Fast, Scalable Quantized Neural Network Inference on FPGAs with **FINN & LogicNets**

@ H2RC at Supercomputing, 2020-11-10

Yaman Umuroglu, Senior Research Scientist
Xilinx Research Labs
Xilinx Research, Dublin

- Established over 14 years ago
- Slowly expanding and increasingly leveraging external funding (IDA, H2020)
- 6 full-time researchers + interns
- Applications & Architectures
- Quantifying the value proposition of Xilinx devices in machine learning
- In collaboration with Partners, Customers and Universities

Lucian Petrica, Giulio Gambardella, Alessandro Pappalardo, Ken O’Brien, Michaela Blott (leader), Nick Fraser, Yaman Umuroglu (from left to right)
DNNs in Extreme-Throughput Applications

- How do we mix DNNs into *extreme-throughput* applications?
  - Need DNNs running at 100Ms of FPS, sub-microsecond latency

Source: Thomas James, CERN

CERN CMS Experiment

Network Intrusion Detection
How Efficient Does Your DNN Need To Be?  
A Spectrum of FPGA Inference Alternatives

less efficient
generic
broad scope

more efficient
co-designed
specialized
How Efficient Does Your DNN Need To Be?
A Spectrum of FPGA Inference Alternatives

DPU, overlays
(10k+ FPS)

Layer-by-layer compute
(Matrix of Processing Engines)

Optimizing compiler/scheduler
Down to 4-bit

FINN
(10M+ FPS)

Generated heterogeneous streaming architecture

Custom topologies, arithmetic and hardware

© Copyright 2020 Xilinx
Customization for Efficient Inference

Customization of Algorithm

Customization of Hardware Architecture
Two Key Techniques for Customization

- Few-bit weights & activations (tailored to requirements)
  - k-bit weights and activations, k ≤ 4
    - keep all on-chip!

- Streaming dataflow architecture (tailored to requirements)
  - allocated resource ~ compute requirement per layer

FPGA

CNN
Accuracy-Performance Trade-offs

Error vs Compute Cost

- Float
- 8-bit
- Reduced Precision

Use precision which
- Provides required accuracy
- At minimal computational cost

Highly Quantized Neural Networks (<4b)

Pareto frontier

8-bit networks

Different network topologies

Floating point networks
Few-bit QNNs + FPGA Dataflow: Showcases

**High Throughput & Low Latency**

- **MNIST MLP on ZC706**
  - 12.3 M FPS @ 20 W
  - 310 ns latency

**Low-Power, Real-Time Image Classification**

- **CIFAR-10 CNV on Pynq-Z1**
  - 3000 FPS @ 2.5 W
  - 1 ms latency

**Complex Topologies**

- **ResNet-50 on Alveo U250**
  - 2000 FPS @ 70 W
  - 2 ms latency

© Copyright 2020 Xilinx
The FINN Project: Mission

Support customizing the algorithms with precision, layer types, topologies

Support hardware architecture exploration around dataflow execution

Flexibility on Algorithms

Codesign

Flexibility on Architectures

End-to-end flow to lower adoption barrier

Transparency and flexibility through open source (if not supported, add your own!)

Open source from the ground-up to encourage community contributions
The FINN Project: Components of the Stack

From PyTorch to FPGA

- QNN training in PyTorch
- Brevitas
- Frontends, Transformation, Dataflow Backend
- FINN Compiler
- Deployment with PYNQ for Alveo or Zynq

Support
- Gitter Channel
- Jupyter Notebooks
- readthedocs
- End2End examples
Quantization-Aware Training in PyTorch with Brevitas
Brevitas: A PyTorch library for Quantization-Aware Training

Precision
Preset or learned

Scaling Factors
Granularities, strategies and constraints

Target Tensors
Weights, activations, accumulators

Loss Function
to take HW implementation cost into account

add quantization resize layers change hyperparameters retrain

FP32

INT

https://github.com/Xilinx/brevitas
The FINN Compiler

QNN training in PyTorch
Brevitas

Frontends, Transformation, Dataflow Backend
FINN Compiler

Deployment with PYNQ
Goal of the FINN compiler:
*Transform QNN into custom dataflow architecture*

- Map each layer to HLS description
- Connect with FIFOs/streams
- Stitch together in IPI

![Diagram showing a CNN and FPGA with allocated resource and compute requirement per layer]
An Overview of the FINN Compiler

- Python library of graph transformations
  - Each consumes and produces an ONNX graph

- User calls sequence of transformations to create their own flow
  - Example end-to-end flows to get started

```python
model = ModelWrapper("fpga4hep-bw%d.onnx" % bw)
model = model.transform(InferShapes())
model = model.transform(FoldConstants())
model = model.transform(GiveUniqueNodeNames())
model = model.transform(GiveReadableTensorNames())
model = model.transform(Infer DataTypes())
model = model.transform(Streamline())
model = model.transform(ConvertBipolarMatMulToXnorPopcount())
model = model.transform(absorb.AbsorbauxIntoMultiThreshold())
model = model.transform(absorb.AbsorbauxIntoMultiThreshold())
model = model.transform(RoundAndClipThresholds())
model = model.transform(to_hls.InferBinaryStreamingFCLayer())
model = model.transform(to_hls.InferQuantizedStreamingFCLayer())
```

https://github.com/Xilinx/finn
The FINN HLS Library

- An optimized, templated Vivado HLS C++ library of 10+ common DNN layers
- Key component: MVTU (Matrix Vector Threshold Unit)

Customizable datatypes
Flexibility through C++ templates

Streaming I/O
Easily compose modules together, low latency

Customizable parallelism
Control resource footprint & throughput
How does the generated architecture look?

- Stream-in, stream-out FPGA IP block
  - Easy "bump-in-the-wire" integration into streaming systems
  - Simple data movement, fully deterministic
Deployment with PYNQ

FINN

QNN training in PyTorch
Brevitas

Frontends, Transformation, Dataflow Backend
FINN Compiler

Deployment with PYNQ
Deployment with PYNQ for Python Productivity

```python
# numpy.ndarray shapes for i/o
ishape_packed = (1, 49, 2)
oshape_packed = (1, 1, 40)

# set up the DMA
dma.sendchannel.transfer(in_buf : numpy.ndarray)
dma.recvchannel.transfer(out_buf : numpy.ndarray)

# wait until all transfers complete
dma.sendchannel.wait()
dma.recvchannel.wait()
```

- Use PYNQ-provided Python abstractions and drivers
- User provides Numpy array in, calls driver, gets Numpy array out
  - Internally use PYNQ DMA driver to wr/rd NumPy arrays into I/O streams

https://github.com/Xilinx/PYNQ
https://github.com/Xilinx/Alveo-PYNQ
Upcoming FINN Features

Distributed (Multi-FPGA) Dataflow

Scale-out performance

Automated Floorplanning for Multi-SLR FPGAs

Extract more performance from Alveo

Automated Folding

Quickly scale performance & resources without synthesis

100k LUT 10M FPS

10k LUT 1M FPS

100Gb TCP/IP

Video Tutorials

How to train QNNs and deploy them with FINN
LogicNets
How Efficient Does Your DNN Need To Be?
A Spectrum of FPGA Inference Alternatives

Layer-by-layer compute (Matrix of Processing Engines)
Optimizing compiler/scheduler

DPU, overlays (10k+ FPS)

FINN (10M+ FPS)

Generated heterogeneous streaming architecture
Custom topologies, arithmetic and hardware

LogicNets (100M+ FPS)

The DNN is the circuit
Fully unfolded, pipelined, feedforward datapaths

© Copyright 2020 Xilinx
LogicNets at a Glance

Dataset

training

Specialized DNN Topology

(with *high sparsity* + *activation quantization*)

PyTorch

convert

FPGA

LUT

LUT

LUT

Fully-Spatial Circuit Implementation

one full sample every clock
low logic depth, high $F_{clk}$
100M’s of samples per second
Key idea: Quantized Neurons as Truth Tables

**Neuron Equivalent (NEQ)**

- **Inputs:** 6 bits
- **Activation:** thresholding
- **Output:** 1 bit
- **Weights:** float

**Hardware Building Block (HBB)**

- **Input:** 6 bits
- **Output:** 1 bit
- **Conversion:** enumerate inputs

**Hardware Cost:** 1 x LUT6

**PyTorch**

**FPGA**
Prohibitive Cost of Implementing Large Truth Tables

Total input: 12 bits
Total output: 3 bit

Truth Table Size: $4096 \times 15 = 2^{12} \times 3 \times 5$

LUT6 cost of the neuron: ~4095 LUT6s

Co-design DNN topology to avoid intractably large LUTs:
high sparsity + few-bit activations
LogicNets Key Results

**UNSW-NB15 Network Intrusion Detection dataset [Moustafa et al.]**

- ~91% accuracy using ~16k LUTs
- at 471 M samples / second
- with 9 ns latency

**hls4ml JSC dataset [Duarte et al.]**

- ~72% accuracy
- using ~38k LUTs
- at 427 M samples / second
- with 13 ns latency

Jet Tagging (CERN LHC)

[Video: https://youtu.be/qCyK5v84jpl](https://youtu.be/qCyK5v84jpl)
Conclusion

- **FINN**
  - QNN solution stack from training to custom dataflow architecture
  - Full co-design environment with growing library examples
  - Flexible, customizable open-source compiler framework

- **LogicNets**
  - Sparse + quantized topology converts directly to LUT circuit
  - Many exciting future research directions
  - To be open-sourced as part of FINN ecosystem (~Q1 2021)
Join our Growing Open-Source Community!

GitHub
https://xilinx.github.io/finn

Japanese documentation effort + "cucumber sorting"

Stanford University
UNC CHARLOTTE
hackster.io
ULTRA96
NTNU

University courses, student/hobbyist projects

Sketch Recognition (Xilinx Edinburgh)
Thank You