

FPGAs as Compute Accelerators

Enrico Calore, Andrea Miola, Giada Minghini,
Valentina Sisini, Sebastiano Fabio Schifano

INFN & University of Ferrara, Italy



- 1 Introduction
 - Technology Tracking
 - EuroEXA Project
- 2 Roofline Model
 - The FPGA Empirical Roofline (FER)
 - Theoretical Model
- 3 AI inference on FPGAs
 - Application
 - Results
- 4 Edit distance on FPGAs
 - Application
 - Results

Performance analysis and benchmarking of novel architectures

- Intel CPUs and many-core processors (e.g., Xeon and Xeon Phi);
- NVIDIA high-end and embedded GPUs (e.g., Tesla and Tegra);
- AMD GPUs (e.g., FirePro and Instinct);
- Arm CPUs (e.g., ThunderX2 and Grace);

Performance analysis and benchmarking of novel architectures

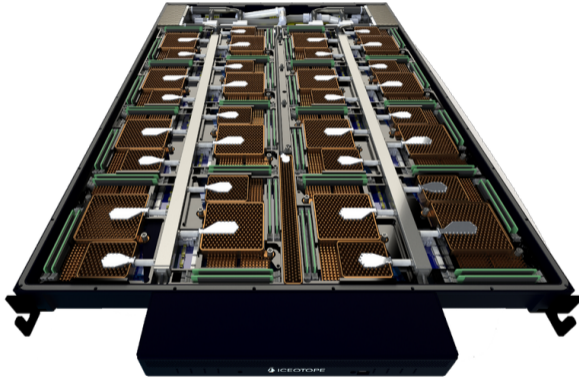
- Intel CPUs and many-core processors (e.g., Xeon and Xeon Phi);
- NVIDIA high-end and embedded GPUs (e.g., Tesla and Tegra);
- AMD GPUs (e.g., FirePro and Instinct);
- Arm CPUs (e.g., ThunderX2 and Grace);
- **AMD-Xilinx FPGAs (i.e., Alveo).**

EuroEXA: *Co-designed innovation and system for resilient exascale computing in Europe: from application to silicon*

A Co-design HPC Project featuring:

- use of FPGAs as accelerators;
- use of FPGAs to implement custom interconnects;
- co-design a balanced architecture for both compute and data-intensive applications;
- programmable using high-level languages.

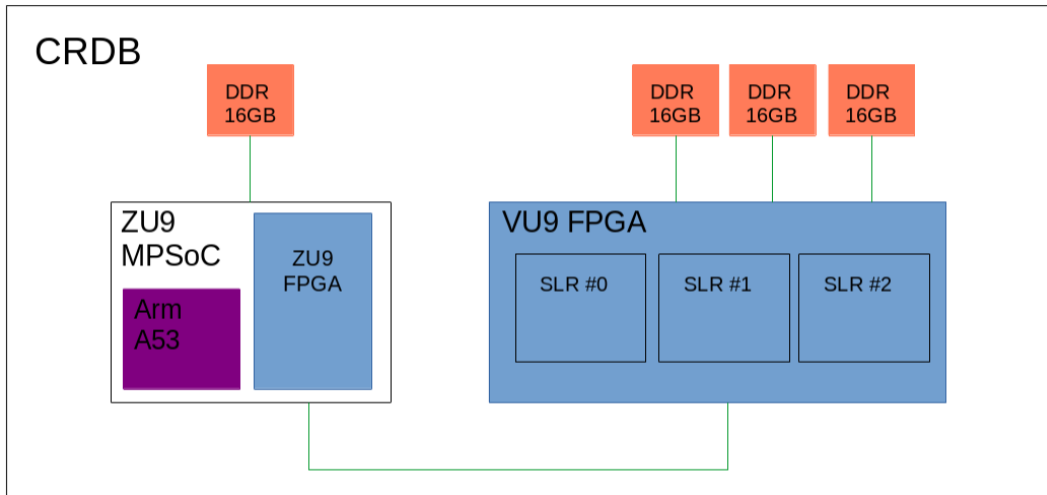
First EuroEXA Prototype at scale



EuroEXA Architecture:

- 16 CRDBs in a Blade.
- 32 EuroEXA Blades in one Rack.
- hierarchical network with hybrid topology: all-to-all at Blade level and torus for inter-Blade level.

EuroEXA Prototype Architecture

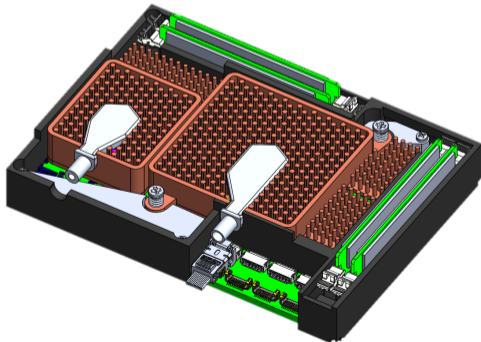


Sketch of the Single CRDB board.

EuroEXA Prototype Architecture

Co-design Recommended Daughter Board (CRDB):

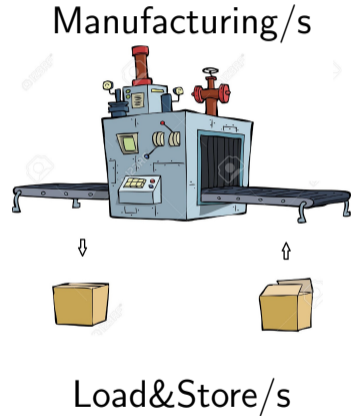
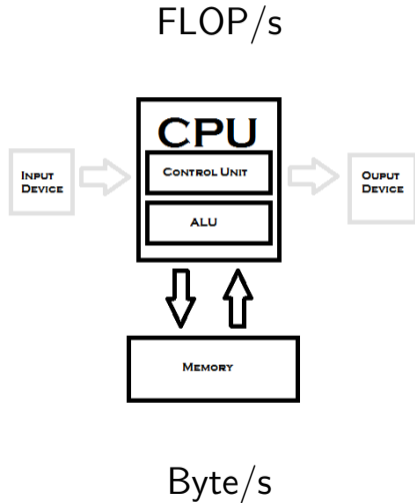
- Zinq UltraScale+ ZU9 for interconnect and compute.
- Virtex UltraScale+ VU9 as a compute accelerator.
- Liquid cooled board.



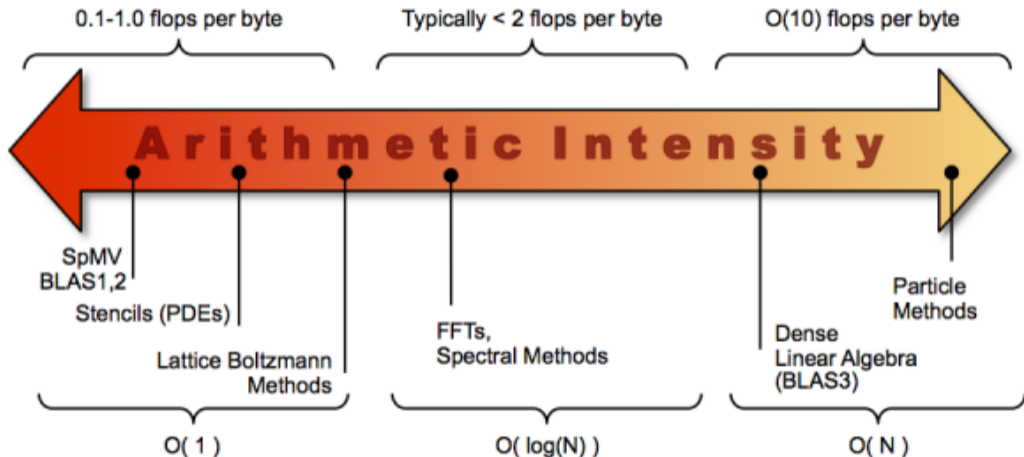
We needed to:

estimate applications
expected performance for
co-design and evaluation.

What is limiting the performance?

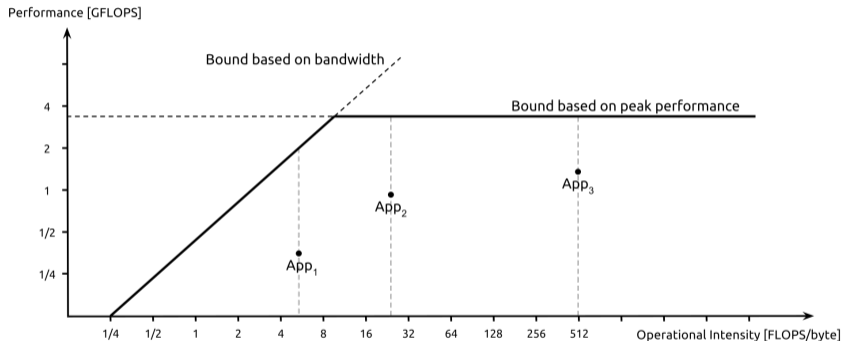


Arithmetic/computational Intensity



Roofline Model

The *Roofline Model* is used to provide performance estimates of a given compute kernel running on a given architecture.

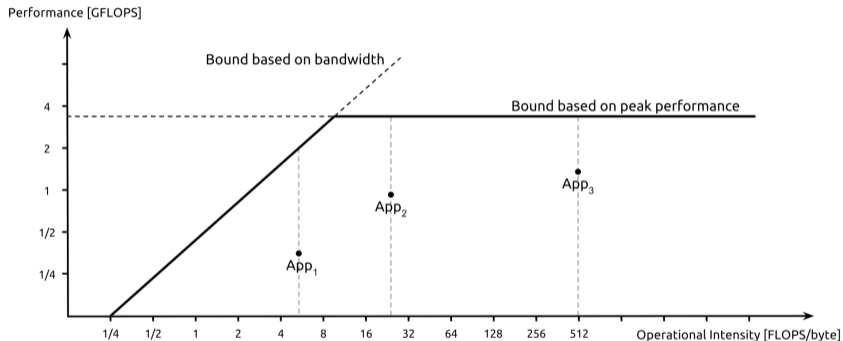


Williams, S., Waterman, A., & Patterson, D. (2009) "Roofline: an insightful visual performance model for multicore architectures" *Communications of the ACM*, 52(4), 65-76.

Peak Bandwidth and Performance could be theoretical ones, or empirically measured...

Roofline Model

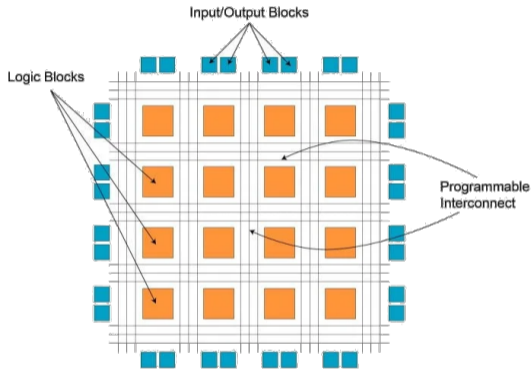
The *Roofline Model* is used to provide performance estimates of a given compute kernel running on a given architecture.



Williams, S., Waterman, A., & Patterson, D. (2009) "Roofline: an insightful visual performance model for multicore architectures" *Communications of the ACM*, 52(4), 65-76.

Peak Bandwidth and Performance could be theoretical ones, or empirically measured...
Not trivial on FPGAs.

Field Programmable Gate Arrays (FPGAs)



- Bunch of hardware resources to be configured
- Some generic resources and some more specialized ones
- Same operation can be implemented in different ways
- Max clock frequency depends on the path lengths, on the used resources, etc.

Generic overview of FPGA Architectures

The FPGA Empirical Roofline (FER)

Empirical Roofline Tool (ERT)

Berkeley Lab

- Empirically find the max FLOPs and Bandwidth
- Kernel with tunable arithmetic complexity
- Targeting CPUs/GPUs
(OpenCL kernel can target also FPGAs)

<https://crd.lbl.gov/departments/computer-science/PAR/research/roofline/software/ert/>

FPGA Empirical Roofline (FER)

INFN & Univ. of Ferrara

- Based on the same principles of ERT
- Written using HLS directives
- Targeting FPGA devices

E. Calore and S. F. Schifano, "FER: A Benchmark for the Roofline Analysis of FPGA Based HPC Accelerators" in IEEE Access, vol. 10, pp. 94220-94234, 2022. doi: 10.1109/ACCESS.2022.3203566

<https://baltig.infn.it/EuroEXA/FER>

FPGA performance model

To estimate the peak performance C of an FPGA in terms of op/s , we can assume that to implement a hardware core performing op , are required R_{op} hardware resources. If an FPGA contains R_{av} of these resources, the maximum number of implementable hardware cores H_c is:

$$H_c = \frac{R_{av}}{R_{op}}$$

If each core operates at a maximum clock frequency f_{op} , and starts a new operation every clock cycle, the theoretical performance C is:

$$\begin{aligned} C &= f_{op} \times H_c \\ &= f_{op} \left(\frac{R_{av}}{R_{op}} \right) \end{aligned}$$

FPGA performance model

Given that on FPGAs are commonly available Ri_{av} different i types of resources, one of them will limit the number of cores:

$$C = f_{op} \times \min_i \left(\frac{Ri_{av}}{Ri_{op}} \right)$$

Such theoretical models, have already been used by FPGA manufacturers to publicize peak performance, but actual applications could be able to reach much lower values.

Intel says:

“For FPGAs lacking hard floating-point circuits, using the vendor-calculated theoretical GFLOPS numbers is quite unreliable. Any FPGA floating-point claims based on a logic implementation at over 500 GFLOPS should be viewed with a high level of skepticism. In this case, a representative benchmark design implementation is essential to make a comparative judgment.”

Intel White Paper: *Understanding Peak Floating-Point Performance Claims* (2017)

Too optimistic theoretical estimations

It is actually too optimistic:

- to assume to be able to exploit all of the available resources of one specific type;
- to assume to reach the maximum clock frequency declared for a single *op* core, when a large fraction of resources is used.

Need for empirical parameters: f_{imp} and u_{Ri}

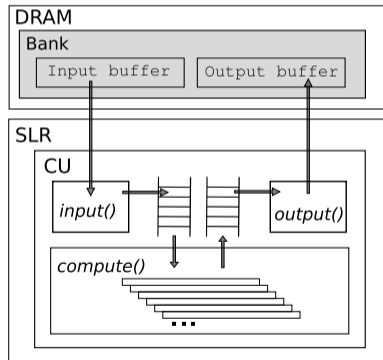
$$C = f_{imp} \times \min_i \left(\frac{Ri_{av}}{Ri_{op}} \times u_{Ri} \right), \quad u_{Ri} < 1$$

Main FER C/HLS kernel function

Implements a task level pipeline (dataflow):

reading elements from an input array, applying to each a given op , for O_e times, and storing the result in an output array.

```
1 void fer( const data_v *input,
2           data_v *output ) {
3
4     hls::stream<data_v> inFifo;
5     hls::stream<data_v> outFifo;
6
7     #pragma HLS dataflow
8
9     readInput(input, inFifo);
10    compute(inFifo, outFifo);
11    writeOutput(output, outFifo);
12 }
```



FER *compute()* function

FER tuning knobs:

$$C = f \times \frac{V \times O_e}{II_c}$$

II_c : Initiation Interval

V : SIMD vector width

O_e : Ops per element

Hardware limit:

$$\frac{V \times O_e}{II_c} < \min_i \left(\frac{Ri_{av}}{Ri_{op}} \times uRi \right)$$

```
1 void compute(hls::stream<data_v> &inFifo,
2             hls::stream<data_v> &outFifo){
3
4     for (i = 0; i < DIM; i++) {
5         #pragma HLS pipeline II=IIc
6
7         data_v in = inFifo.read();
8
9         for (e = 0; e < V; e++) {
10            #pragma HLS unroll
11            data_t elem = in.elem[e];
12            for (o = 0; o < Oe; o++) {
13                #pragma HLS unroll
14                elem = op(elem);
15            }
16            out.elem[v] = elem;
17        }
18
19        outFifo.write(out);
20
21    }
```

Pipeline and Unroll

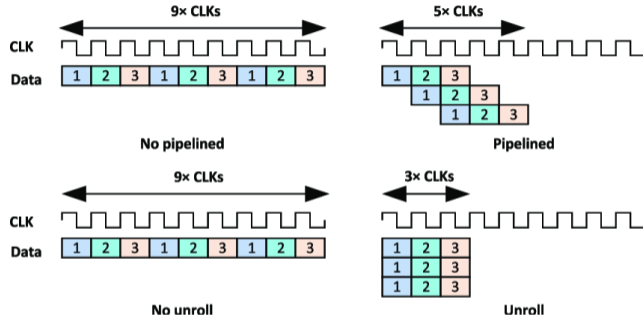


Image credits:

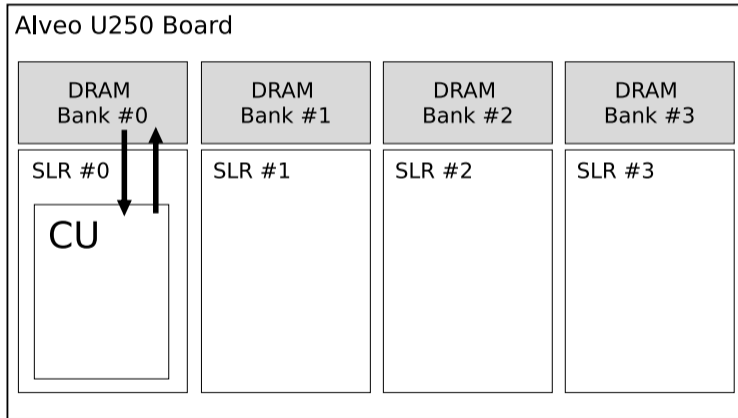
Yang, Xu & Zhuang, Chen & Feng, Wenquan & Yang, Zhe & Wang, Qiang. (2023). FPGA Implementation of a Deep Learning Acceleration Core Architecture for Image Target Detection. Applied Sciences. 13. 4144. 10.3390/app13074144.



Xilinx Alveo U250 Data Center Accelerator Card

Results for a Xilinx Alveo U250

We use as *op* a double-precision floating-point FMA, to allow for cross-architectural comparison, with other HPC processors.

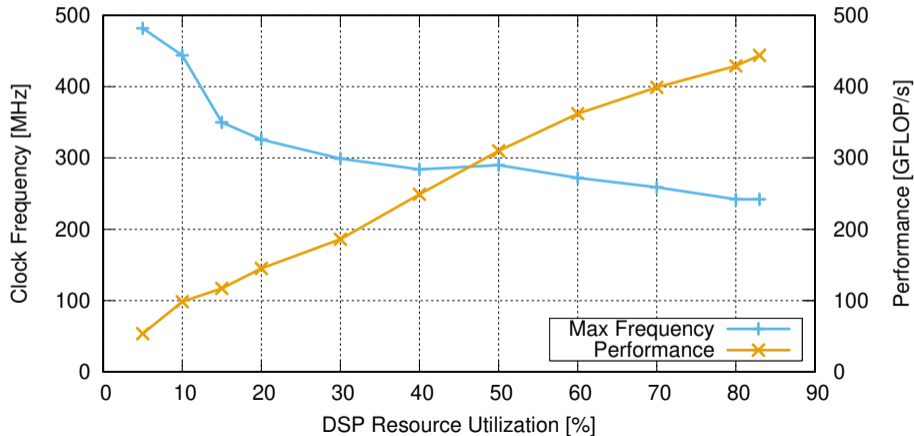


The theoretical performance of this FPGA should be:

$$\begin{aligned} C &= 694\text{MHz} \times \min \left(\frac{1.380 \cdot 10^6}{616 + 172} \text{LUT}, \frac{11508}{8 + 3} \text{DSP} \right) \\ &= 726 \cdot 10^9 \text{ FMA/s} \\ &= 1.45 \text{ TFLOP/s} \end{aligned}$$

Max empirically reachable f_{imp} and C

Synthesized and run FER for different H_c , keeping the arithmetic intensity in the *compute-bound* region.



A more realistic estimation for Alveos

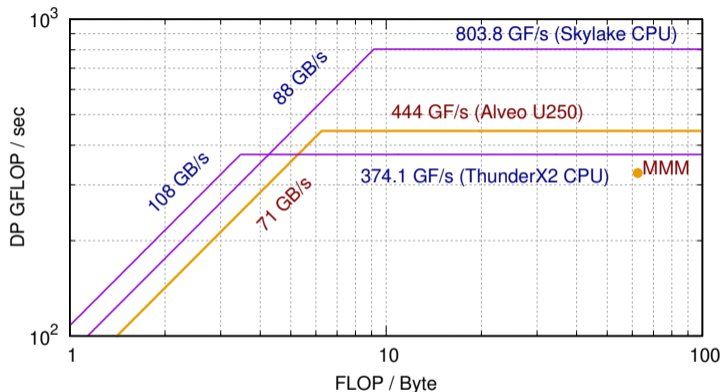
In the Xilinx documentation realistic f_{imp} and u_{Ri} values to be used for performance estimation, could be selected as:

- default clock frequency of 300MHz (provided in the *Alveo Platforms* documentation);
- suggested maximum resources utilization (published in the Vitis Unified Software Platform Documentation as “Timing closure considerations”): i.e., 70% for LUTs and 80% for DSPs.

These would give an estimated performance C of:

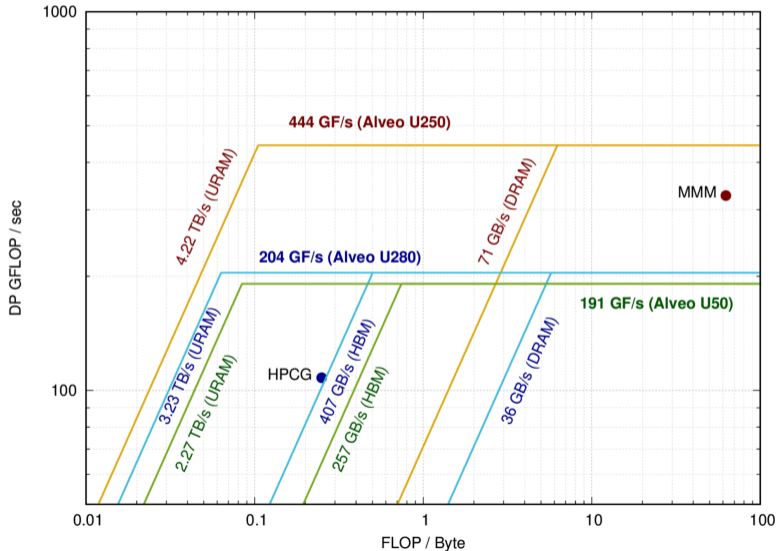
$$\begin{aligned} C &= 300 \times \min \left(\frac{1.380 \cdot 10^6}{616 + 172} \text{LUT} \times 0.7, \frac{11508}{8 + 3} \text{DSP} \times 0.8 \right) \\ &= 251 \cdot 10^9 \text{ FMA/s} \\ &= 502 \text{ GFLOP/s} \end{aligned}$$

Cross-architectural comparison (not fair for FPGA)

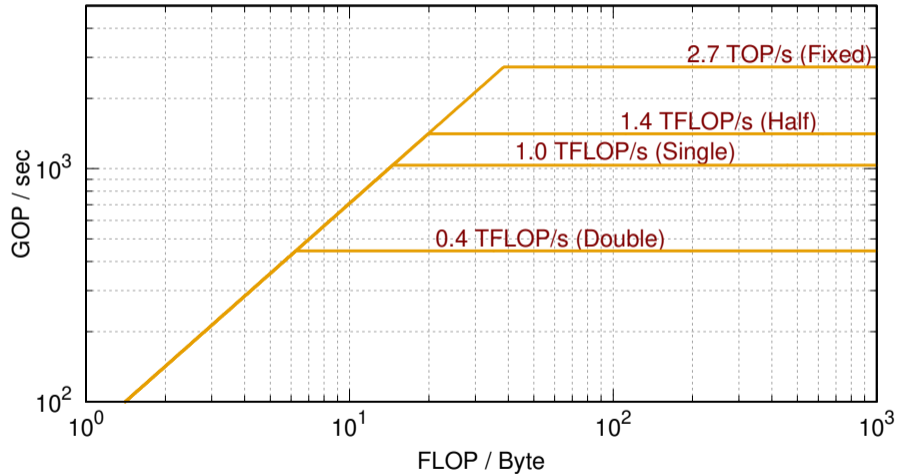


Roofline obtained by FER on the Alveo U250, compared with the ones obtained by ERT on an Intel Skylake CPU and by our custom Arm optimized ERT version on a Marvell ThunderX2 CPU.

Empirical Roofline for other Alveos



Empirical Roofline for lower precision



Much more competitive performance can be obtained using lower precision operations.

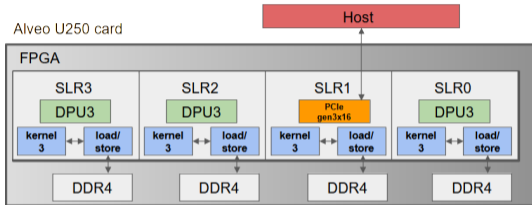
DNN (Deep Neural Network) inference use case

The inference phase of DNN can exploit lower precision weights for the trained model

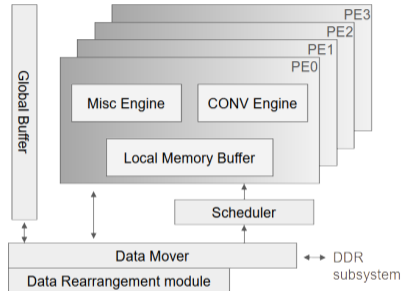
Programming models and frameworks

- Low level programming at RTL (Register Transfer Level), using HDL (Hardware Description Languages) e.g., Verilog, VHDL
- Programming in C with pragma directives, using HLS (High Level Synthesis)
- Generate HLS code using an higher level tool
- Deploy on the FPGA a DPU (Deep Learning Processor Unit) and then compile a trained model as a sequence of DPU instructions

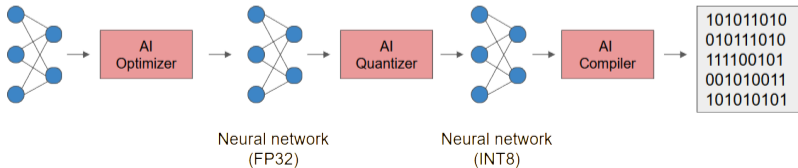
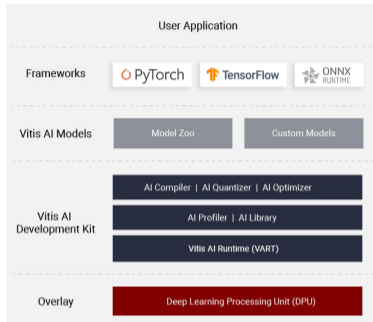
Alveo U250 Card and DPUCADF8H Architecture



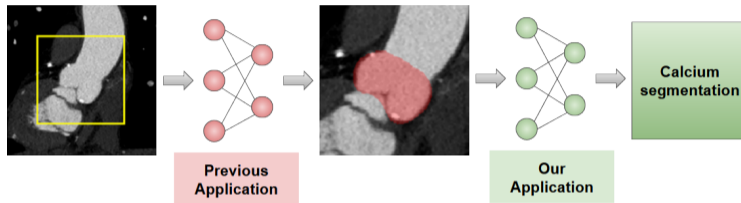
- PCIe Gen3 x16
- 4 × 16 GB DDR4 memory banks
- 16 nm XCU250 FPGA
- Power consumption: 215 Watts (TDP)



- **3 cores**
- **4 Process Elements (PE) per core**
 - Convolution Engines
 - Misc Engines
 - Feature Map Buffer



Segmentation of Aortic Valve Calcium lesions using a CNN model of a Unet



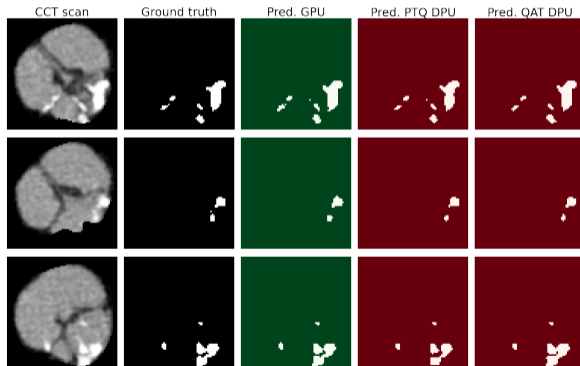
The full pipeline

- 27392 image slices extracted from annotated gated **contrast-enhanced** Cardiac Computed Tomography (**CCT**) acquired with 256-slide scanner
- **ROI extraction** around the aortic root
- **Aortic root segmentation** within the ROI with the previous application
- **Calcium segmentation** within the aortic root segmentation with our application

Accuracy Comparison

Dice Score

Achieved a Dice score of approximately 93% across all trials.



Hardware Accelerators Details

Xilinx Alveo U250



FinFET+ 16nm

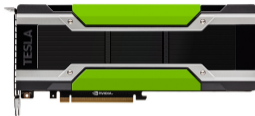
77 GB/s off-chip
38 TB/s on-chip

225 W

33 TOPS INT8

4 SLR

NVIDIA P100



FinFET+ 16nm

732 GB/s off-chip

250 W

9340 GFLOPS SP

SMs 56

NVIDIA V100



FFN 12nm

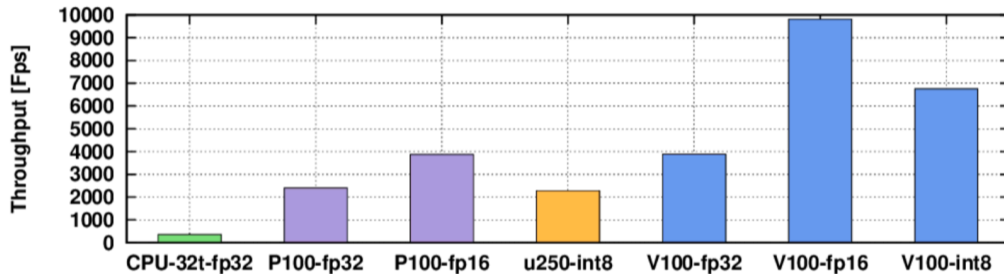
900 GB/s off-chip

250 W

112 TFLOPS HP
14 TFLOPS SP

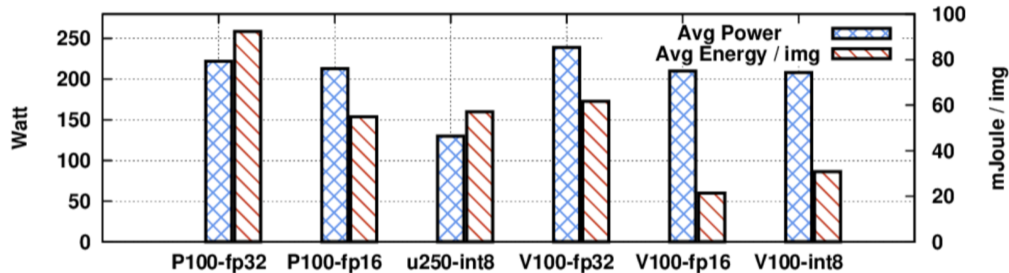
SMs 80

Throughput Performance Comparison



Inference performance on different architectures and numerical precision.

Power Comparison



Total power drained as reported by nvidia-smi and xutil.
Energy derived as integral over the execution time.

Edit distance (Used for DNA “strings”)

The **edit distance** is the number of operations required to morph a string S_1 into a string S_2 using three kinds of operations:

- substitution or replacement of a character;
- insertion of a new character;
- deletion of a character.

If the cost of each operation is unitary, the sum of the costs of the corresponding operations required to align the two strings is named the **Levenshtein distance**.

Edit distance computation

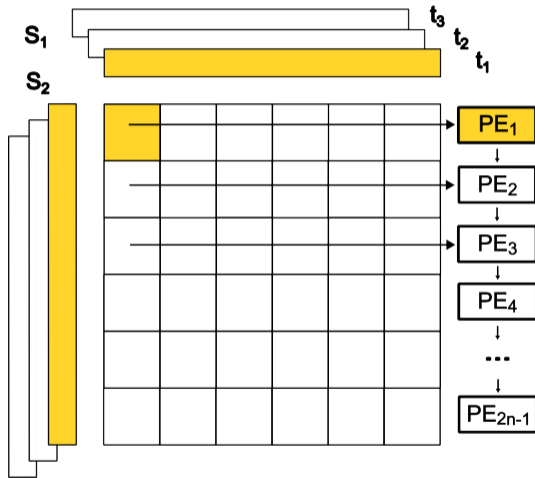
		A	C	A	G	C	G	G	T
	0	1	2	3	4	5	6	7	8
G	1	1	2	3	3	4	5	6	7
T	2	2	2	3	4	4	5	6	6
A	3	2	3	2	3	4	4	5	6
C	4	3	2	3	3	3	4	5	6
A	5	4	3	2	3	4	4	5	6
G	6	5	4	3	2	3	4	4	5
C	7	6	5	4	3	2	3	4	5
G	8	7	6	5	4	3	2	3	4

		A	C	A	G	C	G	G	T
	0	1	2	3	4	5	6	7	8
G	1	1	2	3	3	4	5	6	7
T	2	2	2	3	4	4	5	6	6
A	3	2	3	2	3	4	5	6	7
C	4	3	2	3	3	3	4	5	6
A	5	4	3	2	3	4	4	5	6
G	6	5	4	3	2	3	4	4	5
C	7	6	5	4	3	2	3	4	5
G	8	7	6	5	4	3	2	3	4

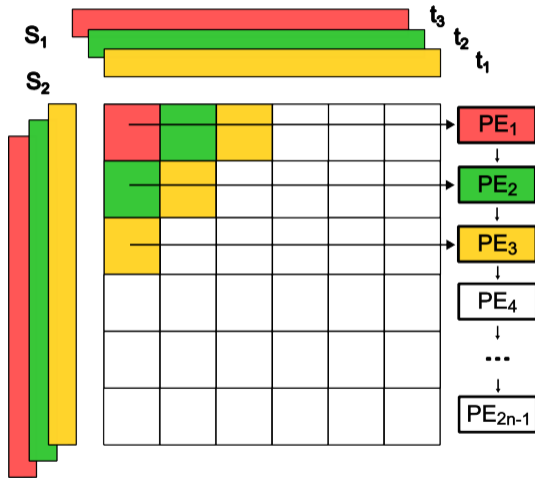
		A	C	A	G	C	G	G	T
	0	1	2	3	4	5	6	7	8
G	1	1	2	3					
T	2	2	2	3	4				
A	3	2	3	2	3	4			
C	4		2	3	3	3	4		
A	5			2	3	4	4	5	
G	6				2	3	4	4	5
C	7					2	3	4	5
G	8						2	3	4

(Left) Example of computation of the Levenshtein distance between two strings of 8 characters each. (Center) Example of distance path built by the algorithm during the execution. (Right) Example of Banded Levenshtein distance with a threshold of 4; only the white cells are computed while dark cells correspond to those for which the distance is higher than the threshold.

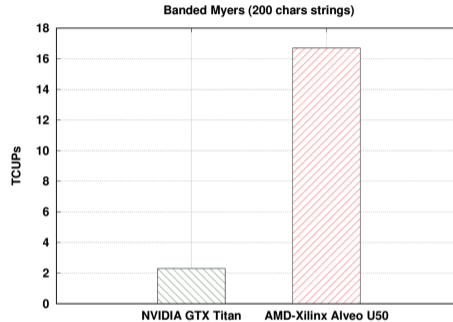
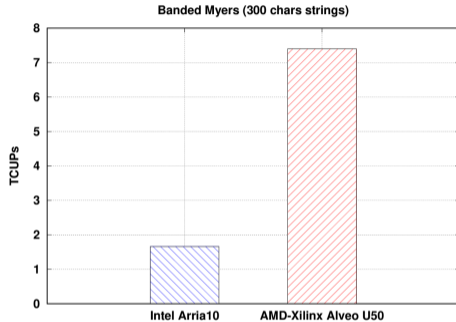
Edit distance computation



Edit distance computation



Results



Comparison with previous research works implementing the Banded Myers algorithm on different architectures, in terms of CUPs (Cell Updates per Second).

Conclusions

- The use of HLS, and even higher level abstractions, are maturing;
- Using SP and DP floating-point operations FPGAs are competitive with CPUs;
- Using low precision arithmetic, FPGAs can be competitive with GPUs.

Conclusions

- The use of HLS, and even higher level abstractions, are maturing;
- Using SP and DP floating-point operations FPGAs are competitive with CPUs;
- Using low precision arithmetic, FPGAs can be competitive with GPUs.

Specific use cases

- The inference phase of DNN algorithms;
- The edit distance computation;

Conclusions

- The use of HLS, and even higher level abstractions, are maturing;
- Using SP and DP floating-point operations FPGAs are competitive with CPUs;
- Using low precision arithmetic, FPGAs can be competitive with GPUs.

Specific use cases

- The inference phase of DNN algorithms;
- The edit distance computation;
- When low precision arithmetic can be used;
- When results latency has to be predictable.

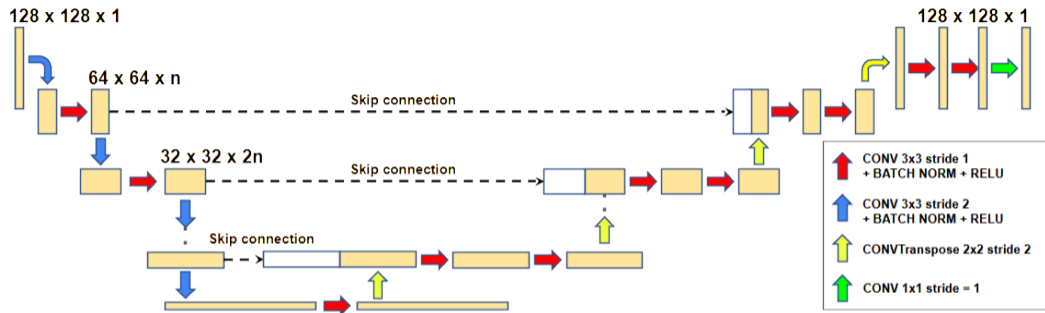
Future works

- Test the same applications on newer FPGAs (e.g., Alveo V70, or V80);
- Test newer versions of Vivado / Vitis-AI;
- Test performance and programmability of different tools (e.g., HLS4ML);
- Try to port and implement other applications.

Thanks for Your Attention



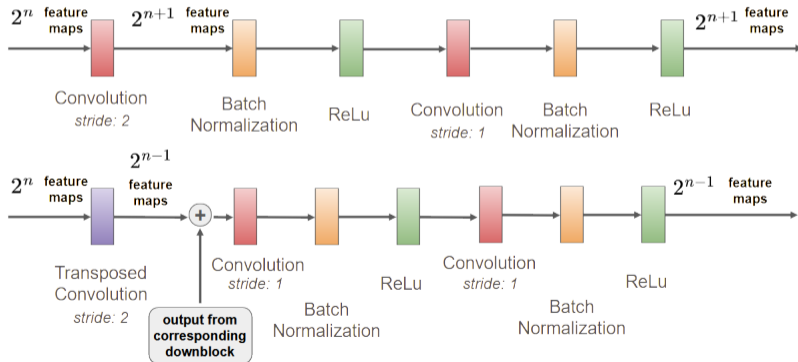
Unet Structure



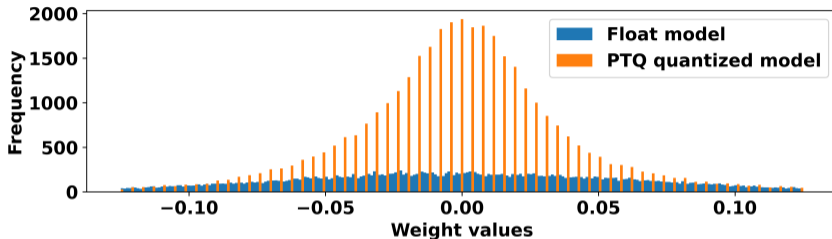
- Based on the 2D Unet developed in 2015 at the University of Freiburg
- U-shape pattern:
 - Encoder branch: extract feature maps and reduce the dimensions
 - Decoder branch: Reconstructs image reducing feature maps and restoring dimensions
- Skip connections

Unet implementation

- Max-pooling (not supported by the DPU) replaced by convolutions with stride 2
- Added batch normalizations
- Implemented using **Python 3.7.12** and **Pytorch 1.10.1**, in **Vitis-AI Docker 2.5**



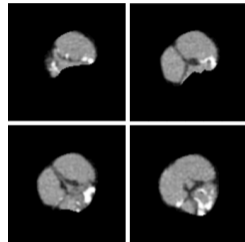
Float and Quantized Weights Distributions



- Comparison between float and quantized weights distributions of the same layer (a convolution of a downblock)
- Float model: continuous distribution
- Quantized model (PTQ): discrete distribution at regular intervals

The Dataset

- Cropped original images to a volume $128 \times 128 \times 128$ pixels around the centroid of the aortic root
- Clipped within the range $[0, 1000]$
- Normalized in range $[0.0, 1.0]$
- Stored as 32-bit floating point values
- extracted all 2D slices containing the aortic root



Result: 1350 images

Augmentation

With probability 0.5.

Horizontal/vertical flip, rotation between -180 and 180 degrees

We thank the San Carlo di Nancy Hospital (Rome) for access to the dataset.

Loss function

Weighted sum of Dice and Cross Entropy (CE)

$$\text{Dice} = 1 - \frac{2 |A \cap B|}{|A| + |B|} \quad \text{CE}(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

Accuracy metrics

- Dice Score
- Intersection over Union (IoU)

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|}$$

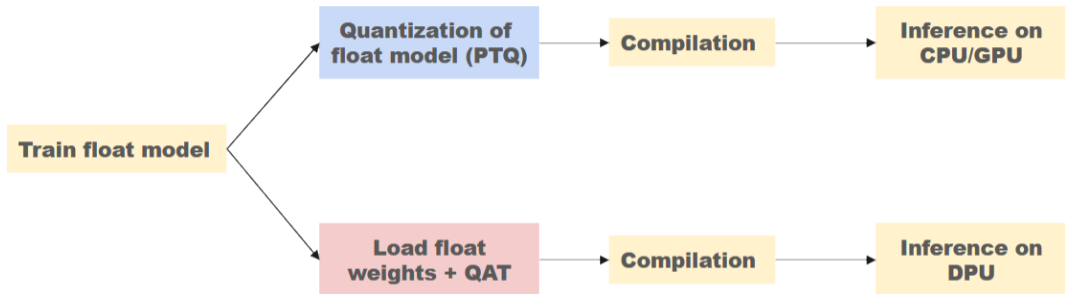
Deploy on FPGA

```
1 g = xir.Graph.deserialize(config.PQT_COMPILED_MODEL_PATH)
2 subgraphs = get_child_subgraph_dpu(g)
3 runner = vart.Runner.create_runner(subgraphs[0], "run")
4 qimgs = quantization_img(imgs, input_scale)
5 rid = runner.execute_async(qimgs, predictions)
6 runner.wait(rid)
7 pred_imgs = np.argmax(predictions, axis=-1)
```

Deployment Process

- 1 Extract the subgraphs executable on the DPU from the compiled model
- 2 Instantiate a Vitis AI Runtime runner to handle asynchronous data transfer and communication between CPU and DPU
- 3 Quantize input images for DPU processing
- 4 Launch the runner for asynchronous inference execution on DPU

PTQ and QAT Pipelines



- Symmetric quantization
- Bitwidth: 8
- Calibration set: 1000 randomly augmented images from the training set