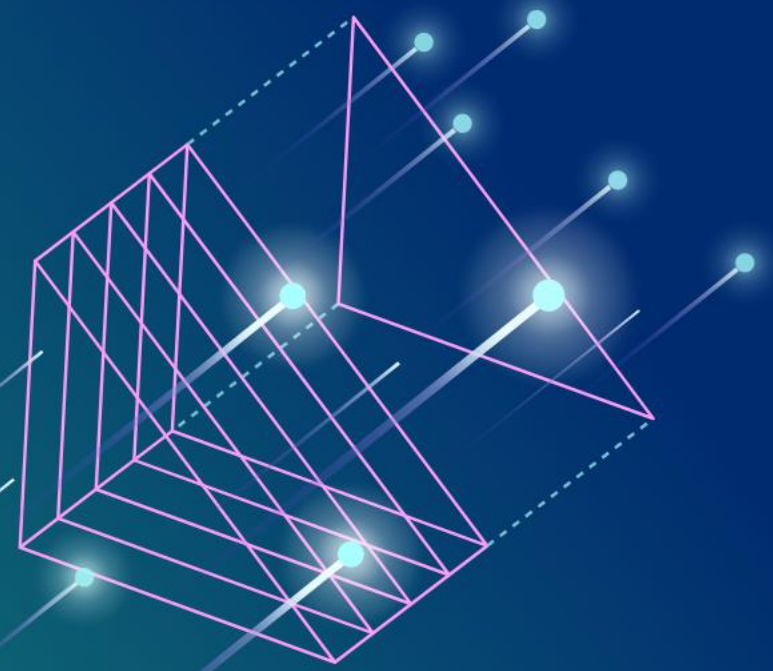
EUROPEAN BIG DATA VALUE FORUM

Technology, Platforms, and impact

EVEREST SDK: A System Development Kit for Extreme-Scale Data Analytics on High-Performance and Heterogeneous Platforms

Christian Pilato, Politecnico di Milano

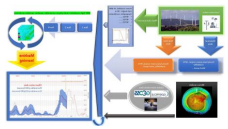


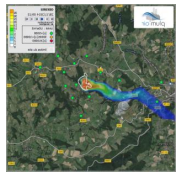
ORGANISED BY
BDV BIG DATA VALUE ASSOCIATION

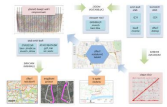
IN COLLABORATION WITH
ITI INVESTIGATE TO INNOVATE
European Commission
INCLIVA | VLC Biomedical Research Institute
POLITÉCNICA
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNDER THE AUSPICES OF
EU23 SPANISH PRESIDENCY COUNCIL OF THE EUROPEAN UNION

The EVEREST Project







Big data applicati
heterogeneous d

- App designers are not FPGA experts
- Hardware accelerators require many
- an have different

H2020 Project – 10 partners, 6 countries

Project Coordinator: Christoph Hagleitner, IBM Research Europe

Scientific Coordinator: Christian Pilato, Politecnico di Milano

Budget: ~5M€

Start date: October 1, 2020 (36+6 months)

Compilation

Unified hardware gene
(high-level synth

MLIR

Generation of **variants**

Increase quality of accelerators

Improve applications' results

Runtime

adaptation to variants

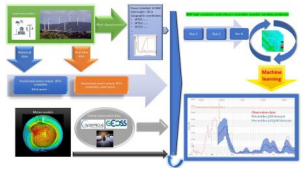
Virtualization of resources

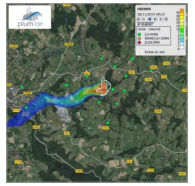
Multi-node support

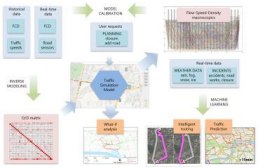
EVEREST Approach

Big data applications with heterogeneous data sources

Three use cases







What are the relevant requirements for data, languages and applications?

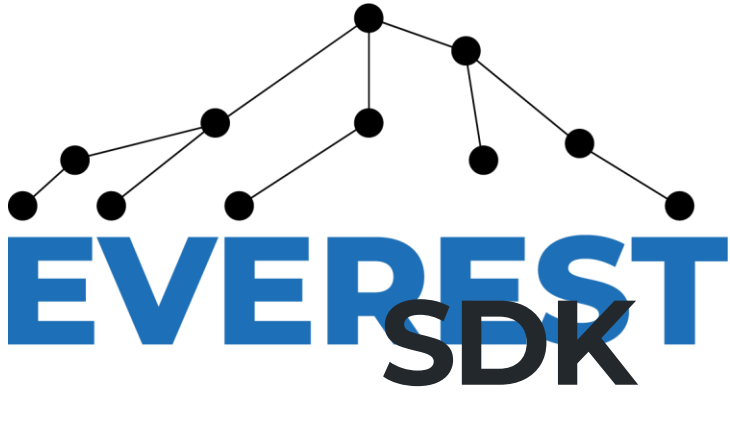
How to design data-driven policies for computation, communication, and storage?

How to create FPGA accelerators and associated binaries?

How to manage the system at runtime?

How to evaluate the results?

How to disseminate and exploit the results?



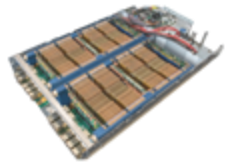
Open-source framework to support the optimization of selected workflow tasks

FPGA-based architectures to accelerate selected kernels

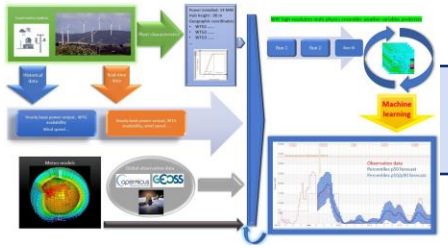
CPU-based infrastructure

+

Two FPGA-based clusters

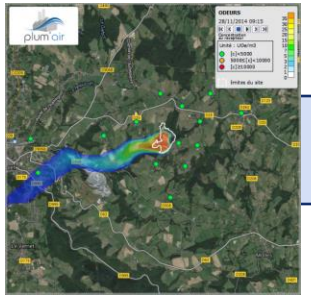


EVEREST Use Cases



Renewable energy production prediction

★ Improve **quality of the predictions**

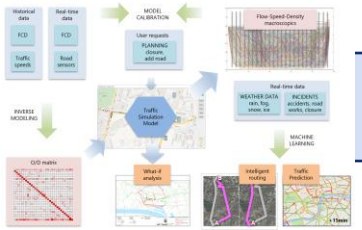


Air-quality monitoring of industrial sites

★ Improve the **response time of predictions**

Weather prediction modelling (WRF)

★ Accelerate kernels to execute more tests



Traffic modeling for intelligent transportation

★ Improve the **overall performance of traffic simulation**

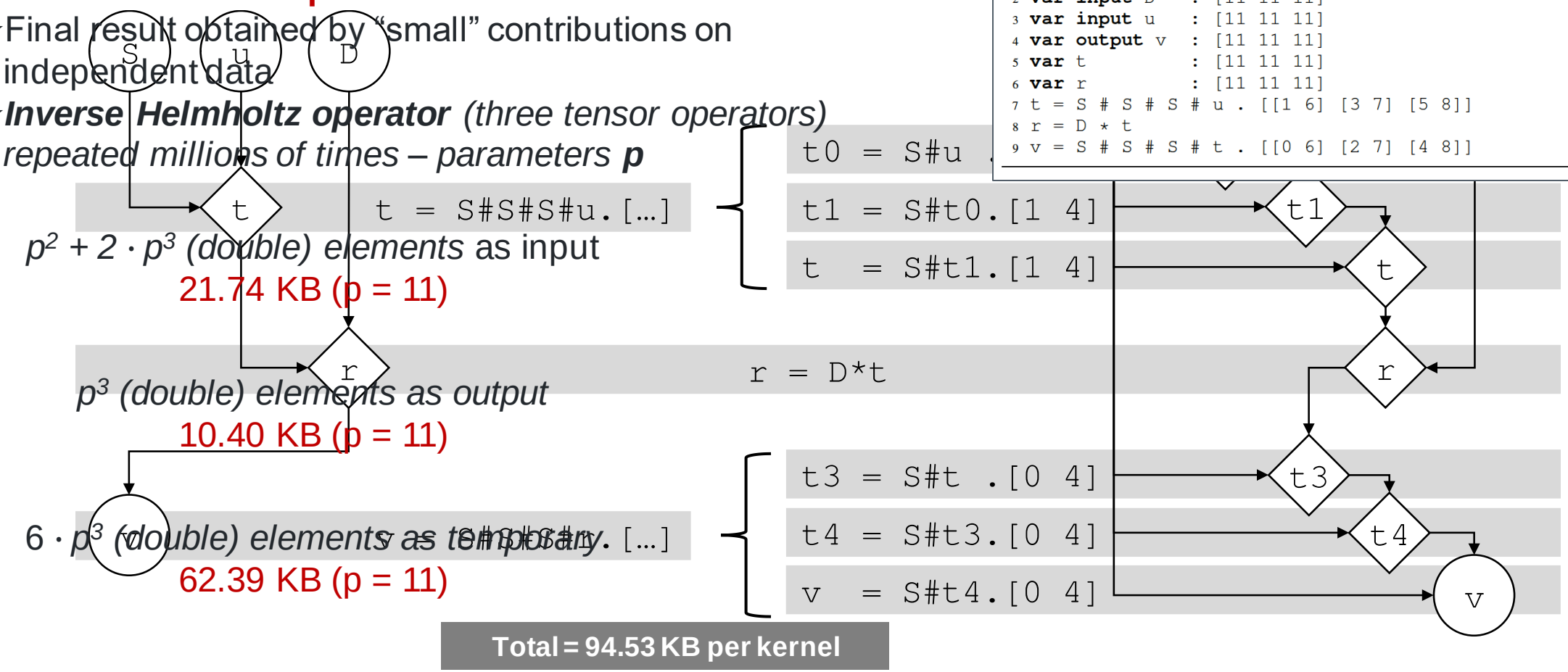
Accelerated computationally-intensive kernels + **Machine-learning** kernels

The Case of Computational Fluid Dynamics

Numerical simulation application that requires to solve **partial differential equations**

★Final result obtained by "small" contributions on independent data

★***Inverse Helmholtz operator*** *(three tensor operators) repeated millions of times – parameters* ***p***



Total = 94.53 KB per kernel

EVEREST Target System

cloudFPGA

- **Disaggregated FPGAs** directly attached to the network (64 FPGA instances)
- **Low latency** and **high bandwidth** system
- Separation between **Shell** and **Role** modules
- **cFDK framework** for system generation

FPGA-Accelerated HPC Cluster

- Cluster of **PCIe-attached FPGAs** (Alveo) with HBM architecture (up to 460 GB/s per board)
- **Xilinx Vitis framework** for HLS and system integration
- Support for the integration of **custom HDL**

CPU Reference System

- CPU-based infrastructure to **execute end-to-end workflows**, **manage storage**, and **data transfers**
- Extended to support the **offloading of tasks** to **FPGA servers**

LEXIS

Exploit **spatial parallelism**

High **memory bandwidth**

Different nodes to better **match applications**

Data-intensive (memory-bound) applications

Seamless support for multiple nodes

Limited **FPGA resources** (esp. memories)

FPGA Memory Architecture

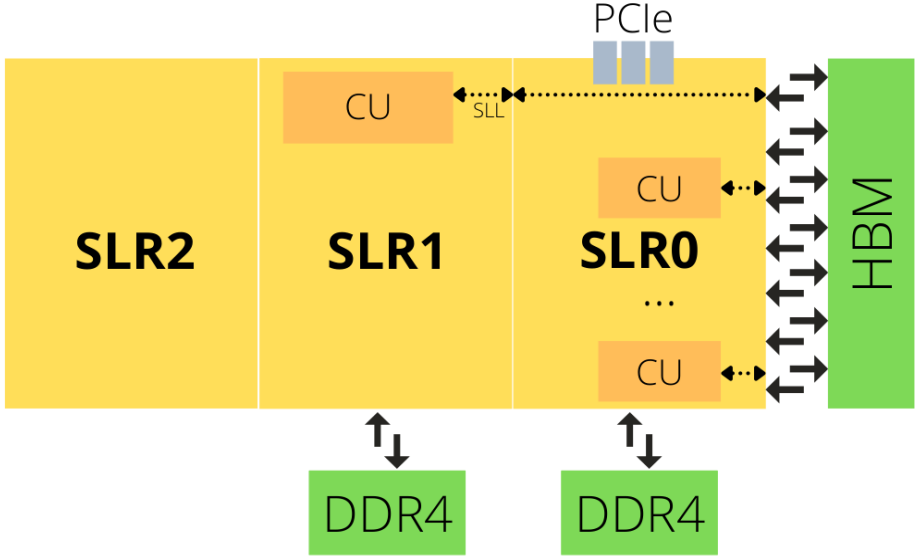
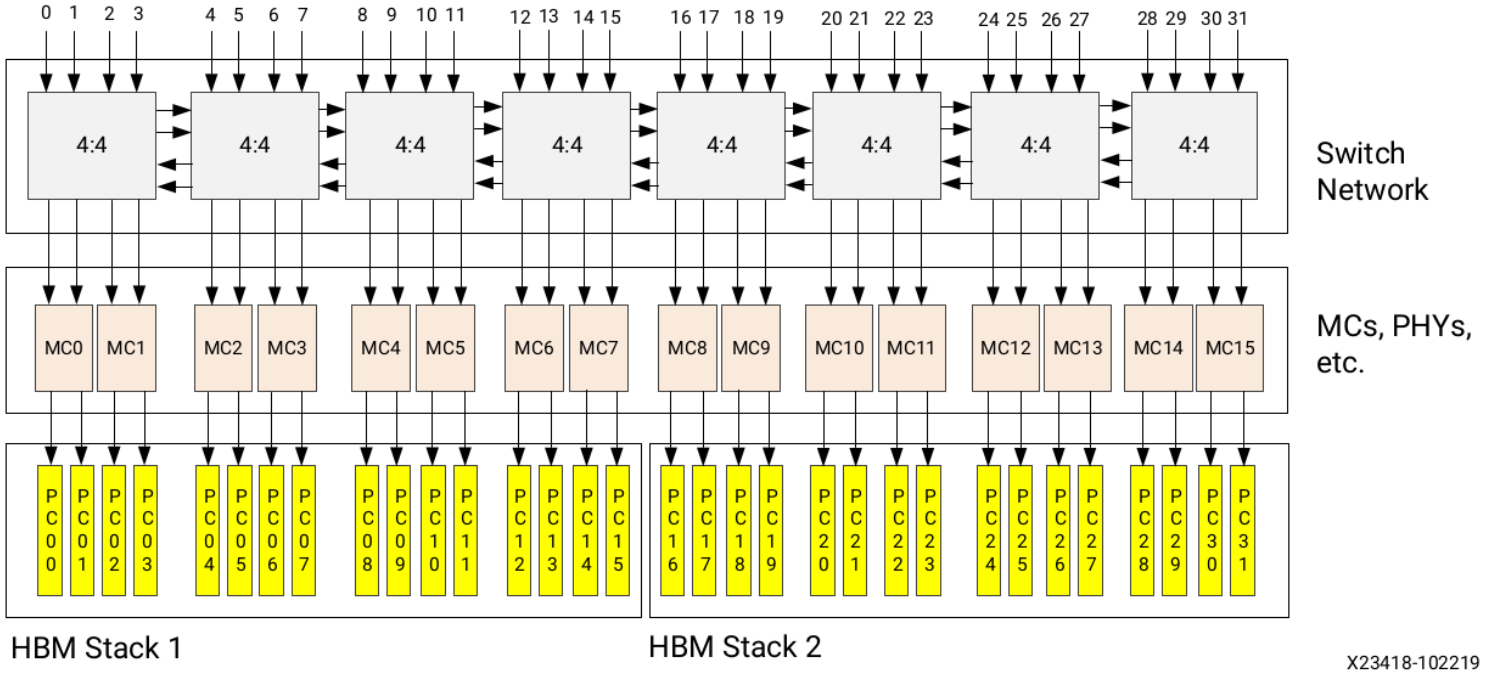


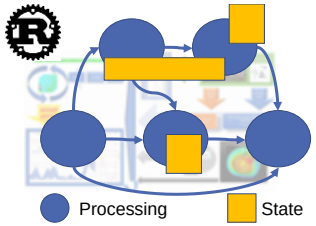
Table 1. Alveo U280 SLR resources.

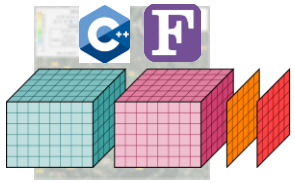
Resources	SLR0	SLR1	SLR2
HBM	32×256MB	-	-
DDR4	16GB	16GB	-
PLRAM	2×4MB	2×4MB	2×4MB
CLB LUT	369K	333K	367K
CLB Register	746K	675K	729K
Block RAM tile	507	468	512
UltraRAM	320	320	320
DSP	2,733	2,877	2,880

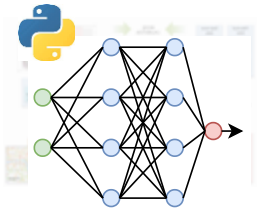


Max theoretical bandwidth: 460.8 GB/s

EVEREST System Development Kit (SDK)







Different **input flows**
starting from different **input languages**

Support for **multiple target boards**

Collection of interoperable and open-source tools to create hardware/software systems that can adapt to the target system, the application workflow, and the data characteristics

- Compilation framework based on **MLIR** to unify the input languages
- **High-level synthesis** and **hardware generation flow** to automatically create optimized architectures
- Creation of hardware and software variants to match architecture features
- Use of state-of-the-art frameworks and commercial toolchains for FPGA synthesis

tvm
MLIR
DandA
XILINX VITIS
cFDK
HEAppE
HyperLoom
(and more...)

Additional Data-Related Issues

Application-specific optimizations

★ For example, in Helmholtz, one of the tensors is constant over all elements and another is diagonal

how to specify them at the language level and exploit them during design?

Limited BRAM resources

★ *The **number of parallel kernels** can be limited*

how to optimize local storage?

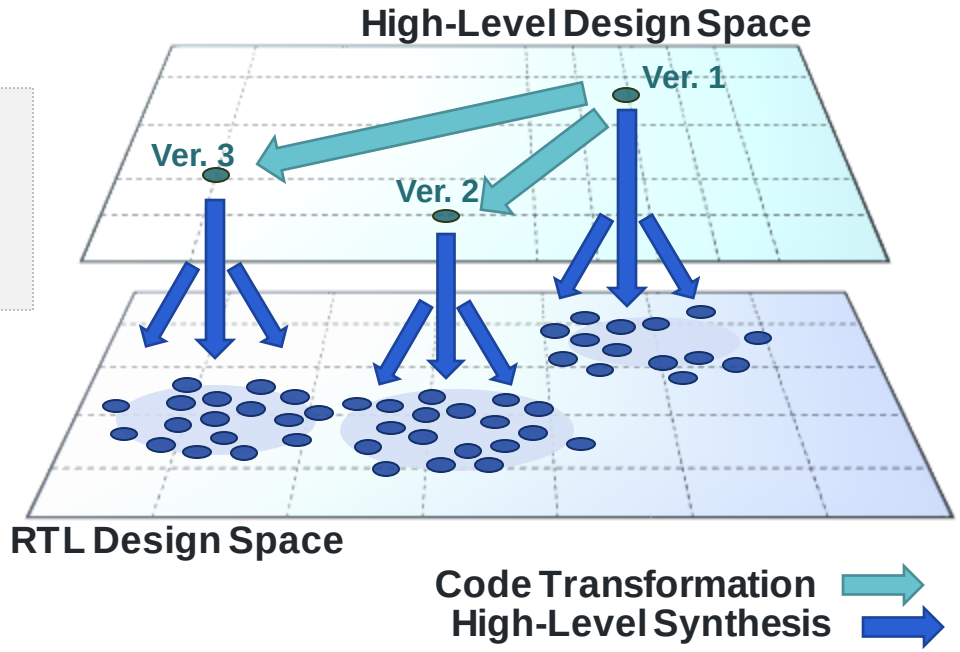
System-level optimization

★ *Creation of **batch of elements** to be executed in series by each kernel*

how to size the batches and hide communication latency?

From Software to Hardware

High-Level Synthesis is a collection of **algorithms and design methodologies** to automatically derive a synthesizable hardware description from a high-level description

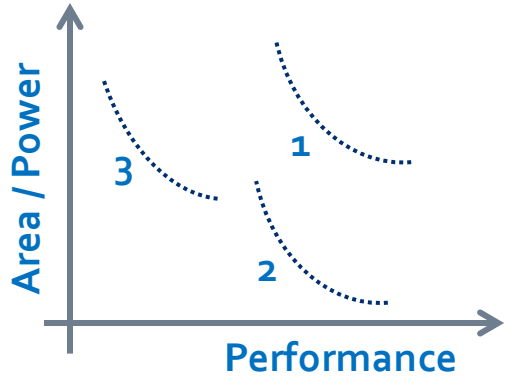


Hardware generation is highly dependent on **how the code is written**

- ★ C does not express concurrency
- ★ Arrays vs. pointers
- ★ Multi-port interfaces

C is a **low-level language** for hardware design

Input language for system specification is an open question



Hardware HPC (Memory) Architectures

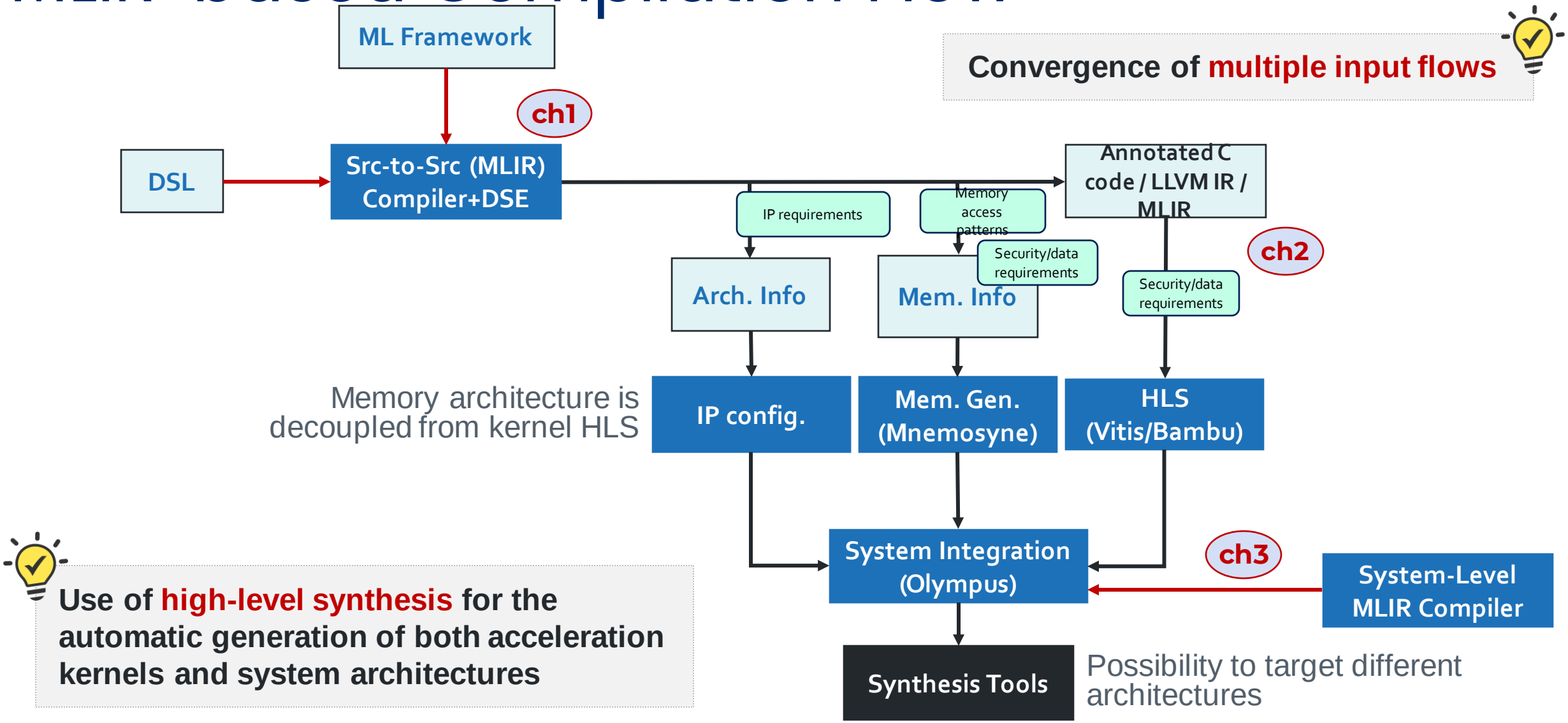
What do we mean with **memory architecture**?

Every hardware module that is responsible to provide data to the accelerator kernels

Additional issues:

- BRAM resources are limited
  - Helmholtz operator requires >94 KB of local data
  - If local storage is not optimized, the number of parallel kernels can be limited
- Application-specific details can be used to optimize the data transfers
  - In Helmholtz, one of the tensors is constant over all elements – how to match these details with platform characteristics?
  - Better to transfer data for a "batch" of elements and then execute them in series – how many? again, limited storage

MLIR-based Compilation Flow



Convergence of **multiple input flows**

Memory architecture is decoupled from kernel HLS

Possibility to target different architectures

Use of high-level synthesis for the automatic generation of both acceleration kernels and system architectures

Multi-Level Intermediate Representation (MLIR)

- **Novel compiler infrastructure** centered on reuse and extensibility
  - Becoming popular as a framework for domain-specific language (DSL) compilers for heterogeneous systems

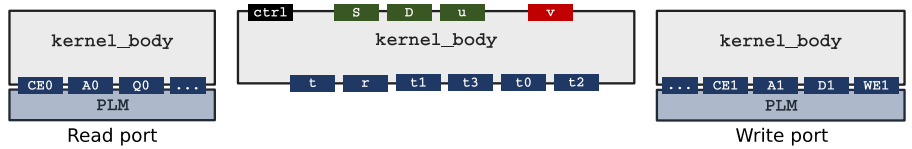
- MLIR is a **collection of dialects**, each representing different layers of abstraction through various operators, types, and attributes.
  - Custom dialects can easily be added for domain-specific problems while reusing existing infrastructure
  - Dialects can be integrated into larger language stacks via lowering,

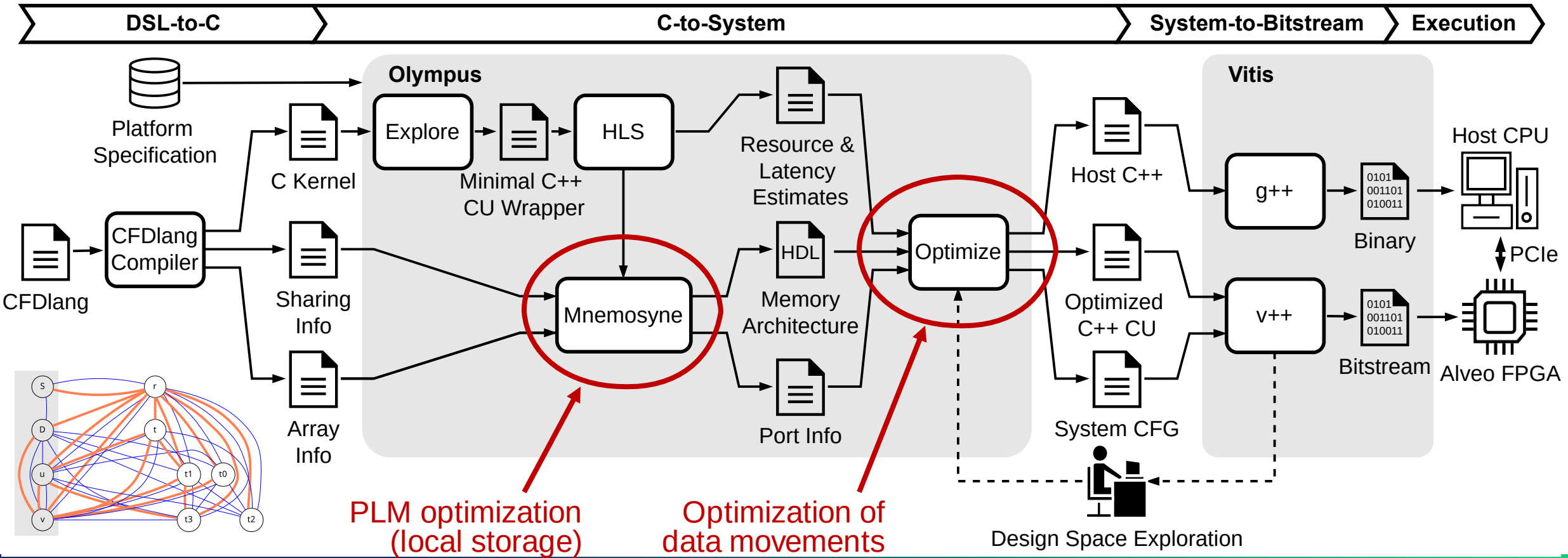
From DSL to Bitstream – Focus on Memory

```
var input S    : [11 11]
var input D    : [11 11 11]
var input u    : [11 11 11]
var output v   : [11 11 11]
var t          : [11 11 11]
var r          : [11 11 11]
t = S # S # S # u . [[1 6] [3 7] [5 8]]
r = D * t
v = S # S # S # t . [[0 6] [2 7] [4 8]]
```

Memory architecture generation is decoupled from kernel HLS

```
void kernel_body(double S[11][11], double D[11][11][11], double u[11][11][11],
                 double v[11][11][11],
                 double t[11][11][11], double r[11][11][11], double t1[11][11][11],
                 double t3[11][11][11], double t0[11][11][11], double t2[11][11][11])
```





Mnemosyne – Reuse What is not Used

Generally, we use one **PLM unit** (possibly composed of many banks) for each **data structure** (array)

Reuse the same memory IPs
for several data structures

"Two data structures are compatible if they can be allocated to the same PLM unit (memory IPs)"

A common case: accelerator kernels never executed at the same time

- Possible only at system-level, when integrating the components
- Optimizations of accelerator logic and memory subsystem are independent

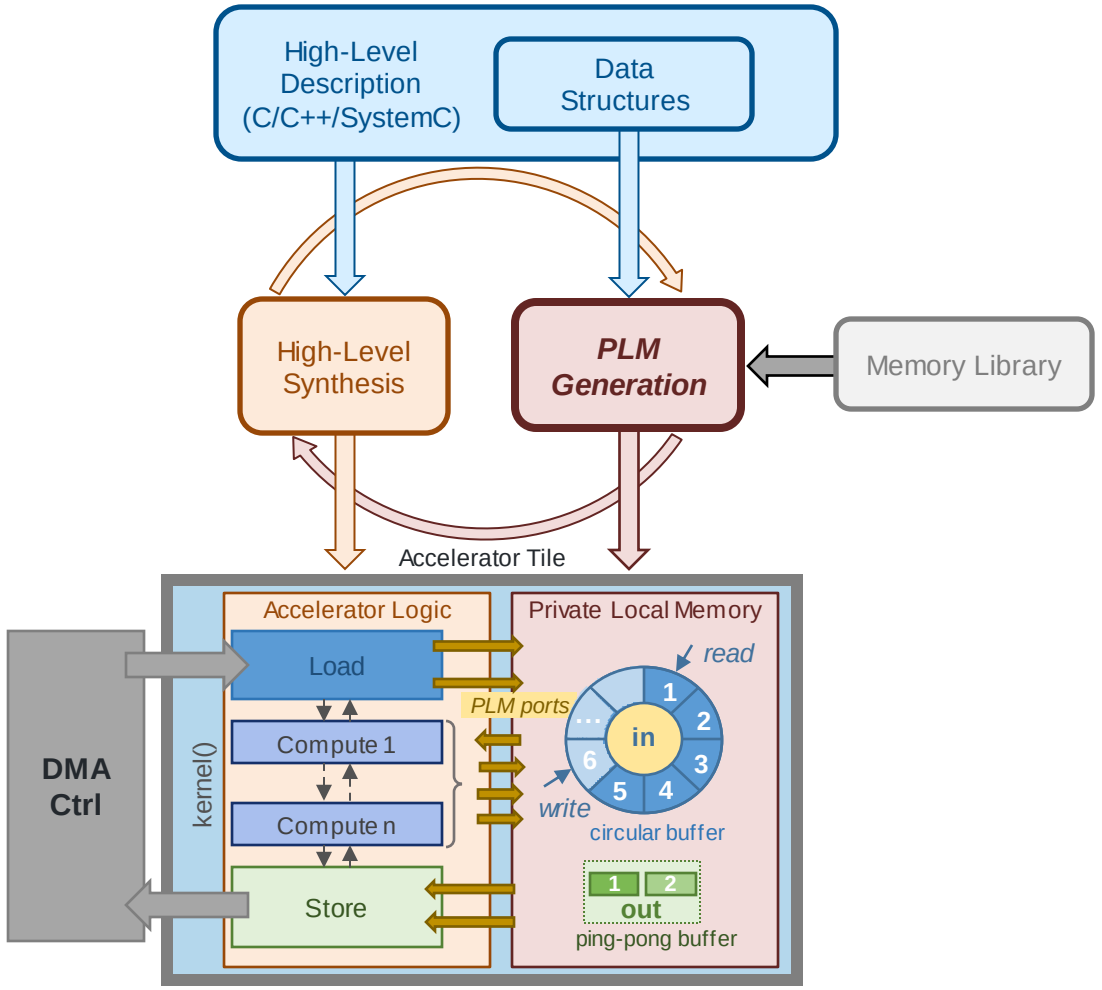
PLM Customization for Heterogeneous SoCs

High-Level Synthesis (HLS) to create the accelerator logic

- Definition of memory-related parameters (e.g., number of process interfaces)

Generation of **specialized PLMs**

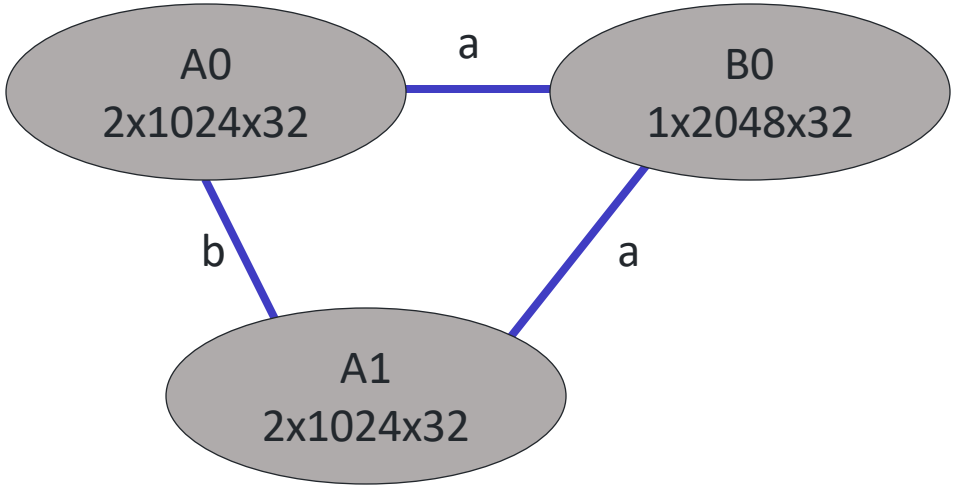
- Technology-related optimizations
- Possibility of system-level optimizations across accelerators



Memory Compatibility Graph (MCG)

Graph to represent the possibilities for optimizing the data structures

- Each node represents a data structure to be allocated, annotated with its data footprint (after data allocation)
- Each edge represents compatibility between the two data structures
- **Automatically extracted** from the **MLIR-based compiler flow**
  - Variant exploration to achieve the "best solutions"

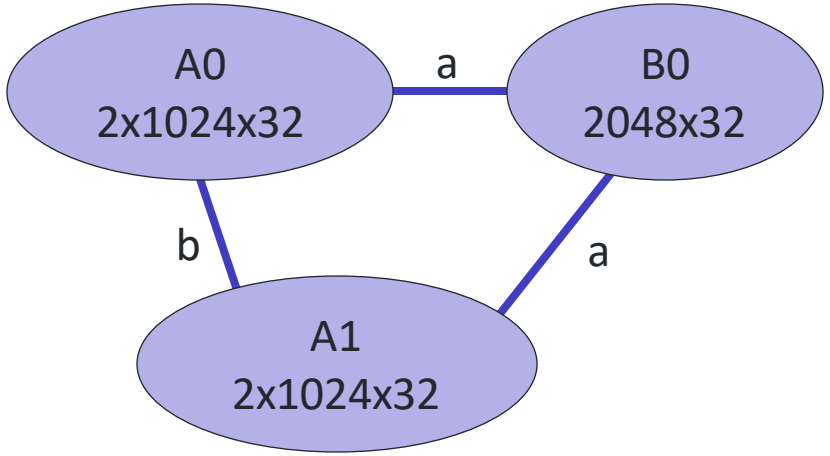


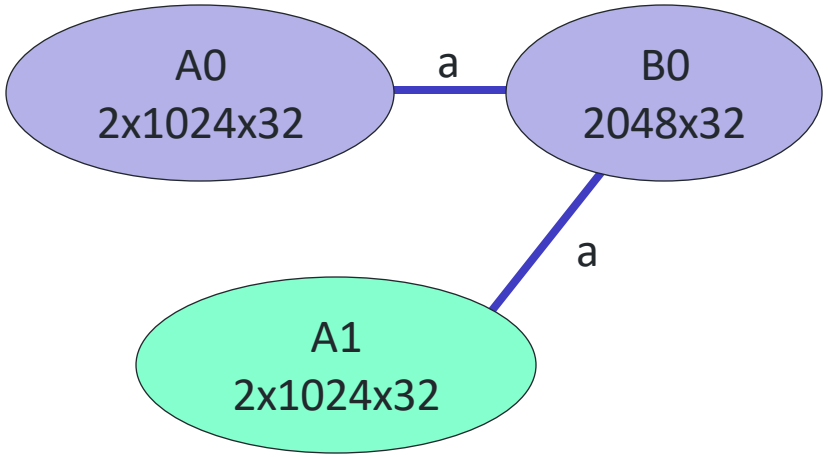
a) **Address-space compatibility**: the data structures are compatible and can use the same memory IPs
b) **Memory-interface compatibility**: the ports are never accessed at the same time and the data structures can stay in the same memory IP

Clique Definition

“A clique is a subset of the vertices of the memory compatibility graph such that every two vertices are connected by an edge”

A clique represents a **set of data structures** that can share the same memory IPs



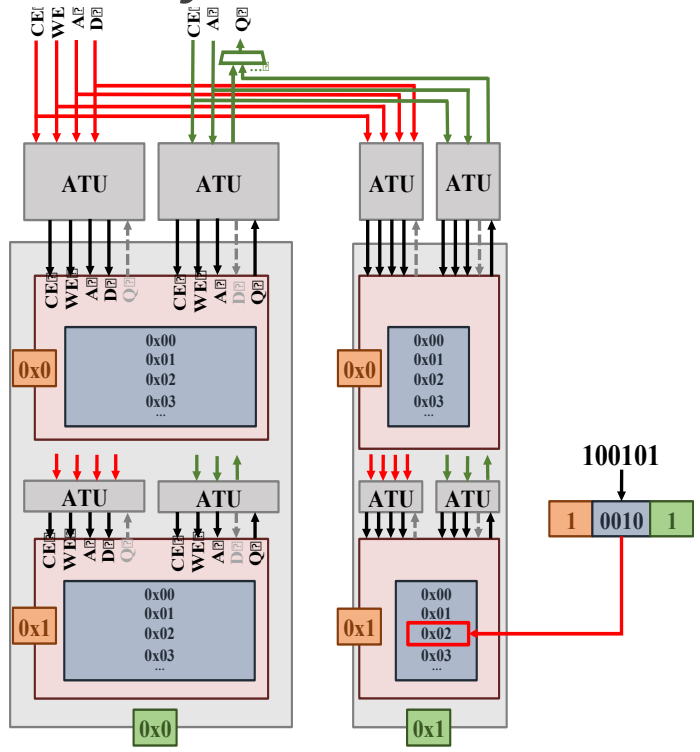


We need two distinct configurations!
{A0,B0} and {A1} or {A1,B0} and {A0}?

PLM Controller Generation

A **lightweight PLM controller** is created for each compatibility set (clique) based on the bank configuration

- Accelerator logic is not aware of the actual memory organization
- Array offsets need to be translated into proper memory addresses



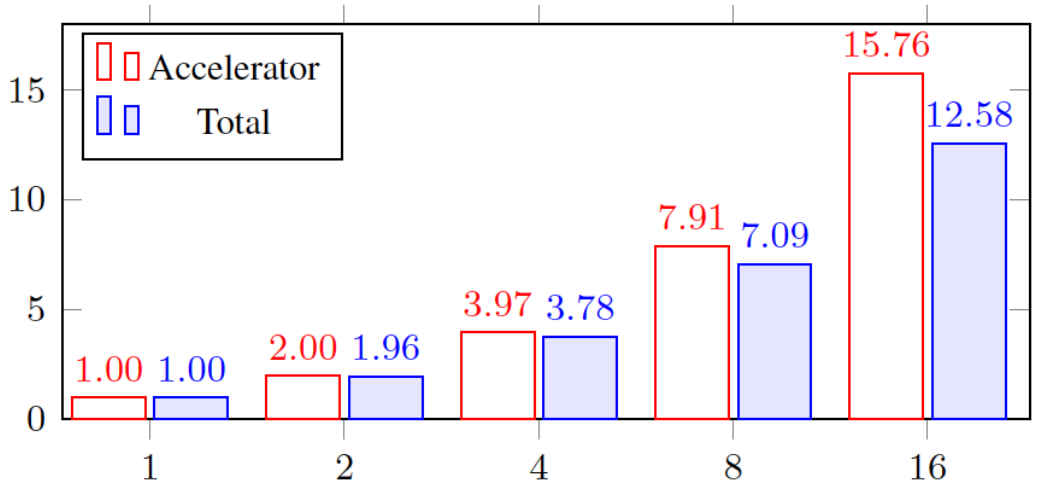
Custom logic with negligible overhead, especially when the number of banks and their size is a power of two

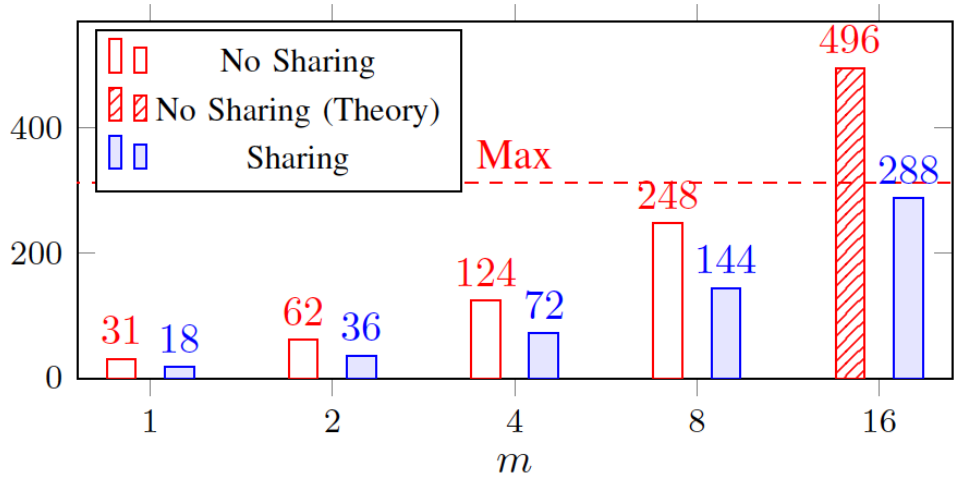
Sharing Evaluation

Xilinx Zynq UltraScale+ MPSoC ZCU106 board

★ Small CFD simulation of 50,000 elements

	m, k	LUT		FF		DSP	
No Sharing	1	11,318	(4.9%)	9,523	(2.1%)	15	(0.9%)
	2	15,929	(6.9%)	12,583	(2.7%)	30	(1.7%)
	4	25,728	(11.2%)	18,663	(4.1%)	60	(3.5%)
	8	42,679	(18.5%)	30,795	(6.7%)	120	(6.9%)
Sharing	1	11,292	(4.9%)	9,533	(2.1%)	15	(0.9%)
	2	15,572	(6.8%)	12,596	(2.7%)	30	(1.7%)
	4	24,480	(10.6%)	18,663	(4.1%)	60	(3.5%)
	8	42,141	(18.3%)	30,782	(6.7%)	120	(6.9%)
	16	77,235	(33.5%)	55,053	(12.0%)	240	(13.9%)





Memory sharing allows us to fit more kernels and better exploit the parallelism

K. F. A. Friebel, S. Soldavini, G. Hempel, C. Pilato, J. Castrillon: **From Domain-Specific Languages to Memory-Optimized Accelerators for Fluid Dynamics**. HPCFPGA (2021)

Olympus – Automated System-Level Integration

Complete **hardware architecture generation flow** from high-level specification

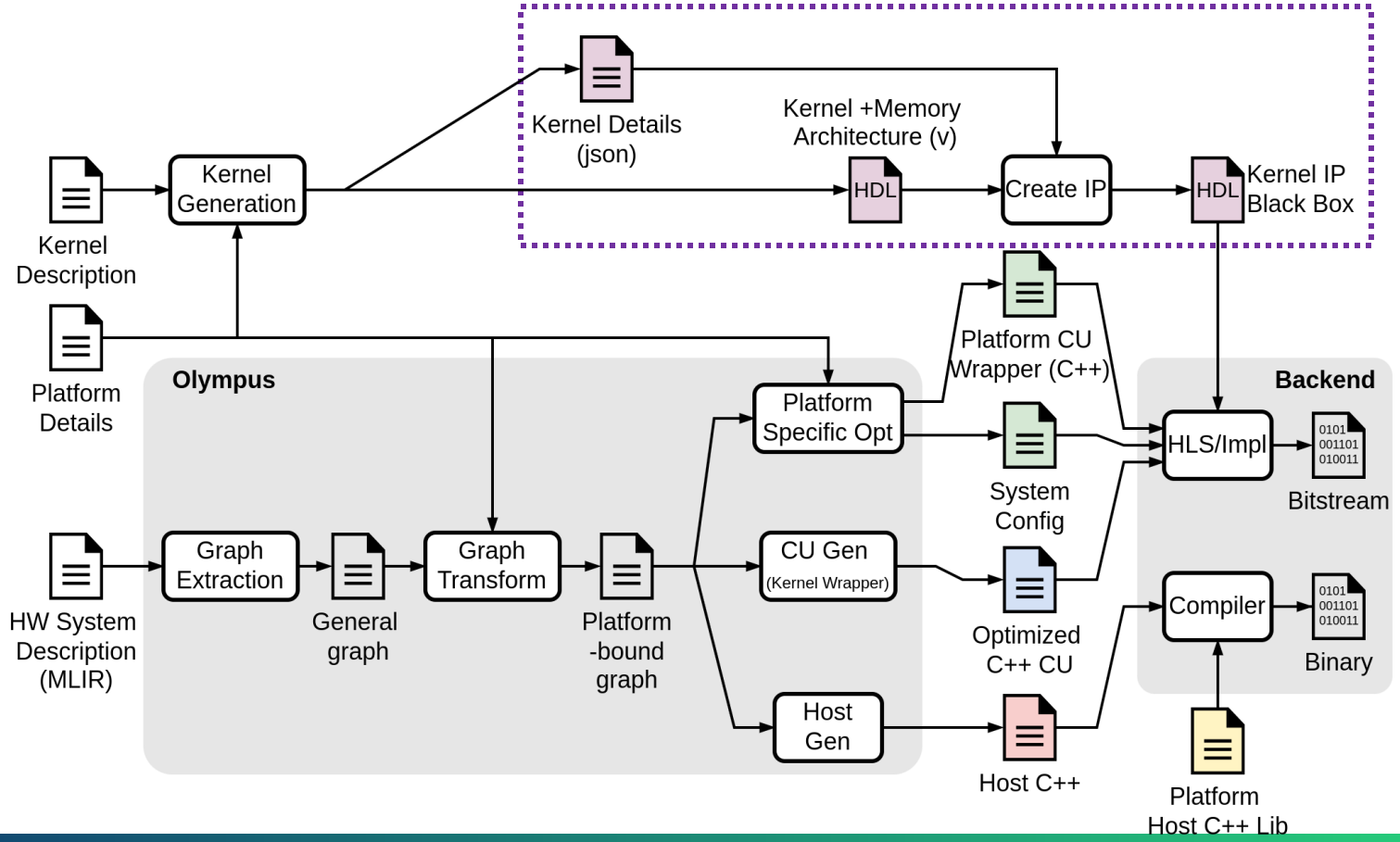
Possibility to use alternative HLS tools/HDL generators

Platform-specific description

- HBM-based Xilinx Alveo
- IBM CloudFPGA
- …

Host code generation

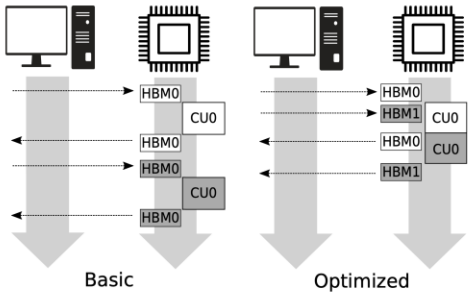
- Based on **platform libraries** for the specific target



Olympus Optimizations

Double buffering

★ To hide latency of host-FPGA data transfers

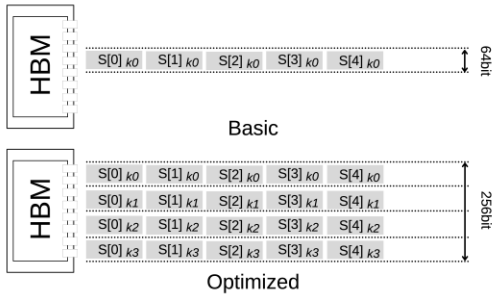


automatic batch sizing

Bus optimization and data interleaving

★ To maximize bandwidth (e.g., 256-bit AXI channels)

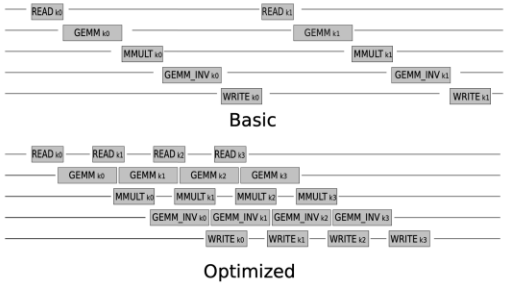
S. Soldavini, D. Sciuto, C. Pilato: **Iris: Automatic Generation of Efficient Data Layouts for High Bandwidth Utilization**. ASP-DAC (2023)



algorithms for efficient data layout on the bus

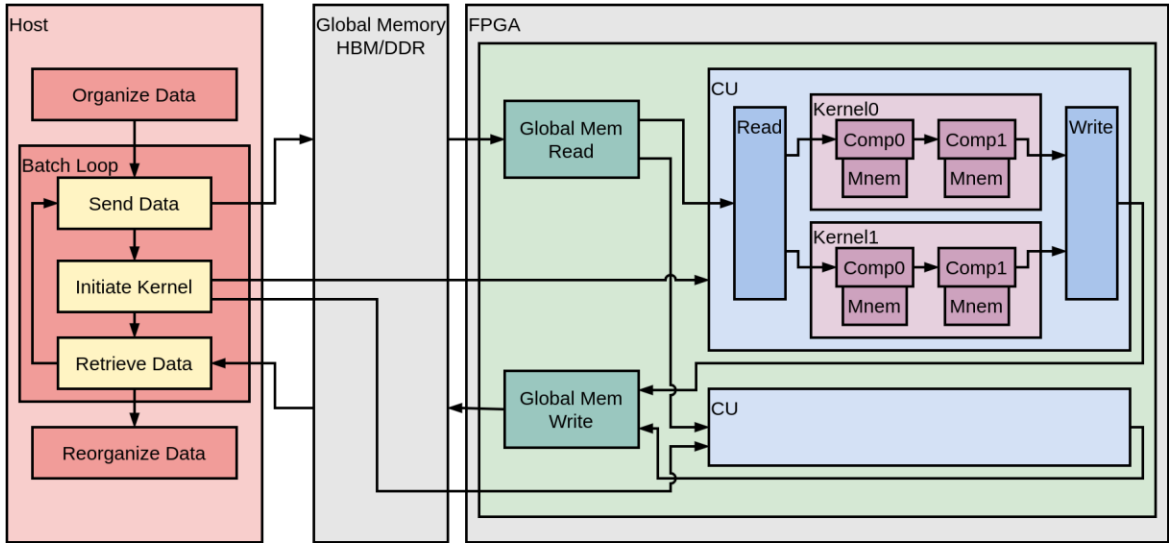
Dataflow execution model

★ To enable kernel pipelining



automatic (pre-HLS) code transformations

Olympus – System Generation Flow



Parallel computing units

Inputs

- ★ Algorithm parallelism
- ★ Characteristics of the target platform(s)
- ★ Interfaces of the modules (HLS tools)

Outputs

- ★ Synthesizable C++ code
- ★ Host library implementation
- ★ System configuration file

“Intelligent” policies to coordinate and/or protect data transfers

Olympus MLIR Dialect: Kernel Operator

- **`callee`**: name of the kernel
- **`latency, ii`**: timing estimates fr
- **`ff, lut, bram, uram, dsp`**: res HLS/synthesis
- **`operand_segment_sizes`**: lists ho inputs (first number) and outputs

```
"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

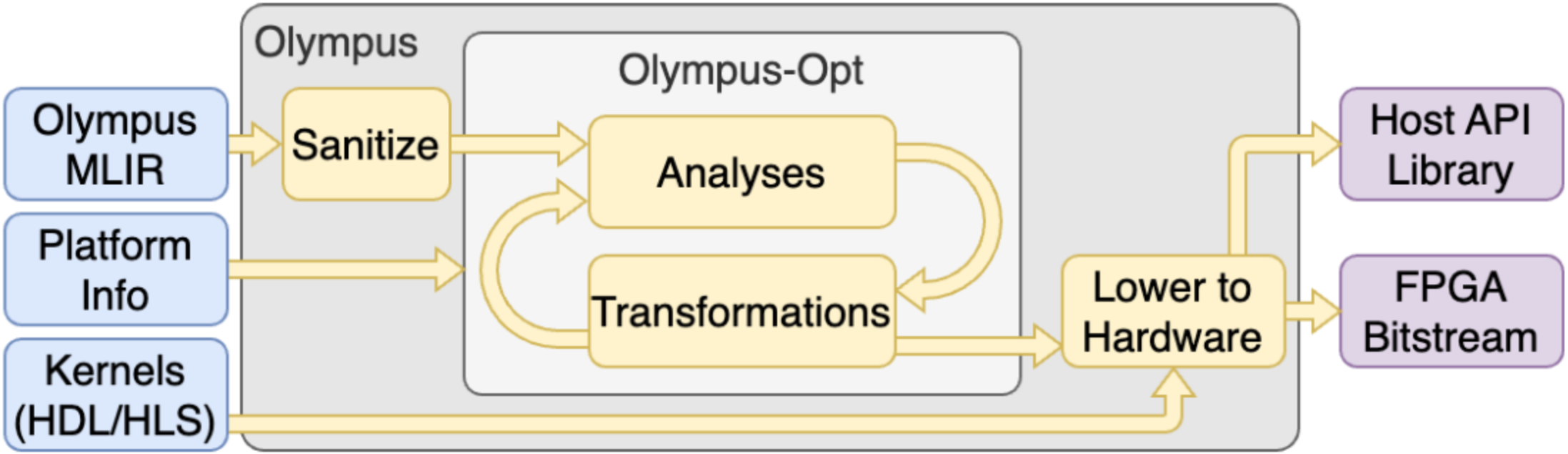
Olympus MLIR Dialect: Channel Operator

- **`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>
)
```

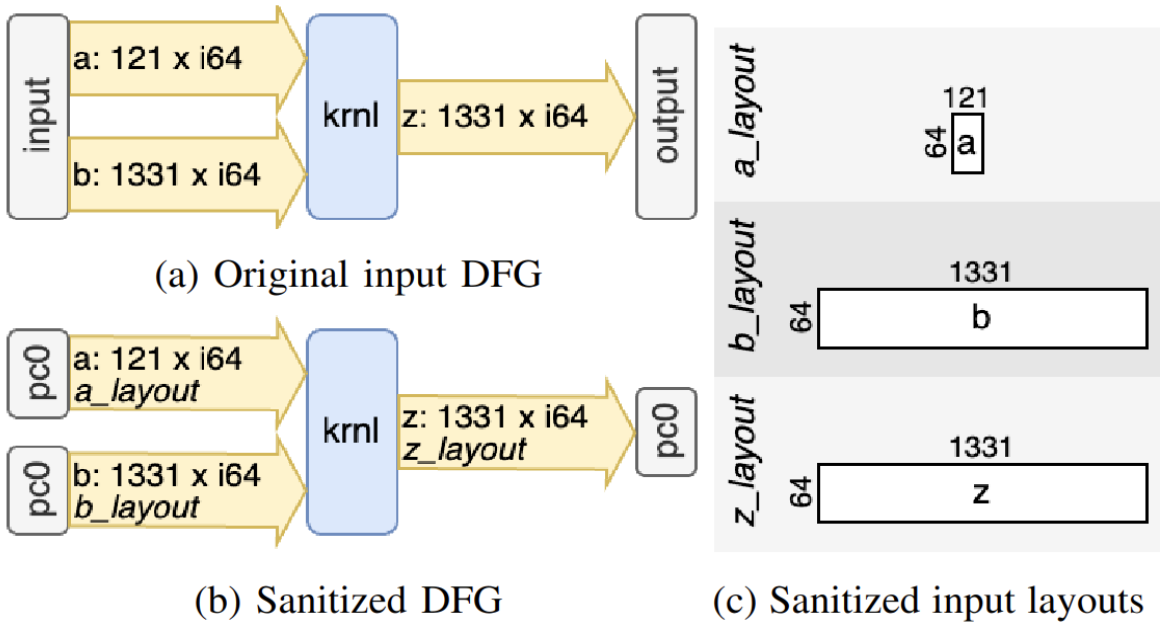
Fig. 1: Sample channel operator

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

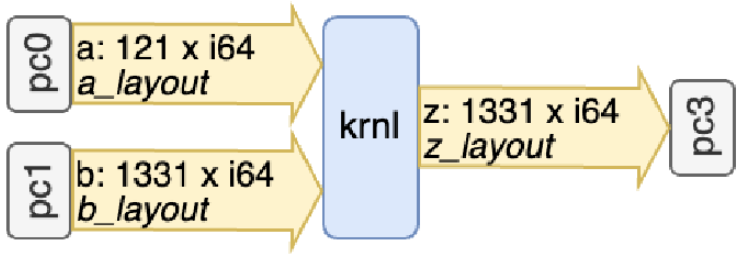
(c) Sanitized input layouts

Analyses

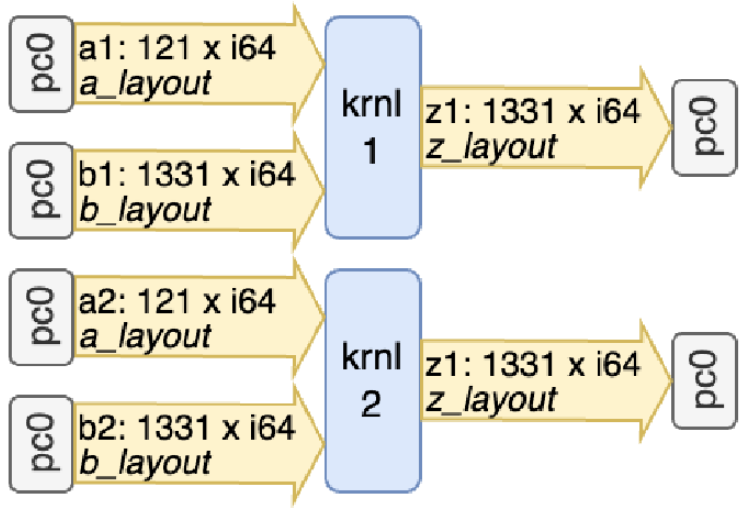
- **Bandwidth utilization**:
  - From target PC information and the attributes of each data channel
- **Resource utilization**:
  - From total resource availability and the kernel resource utilization
  - Using the results of these analyses, transformation passes can be chosen to alter the DFG to increase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

Transformations

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



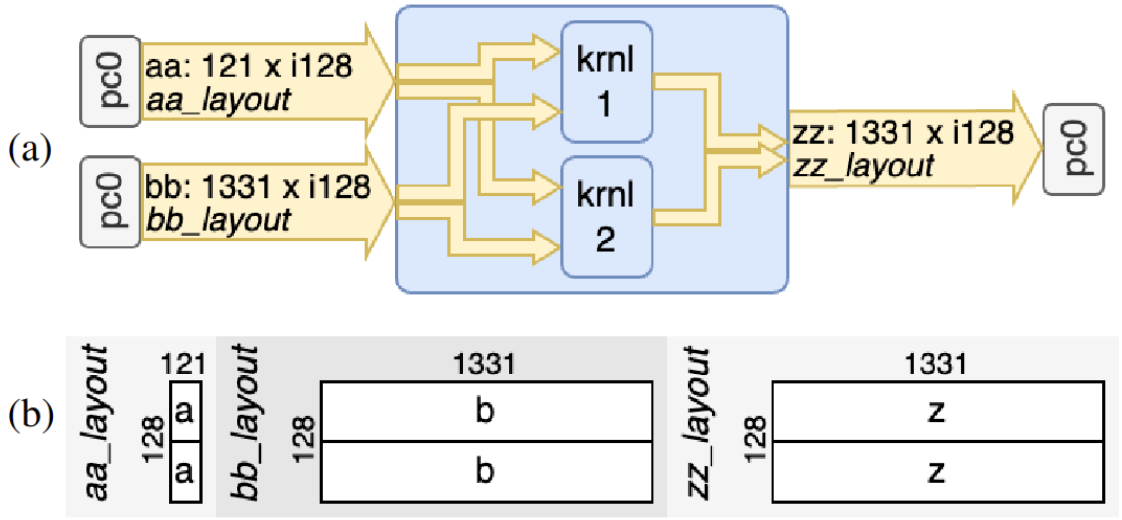
- Replication [1]
  - Instantiate more DFG instances
  - Increases parallelism when using available



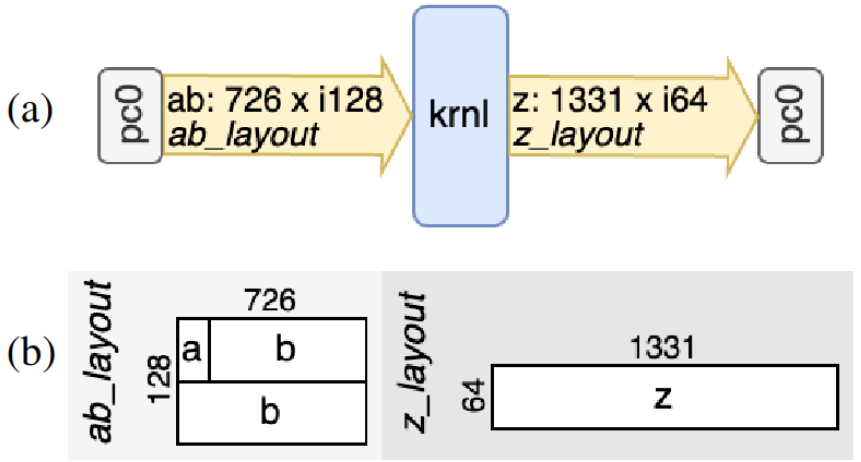
[1] S. Soldavini et. al. “Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics”. In: ACM TRETS (Sept. 2022). doi: 10.1145/3563553

Transformations

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



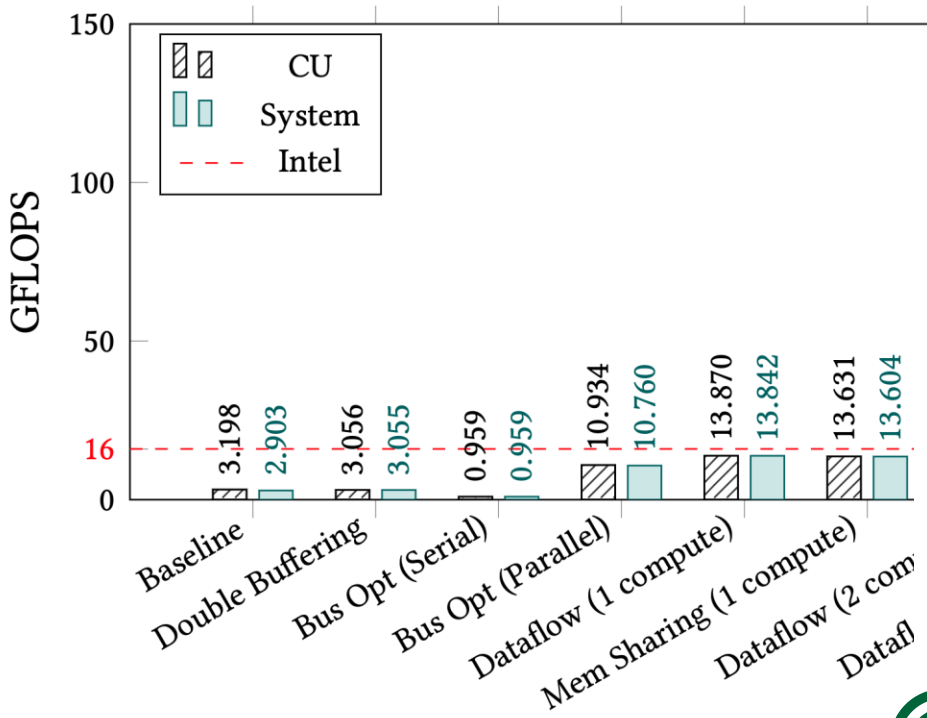
- Bus optimization [2]
  - Iris algorithm: Efficiently "pack" multiple inputs o
  - Increases bandwidth utilization, frees channels



[1] S. Soldavini et al. "Automatic Creatn 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.221io1.04361

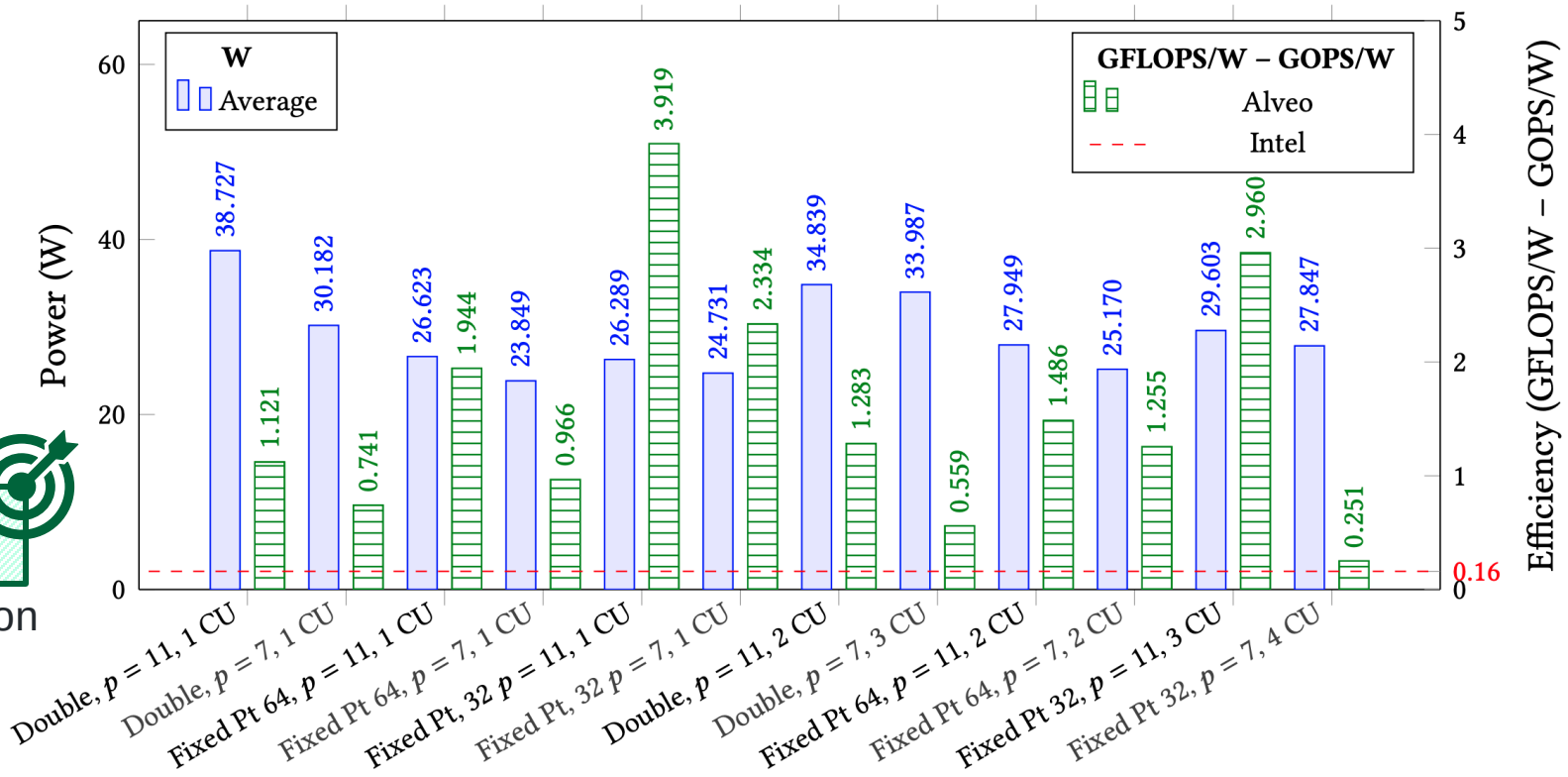
Results on HBM FPGA (Alveo u280)



Best performance: 103 GOPS
(118x faster than "starting point")

Results are 6x better than Intel CPU

★ Intel is an optimized, vectorized implementation



Configuring PLM and data transfers based on custom data formats

S. Soldavini, K. F. A. Friebel, M. Tibaldi, G. Hempel, J. Castrillón, C. Pilato: **Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics.** ACM TRETS (2022)

Conclusions (i)

Accelerators are becoming **key components** in modern architectures

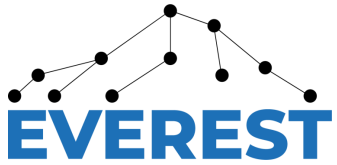
★ **Data management optimizations** are becoming the key for the creation of efficient FPGA architectures (… more than pure kernel optimizations)

★ Novel **HBM architectures** offer high bandwidth (that's why they are called *high-bandwidth* memory architectures… ☺) but their design is complex

The increasing **design complexity** requires embracing **high-level synthesis** to increase productivity

★ Use of HLS to generate both **accelerator kernel** and the **associated memory architecture**

★ Possibility to **target different platforms**



Mnemosyne

Olympus

Conclusions (ii)

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

EUROPEAN BIG DATA VALUE FORUM

Thank you!

ORGANISED BY

BDV BIG DATA VALUE ASSOCIATION

IN COLLABORATION WITH

ITI INVESTIGATE TO INNOVATE

European Commission

INCLIVA | VLC Biomedical Research Institute

POLITÉCNICA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNDER THE AUSPICES OF

EU23 SPANISH PRESIDENCY COUNCIL OF THE EUROPEAN UNION