OpenMP to FPGA Offloading Prototype Using the Intel FPGA SDK for OpenCL

An IXPUG Success Story

Marius Knaust [1]

[1] Zuse Institute Berlin, Germany

ZUSE INSTITUTE BERLIN

orka

High Performance Computing

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

24th September, 2019

Introduction

IXPUG Annual Fall Conference 2018

- Presented plan for OpenMP to FPGA offloading
- Asked for feedback

Introduction

IXPUG Annual Fall Conference 2018

- Presented plan for OpenMP to FPGA offloading
- Asked for feedback

PyGA: a Python to FPGA compiler prototype

- Python to FPGA offloading
- Using Intel FPGA SDK for OpenCL

Motivation

High-level synthesis (HLS) opens FPGA world to Software Developers

- Vendor specific platforms
- Not portable

Motivation

High-level synthesis (HLS) opens FPGA world to Software Developers

- Vendor specific platforms
- Not portable

Both Intel and Xilinx introduced OpenCL support

- Standardized interface

Motivation

High-level synthesis (HLS) opens FPGA world to Software Developers

- Vendor specific platforms
- Not portable

Both Intel and Xilinx introduced OpenCL support

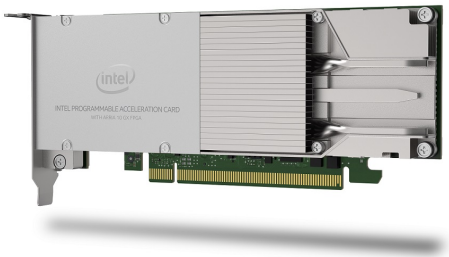
- Standardized interface

OpenMP

- Used in many legacy code bases
- Version 4 introduced `target` pragma

Agenda

- OpenCL SDK
- Implementation
  - Code Generation for the FPGA Target
  - Code Generation for the Host
  - Current Limitations
- Evaluation of the Prototype
- Future work



OpenCL SDK

OpenMP `target` Offloading Vision

In a perfect world ...

```
// ...

int A[n][m], B[m][p], C[n][p];

// ...

#pragma omp target map(to: A, B) map(from: C)
for (int i = 0; i < n; i++)
    for (int j = 0; j < p; j++)
    {
        C[i][j] = 0;

        for (int k = 0; k < m; k++)
            C[i][j] += A[i][k] * B[k][j];
    }

// ...
```

OpenMP `target` Offloading Vision

```
// ...

int A[n][m], B[m][p], C[n][p];

// ...

#pragma omp target map(to: A, B) map(from: C)
for (int i = 0; i < n; i++)
    for (int j = 0; j < p; j++)
    {
        C[i][j] = 0;

        for (int k = 0; k < m; k++)
            C[i][j] += A[i][k] * B[k][j];
    }

// ...
```

In a perfect world ...

1. OpenMP `target` outlining
   - By compiler frontend (*clang*)
   - Output IR (intermediate representation)

OpenMP `target` Offloading Vision

```
// ...

int A[n][m], B[m][p], C[n][p];

// ...

#pragma omp target map(to: A, B) map(from: C)
for (int i = 0; i < n; i++)
    for (int j = 0; j < p; j++)
    {
        C[i][j] = 0;

        for (int k = 0; k < m; k++)
            C[i][j] += A[i][k] * B[k][j];
    }

// ...
```

In a perfect world ...

1. OpenMP `target` outlining
   - By compiler frontend (*clang*)
   - Output IR (intermediate representation)
2. Vendor HLS
   - Input IR
   - Follow default HLS workflow

OpenMP `target` Offloading Vision

```
// ...

int A[n][m], B[m][p], C[n][p];

// ...

#pragma omp target map(to: A, B) map(from: C)
for (int i = 0; i < n; i++)
    for (int j = 0; j < p; j++)
    {
        C[i][j] = 0;

        for (int k = 0; k < m; k++)
            C[i][j] += A[i][k] * B[k][j];
    }

// ...
```

In a perfect world ...

1. OpenMP `target` outlining
   - By compiler frontend (*clang*)
   - Output IR (intermediate representation)
2. Vendor HLS
   - Input IR
   - Follow default HLS workflow

Problem

- IR not (officially) supported as an input to the HLS Tools

Advantages of Using the OpenCL SDK

	Intel		Xilinx	
	HLS	OpenCL	OpenCL	HLS
LLVM based	X	X	X	X

Advantages of Using the OpenCL SDK

	Intel		Xilinx	
	HLS	OpenCL	OpenCL	HLS
LLVM based	X	X	X	X
Standard API		X	X	

Standard API

- Across both big vendors (Intel & Xilinx)
- Implementation for Intel PAC
- Should be possible with Xilinx SDAccel as well

Advantages of Using the OpenCL SDK

	Intel		Xilinx	
	HLS	OpenCL	OpenCL	HLS
LLVM based	X	X	X	X
Standard API		X	X	
Transfer & Low Level Platform	X	X	X	

Standard API

- Across both big vendors (Intel & Xilinx)
- Implementation for Intel PAC
- Should be possible with Xilinx SDAccel as well

Low Level Platform

- Provided by the vendor tools
- Handy for a first prototype
- Might limit optimizations later on

Runtime Tracing

No official support for IR as input to HLS

- As a consequence not documented
- Runtime tracing nessasary

Runtime Tracing

No official support for IR as input to HLS

- As a consequence not documented
- Runtime tracing nessasary

Results

- Both Intel & Xilinx use LLVM for HLS
  - Intel: LLVM 3.0
  - Xilinx: LLVM 3.1

Runtime Tracing

No official support for IR as input to HLS

- As a consequence not documented
- Runtime tracing nessasary

Results

- Both Intel & Xilinx use LLVM for HLS
  - Intel: LLVM 3.0
  - Xilinx: LLVM 3.1

Problems

- *clang* OpenMP `target` offloading
  - Starting LLVM 3.8

Runtime Tracing

No official support for IR as input to HLS

- As a consequence not documented
- Runtime tracing nessasary

Results

- Both Intel & Xilinx use LLVM for HLS
  - Intel: LLVM 3.0
  - Xilinx: LLVM 3.1

Problems

- *clang* OpenMP `target` offloading
  - Starting LLVM 3.8
- Requires mixed version approach
  - Adjust outlined function from up to date LLVM IR to old IR
  - Could become obsolete with newer versions (or standardized IR like SPIR-V)

Runtime Tracing Results (Intel)

```
aocl-clang -cc1 -O3 -emit-llvm-bc -Wuninitialized -triple fpga64 -mllvm -board
        -mllvm $AOCL_BOARD_PACKAGE_ROOT/hardware/pac_a10/board_spec.xml
        -DACL_BOARD_pac_a10=1 -DAOCL_BOARD_pac_a10=1 "$name.cl"
        -o "$name.pre.bc" -g

aocl-link "$name.pre.bc" $ALTERAOCLSDKROOT/share/lib/acl/acl_early.bc -o "$name.1.bc"

aocl-opt --acle $acle_key -board $board_spec -dbg-info-enabled --grif
    --soft-elementary-math=false --fas=false --wiicm-disable=true "$name.1.bc"
    -o "$name.kwgid.bc"
aocl-opt -insert-ip-library-calls -dbg-info-enabled --grif --soft-elementary-math=false
    --fas=false --wiicm-disable=true "$name.kwgid.bc" -o "$name.lowered.bc"

aocl-link "$name.lowered.bc" $ALTERAOCLSDKROOT/share/lib/acl/acl_late.bc
    -o "$name.linked.bc"

aocl-opt -board $board_spec -always-inline -add-inline-tag -instcombine -adjust-sizes -dce
    -stripnk -rename-basic-blocks -dbg-info-enabled --grif --soft-elementary-math=false
    --fas=false --wiicm-disable=true "$name.linked.bc" -o "$name.bc"

aocl-llc -march=griffin -board $board_spec -dbg-info-enabled "$name.bc" -o "$name.v"

system_integrator --bsp-flow green_top $board_spec $name.bc.xml system.tcl kernel_system.tcl
```

Runtime Tracing Results (Intel)

```
aocl-clang -cc1 -O3 -emit-llvm -Wuninitialized -triple fpga64 -mllvm -board
        -mllvm $AOCL_BOARD_PACKAGE_ROOT/hardware/pac_a10/board_spec.xml
        -DACL_BOARD_pac_a10=1 -DAOCL_BOARD_pac_a10=1 "$name.cl"
        -o "$name.pre.bc" -g

# TODO: Modify IR here

aocl-link "$name.pre.bc" $ALTERAOCLSDKROOT/share/lib/acl/acl_early.bc -o "$name.1.bc"

aocl-opt --acle $acle_key -board $board_spec -dbg-info-enabled --grif
    --soft-elementary-math=false --fas=false --wiicm-disable=true "$name.1.bc"
    -o "$name.kwgid.bc"
aocl-opt -insert-ip-library-calls -dbg-info-enabled --grif --soft-elementary-math=false
    --fas=false --wiicm-disable=true "$name.kwgid.bc" -o "$name.lowered.bc"

aocl-link "$name.lowered.bc" $ALTERAOCLSDKROOT/share/lib/acl/acl_late.bc
    -o "$name.linked.bc"

aocl-opt -board $board_spec -always-inline -add-inline-tag -instcombine -adjust-sizes -dce
    -stripnk -rename-basic-blocks -dbg-info-enabled --grif --soft-elementary-math=false
    --fas=false --wiicm-disable=true "$name.linked.bc" -o "$name.bc"

aocl-llc -march=griffin -board $board_spec -dbg-info-enabled "$name.bc" -o "$name.v"

system_integrator --bsp-flow green_top $board_spec $name.bc.xml system.tcl kernel_system.tcl
```

Implementation

Rough Implementation Idea

1. Use *clang* to outline OpenMP `target` pragma
   - Misuse *x86-64* OpenMP target
   - Results in *pure* IR of outlined function

```
; ...

define dso_local i32 @main() #0
{
    ; ...
}

define internal void @__omp_offloading_main_t1(i64) #0
{
    ; ...

    ret void
}

; ...
```

Rough Implementation Idea

1. Use *clang* to outline OpenMP `target` pragma
   - Misuse *x86-64* OpenMP target
   - Results in *pure* IR of outlined function
2. Target related
   - Inject outlined function body as IR into OpenCL SDK

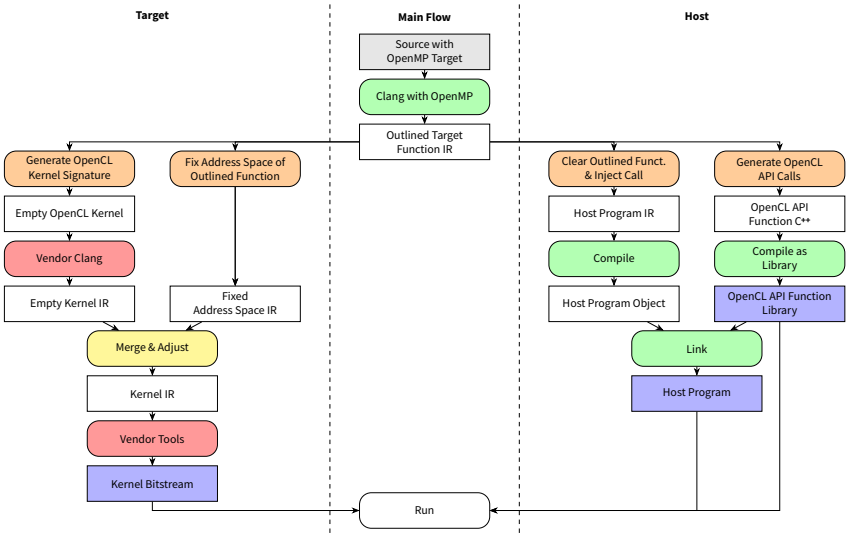
```
; ...

define dso_local i32 @main() #0
{
    ; ...
}

define internal void @__omp_offloading_main_t1(i64) #0
{
    ; ...

    ret void
}

; ...
```

Rough Implementation Idea

1. Use *clang* to outline OpenMP `target` pragma
   - Misuse *x86-64* OpenMP target
   - Results in *pure* IR of outlined function
2. Target related
   - Inject outlined function body as IR into OpenCL SDK
3. Host related
   - Replace outlined function body by OpenCL API calls

```
; ...

define dso_local i32 @main() #0
{
    ; ...
}

define internal void @__omp_offloading_main_t1(i64) #0
{
    ; ...

    ret void
}

; ...
```

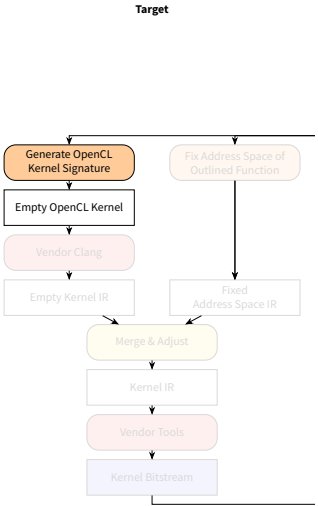
Flowchart of tool invocation



Code Generation for the FPGA Target

Target side

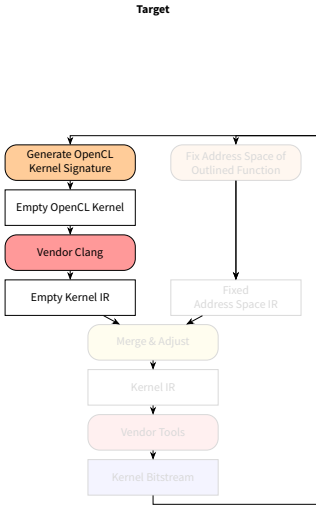
1. Generate *empty* OpenCL kernel source code from signature[1]
   - By LLVM pass



[1] PyGA: a Python to FPGA compiler prototype; Y. Uguen and E. Petit; AI SEPS 2018

Target side

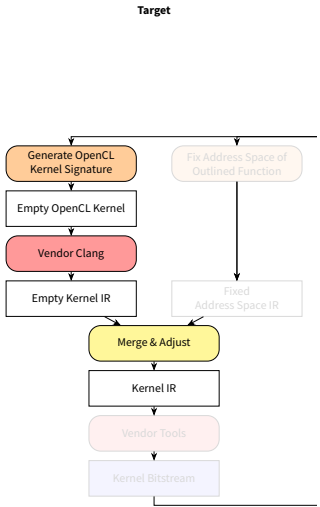
1. Generate *empty* OpenCL kernel source code from signature[1]
   - By LLVM pass
2. Translate OpenCL into IR
   - Using vendor *clang*



[1] PyGA: a Python to FPGA compiler prototype; Y. Uguen and E. Petit; AI SEPS 2018

Target side

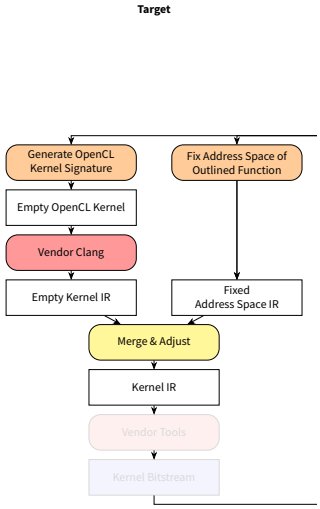
1. Generate *empty* OpenCL kernel source code from signature[1]
   - By LLVM pass
2. Translate OpenCL into IR
   - Using vendor *clang*
3. Merge outlined function body into kernel IR



[1] PyGA: a Python to FPGA compiler prototype; Y. Uguen and E. Petit; AI SEPS 2018

Target side

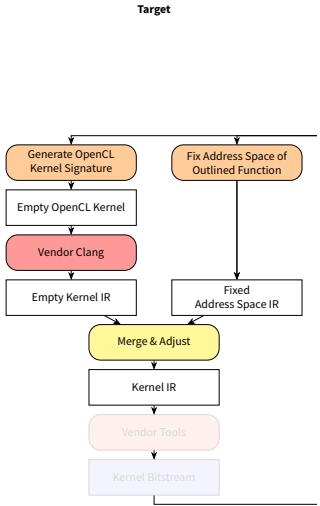
1. Generate *empty* OpenCL kernel source code from signature[1]
   - By LLVM pass
2. Translate OpenCL into IR
   - Using vendor *clang*
3. Merge outlined function body into kernel IR
   - Address space adjustments (LLVM pass)
     - Arguments in global memory
     - Propagate recursively to all IR instructions using it



[1] PyGA: a Python to FPGA compiler prototype; Y. Uguen and E. Petit; AI SEPS 2018

Target side

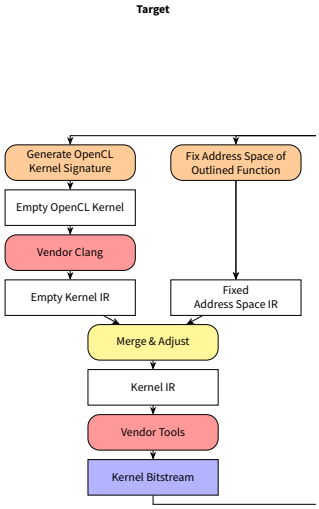
1. Generate *empty* OpenCL kernel source code from signature[1]
   - By LLVM pass
2. Translate OpenCL into IR
   - Using vendor *clang*
3. Merge outlined function body into kernel IR
   - Address space adjustments (LLVM pass)
     - Arguments in global memory
     - Propagate recursively to all IR instructions using it
   - Version adjustments
     - Up to date LLVM IR to LLVM 3.0 IR
     - loop unrolling annotations
     - special vendor functions (e.g. `sqrt`)



[1] PyGA: a Python to FPGA compiler prototype; Y. Uguen and E. Petit; AI SEPS 2018

Target side

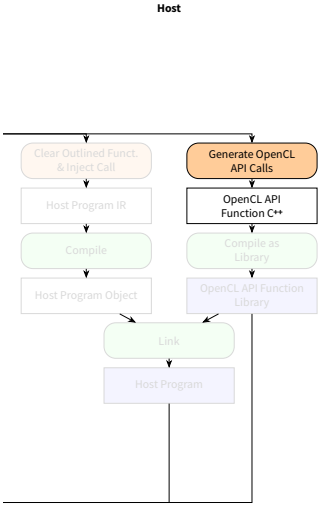
1. Generate *empty* OpenCL kernel source code from signature[1]
   - By LLVM pass
2. Translate OpenCL into IR
   - Using vendor *clang*
3. Merge outlined function body into kernel IR
   - Address space adjustments (LLVM pass)
     - Arguments in global memory
     - Propagate recursively to all IR instructions using it
   - Version adjustments
     - Up to date LLVM IR to LLVM 3.0 IR
     - loop unrolling annotations
     - special vendor functions (e.g. `sqrt`)
4. Use Vendor tools for HLS



[1] PyGA: a Python to FPGA compiler prototype; Y. Uguen and E. Petit; AI SEPS 2018

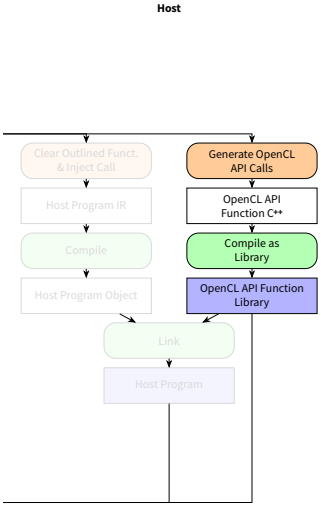
Code Generation for the Host

Host side



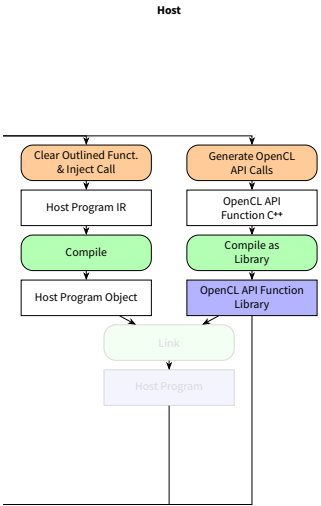
1. Generate C++ source code with OpenCL API calls
   - Based on the arguments
   - LLVM pass

Host side



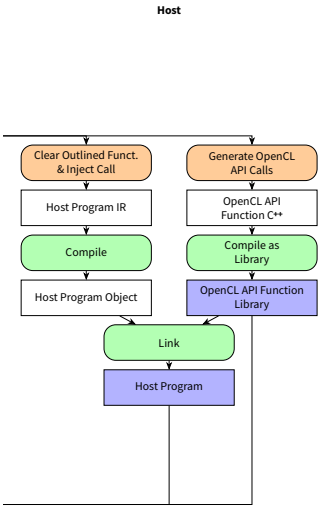
1. Generate C++ source code with OpenCL API calls
   - Based on the arguments
   - LLVM pass
2. Compile into library

Host side



1. Generate C++ source code with OpenCL API calls
   - Based on the arguments
   - LLVM pass
2. Compile into library
3. Clear outlined function body and inject call to library
   - LLVM pass

Host side



1. Generate C++ source code with OpenCL API calls
   - Based on the arguments
   - LLVM pass
2. Compile into library
3. Clear outlined function body and inject call to library
   - LLVM pass
4. Link to host program

Current Limitations

Current Limitations

OpenCL API calls generated from signature

Current Limitations

OpenCL API calls generated from signature

→ `map` clause not considered

Current Limitations

OpenCL API calls generated from signature

- → `map` clause not considered
- → no array selections

Current Limitations

OpenCL API calls generated from signature

- → `map` clause not considered
- → no array selections
- → only static buffer (fixed-size arrays)

Current Limitations

OpenCL API calls generated from signature

→ map clause not considered

→ no array selections

→ only static buffer (fixed-size arrays)

Source code generation is not elegant

- But quick & easy (prototype)
- Better directly as LLVM IR

Current Limitations

OpenCL API calls generated from signature

→ map clause not considered

→ no array selections

→ only static buffer (fixed-size arrays)

Source code generation is not elegant

- But quick & easy (prototype)
- Better directly as LLVM IR
- Or even better …

Tighter compiler framework integration

- *libomptarget*
- Dedicated transfer infrastructure
- But requires *clang* modification

Evaluation of the Prototype

Workload

Sobel filter

- 4k integer RGB image

Workload

Sobel filter

- 4k integer RGB image

Two implementations

- Naïve version
- FPGA optimized version

Workload

Sobel filter

- 4k integer RGB image

Two implementations

- Naïve version
- FPGA optimized version
  - Analog to Intel OpenCL Example
    - `unroll` pragma
    - local buffer (to avoid global memory access)
    - structured to employ shift register

Workload

Sobel filter

- 4k integer RGB image

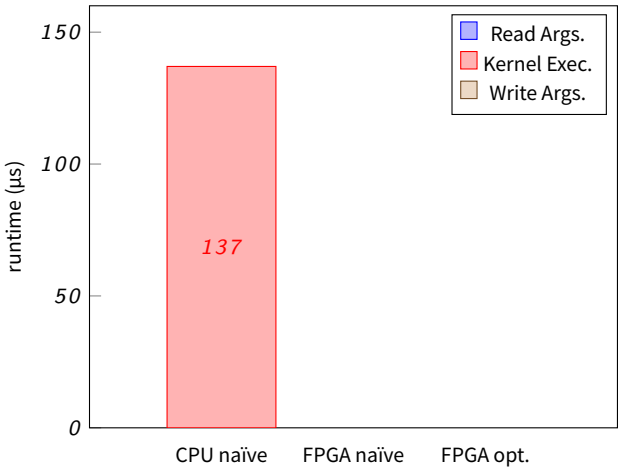
Two implementations

- Naïve version
- FPGA optimized version
  - Analog to Intel OpenCL Example
    - `unroll` pragma
    - local buffer (to avoid global memory access)
    - structured to employ shift register

Hardware

- Intel PAC with a Arria 10 GX
- Intel Xeon W-2125 Skylake (Host & Reference)
- KVM virtual machine with PCI passthrough
  - Might have a small performance penalty

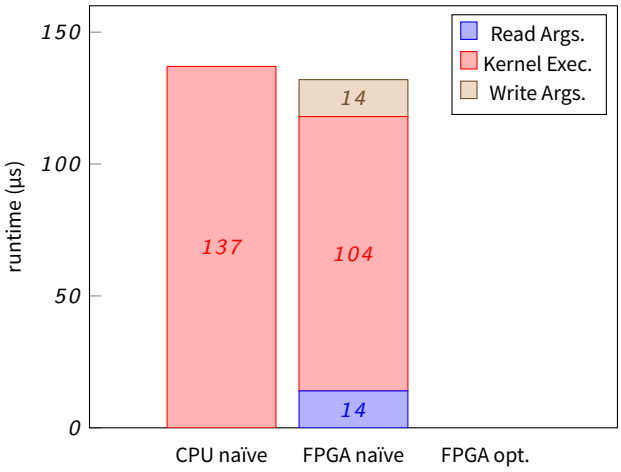
Baseline



Naïve CPU version

- Serves as a baseline
- No manual optimizations
  - Single threaded
  - No SIMD
  - ...

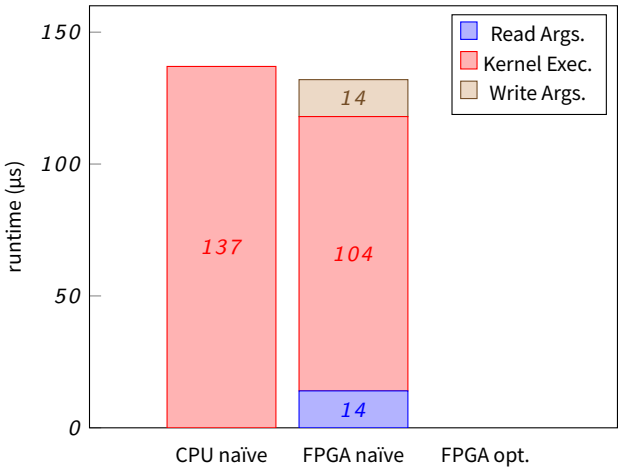
Naïve FPGA Version



Naïve FPGA version has equal runtime

- Actual Kernel is *slightly* faster

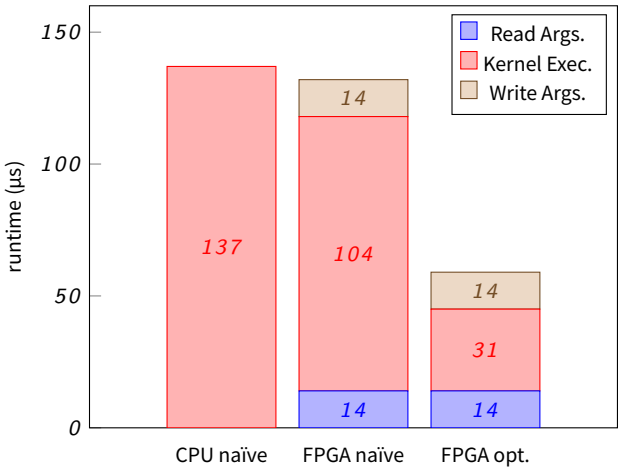
Naïve FPGA Version



Naïve FPGA version has equal runtime

- Actual Kernel is *slightly* faster
- Transfer Overhead
  - `map` clause not considered
    - `to` & `from` handled equal to `tofrom`
    - When considered, decrease by half

Optimized FPGA Version



Kernel of FPGA optimized version way better

- Shows simple annotation not enough (as expected)
- Requires code adaption

Future work

Outlook

- *libomptarget* integration
  - More elegant and powerful host code
  - `map` clause support
    - Map-type
    - Array sections

Outlook

- *libomptarget* integration
  - More elegant and powerful host code
  - `map` clause support
    - Map-type
    - Array sections
- Integration into Intel LLVM (Sycl) toolchain
  - Spir-V supported as an input
    - Khronos SPIRV-LLVM Translator
    - Intels new OpenCL FPGA Tools (`aoc -sycl kernel.spv`)