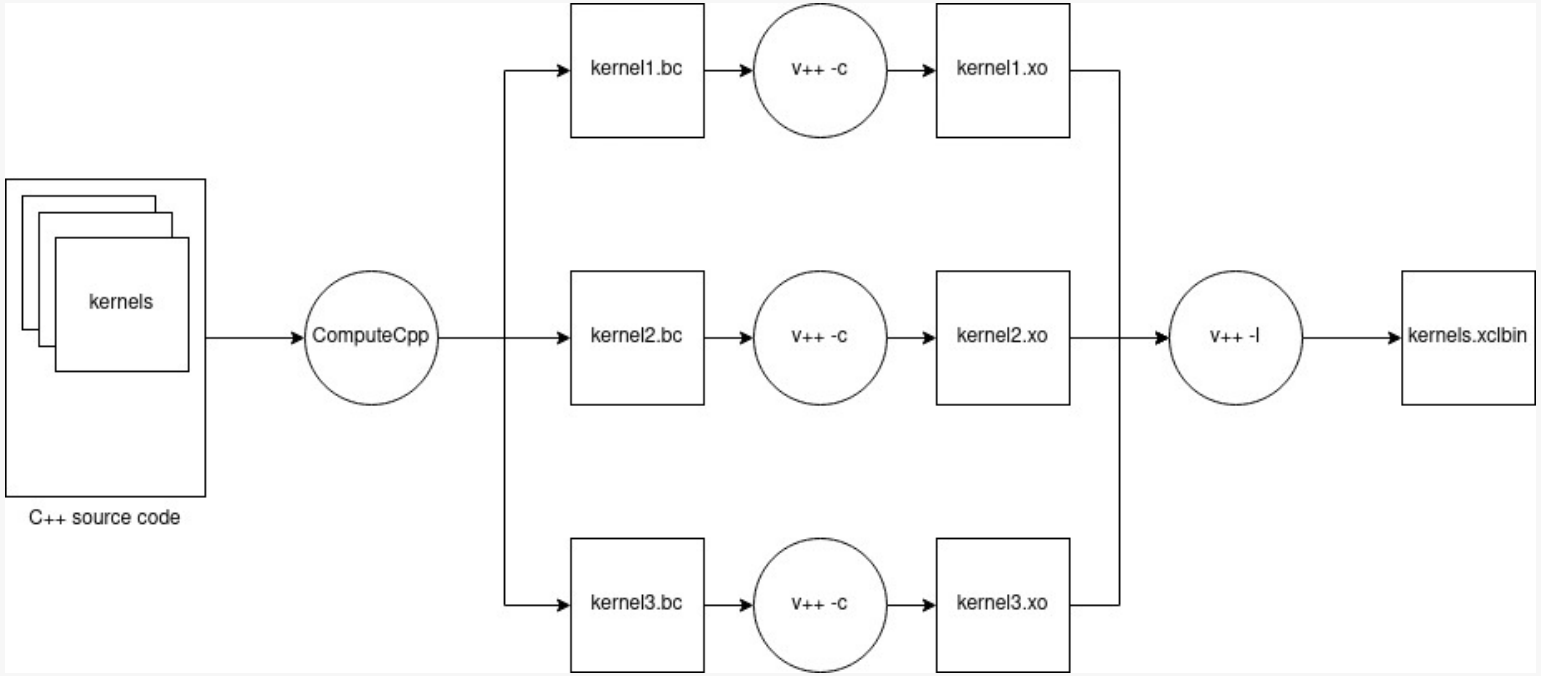
- triSYCL: Xilinx FPGAs.
- DPC++: Intel FPGAs.
- Both the implementations support OpenCL as a backend.
- Both provide FPGA-specific extensions.
- ComputeCpp: Xilinx (embedded) FPGAs.
- SYCL written for FPGAs can differ quite drastically from SYCL written for CPUs/GPUs.
  - For example no parallel_for, single task + specific extensions).

Experimental setup

- Zynq UltraScale+™ MPSoC ZCU106 Evaluation Kit.
- Arm® Cortex®- A53 processor.
- Targets low-power embedded applications.

ComputeCpp FPGA integration

- Mostly compiler driver work, use v++ (Xilinx HLS compiler), feed it SPIR produced by ComputeCpp.



Issues/limitations of targeted FPGA platform

- Kernel naming: the kernel name must be at most 28 characters long without containing underscores; this can cause issues with heavily templated kernels.
- Enqueue first kernel: data can't be written on the FPGA device before a kernel gets enqueued; need to enqueue a "fake kernel" first.
- All kernels linked together: all the kernels from a SYCL program must appear in the same bitstream file.

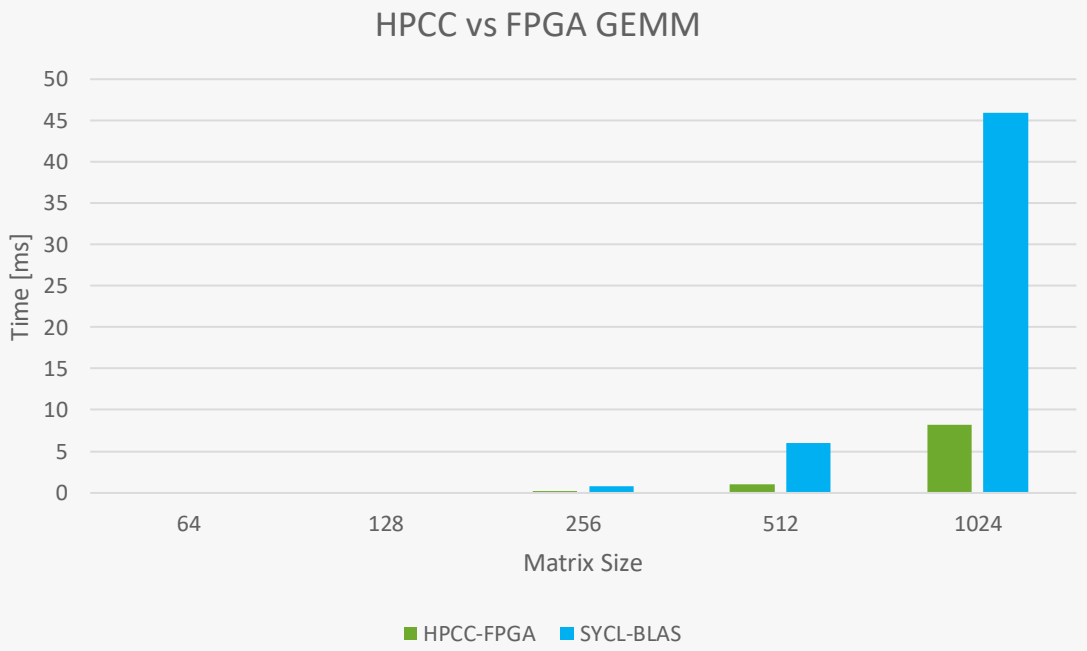
Addressed in ComputeCPP 

Performance numbers

- Compared SYCL-BLAS GEMM (CPU/GPU specific implementation) against GEMM from the HPCC_FPGA benchmark suite (OpenCL kernels that explicitly targets FPGAs).
- CPU/GPU specific implementation doesn't map very well to the FPGA, showing a slowdown that ranges from 1.5x to 5x, growing with the matrix size.
- SYCL-BLAS provides an autotuner, but the kernel linking issue limits its applicability on the FPGA.

Performance numbers

Size	Slowdown
64	1.6x
128	4.3x
256	5.7x
512	5.7x
1024	5.6x



FPGA-specific program transformations

Single task conversion

- Compiler-side transformation (+ SYCL runtime support) that allows to enqueue a SYCL parallel_for as a single_task.
- Exposes the computation of the entire parallel_for range (e.g. NDrange) to the HLS compiler.
  - Enables better pipelining and other optimizations.
- Allows us to explore further optimizations.

Example

Vanilla

```
define spir_kernel void @SimpleVadd(i32 *, i32 , i32*) {
  %4 = tail call spir_func i64 @_Z13get_global_idj(i32 0) #2
  %5 = getelementptr inbounds i32, i32 addrspace(1)* %1, i64 %4
  %6 = load i32, i32 addrspace(1)* %5, align 4, !tbaa !9
  %7 = getelementptr inbounds i32, i32 addrspace(1)* %2, i64 %4
  %8 = load i32, i32 addrspace(1)* %7, align 4, !tbaa !9
  %9 = add nsw i32 %8, %6
  %10 = getelementptr inbounds i32, i32 addrspace(1)* %0, i64 %4
  store i32 %9, i32 addrspace(1)* %10, align 4, !tbaa !9
  ret void
}
```

Single-task

```
define spir_kernel void @SimpleVadd(i32*, i32*, i32*, i64, i64, i64,
i64, i64, i64, i64) {
; … Some phi nodes removed for brevity
24:                                               ; preds = %24, %22
  %25 = phi i64 [ 0, %22 ], [ %33, %24 ]
  %26 = add i64 %19, %25
; … The actual sva computation, no more calls to get_global_id
  br i1 %34, label %24, label %35


35:                                               ; preds = %24
  %36 = add nuw i64 %23, 1
  %37 = icmp ult i64 %36, %4
  br i1 %37, label %22, label %38
; … increases the other loop counters
}
```

Barrier removal

- Having the single task conversion performed by the compiler allows to integrate it with other compiler passes.
- Calls to OpenCL barrier can be handled by performing opportune transformations.
- This is necessary to correctly convert to single task SYCL nd_range kernels.

Whole Function Vectorization

- Program transformation that, given a scalar kernel, creates a new one s.t. one execution of the new kernel corresponds to N executions of the original one (N is the vector width).
  - Executes N work items in one thread instead of N threads!
- Employs vector instructions to improve performances.
- Implementation: ComputeAorta's vecz.

Whole Function Vectorization - Example

Original kernel

```
define void @SimpleVadd(i32*, i32*, i32*) {
  %5 = call i64 @_Z13get_global_idj(i32 0)
  %6 = getelementptr inbounds i32, i32* %1, i64
%5
  %7 = load i32, i32* %6, align 4
  %8 = getelementptr inbounds i32, i32* %2, i64
%5
  %9 = load i32, i32* %8, align 4
  %10 = add nsw i32 %9, %7
  %11 = getelementptr inbounds i32, i32* %0,
i64 %5
  store i32 %10, i32* %11, align 4
  ret void
}
```

Vectorized kernel

```
define void @SimpleVadd_v16(i32*, i32*, i32*) {
  %5 = call i64 @_Z13get_global_idj(i32 0)
  %6 = getelementptr inbounds i32, i32* %1, i64 %5
  %7 = bitcast i32* %6 to <16 x i32>*
  %8 = load <16 x i32>, <16 x i32>* %7, align 4
  %9 = getelementptr inbounds i32, i32* %2, i64 %5
  %10 = bitcast i32* %9 to <16 x i32>*
  %11 = load <16 x i32>, <16 x i32>* %10, align 4
  %12 = add nsw <16 x i32> %11, %8
  %13 = getelementptr inbounds i32, i32* %0, i64 %5
  %14 = bitcast i32* %13 to <16 x i32>*
  store <16 x i32> %12, <16 x i32>* %14, align 4
  ret void
}
```

WFV + single task

Vectorized kernel

```
define void @SimpleVadd_v16(i32*, i32*, i32*) {
  %5 = call i64 @_Z13get_global_idj(i32 0)
  %6 = getelementptr inbounds i32, i32* %1, i64 %5
  %7 = bitcast i32* %6 to <16 x i32>*
  %8 = load <16 x i32>, <16 x i32>* %7, align 4
  %9 = getelementptr inbounds i32, i32* %2, i64 %5
  %10 = bitcast i32* %9 to <16 x i32>*
  %11 = load <16 x i32>, <16 x i32>* %10, align 4
  %12 = add nsw <16 x i32> %11, %8
  %13 = getelementptr inbounds i32, i32* %0, i64 %5
  %14 = bitcast i32* %13 to <16 x i32>*
  store <16 x i32> %12, <16 x i32>* %14, align 4
  ret void
}
```

Vectorized + single task

```
define spir_kernel void @SimpleVadd(i32*, i32*, i32*, i64, i64, i64,
i64, i64, i64, i64) {
; … Some phi nodes removed for brevity
24:                                               ; preds = %24, %22
  %25 = phi i64 [ 0, %22 ], [ %33, %24 ]
  %26 = add i64 %19, %25
; … The vectorized vector add, no more calls to get_global_id
  br i1 %34, label %24, label %35

35:                                               ; preds = %24
  %36 = add nuw i64 %23, 16
  %37 = icmp ult i64 %36, %4
  br i1 %37, label %22, label %38
; … increases the other loop counters + eventually peel
}
```

WFV: Some caveats

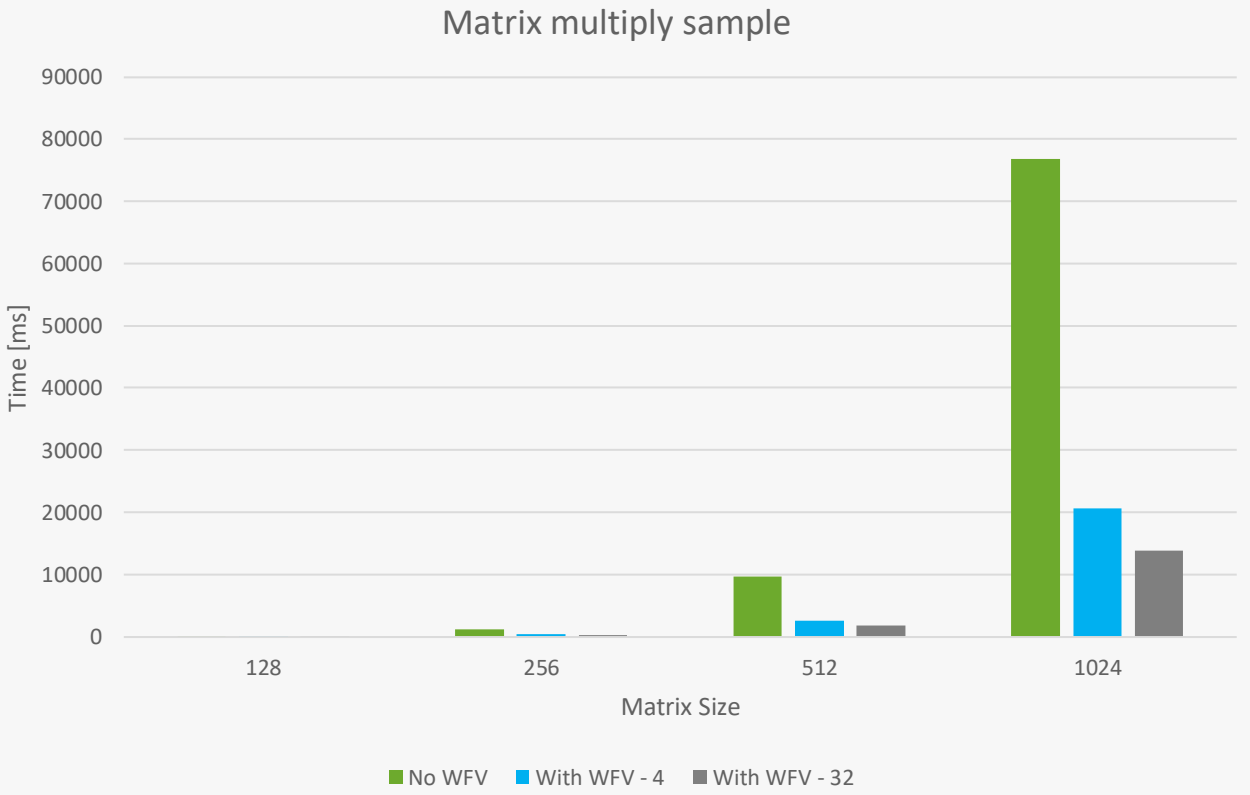
- Vecz/barrier pass emit some instructions that are not supported by SPIRV.
  - Variable length LLVM alloca are not supported -> Use a compile-time fixed size.
  - Vector of pointers are not supported -> Unpack them into scalars.
- LLVM vector intrinsics (masked scatter/gather etc) are flattened, leading to serial memory accesses.

Performance numbers

- No performance difference by just performing single task conversion, even when unrolling the loops.
- 5x speedup on matrix multiply sample from ComputeCpp-SDK with vectorization.
- Memory bandwidth limits benefit from vectorization.
- v++ -c crashes on the vectorized version of the SYCL-BLAS GEMM kernels. The IR is valid SPIR.

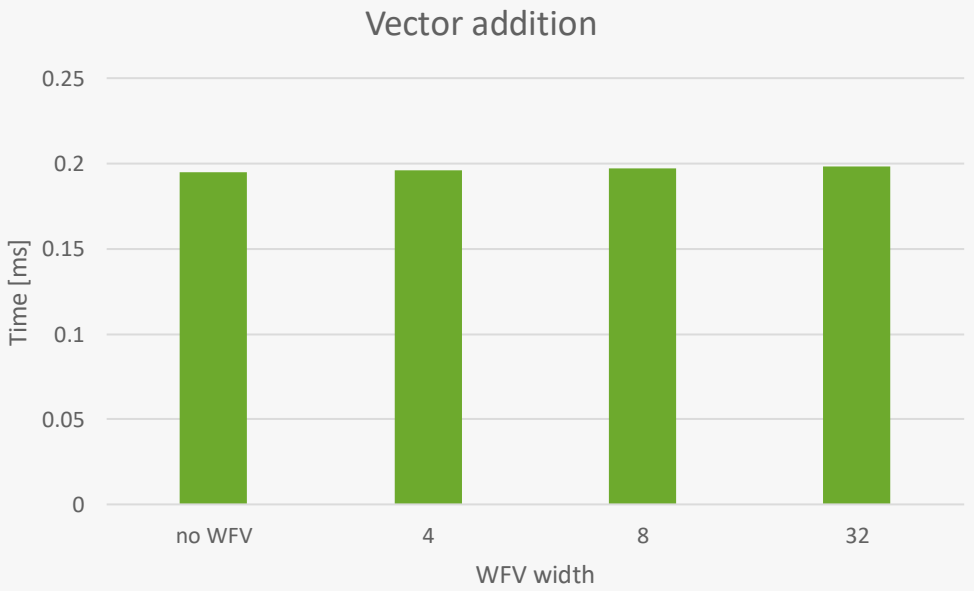
Performance numbers – matrix multiply

Size	V32 vs none	V4 vs none
128	5.27	3.55
256	5.43	3.67
512	5.55	3.73
1024	5.55	3.72



Performance numbers – vector addition

Performances are basically Unaffected by WFV in a memory-bound application.



Conclusions

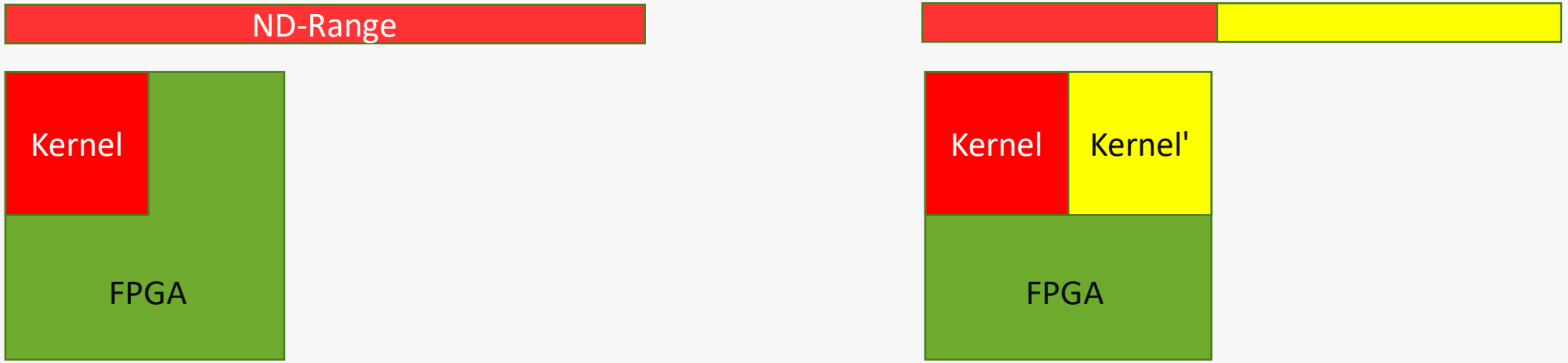
- Enabled FPGA support in ComputeCpp, specifically targeting embedded Xilinx boards. Successfully ran complex BLAS kernels.
- Single task conversion and vectorization show promising results on simple kernels, but the current HLS tool fails to synthetize more complex kernels.

Future work

- Enabling autotuning (requires either XRT to lift the restriction of requiring all kernels in the same bitstream file, or work arounds to enable kernels to be in separate bitstreams).
- Implement some FPGA-specific SYCL extensions in ComputeCpp (e.g. array partitioning).
- Automated kernel replication.
- MLIR/CIRCT.

Automatic kernel replication

- Multiple enqueues of the same kernel, each enqueued kernel operates on a subset of the nd-range.



MLIR/CIRCT

- CIRCT is an LLVM incubator project that aims to apply MLIR and LLVM in the hardware design domain.
- circt-hls is a sub-project within CIRCT that specifically targets HLS. Uses MLIR standard dialects, outputs SystemVerilog.
- Still in a very experimental state, and ComputeCpp doesn't include MLIR in its pipeline, but we are keeping an eye on the project.

We're Hiring!

codeplay.com/careers/

codeplay

Enabling AI to be Open, Safe & Accessible to All

Thank You!

@codeplaysoft

/codeplaysoft

codeplay.com