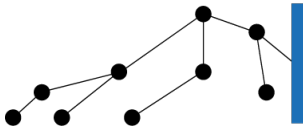
POLITECNICO
MILANO 1863



EVEREST

Universidad Complutense de Madrid – April 21, 2022

Automatic generation of hardware memory architectures for HPC

Christian Pilato

Assistant Professor

christian.pilato@polimi.it

About Me

Assistant Professor (RTD-B - Ricercatore a Tempo Determinato Senior)

Office: DEIB (Building 20 - 1st floor – Room 029) **Website**: http://pilato.faculty.polimi.it

PhD Student
2008-2011

Research Assistant
2011-2013

Postdoc Research Scientist
2013-2016

Postdoc Research Assistant
2016-2018

Assistant Prof.
2018-*now*

R&D Internship
6 months

Visiting Researcher
3 months

https://github.com/sld-columbia/esp

Visiting Researcher
4 months

Visiting Researcher
9 months

R&D Projects

FP6 HARTES
FP7 SYNAPTIC
FP7 FASTER
DARPA PERFECT
SRC CFAR
H2020 CERBERO
DARPA CRAFT
H2020 EVEREST

EVEREST: Big Data Analytics on FPGA

H2020 project funded under the call – "**Big Data technologies and extreme-scale analytics**" [Kick-off on Oct 1, 2020] – http://www.everest-h2020.eu

Project Coordinator: Christoph Hagleitner, IBM Research Europe, Zurich, Switzerland

Scientific Coordinator: Christian Pilato, Politecnico di Milano, Italy

Key idea: a coordinated action with the appropriate technology areas (e.g., AI, analytics, software engineering, HPC, Cloud technologies, IoT and edge/fog/ubiquitous computing) ➔ **Computing continuum to enable cloud-to-edge integration**

system engineering/tools to contribute to the co-design of federated/distributed systems ➔ **EVEREST *system* development kit**

architectures for collecting, managing and exploiting

virtualization hardware acceleration data security

standardized interconnection methods

runtime management

domain-specific extensions

We can use AI/ML to optimize data analytics and knowledge extraction, but this is not strictly an AI/ML-related project!

POLITECNICO MILANO 1863

EVEREST Partners

IBM Reseach Lab, Zurich (Switzerland)
Project administration, prototype of the target system
PI: Christoph Hagleitner

Politecnico di Milano (Italy)
Scientific coordination, high-level synthesis, flexible memory managers, autotuning
PI: Christian Pilato

Università della Svizzera italiana (Switzerland)
Data security requirements and protection techniques
PI: Francesco Regazzoni

TU Dresden (Germany)
Domain-specific extensions, code optimizations and variants
PI: Jeronimo Castrillon

Centro Internazionale di Monitoraggio Ambientale (Italy)
Weather prediction models
PI: Antonio Parodi

IT4Innovations (Czech Republic)
Exploitation leaders, large HPC infrastructure, workflow libraries
PI: Katerina Slaninova

Virtual Open Systems (France)
Virtualization techniques, runtime extensions to manage heterogeneous resources
PI: Michele Paolino

Duferco Energia (Italy)
Application for prediction of renewable energies
PI: Lorenzo Pittaluga

Numtech (France)
Application for monitoring the air quality of industrial sites
PI: Fabien Brocheton

Sygic A/S (Slovakia)
Application for intelligent transportation in smart cities
PI: Radim Cmar

Application Concepts

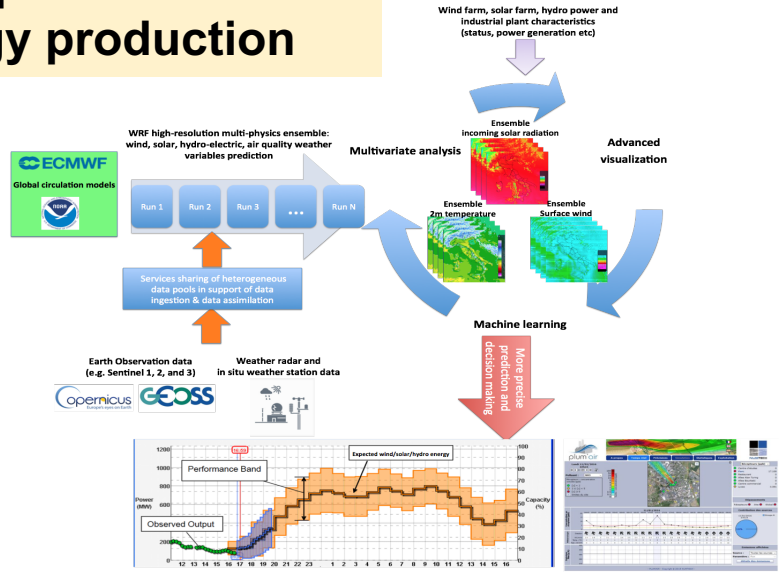
Three use cases provided by the application partners

Looking for hardware acceleration (intense data computation) with efficient and secure data management (distributed data sources)

Possibility of AI/ML-based decision making

Combination of the tasks in different pipelines

Weather-based prediction of renewable energy production



Air-quality monitoring in industrial sites

Traffic modelling for intelligent transportation

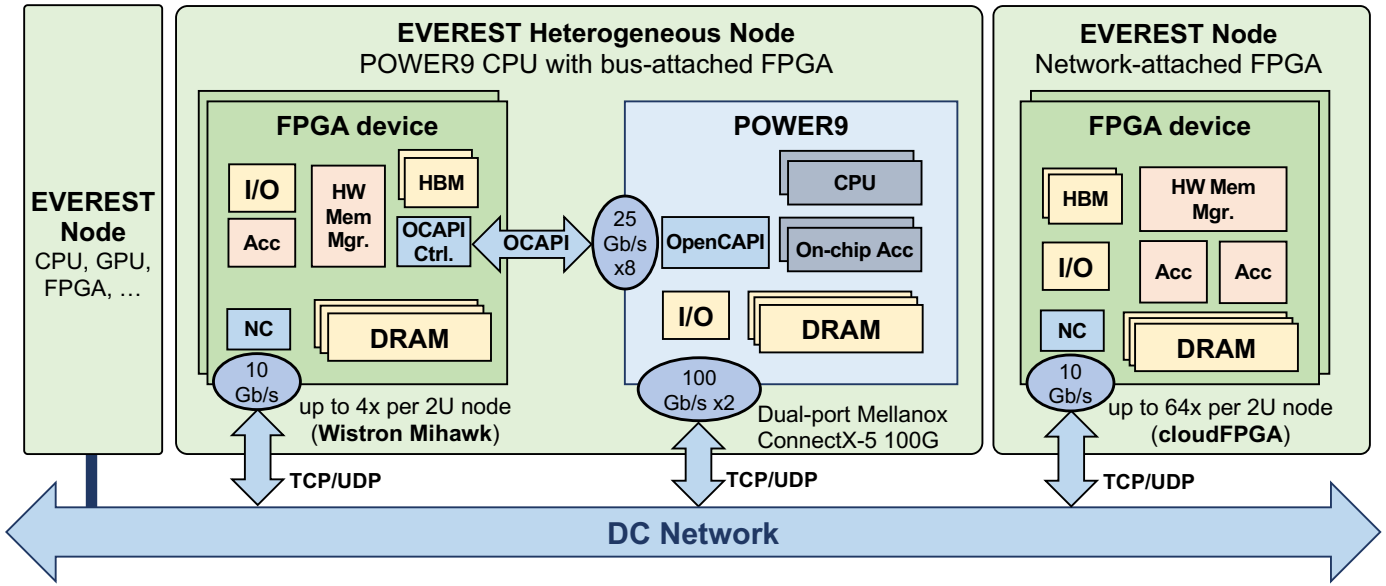
EVEREST Target System

Target prototype based on **IBM** products and internal projects

- Combination of **CPU-managed systems** with **tightly-coupled bus-attached FPGAs** and **FPGA-disaggregated systems** with **loosely-coupled network-attached FPGAs**

Other platforms (e.g., from **IT4I** and **NUM** or existing SoCs)

- Combination of devices and architectures: Xilinx Alveo, cloudFPGA, Columbia ESP, …



Standard communication formats (e.g., OpenCAPI)

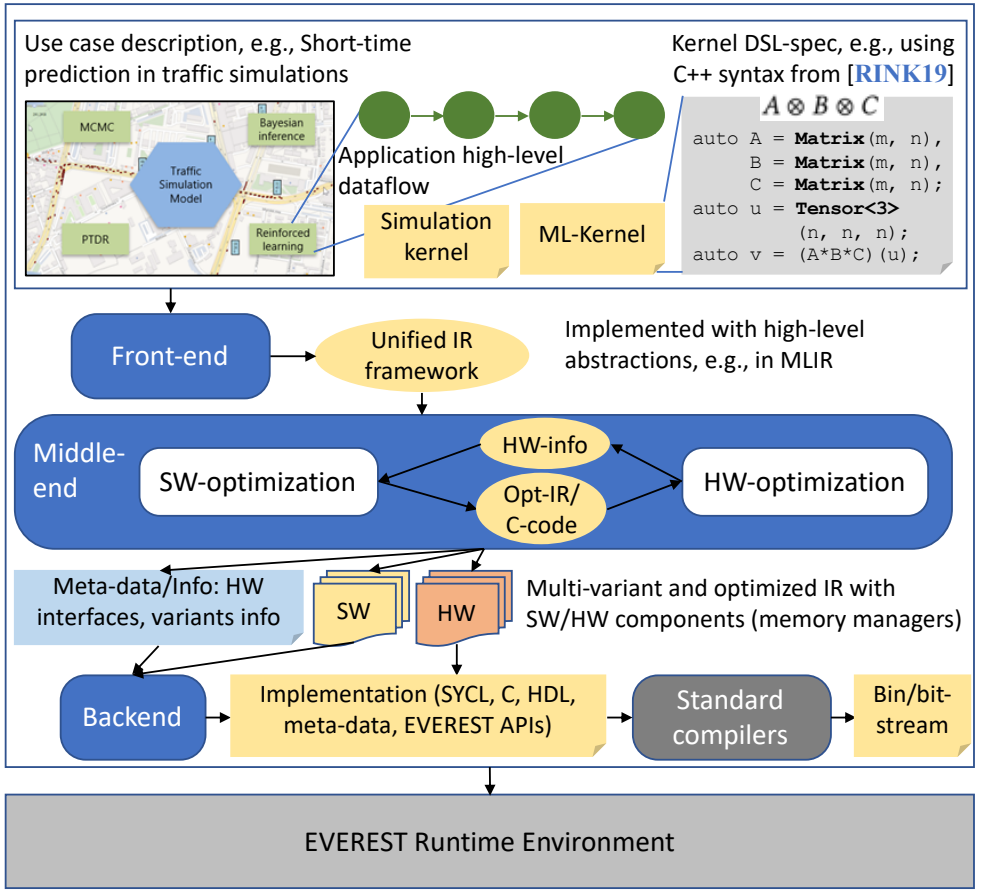
Efficient communication libraries with clear API

Virtualization of resources

Seamless integration of additional EVEREST nodes

EVEREST Programming Environment

1. **Compilation Environment:** analyzes application and creates all "variants" based on *architecture abstraction* and *application/data requirements*
   - **Exploring unified IR framework** (e.g., **MLIR**)
   - **Integration of non-functional properties with domain-specific extensions**
   - **Hardware acceleration and High-level synthesis** (**Bambu**, **Vivado HLS**)



Standard IR format and exchange files

Novel domain-specific extensions (format)

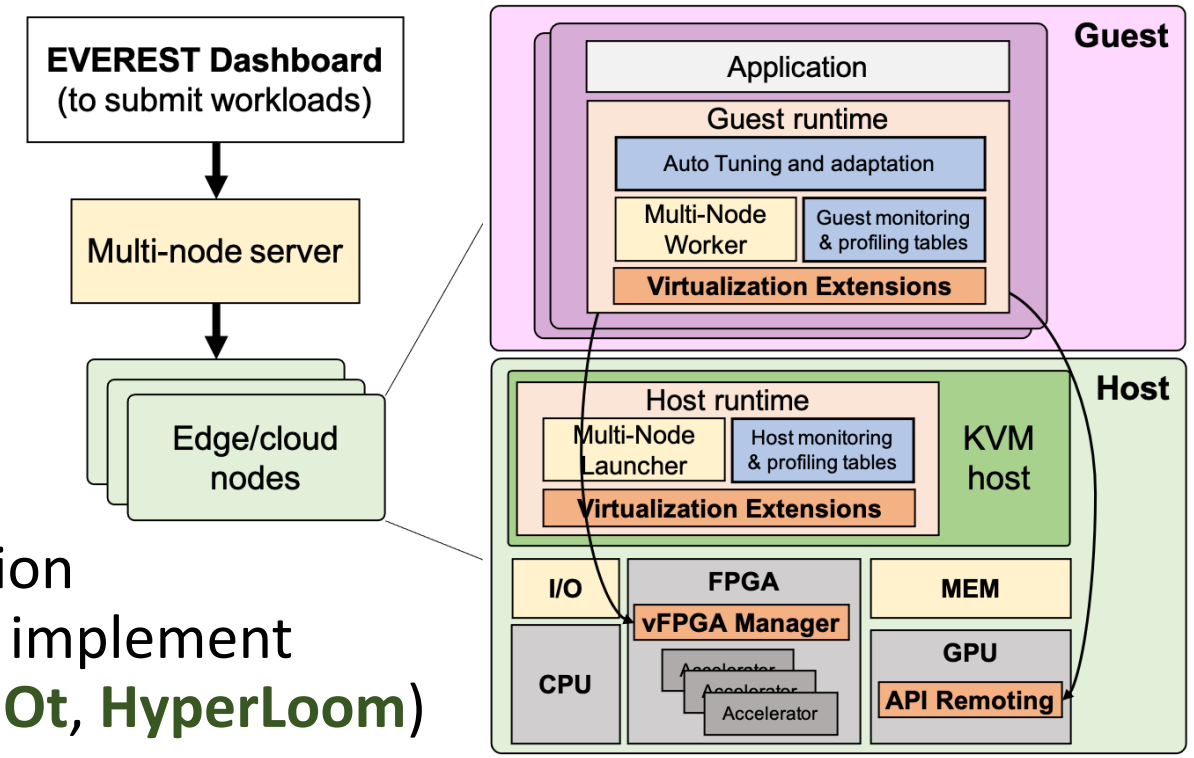
System and resource description (format)

Possibility of using different (ML) frameworks

Interoperability with different HLS tools

EVEREST Programming Environment

2. **Runtime Environment:** implements the *selection of "variants"* and the *hardware configuration* based on the *system status*
   - **Dynamic adaptation and autotuning** (**mARGOt**)
   - **Two-level runtime** for (1) virtualization of hardware resources regardless their distribution and the low-level details of the platforms; (2) implement functional decisions (**VOSYS solutions**, **mARGOt**, **HyperLoom**)



How to collect system status and expose it to the runtime?

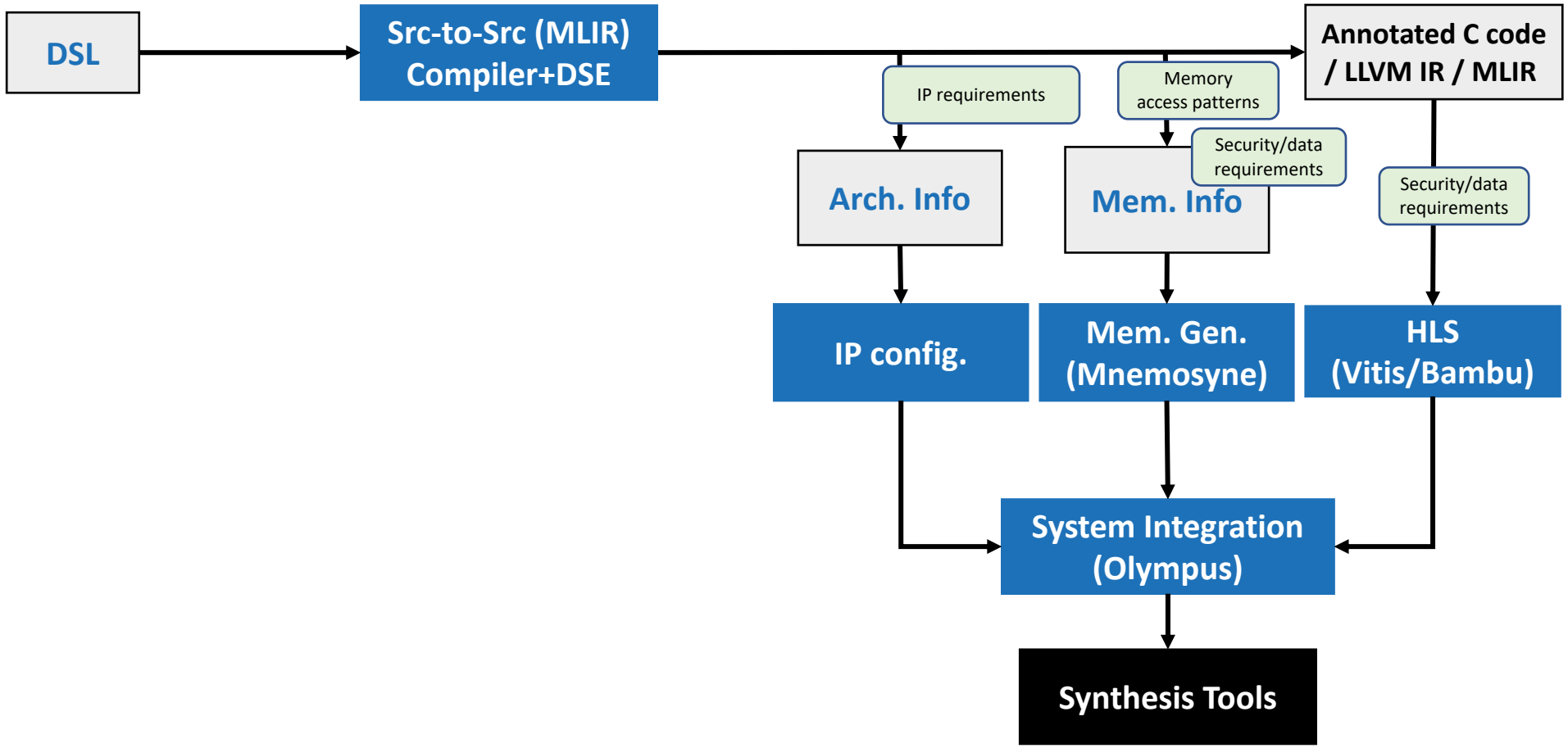
Runtime API

Autotuning API

Hiding communication latency (e.g., prefetching)

Seamless execution when varying the system configuration (resources, nodes, data, etc.)

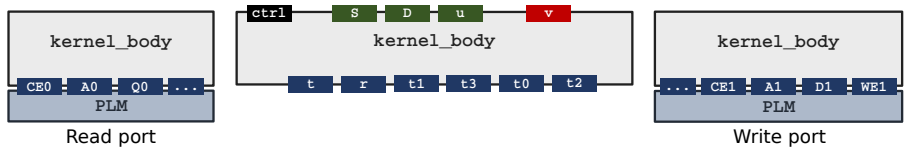
Hardware Compilation Flow

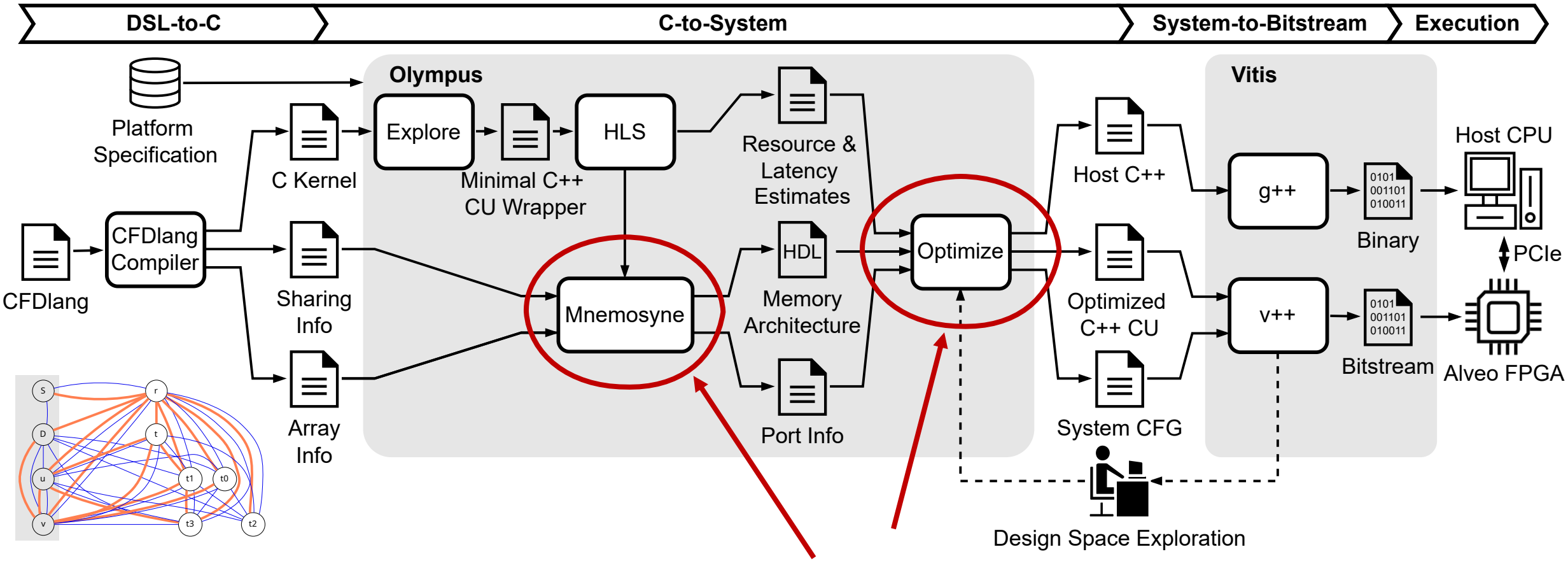


From DSL to Bitstream

```
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]]
```

```
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])
```





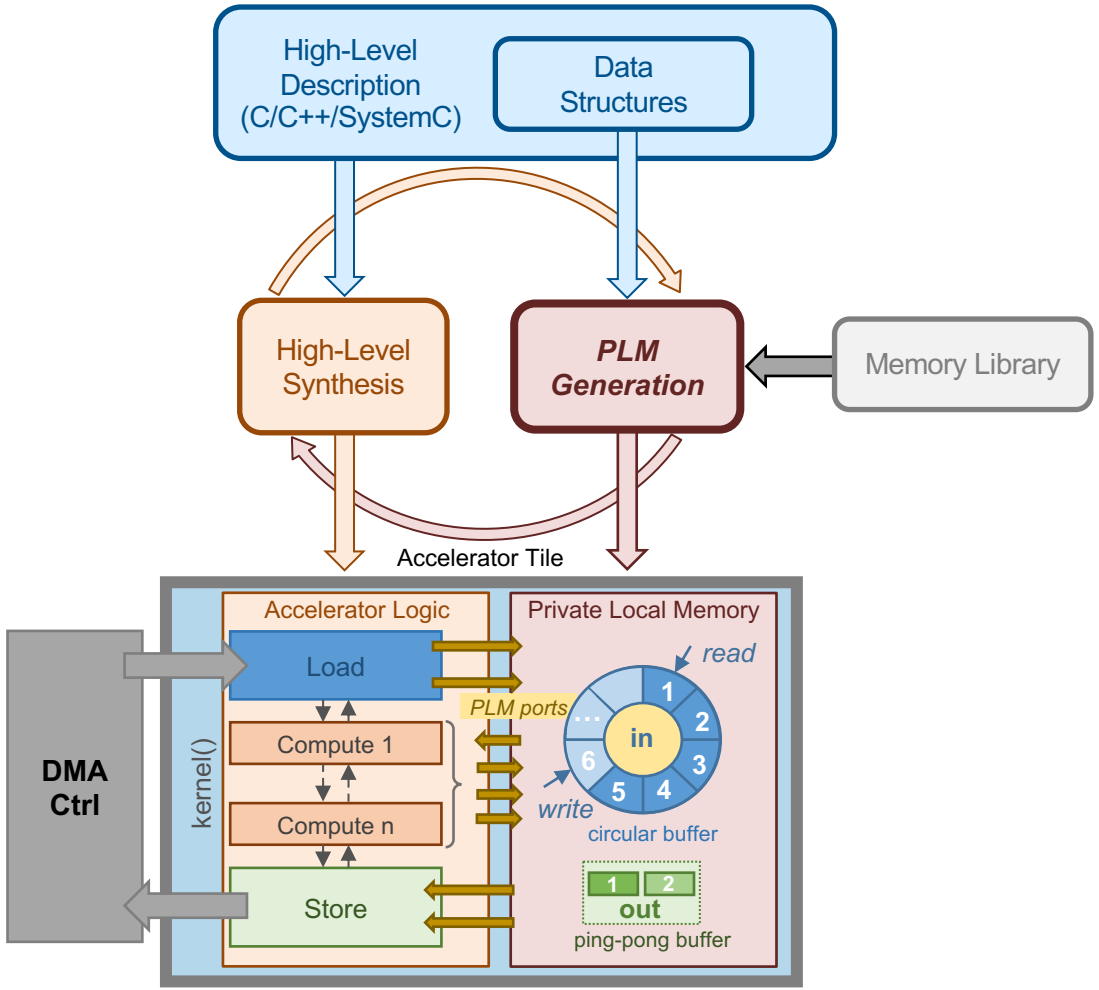
PLM Customization for Heterogeneous SoCs

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

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

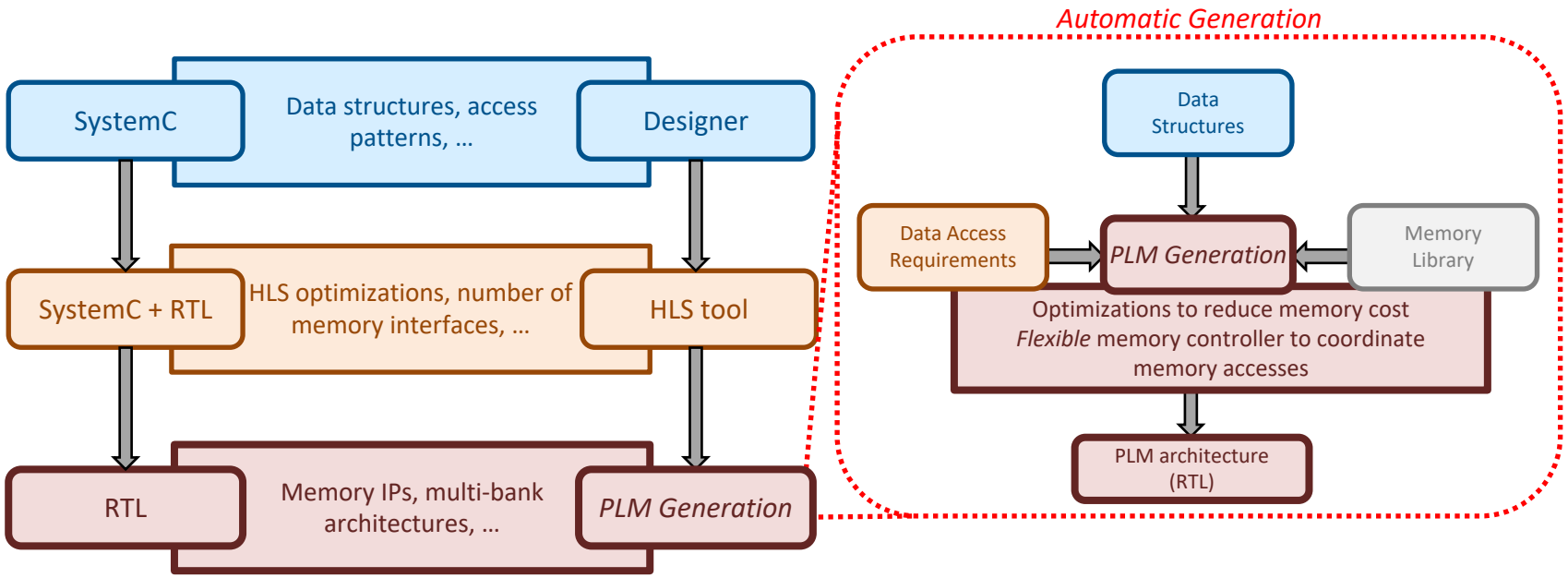
Generation of **specialized PLMs**

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



PLM Customization

System-level methodology for **PLM customization**



Performance optimization: *HLS* defines how the accelerator logic accesses the data structures (e.g. number of parallel accesses)

Cost optimization: *PLM Customization* defines the best PLM microarchitecture to achieve the desired performance (e.g. number of banks, data allocation)

Reuse What is not Used

Generally we can use one **PLM unit** (eventually composed of many banks) for each data structure

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)”

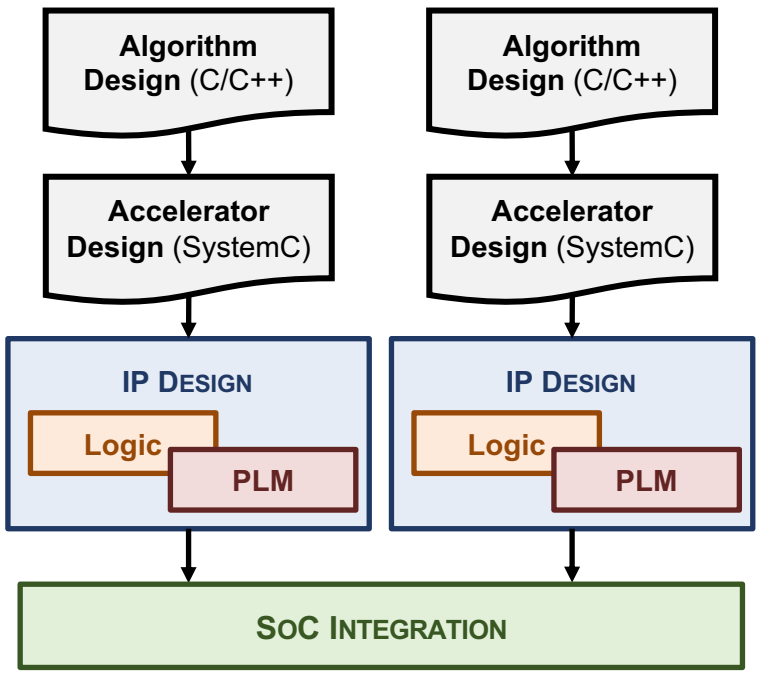
A common case: accelerators never executed at the same time

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

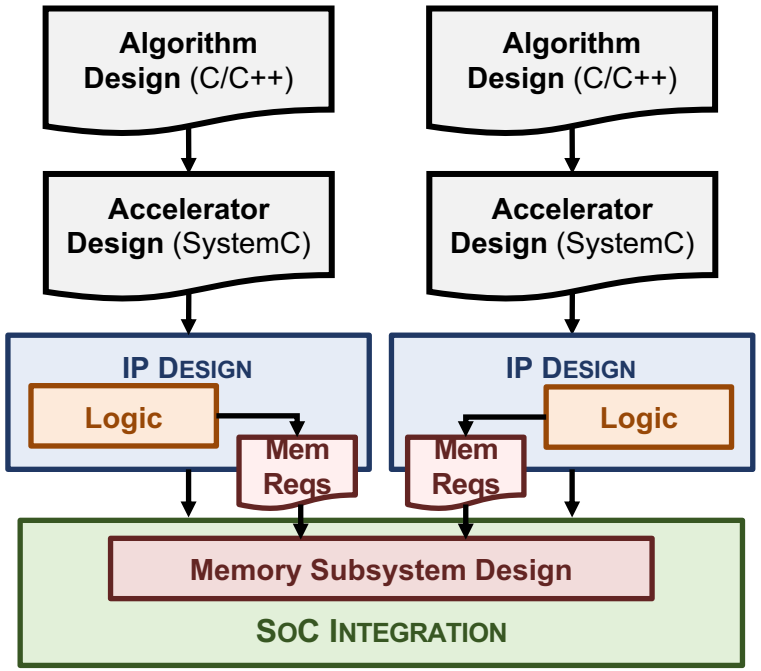
Optimization only at the System-Level

Accelerator(s) memory subsystem is defined during SoC integration

- Possibility for more optimizations

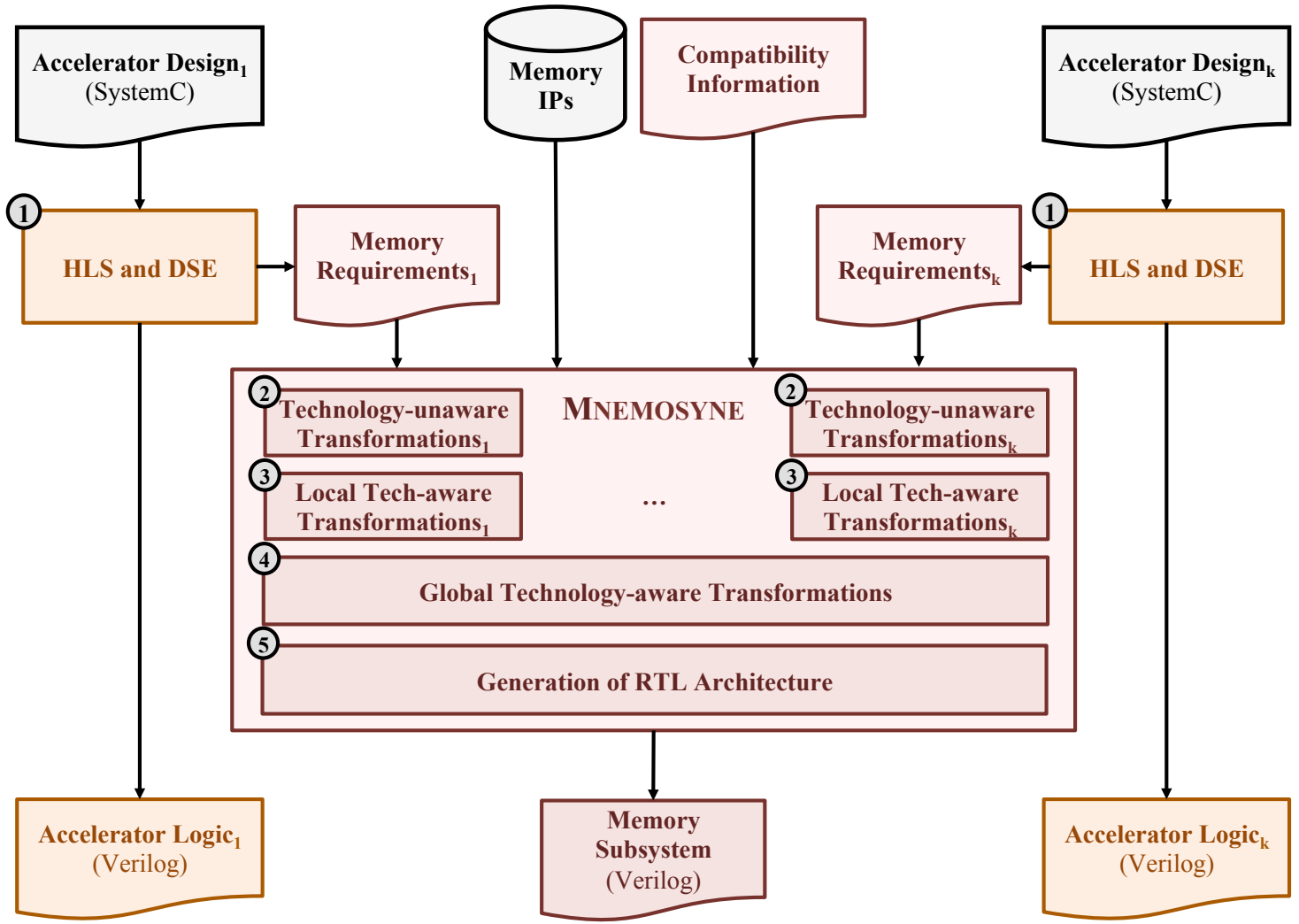


Component-based Approach



System-Level Approach

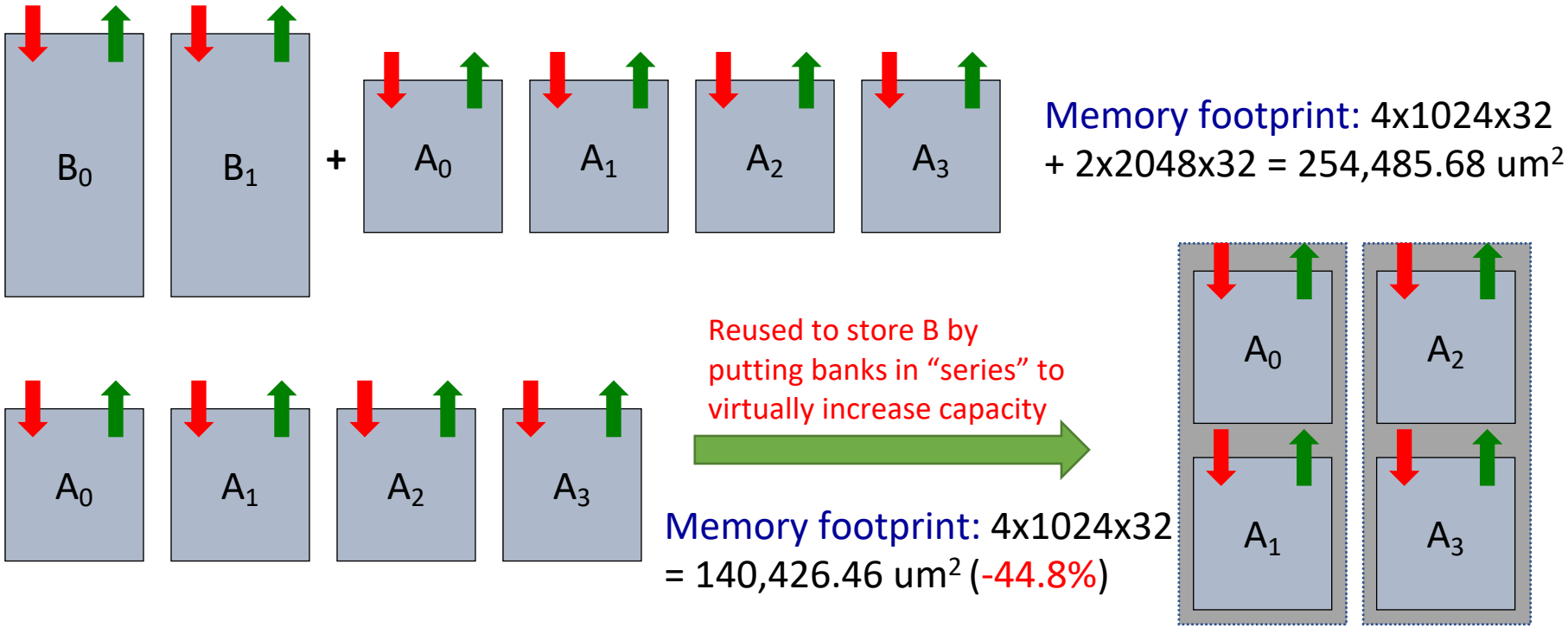
PLM Optimization for Multiple Accelerators



Address-Space Compatibility

Let us assume to have the two following data structures that are never *alive* at the same time

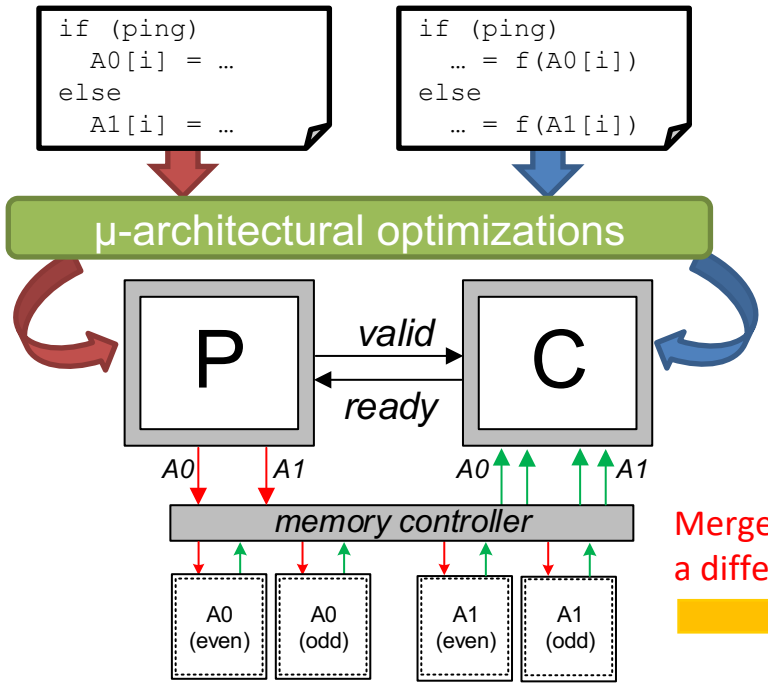
- A[1024] with *data duplication* over 4 parallel banks
- B[4096] with *data distribution* over 2 parallel banks



Memory-Interface Compatibility

A classical example is the ping-pong buffer (two 2048x16 arrays – A0/A1)

- When process P writes A0 (A1), it never writes A1 (A0)
- When process C reads from A0 (A1), it never reads from A1 (A0)

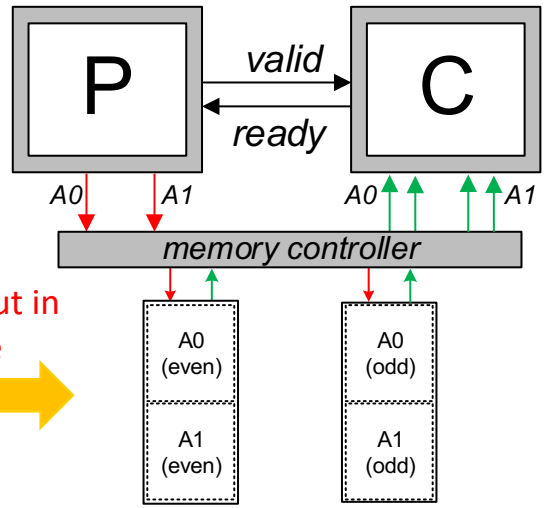


Memory footprint: 4x1024x32 = 140,426 um^2

Merged in the same IP, but in a different memory space

Memory footprint: 2x2048x32 = 114,059.2 um^2

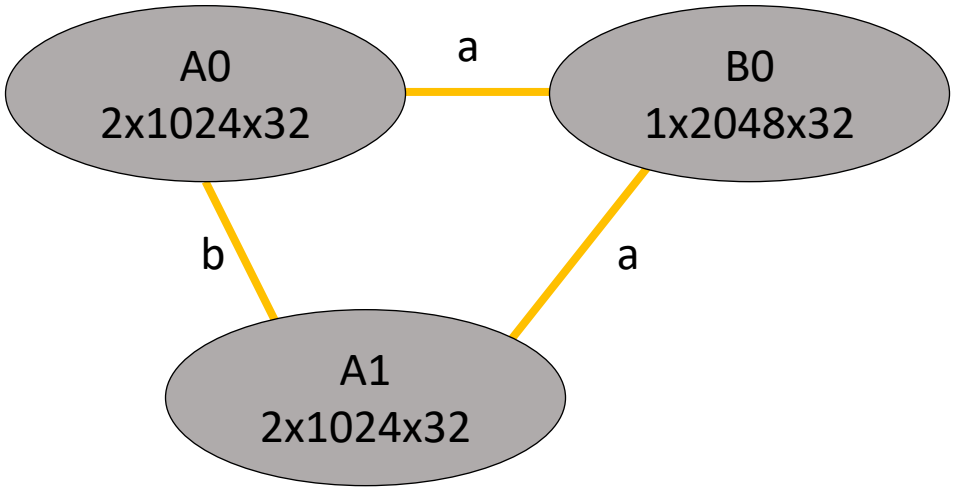
Area reduced by 18% without any performance overhead!



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

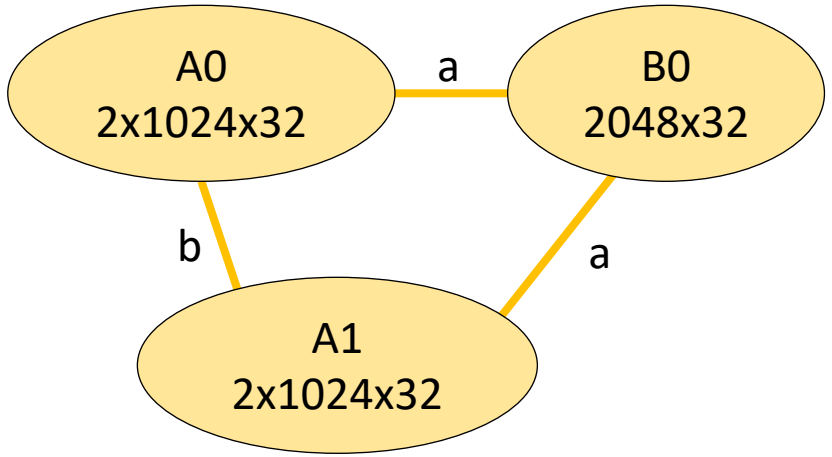


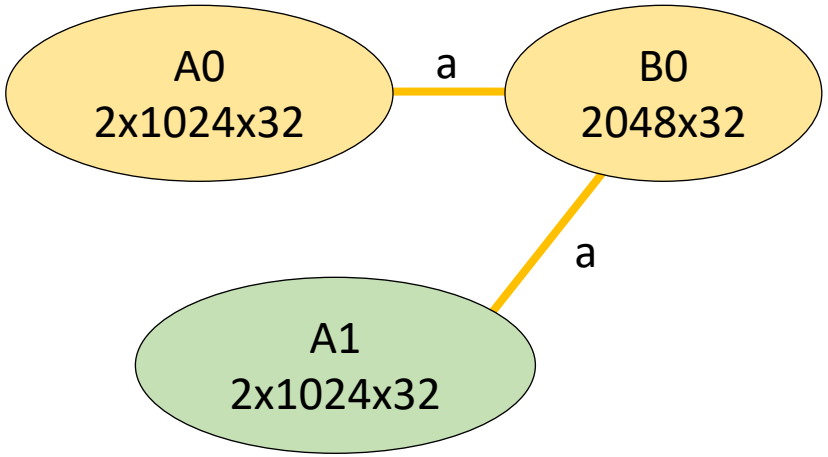
a) **Address-space compatibility**: the data structures are compatible and can use the same memory IPs
b) **Memory-interface compatibility**: the ports are never accessed at the same time and the data structures can stay in the same memory IP

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



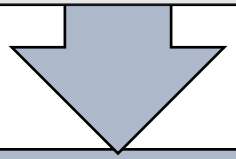


We need two distinct configurations!
{A0,B0} and {A1} or {A1,B0} and {A0}?

How to Determine the Memory Subsystem

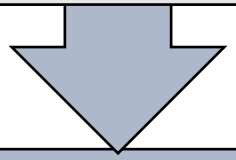
Clique Enumeration

To define the list of admissible cliques in the MCG



Clique Characterization

To determine the memory architecture of all cliques and their memory cost



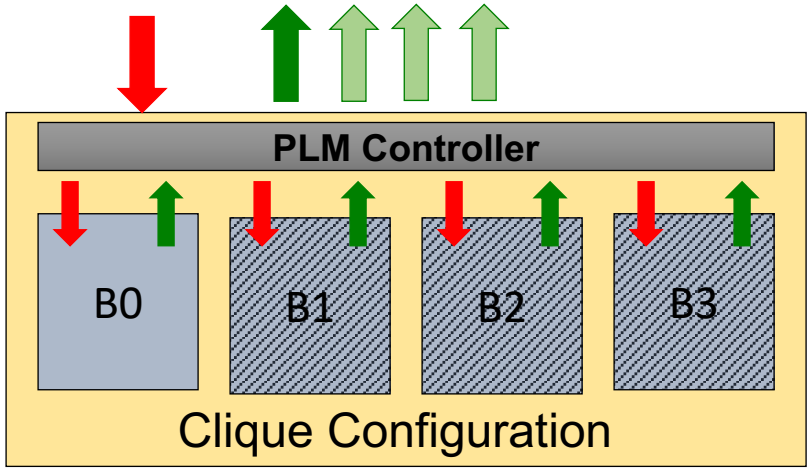
Memory Cost Minimization

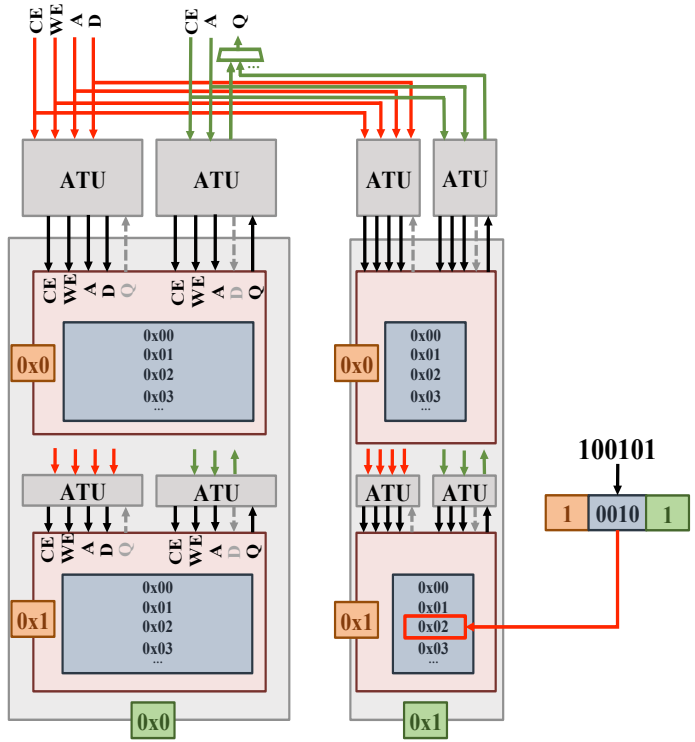
To determine how to partition the MCG such that the total memory cost is minimized

PLM Controller Generation

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

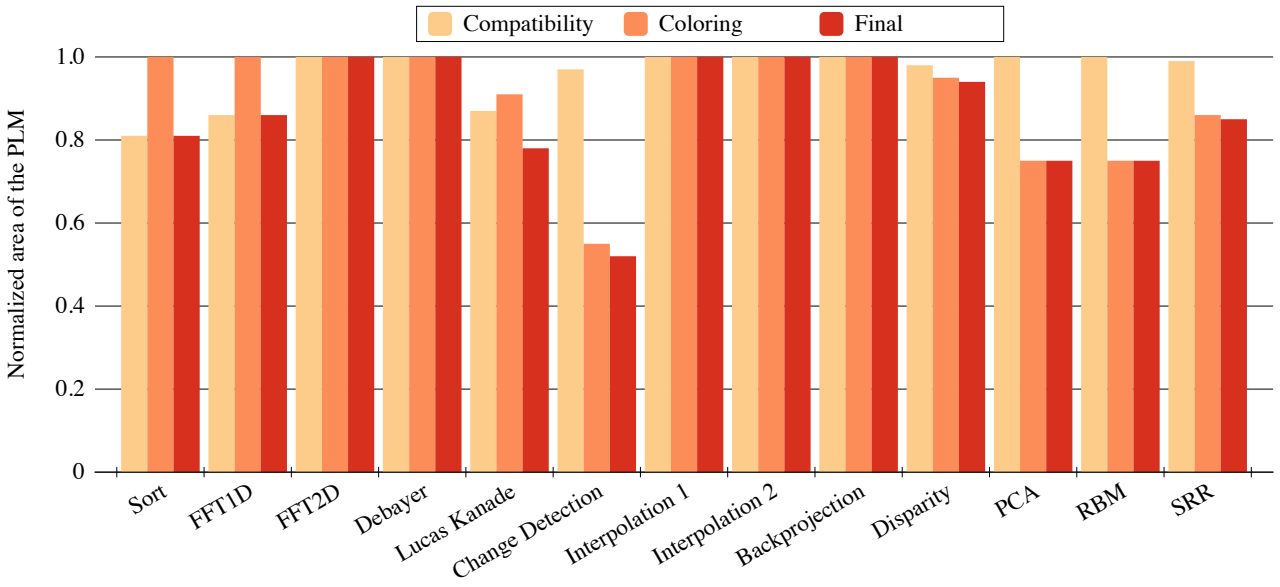
- Accelerator logic is not aware of the actual memory organization
- Array offsets need to be translated into proper memory addresses





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

Impact of Optimizations

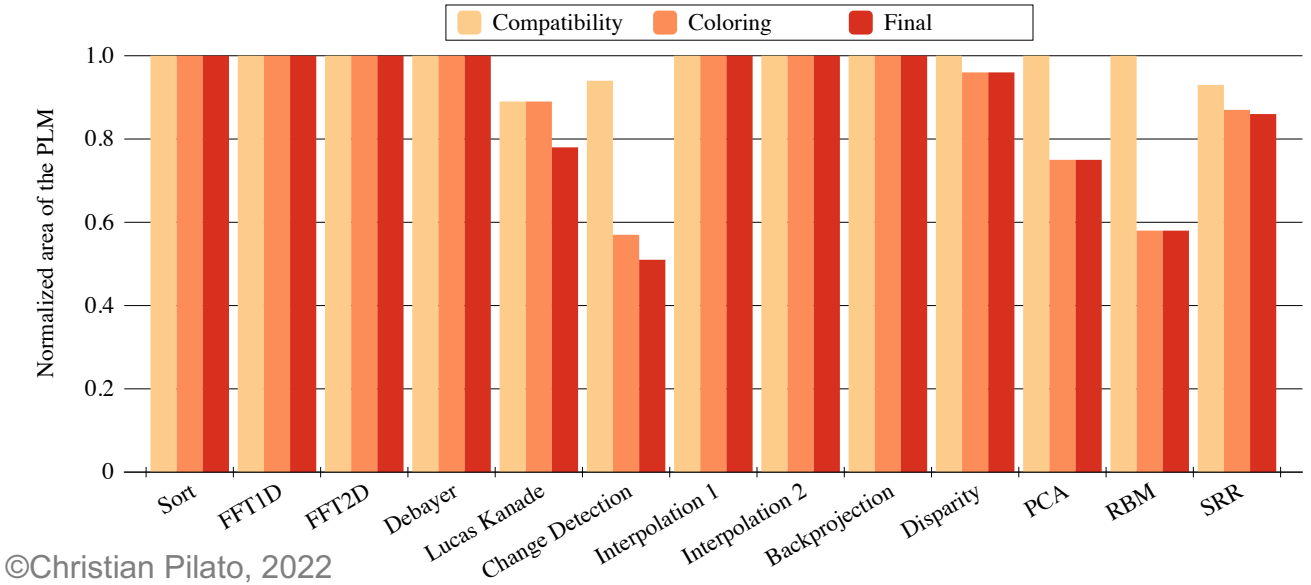


Industrial 32nm CMOS technology

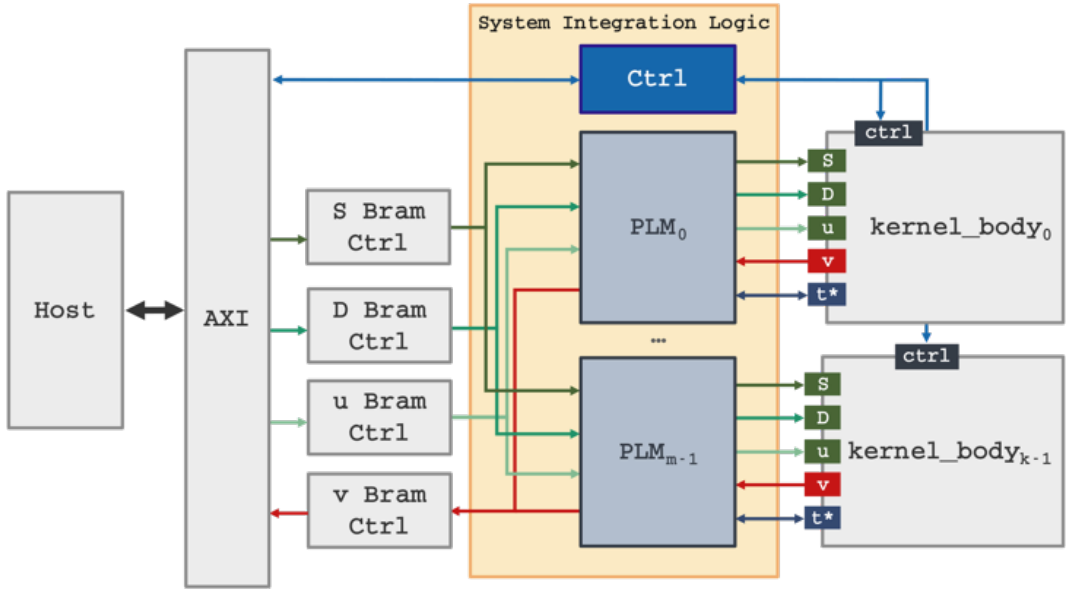
- Memory library with 18 SRAMs

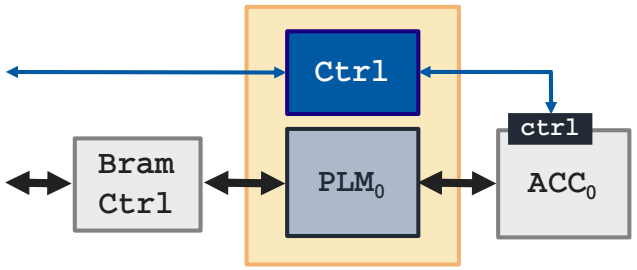
Xilinx Virtex-7 FPGA

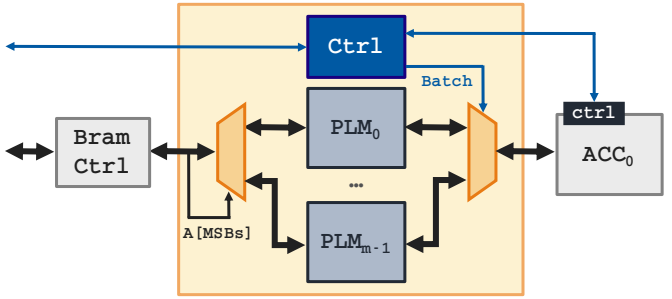
- Memory library with 6 BRAM configurations

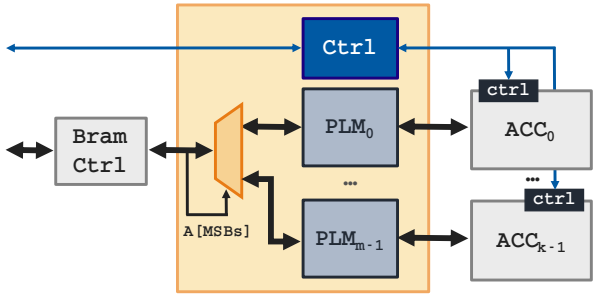


Creation of Parallel Architectures



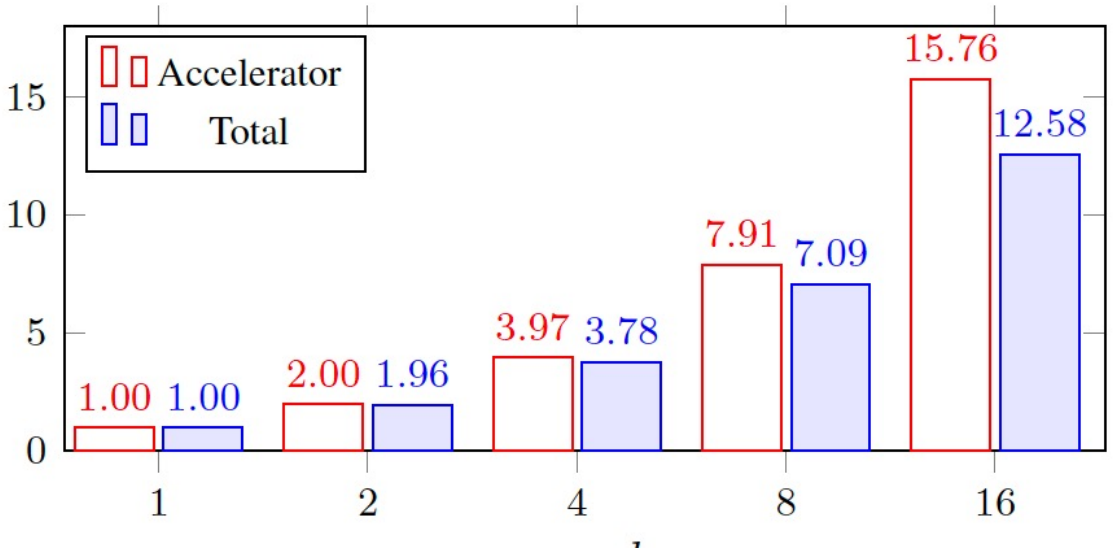


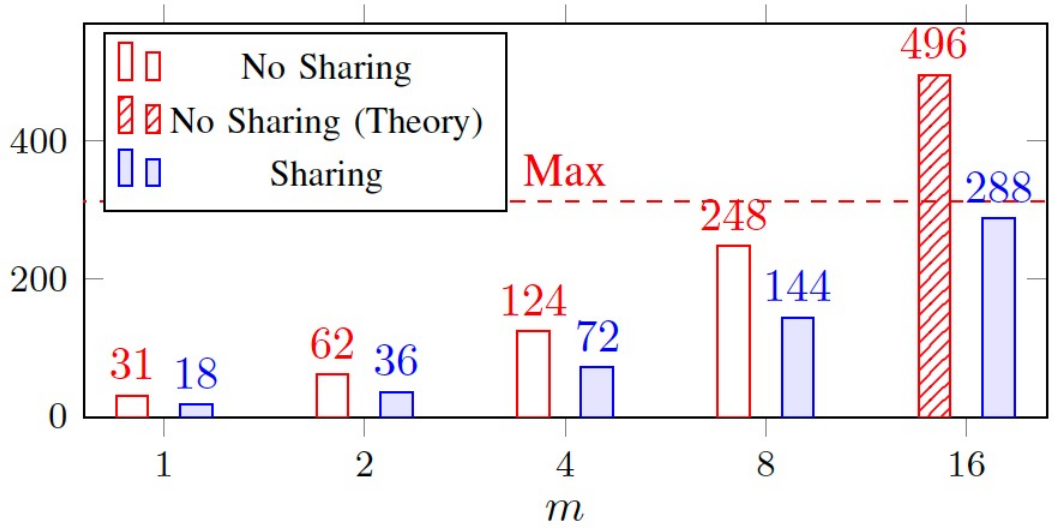




Preliminary Evaluation

- Xilinx Zynq UltraScale+ MPSoC ZCU106 board
  - CFD simulation of 50,000 elements
- Memory sharing allows us to fit more kernels





	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%)

Next Step: System-Level DSL

DSL for representing the kernel

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

Moving to a system-level representation

- Simple example for a massively parallel architecture:

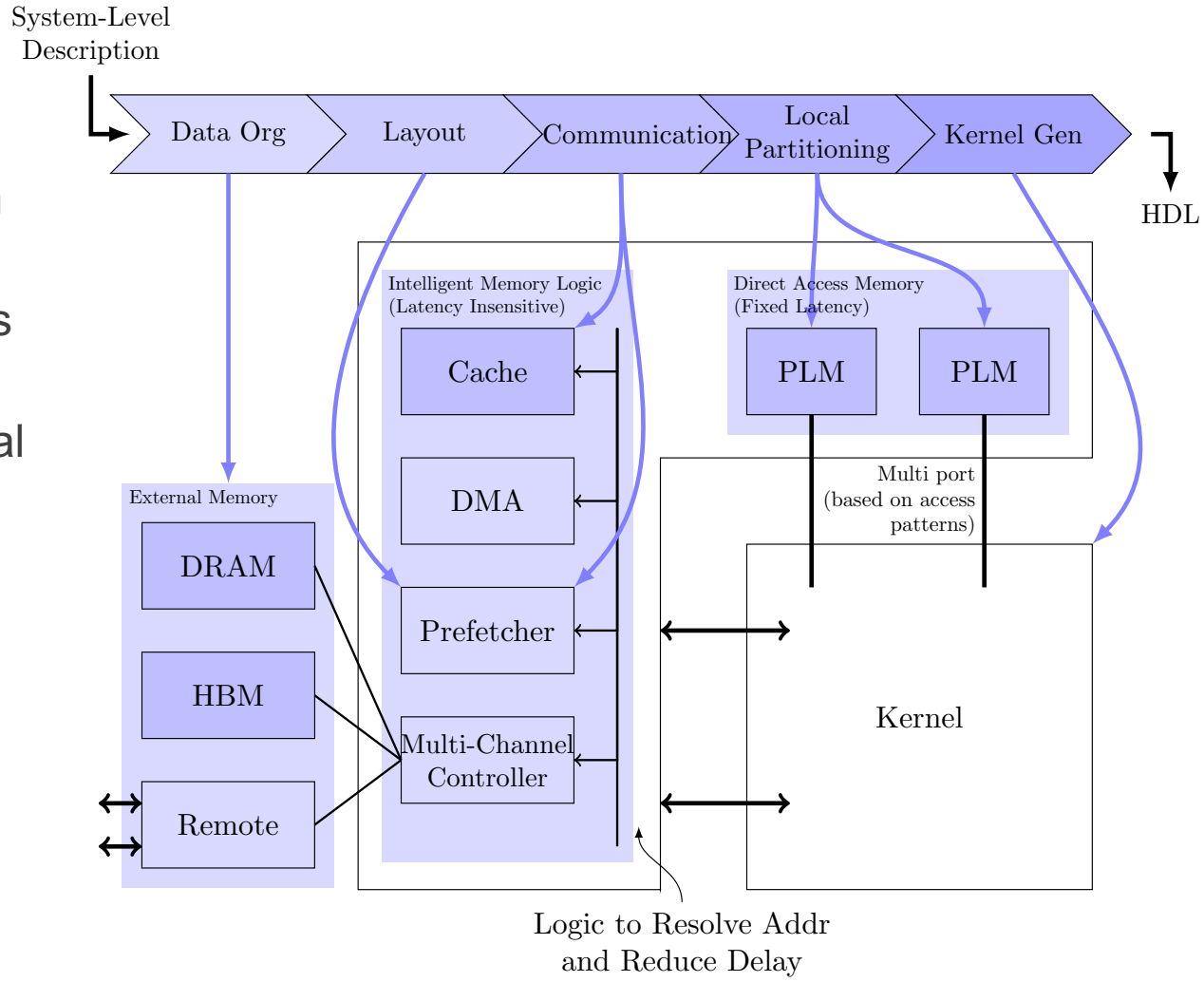
LOOP ~ KERNEL(S, D, u, v)

Possibility to decide the memory layout and configure DMA/prefetchers based on the target architecture/platform

Next Step: Let's Put Memory First

We are building a **compilation flow** based on **LLVM MLIR** for **automatic specialization**:

- **MLIR Input** – From DSL descriptions of the system functionality
- **Data Organization** – Determine which data resides off chip (also based on user/compiler annotations)
- **Layout** – Reorganize communication to exploit local memories (cache/PLM)
- **Communication** – Configure prefetcher to hide transfer latency
- **Local Partitioning** – Determine multi-bank PLM architecture (Mnemosyne[1])
- **HLS** – Generate computation part (interfacing with existing HLS tools, e.g., open-source Vitis HLS frontend)
- **HDL Output** – Automated code generation and system-level integration based on the target platform



S. Soldavini and C. Pilato. "Compiler Infrastructure for Specializing Domain-Specific Memory Templates" LATTE'21

Olympus – Automated System-Level Integration

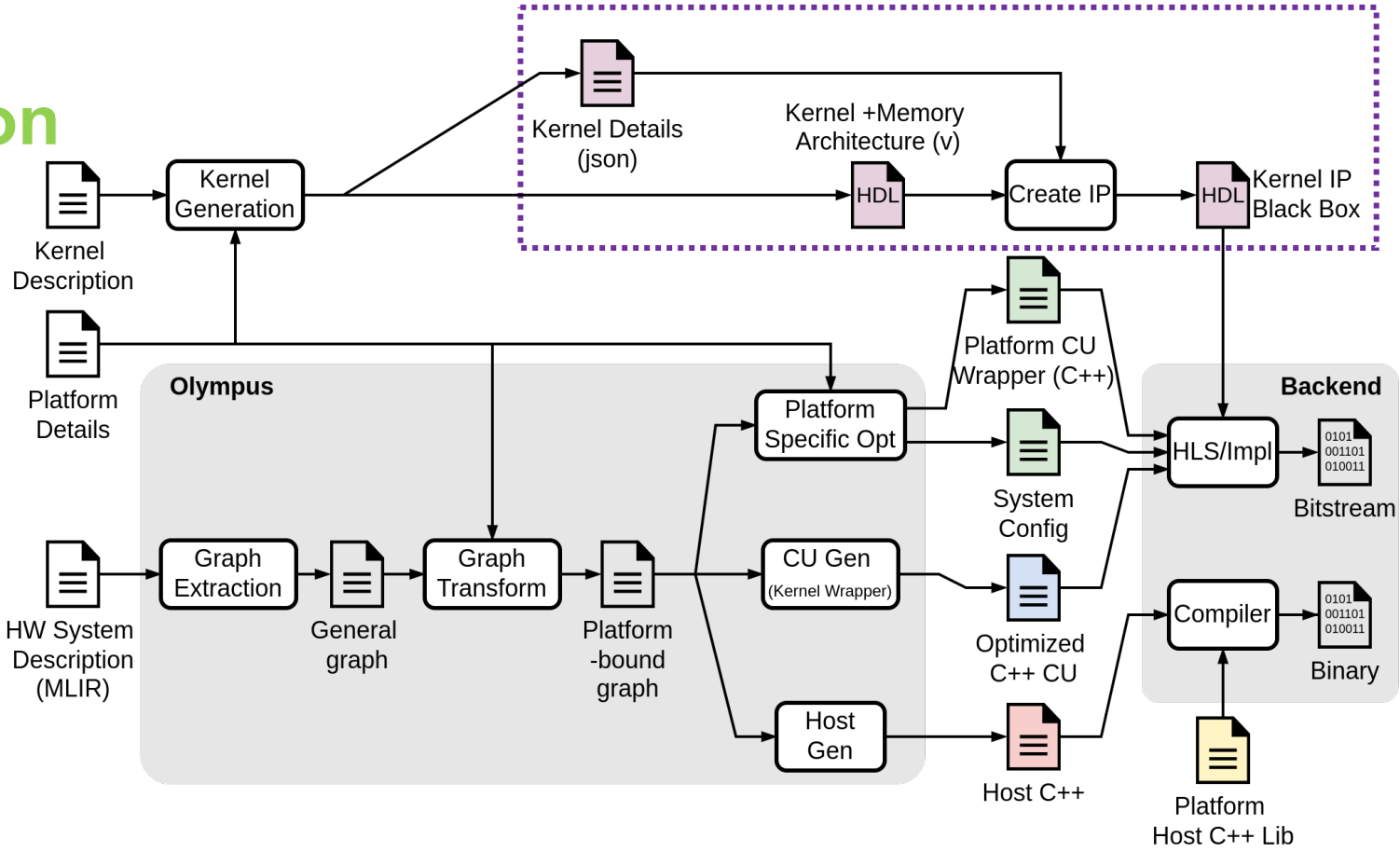
We are developing a complete hardware architecture generation flow based on **MLIR description** of the **system functionality**

Platform-specific description

- HBM-based Xilinx Alveo
- IBM CloudFPGA
- …

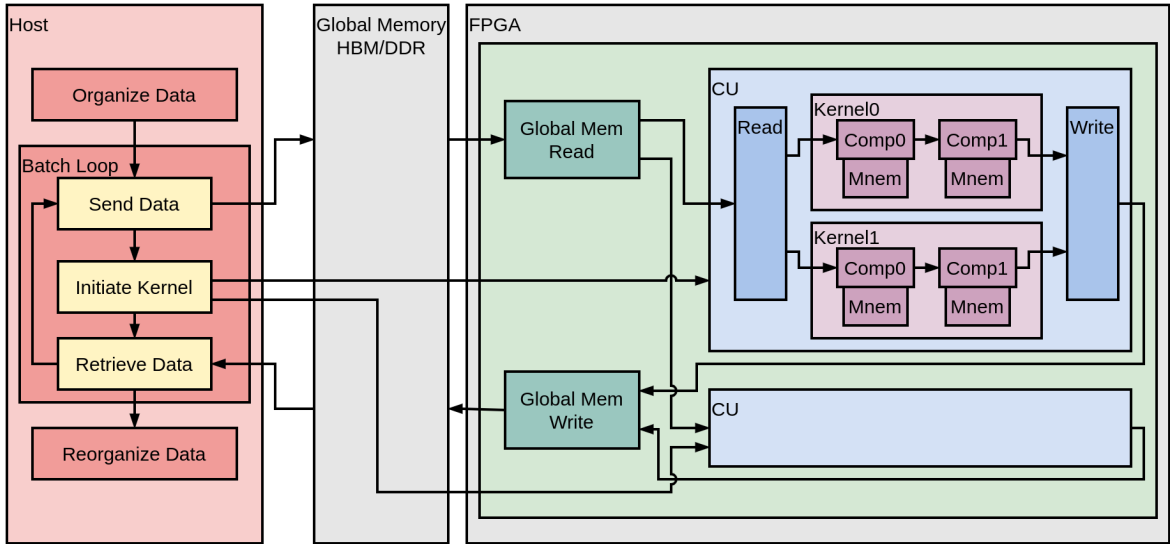
Host code generation

- Based on **platform libraries** to be developed for the specific target



Olympus – System generation flow

- Determines the **system-level architectures** based on:
  - **Algorithm parallelism**
  - Characteristics of the **target platform(s)**
  - Interfaces of the modules (**HLS tools**)

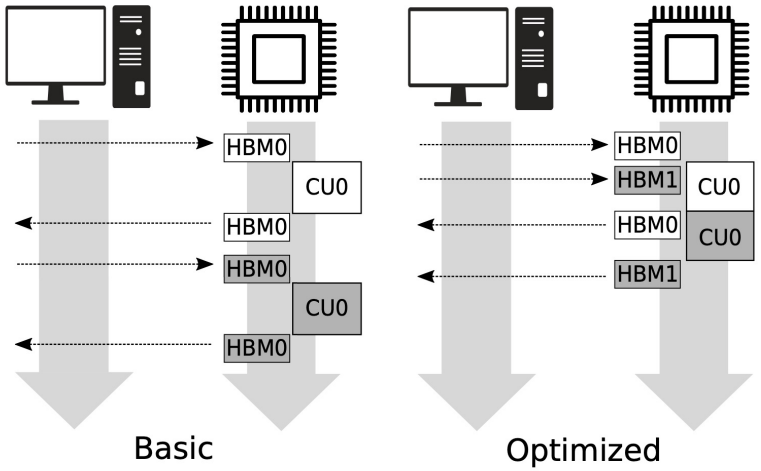


- Produces
  - **Synthesizable C++ code** that includes:
    - **Accelerators** and **PLM** generated with HLS
    - **Communication modules** to match interfaces
      - *Standard AXI interfaces to the system (either cloudFPGA SHELL or HBM channels)*
      - May include **"intelligent" policies** to coordinate data transfers
  - **System configuration file** to create the overall architecture
    - Support for multiple computing units executing in parallel
    - *Interfacing with Xilinx HLS and synthesis tools*

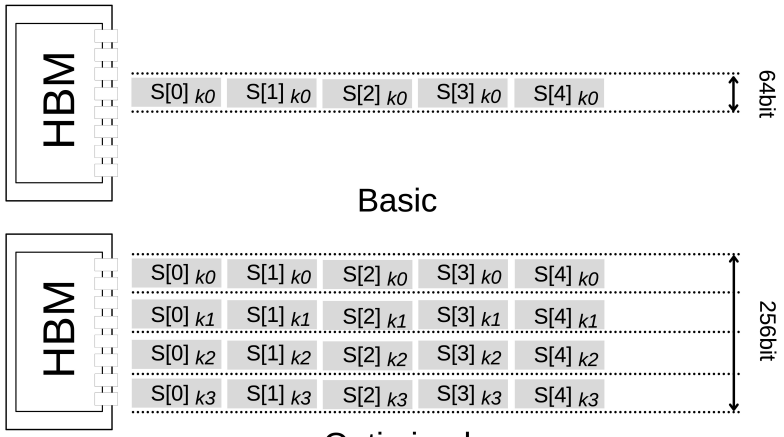
From MLIR to System Architecture

Automatic integration of memory optimizations for **high-performance data transfers**, such as:

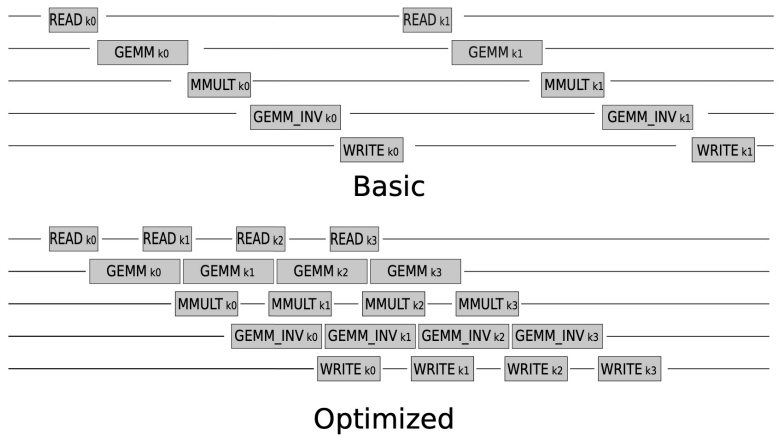
- **Double buffering** to hide latency of host-FPGA data transfers
- **Bus optimization** (and data interleaving) for maximizing bandwidth (e.g., 256-bit AXI channels) – algorithms for efficient data layout on the bus
- **Dataflow execution model** to enable kernel pipelining – automatic (pre-HLS) code transformations



Double buffering

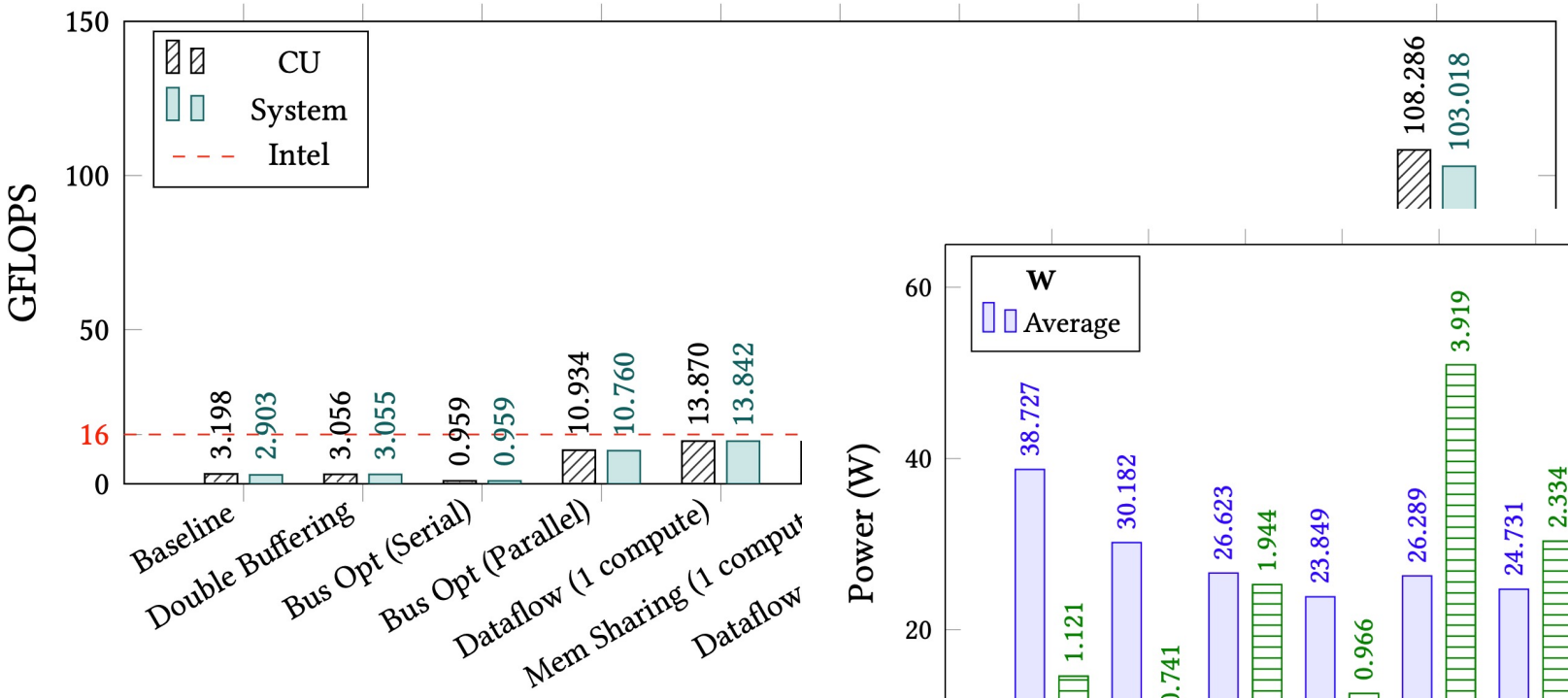


Bus optimization

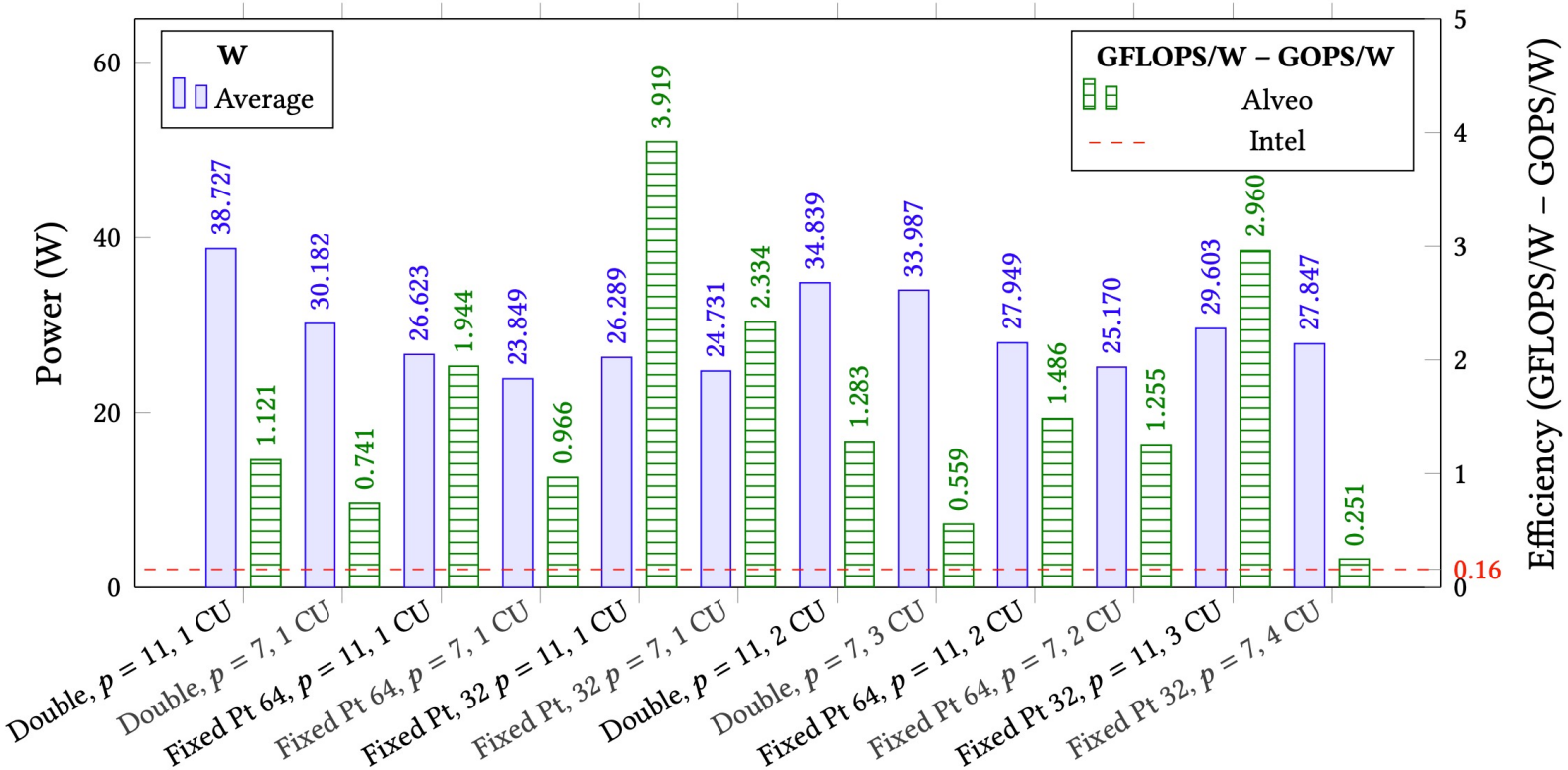


Dataflow

Results on HBM FPGA



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



Results are 6x better than HPC ones (~25x more energy efficient)

Possibility of integrating **custom data formats** and configure memories and data transfers accordingly

Conclusions

Data management optimizations are becoming the key element for the creation of **efficient FPGA architectures**

HLS is now used not only to create accelerator kernels but also to generate the **system-level architecture**

- **Portable solutions** across multiple target platforms

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

- Necessary to match **application requirements** and **technology characteristics**

Work done in collaboration with Stephanie Soldavini (Politecnico di Milano), Jeronimo Castrillon (TU Dresden), Karl F. A. Friebel (TU Dresden), and Gerald Hempel (TU Dresden)

Thank you!

Christian Pilato, christian.pilato@polimi.it

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957269