PyTorch-to-FPGA for QNNs with FINN

@ ONNX Community Virtual Meetup, 2020-04-09

Yaman Umuroglu
Xilinx Research Labs
- 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, me (from left to right)
Exploring Custom Hardware + Algorithms for DNNs

k-bit weights and activations, \( k \leq 4 \)
keep all on-chip!

allocated resource ~
compute requirement per layer

FPGA

CNN

sum \( +2 \)
act. \( -4 \)
+1

\( * +1 \)
\( * -1 \)
\( * -1 \)

+1

© Copyright 2020 Xilinx
Few-bit QNNs + FPGA Dataflow: Showcases

High Throughput & Low Latency

MNIST MLP on ZC706
12.3 M FPS
310 ns latency

Low-Power, Real-Time Object Detection

Tincy-YOLO on Ultra96
55 FPS @ 10 W

Complex Topologies

ResNet-50 on Alveo U250
2 ms latency
2000 FPS
The FINN Project: Mission

- Support customizing the algorithms with precision, layer types, topologies
- Support hardware architecture exploration around dataflow execution
- Transparency and flexibility through open source (if not supported, add your own!)
- Open source from the ground-up to encourage community contributions
- End-to-end flow to lower adoption barrier
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

Customization of Algorithm

Customization of Hardware Architecture
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

› Primary backend target is our own Vivado HLS library
  » Templated datatypes, configurable parallelism
How We Use ONNX in the FINN Compiler

Custom quantization annotations for few-bit types

```python
class DataType(Enum):
    FLOAT32 = 0
    BINARY = 1
    BIPOLAR = 2
    UINT2 = 3
    UINT3 = 4
    UINT4 = 5
    UINT8 = 6
    UINT16 = 7
```

Use `quantization_annotation`

ONNX float32 tensor as container, values restricted to few-bit types indicated by type name (string)

Custom op_types at different abstraction levels

Identified by `op_type` and `domain`

Python wrappers to provide implementations for exec, codegen..

Hybrid runtime to verify model correctness

Mix onnxruntime, Vivado HLS C++ and RTL (PyVerilator)

Not performance-oriented
How We Use ONNX in the FINN Compiler

Library of Python graph transformations
Operating directly on ONNX representation
Including constant folding, shape inference, convolution lowering ++

Tensors with custom data layout and packing
e.g. pack 8x1-bit as 1x8-bit

Use Netron for debugging
Visually inspect transformed graphs
Check node attributes, initializers

Example code:
```python
# Model preparation
model = ModelArcher("fgpa4hop-bwGd.onnx" % bw)
model = model.transform(inferShapes())
model = model.transform(constantfolding())
model = model.transform(giveuniqueNodeNames())
model = model.transform(giveReadableTensorNames())
model = model.transform(InferDataTypes())
model = model.transform(streamline())
model = model.transform(convertToNonLatentMultiXnorPopcount())
model = model.transform(absorb.AbsorbingIntoMultiThreshold())
model = model.transform(absorb.AbsorbingIntoMultiThreshold())
model = model.transform(roundAndClipThresholds())
model = model.transform(to_hls.InferBinaryStreamingLayer())
model = model.transform(to_hls.InferQuantizedStreamingLayer())

# Data Type BINARY, carried as float32
ishape_normal = (1, 784)

# Data Type BINARY, carried as float32
# with stream time multiplexing
ishape_folded = (1, 49, 16)

# Data Type BINARY, packed as uint8
# with stream time multiplexing
ishape_packed = (1, 49, 2)
```
Join our Growing Open-Source Community!

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

Japanese documentation effort + «cucumber sorting»

University courses, student/hobbyist projects

Sketch Recognition (Xilinx Edinburgh)
Thank You