Machine Learning with AI Engines and Vitis

Dennis Layh
Institute of Physics Mainz

Introduction

- ATLAS Phase II-upgrade for HL-LHC
- Machine Learning for FPGA-based Level-0 Trigger
  - Improve signal identification and background rejection
  - Mainz: fFEX (forward Feature EXtractor)
- Calorimeter Data
  - fFEX: forward region
- Submicrosecond(!) latency

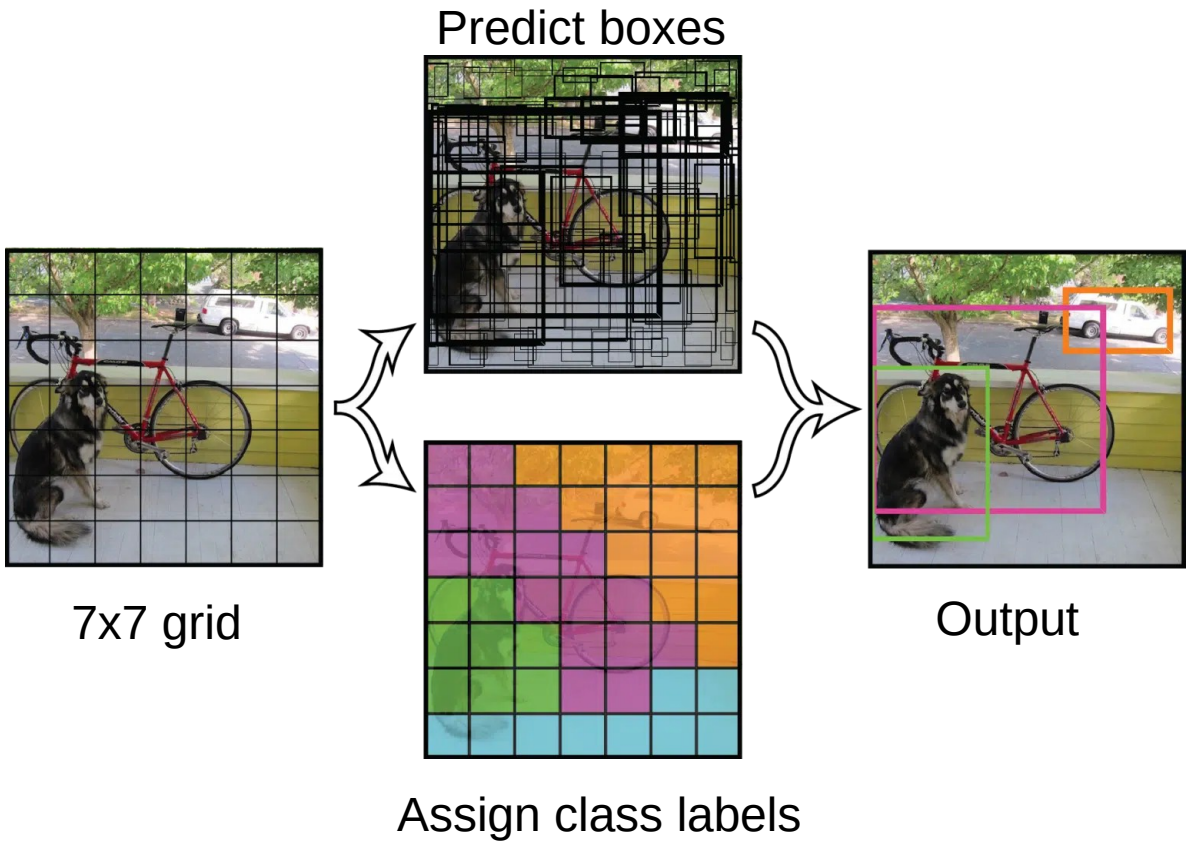
Calorimeter Data

Dennis Layh
NFDI 2023
AIEs and Vitis

- Image like
- No RGB but different layers and calorimeters
- CNNs can extract features
- Groups of deposits can be treated as objects in image
- Idea: object detection for calorimeter data

YOLO Net

- You Only Look Once
- Divide image into grid
- Each grid predicts:
  - N boxes and their size
  - A class label for every box
- Faster than two-stage object detection

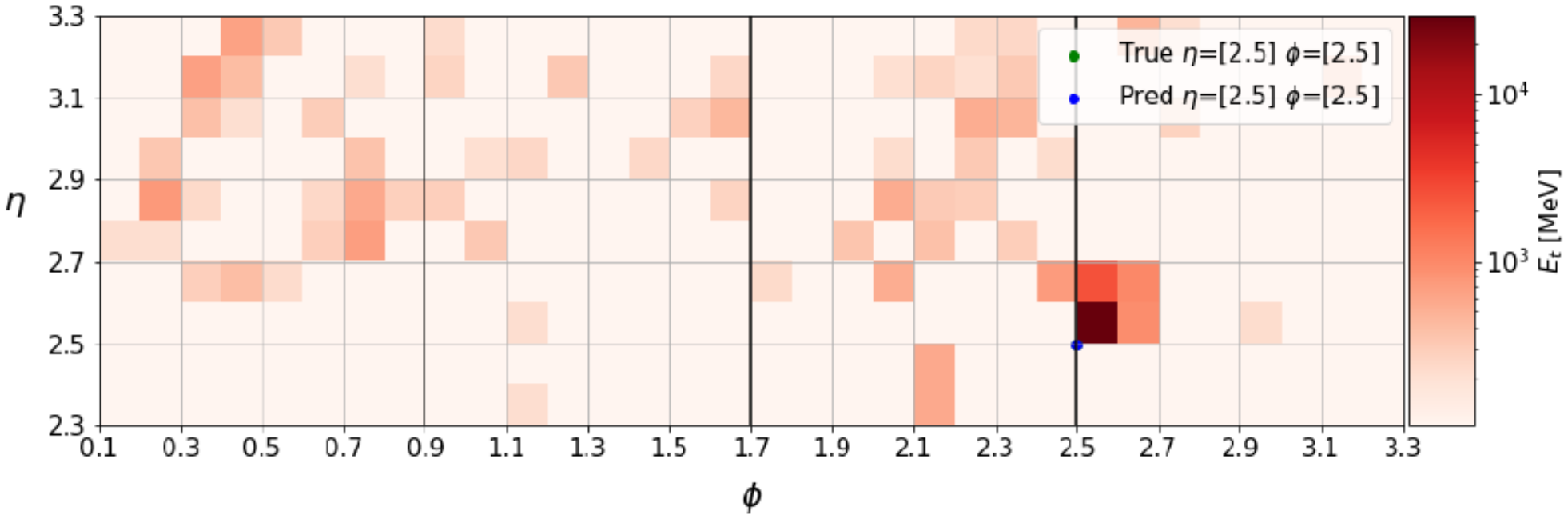


YOLO Redmon, Divvala, Girshick, Farhadi
https://arxiv.org/abs/1506.02640

CALONet

Dennis Layh
NFDI 2023
AIEs and Vitis

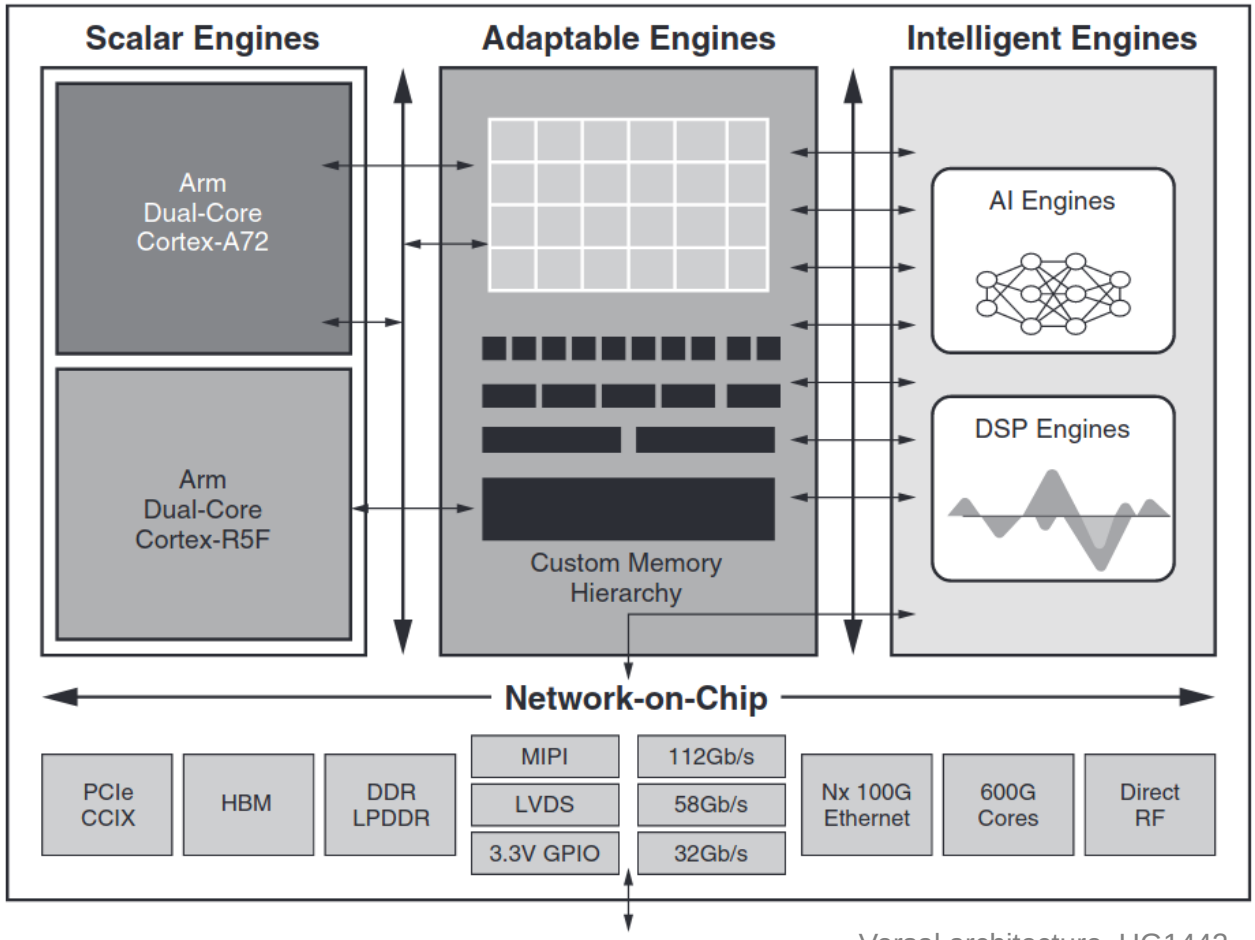
- YOLO-like object detection algorithm
  - Divide calorimeter in to grid
  - Look for a signature in every grid-cell



ML on FPGAs

- How to deploy CALONet on FPGAs?
- Many options:
  - Hls4ml, FINN, Vitis-AI, ….
- But:
  - ML models are heavy on FPGA resources
  - fFEX is already using resources for other processing
- Possible solution:
  - Use AI Engines (AIEs) on new Versal devices

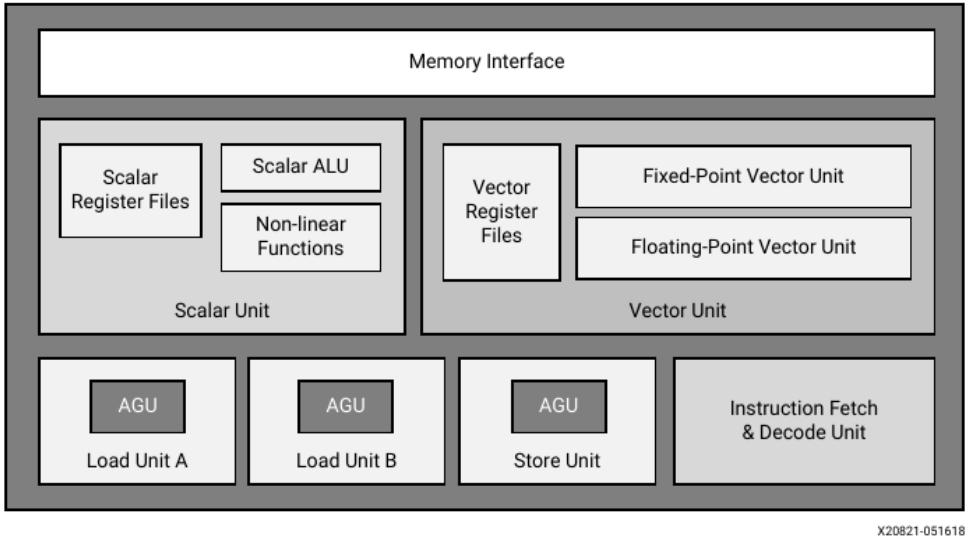
Xilinx Versal Architecture



Versal architecture, UG1442

AI Engine

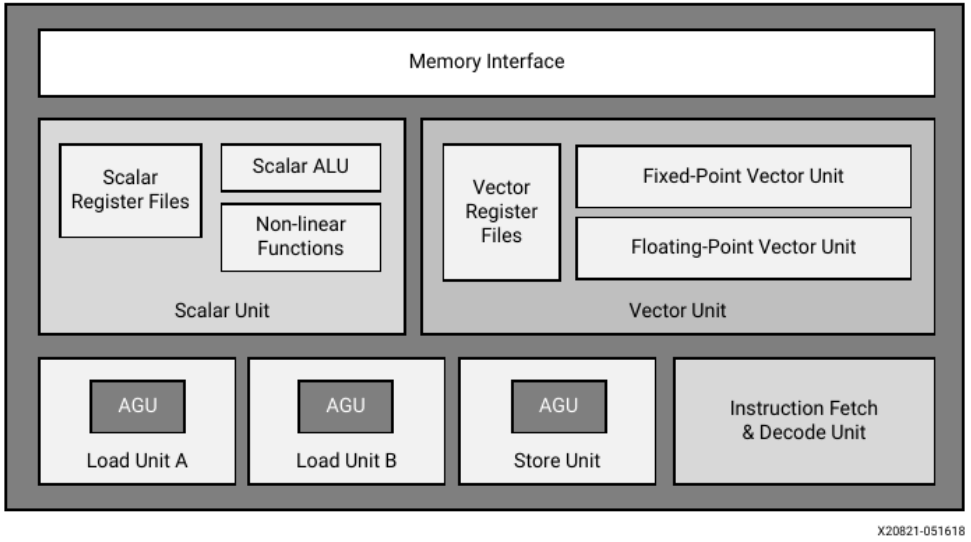
- 16 KB of programming memory
- 32 KB memory module
- Vector + scalar processing unit
- Two load units
- One store unit



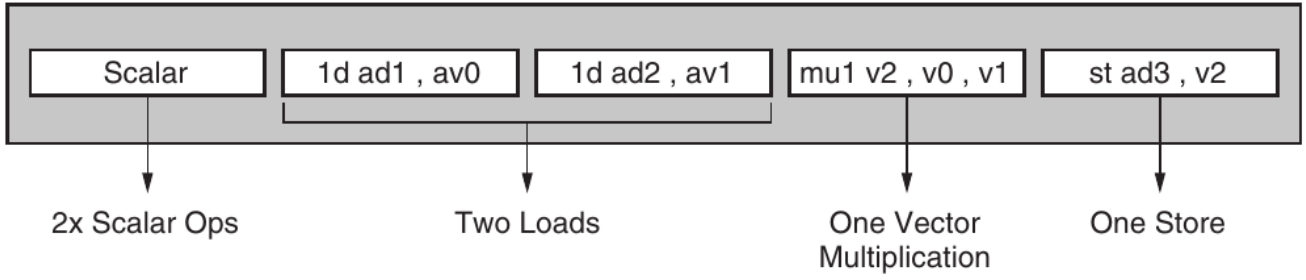
AI Engine, AM009 (v1.1)

AI Engine

- 16 KB of programming memory
- 32 KB memory module
- Vector + scalar processing unit
- Two load units
- One store unit
- Single Instruction Multiple Data
- 1 Very Long Instruction Word per CC

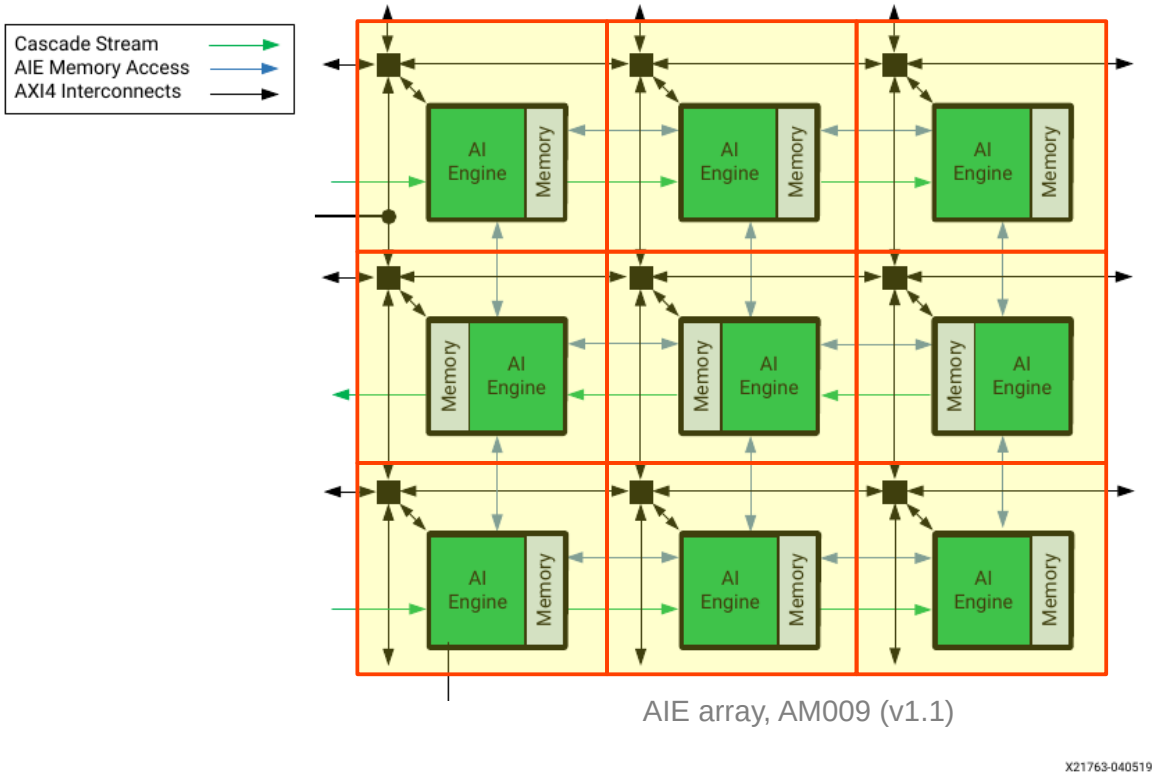


AI Engine, AM009 (v1.1)



AI Engine Array

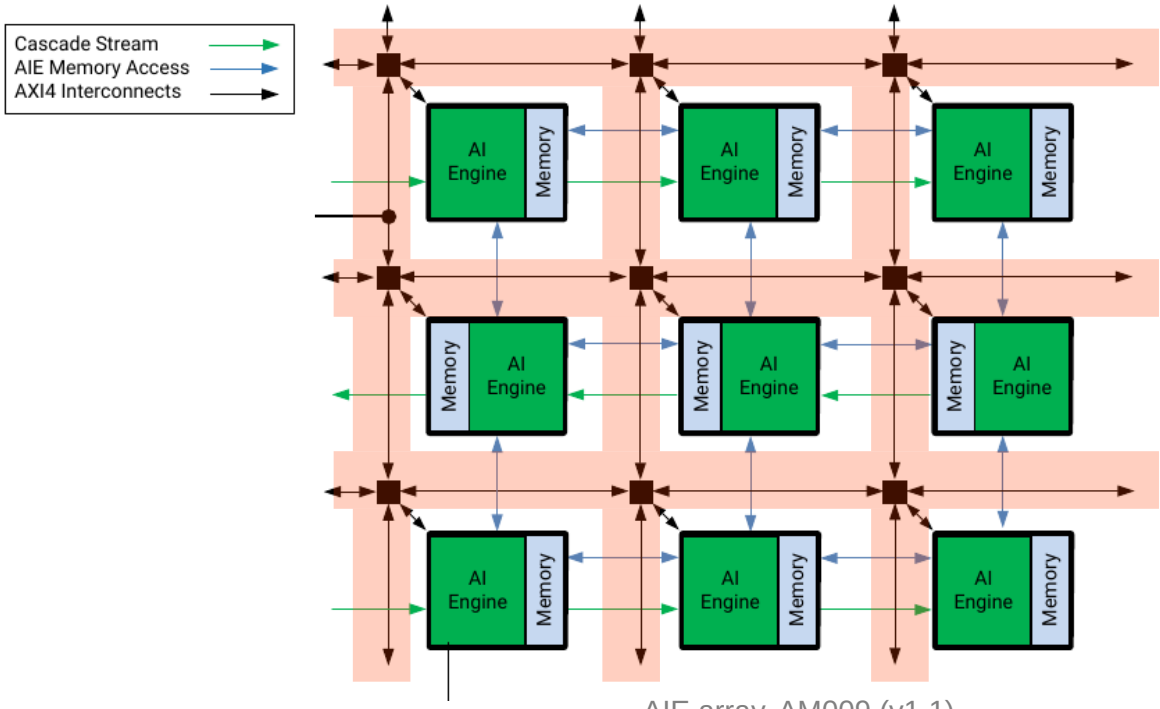
- AI Engine (AIE) Tile:
  - AIE + memory + AXI4



AIE array, AM009 (v1.1)

AI Engine Array

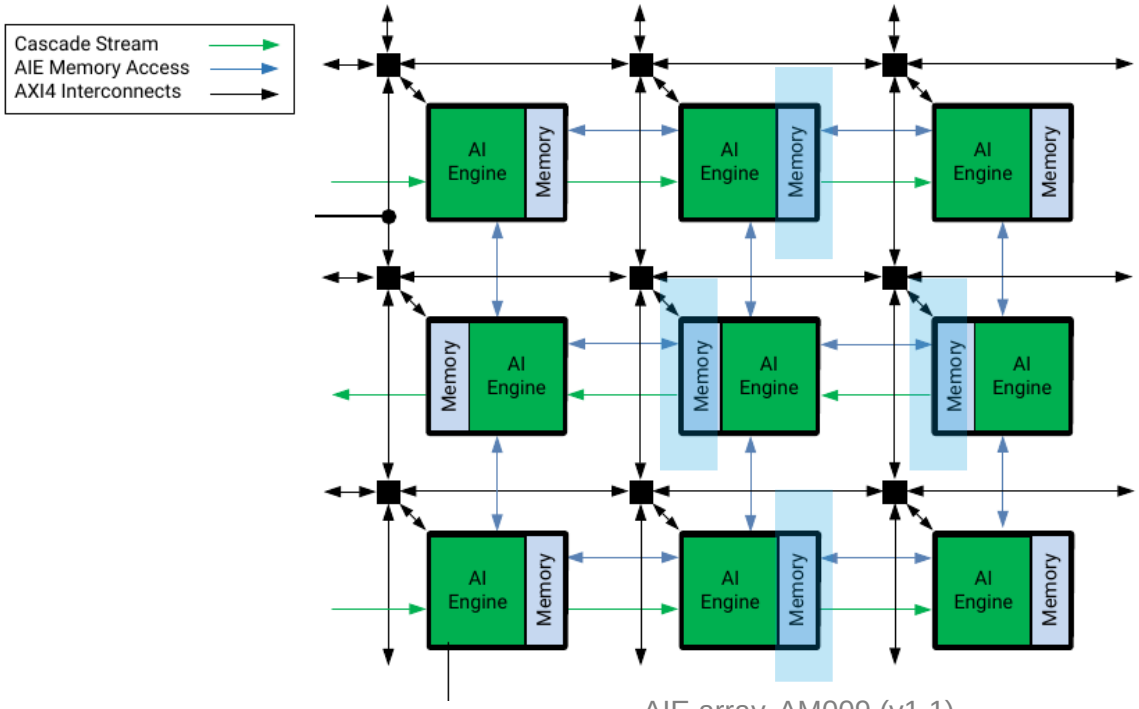
- AI Engine (AIE) Tile:
  - AIE + memory + AXI4
- AXI4 streaming Interconnect
  - In all four directions



AIE array, AM009 (v1.1)

AI Engine Array

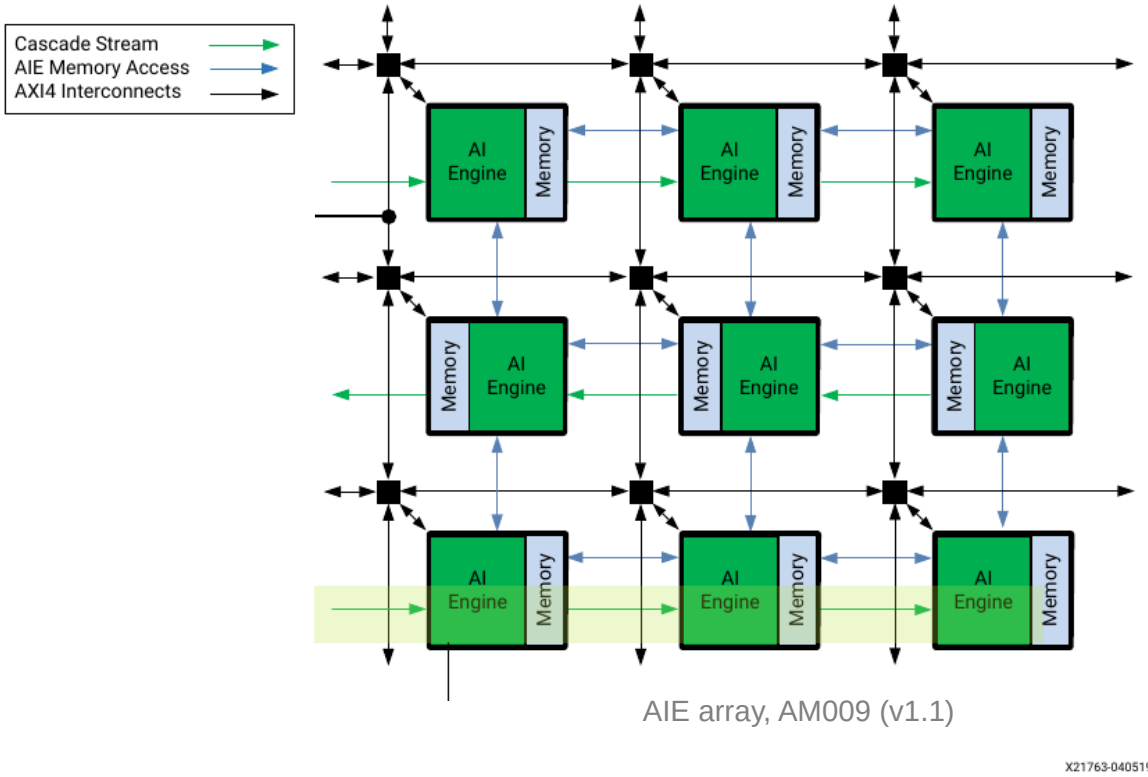
- AI Engine (AIE) Tile:
  - AIE + memory + AXI4
- AXI4 streaming Interconnect
  - In all four directions
- Neighbors share contiguous memory



AIE array, AM009 (v1.1)

AI Engine Array

- AI Engine (AIE) Tile:
  - AIE + memory + AXI4
- AXI4 streaming Interconnect
  - In all four directions
- Neighbors share contiguous memory
- Cascade stream
  - unidirectional stream



AIE array, AM009 (v1.1)

Programming AIEs

- Deep learning Processing Unit (DPU) via Vitis-AI
  - Direct implementation of neural networks
    - Combination of AIEs and PL
  - Restricted to certain architectures

➔ Did not meet our latency requirement

- Kernel programming via Vitis
  - C++ code
  - Directly control instructions, data movement and memory management
  - Reasonable alternative to Vitis-AI for smaller networks

Basic Vitis Structure

- Project with specified device/platform
  - Contains HW – links
- AIE graph
  - Atleast 1 Kernel.cpp
  - One Graph.cpp
  - Corresponding headers
- Kernel
  - Contains instructions for AIE
- Graph
  - Specifies locations and data flow

Example Kernel

- Takes two input streams
- Combines them to a size 16 vector of ints
- Applies a filter (here * 1)
- Returns the output

```
#include <aie_api/aie.hpp>
#include <aie_api/aie_adf.hpp>
#include "aie_api/utils.hpp"
using namespace adf;
aie::vector<int, 16> filter0  = aie::broadcast<int,16>(1);
void cnn(input_stream<int>* in0, input_stream<int>* in1, output_stream<int>* out0) {
    for(int i=0;i<1000;i++) {
    aie::vector<int, 8> v0  = readincr_v<8>(in0);
    aie::vector<int, 8> v1  = readincr_v<8>(in1);
    aie::vector<int,16> temp=aie::concat(v0,v1);
    auto output0 = aie::mul(filter0,temp);
    writeincr(out0, output0.to_vector<int>(0));
    }
}
```

Example Graph

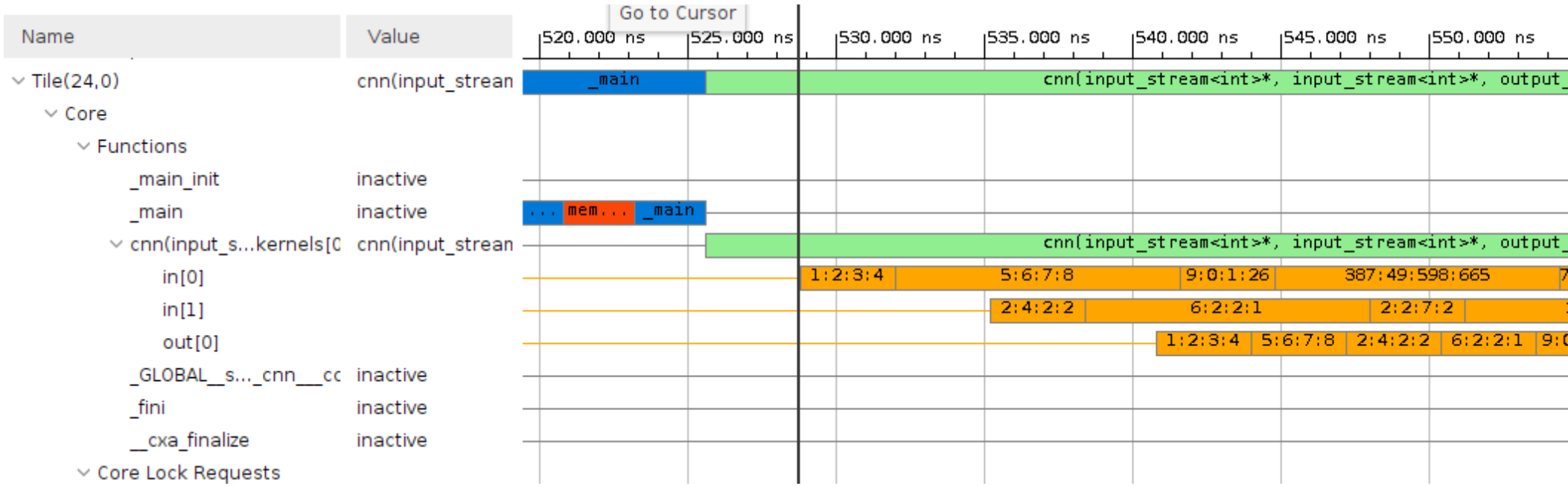
- Define data flow and AIE locations including:
  - Kernel locations
  - Kernel I/Os
  - Data paths and widths
  - Clock frequencies

```
class simpleGraph : public graph {
    private:
        kernel kernels[1];
    public:
        input_plio  in[2];
        output_plio out[1];

    simpleGraph() {

        in[0]  = input_plio::create(plio_32_bits, "data/input0.txt",1250);
        in[1]  = input_plio::create(plio_32_bits, "data/input1.txt",1250);
        out[0] = output_plio::create(plio_32_bits, "data/output0.txt",1250);
        kernels[0] = kernel::create(cnn);
        source(kernels[0]) = "cnn.cc";


        connect<stream> (in[0].out[0], kernels[0].in[0]);
        connect<stream> (in[1].out[0], kernels[0].in[1]);
        connect<stream>(kernels[0].out[0], out[0].in[0]);
        runtime<ratio>(kernels[0]) = 1;
    };
};
```

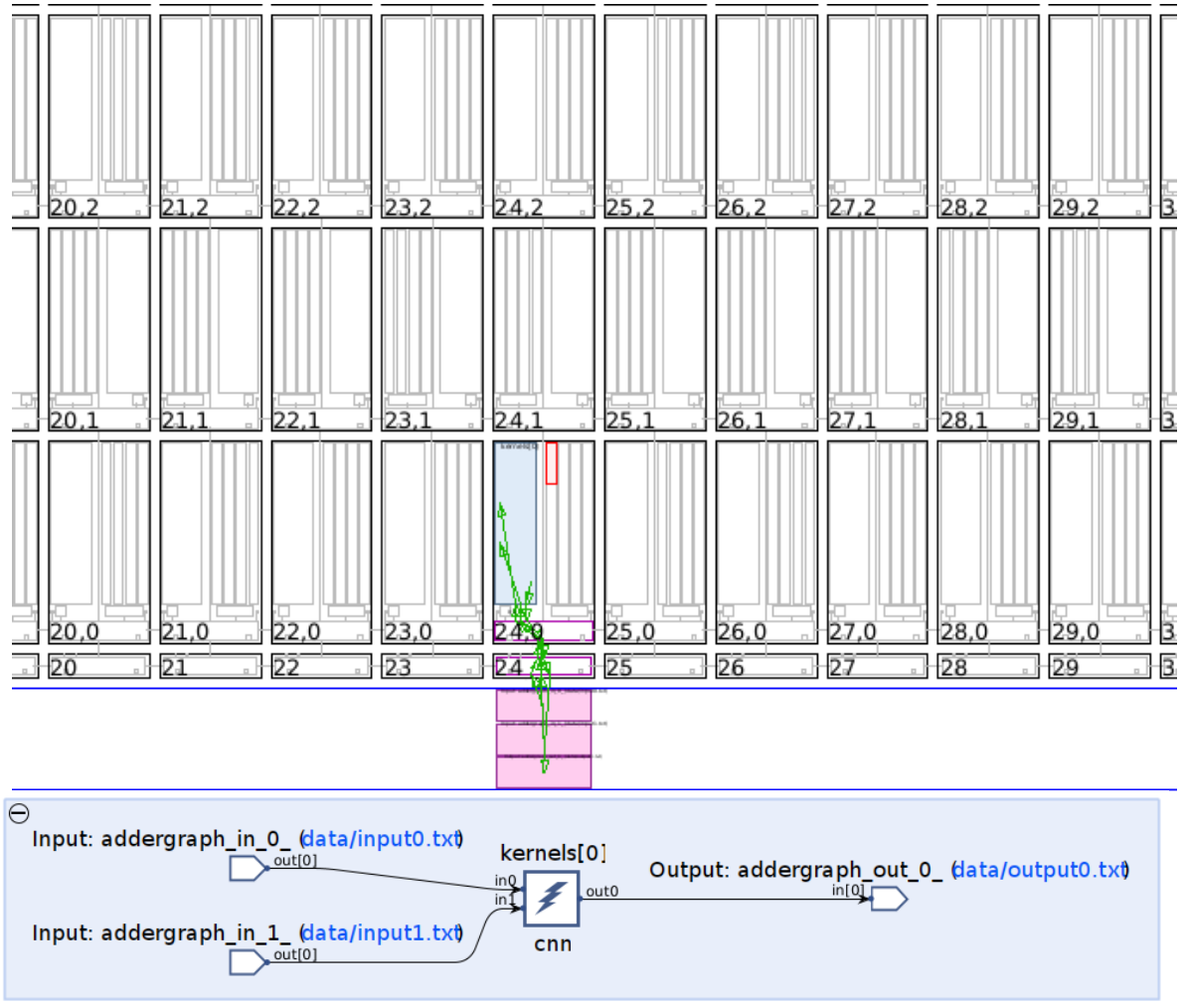
Example Trace

Dennis Layh
NFDI 2023
AIEs and Vitis

- New data output every 4 ns after some initial delay
- Can be further optimized



AIE Array and Graph



Optimization

- Many possible designs for one small network
- Where to put what ?
- One AIE per CNN filter ?
- Plus shaped design to access memory more efficiently ?
- Or line based to use cascade ?
- Also:
  - Optimize network itself!
  - Adjust network to fit AIEs better ?
  - Optimize data ?

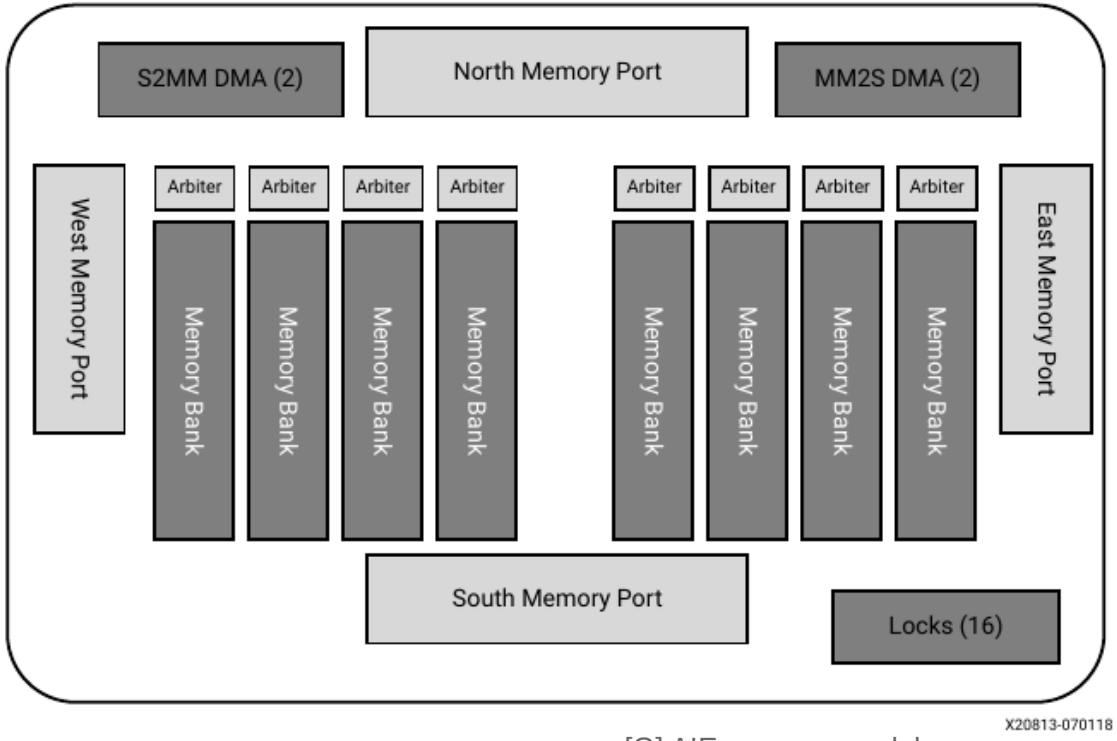
Summary

- CALONet:
  - ML approach for object detection in calorimeter data
  - Outperformed classical sliding window methods offline
  - Vitis-AI could not deliver low enough latency
- Vitis allows custom implementation using C++
  - Difficult to optimize
  - Efficient data and memory management has to be done by the user
  - Can get very complicated for large networks
- Only option right now that could enable sub µs latency with AIEs

Thank you for your attention!

AIE Memory

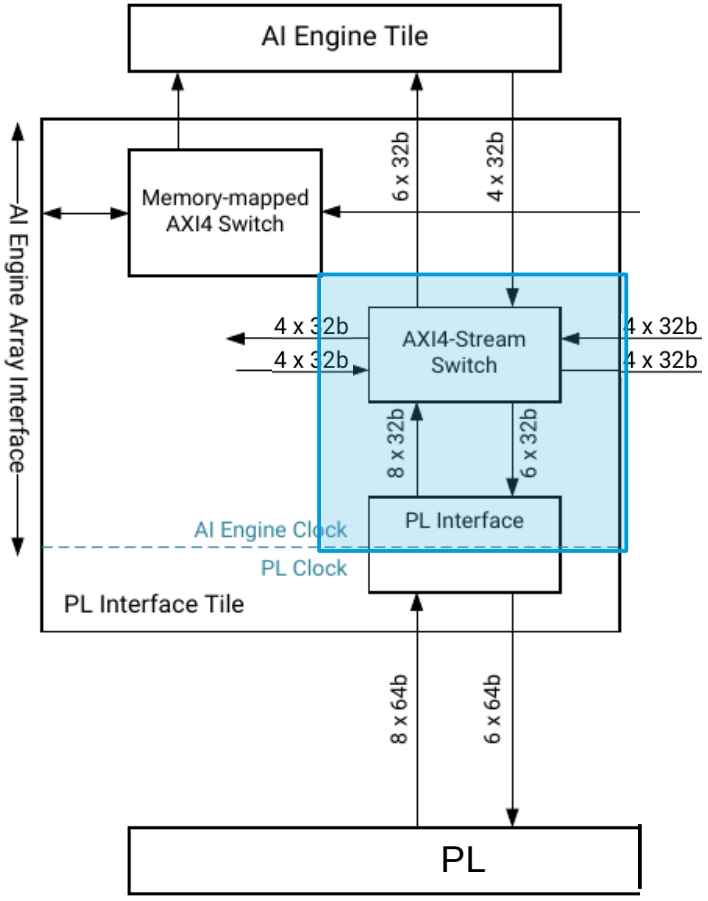
- 8 memory banks
- Every bank holds 128 x 256 bit words
- Locks for every bank
- DMA to access non neighbouring memories



[G] AIE memory module

PL to AI interface

- Data moves from PL to AIE through PL interface
- Is then distributed inside the array using AXI4 stream switches
- Clock domain crossing:
  - AIE clock is between 1 and 1.3GHz
  - PL clock maximum is half of AIE clock



[14] AIE to PL

Vector operations per CC

X Operand	Z Operand	Output	Number of MACs
8 real	8 real	48 real	128
16 real	8 real	48 real	64
16 real	16 real	48 real	32
16 real	16 complex	48 complex	16
16 complex	16 real	48 complex	16
16 complex	16 complex	48 complex	8
16 real	32 real	48/80 real	16
16 real	32 complex	48/80 complex	8
16 complex	32 real	48/80 complex	8
16 complex	32 complex	48/80 complex	4