POLITECNICO
MILANO 1863

Platform-Aware FPGA System Architecture Generation based on MLIR

Stephanie Soldavini, Christian Pilato

Dipartimento di Elettronica, Informazione e Bioingegneria

stephanie.soldavini@polimi.it

Motivation

- Optimizing **data movements** is becoming one of the biggest challenges in heterogeneous computing to cope with modern **big data applications**
- High-level synthesis (HLS) tools are increasingly efficient at optimizing computation, but data transfers have not been adequately improved
- Novel architectures such as **high-bandwidth memory (HBM)** have been developed to be able to transfer more data in parallel
- However, designers must follow strict coding-style rules to exploit this extra bandwidth

Proposal

- Olympus multi-level intermediate representation (MLIR) dialect for representing platform aware system level FPGA architectures
- Olympus-opt: a series of analysis and transformation passes on this dialect
- MLIR enables extensibility and reusability both between many sources of input and many platform-specific back-ends.

Multi-Level Intermediate Representation (MLIR)

- A novel compiler infrastructure centered on reuse and extensibility
- Becoming popular as a framework for domain-specific language (DSL) compilers for heterogeneous systems
- MLIR is a collection of **dialects**, each representing different layers of abstraction through various operators, types, and attributes.
- Custom dialects can easily be added for domain-specific problems while reusing existing infrastructure
- Dialects can be integrated into larger language stacks via **lowering**, transforming a more abstract dialect into a more concrete one.

FPGA Memory Architecture

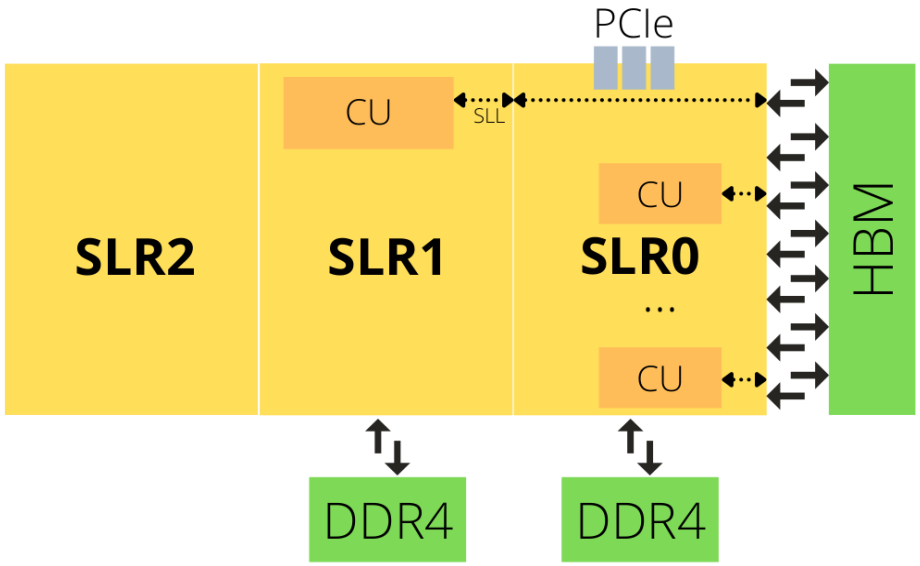
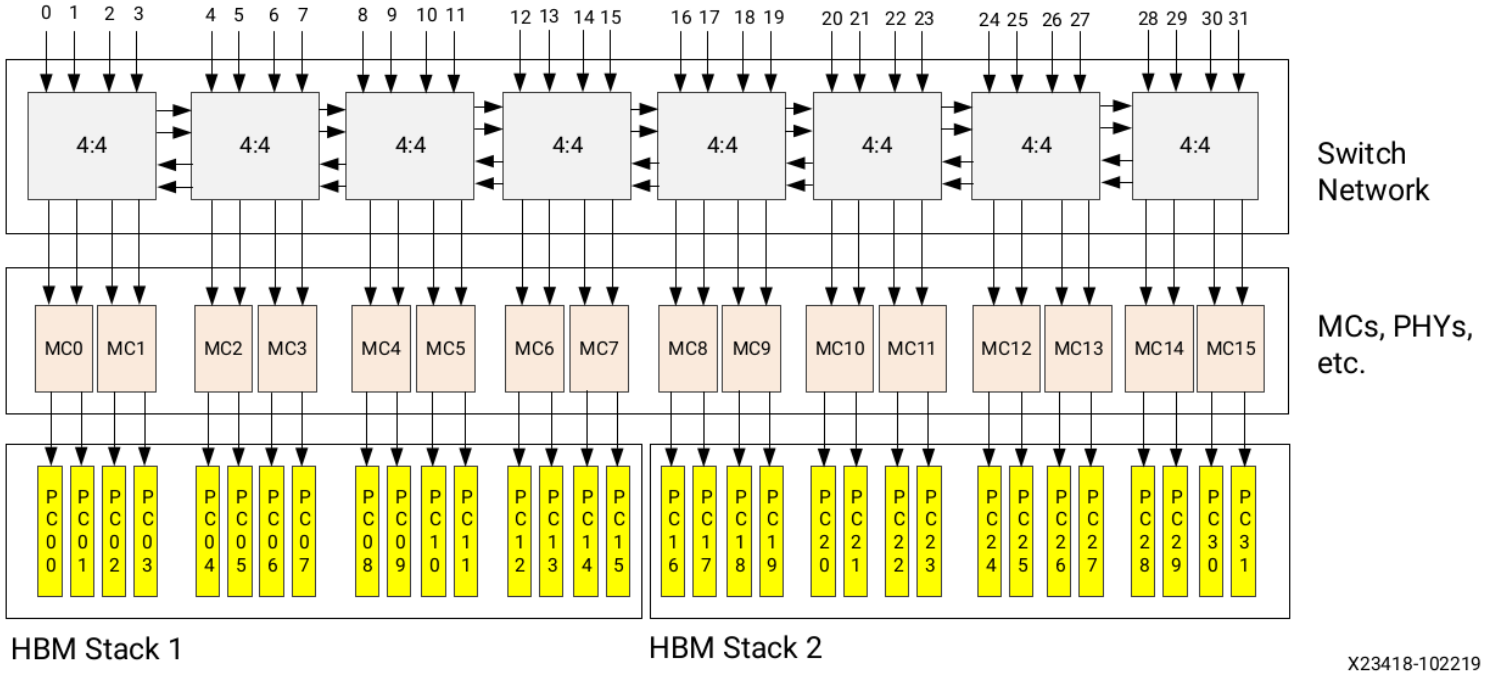


Table 1. Alveo U280 SLR resources.

Resources	SLR0	SLR1	SLR2
HBM	32×256MB	-	-
DDR4	16GB	16GB	-
PLRAM	2×4MB	2×4MB	2×4MB
CLB LUT	369K	333K	367K
CLB Register	746K	675K	729K
Block RAM tile	507	468	512
UltraRAM	320	320	320
DSP	2,733	2,877	2,880



Max theoretical bandwidth: 460.8 GB/s

Olympus MLIR Dialect: Kernel Operator

- **`callee`**: name of the kernel
- **`latency, ii`**: timing estimates from HLS/synthesis
- **`ff, lut, bram, uram, dsp`**: resource estimates from HLS/synthesis
- **`operand_segment_sizes`**: lists how many parameters are inputs (first number) and outputs (second number)

```
"olympus.kernel"(%2, %3, %4) {
    callee = "matmul",
    latency = 795, ii = 268,
    ff = 3106, lut = 6174, bram = 61,
    uram = 0, dsp = 48,
    operand_segment_sizes = array<i32: 2, 1>,
} : (
    !olympus.channel<i32>,
    !olympus.channel<i32>,
    !olympus.channel<i32>
) -> ()
```

Fig. 2: Sample kernel operator

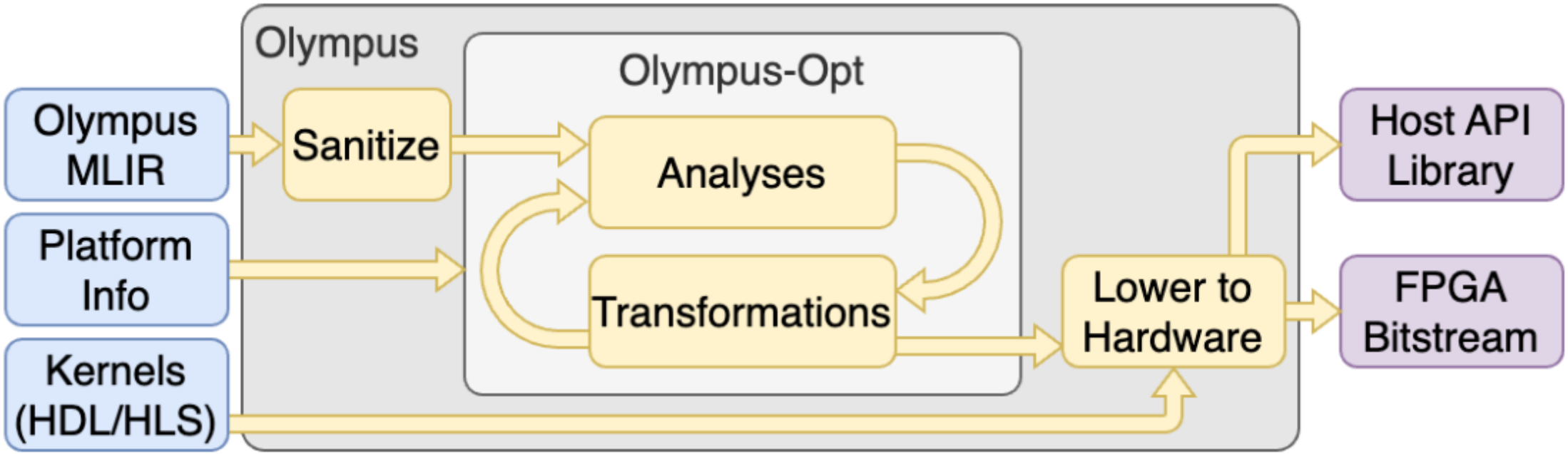
Olympus MLIR Dialect: Channel Operator

- **`encapsulatedType`**: data element width as a signless integer of arbitrary bitwidth
- **`paramType`**:
  - `stream`:
    - produced and consumed in the same order
    - small, statically sized elements
  - `small`:
    - random access
    - a single kernel iteration at most 100s of kB
    - organized of simple structures without nesting or indirection
  - `complex`:
    - can be anything: huge, random access, have indirection, and/or be constructed of nested structures
- **`depth`**: the size of the data structure

```
%2 = "olympus.make_channel"() {
    encapsulatedType = i32,
    paramType = "stream",
    depth = 20
} : () -> (
    !olympus.channel<i32>
)
```

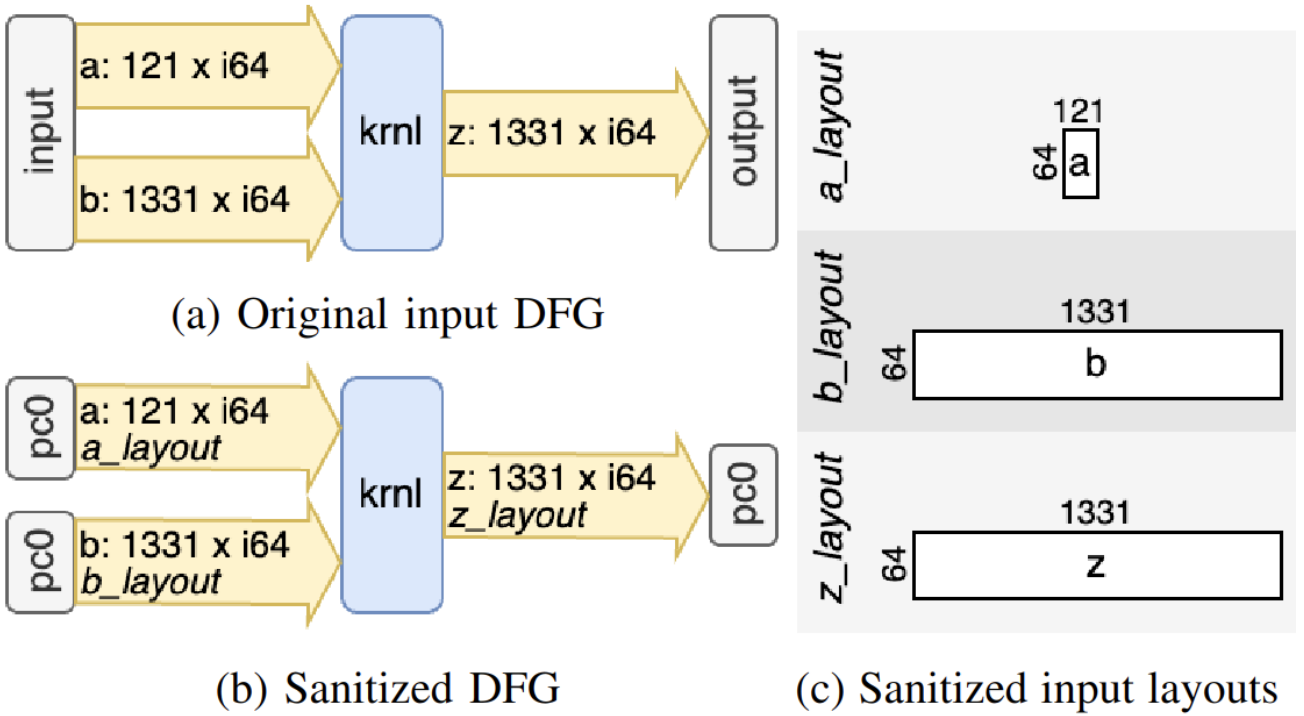
Fig. 1: Sample channel operator

Olympus-opt


Olympus
Olympus-Opt
Olympus MLIR
Platform Info
Kernels (HDL/HLS)
Sanitize
Analyses
Transformations
Lower to Hardware
Host API Library
FPGA Bitstream

Sanitization

- Move MLIR to a form that could be immediately lowered to hardware
  - Map IO to pseudochannels
  - Add basic layouts
- Enables cleaner user input



(a) Original input DFG

(b) Sanitized DFG

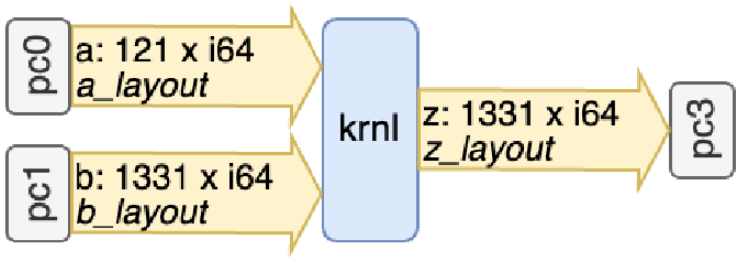
(c) Sanitized input layouts

Analyses

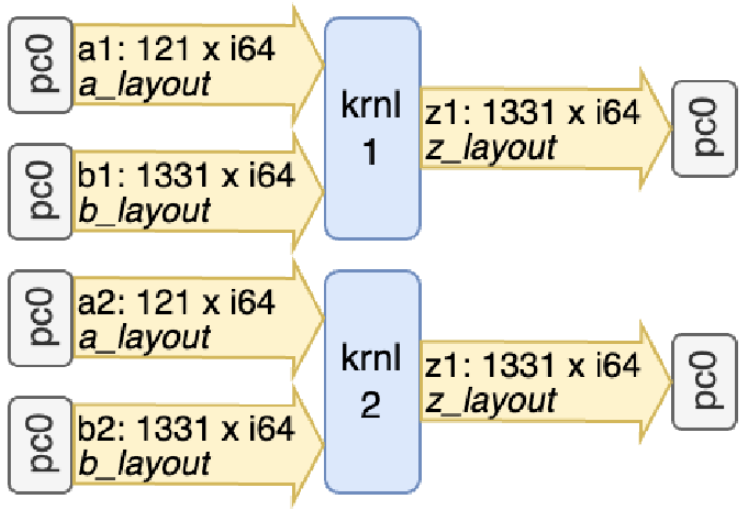
- Bandwidth utilization:
  - From target PC information and the attributes of each data channel
- Resource utilization:
  - From total resource availability and the kernel resource utilization
- Using the results of these analyses, transformation passes can be chosen to alter the DFG to increase expected performance

Transformations

- Channel Reassignment
  - Distribute external IO across PCs
  - Decreases pressure on single PCs



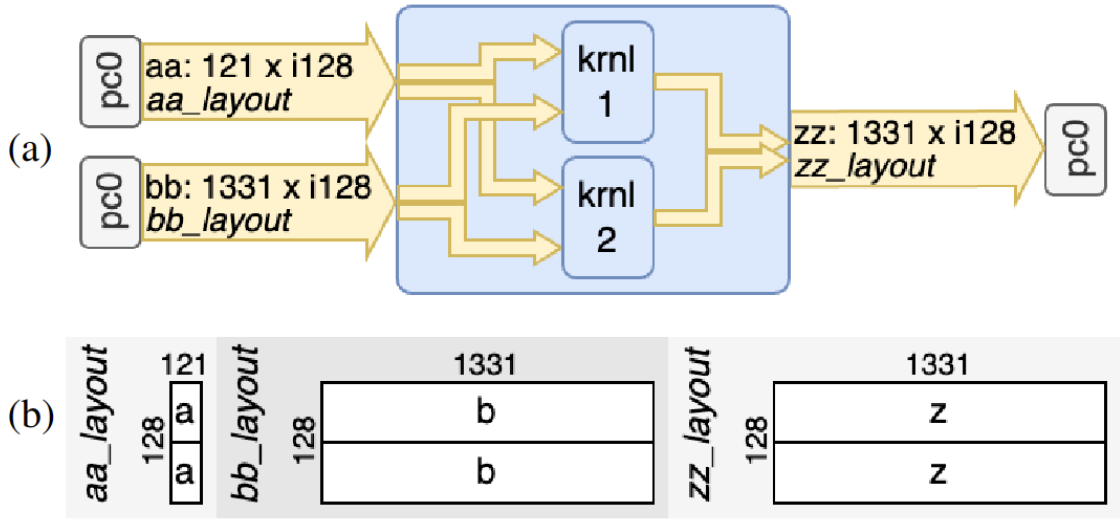
- Replication [1]
  - Instantiate more DFG instances
  - Increases parallelism when using available resources



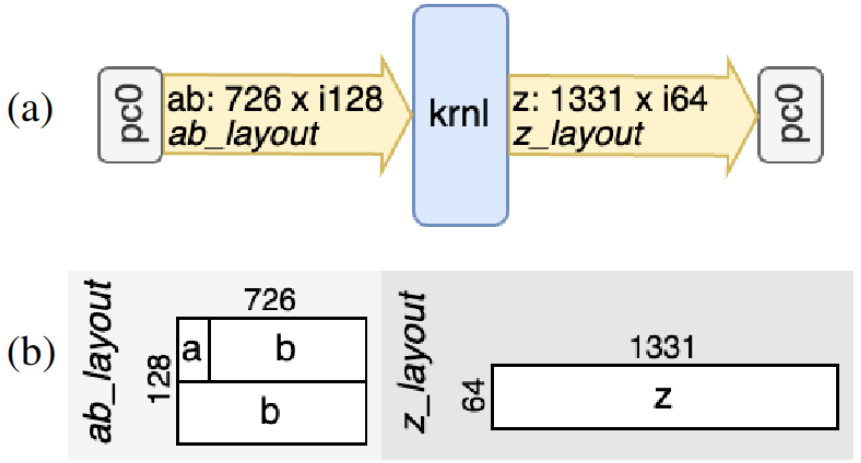
[1] S. Soldavini et. al. “Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics”. In: ACM TRETS (Sept. 2022). doi: 10.1145/3563553

Transformations

- Bus Widening [1]
  - Replicate kernels so their inputs fill the channel width
  - i.e. 64b data fits 4 times on 256b-channel
  - Increases bandwidth utilization



- Bus optimization [2]
  - Iris algorithm: Efficiently “pack” multiple inputs onto a channel
  - Increases bandwidth utilization, frees channels for more replication



[1] S. Soldavini et al. “Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics”. In: ACM TRETS (Sept. 2022). doi: 10.1145/3563553

[2] S. Soldavini, D. Sciuto, and C. Pilato. “Iris: Automatic Generation of Efficient Data Layouts for High Bandwidth Utilization”. In: Proceedings of ASPDAC. Tokyo, Japan: ACM, 2023. doi: 10.48550/ARXIV.2211.04361

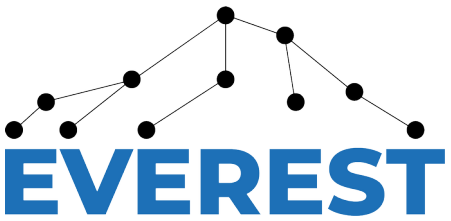
Conclusion

- Olympus MLIR-based infrastructure for platform-aware FPGA system architecture generation
  - Olympus MLIR dialect:
    - Represents the DFG of accelerator kernels
  - Olympus-opt:
    - Analyses and transformations on this DFG to iteratively optimize
    - Take advantage of the characteristics of the FPGA platform, particularly off chip memory bandwidth
  - Extensible for both front end DSL compilers and back end lowering to hardware platforms

Questions?

Find me by my poster :)

stephanie.soldavini@polimi.it



This work was partially funded by the EU Horizon 2020 Programme under grant agreement No 957269 (EVEREST).
http://www.everest-h2020.eu