TVM @ EdgeCortix - Heterogenous AI Hardware Acceleration

Antonio Tomas

DL Compiler Engineer

www.edgecortix.com

EdgeCortix Overview

- Founded in mid 2019
- Engineering driven with nearly equal mix of ML software, ASIC and FPGA experience

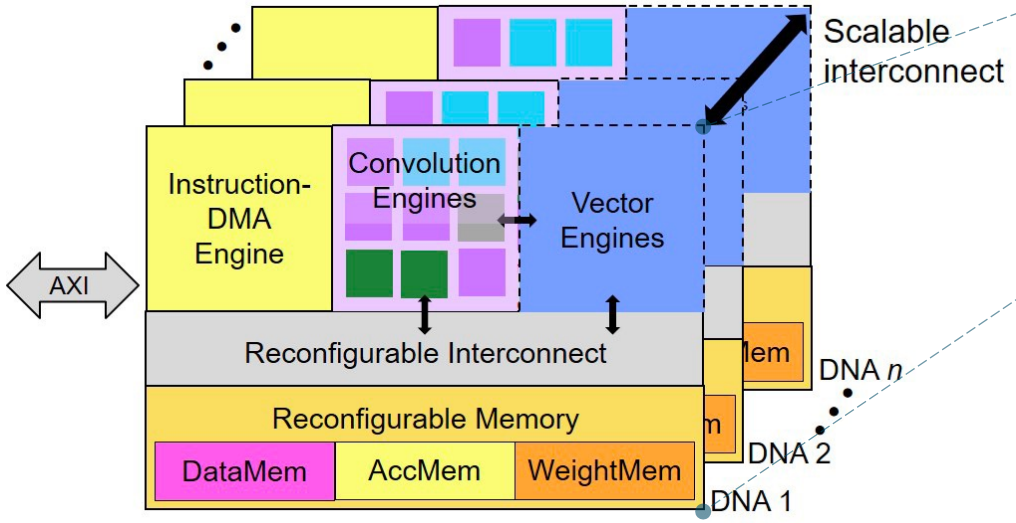
Goal

- ML Acceleration within heterogenous platforms
- Optimized for low-power edge deployment
- Scalable & modular IP
- **Energy Efficient ML Inference System** rather than yet another processor

"Next generation runtime reconfigurable AI processing architecture for edge inference"

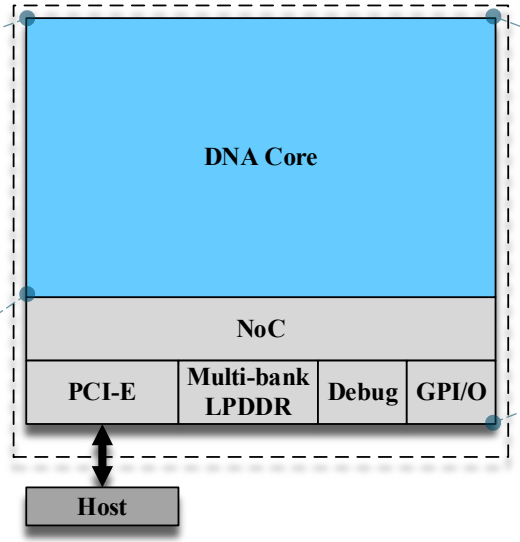
Summary of our Machine Learning Hardware Architecture

Dynamic Neural Accelerator® IP



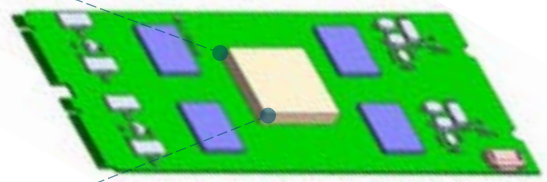
- Dataflow array architecture
- Runtime-reconfigurable Interconnect
- Heterogenous compute engines
- Scalable ML Inference accelerator

SAKURA AI co-processor



- Upto 52 TOPS (INT8)
- Optimized for batch size 1
- 10W-15W TDP

Form factor



- Dual M.2 and Low-profile PCI-e
- PCI-E gen. 3.0
- LPDDR4x
- JTAG
- UART

The Machine Learning Accelerator Deployment Challenge!

Source of your trained model

PyTorch

TensorFlow Lite

TensorFlow

ONNX

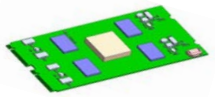
mxnet

Caffe2

Heterogenous compute

CPUs

EdgeCortix **SAKURA**



Custom ASIC

FPGAs

Flexible Compiler Stack

EDGECORTIX®

MERA + tvm

High level summary of TVM with EdgeCortix MERA

We embrace three fundamentals parts of TVM to extend our own software stack - MERA

- Native support for Pytorch and TFLite front-ends
- Relay's QNN dialect for parsing INT8 quantized models at the framework level
- Bring Your Own Code Gen. extension:
    - MergeComposite (patterns)
        - Pattern matching of supported subgraphs (Composites)
    - AnnotateTarget()
    - MergeCompilerRegions()
        - Further merge composites into a single pattern to enable our compiler to have a larger scope for applying further low-level optimizations
    - PartitionGraph()

The MERA Compiler and Software Stack

Pre-optimized

EdgeCortix Model Zoo

- Classification
- Detection
- Segmentation
- Others

Custom Models

Open-source community driven models

PyTorch · TensorFlow · TensorFlow Lite · ONNX

Calibration and Quantization

Relay Intermediate Representation (INT8)

Graph Optimization & Partitioning

Apache **TVM** Frontend

MERA Code generator

CPU / Host Code Generator

MERA Graph IR

LLVM low-level IR

MERA API

LLVM

RUNTIME

Profiler

Interpreter

Simulator

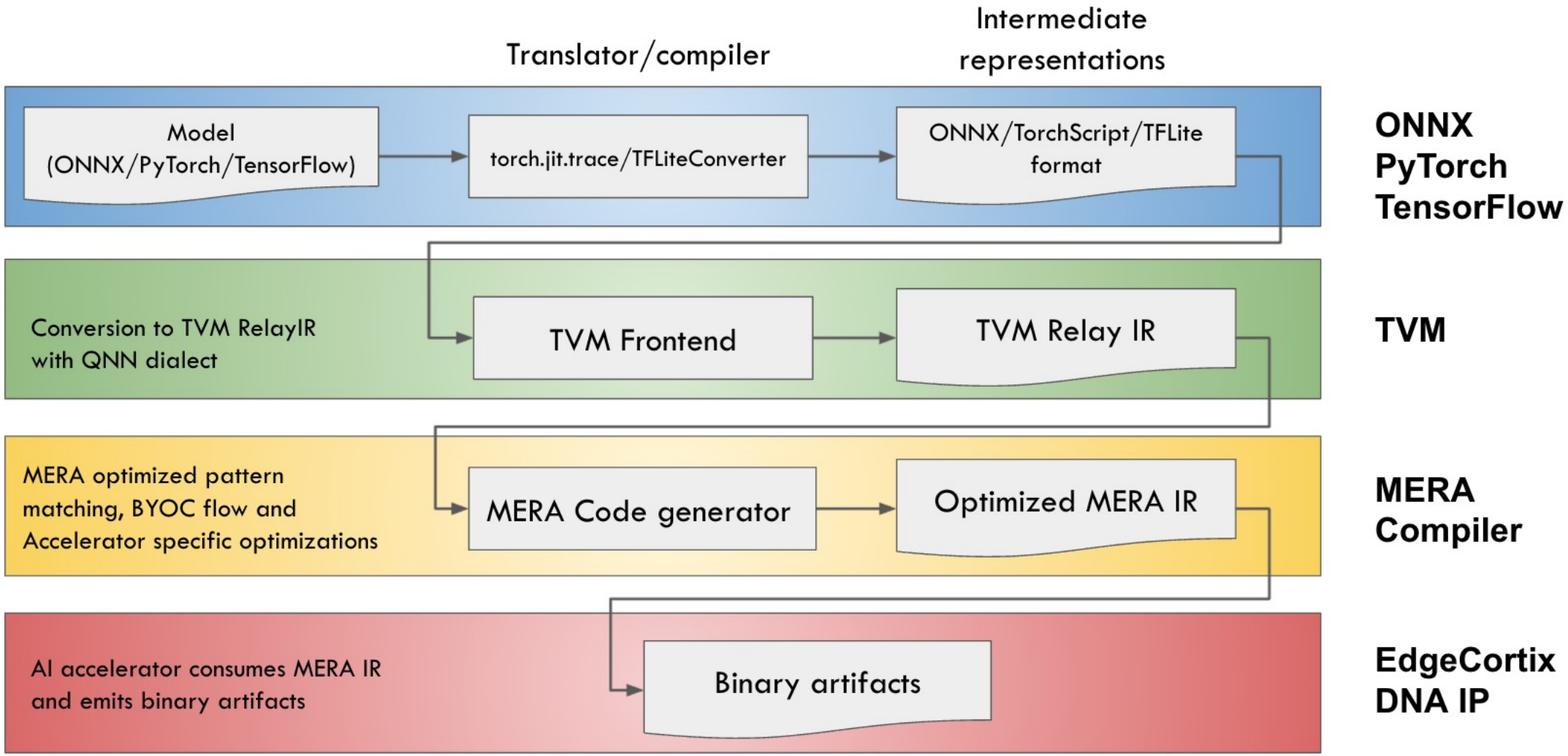
Dynamic Neural Accelerator

Host CPU

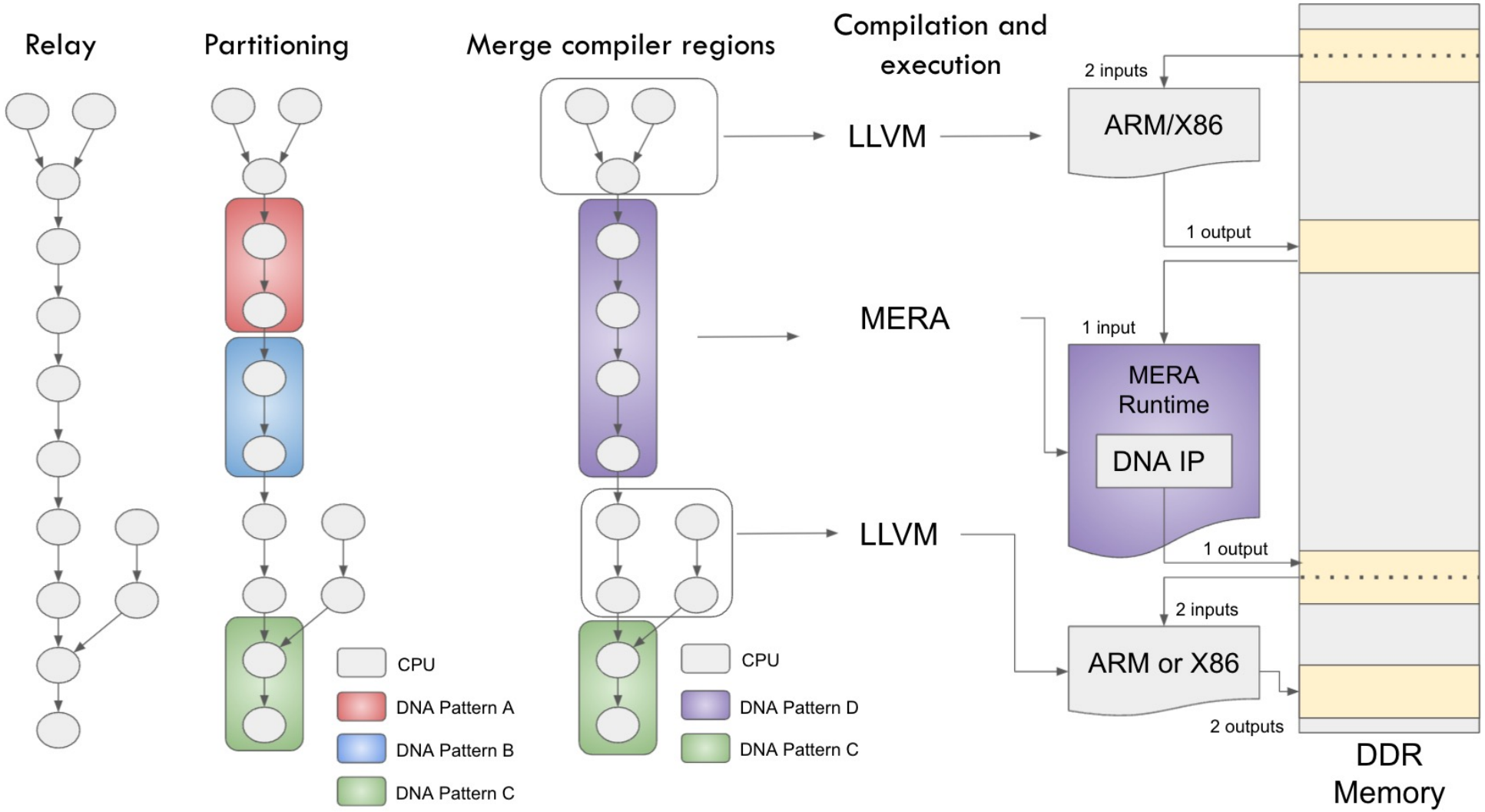
jupyter · docker · ubuntu

FPGA · ASIC

Evolution of the Intermediate Representations



Flexible Graph Partitioning



MERA API to Deploy any Pre-trained Model / Models

1. Framework level Quantization supported

PyTorch TensorFlow Lite ONNX

Pre-trained FP32 Model → Calibration, Quantization to INT8

Take standard machine learning framework dependent steps to calibrate and quantize the model.

2.

Specify input size eg. (1920x1080)

Build quantized graph with MERA API *

Choose Target

- Profiler
- DNA IP on FPGA
- EdgeCortex AI Chip

Pytorch

```
config = {
    "arch": "DNAF632L0001"
}
with mera.build_config(target="IP", **config):
    mera.build(mod, params, host_arch="arm",
        output_dir=output_dir, layout=layout)
```

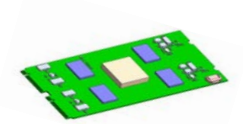
TF Lite

```
config = {
    "arch": "DNAF632L0001"
}
with mera.build_config(target='IP', **config):
    mera.build_for_tflite(mod, params, output_dir=output_dir,
        host_arch="x86")
```

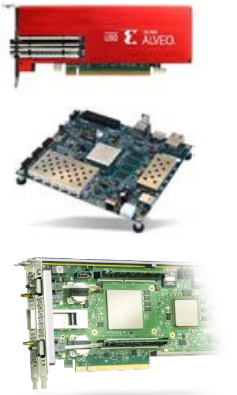
3. Heterogenous platforms

Generates deploy directory with compiled artifacts

EdgeCortix **Sakura** AI co-processor



3rd party FPGAs with EdgeCortix **DNA IP**



* Supports both x86 and Arm based host architectures, enabling execution on SoC or PCI-E type cards

Challenges & Observations of using TVM with a custom AI accelerator

- Software is the most critical element for efficient deployment with any hardware platform.
- A general deep learning compiler like TVM (at a high level) may help achieve a common standard across vendor specific software tools – better user experience & reduced adoption time

However, there are few key challenges (missing features) that we observed needs more attention from the open-source community. These are also some areas we are actively working on.

- **Autonomous and efficient resource scheduling & memory allocation**
- **Efficiently deploying multiple models**
- **Runtime pipelining between processor and co-processors**
- **Built in quantization support for mixed precision networks**
- **Bring your own sparse networks / pruning techniques**