# Accelerator Overlays: Spec What You Want, Build What You Need 

Ephrem Wu
FPL 2019
§. XILINX.

## FPGA Overlay Accelerator

## FPGA Silicon



## FPGA Overlay Accelerator



## FPGA Overlay Accelerator

## Application Software



## Vision Neural Networks

₹. XILINX.

## Image Classification



ImageNet Challenge Accuracy Progress 2010-2017


## 2015 - 2019: Faster, Smaller, More Accurate



## Compute

Graphs based on M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," ICML 2019
Single crop, single model

## 2015-2019: Faster, Smaller, More Accurate, and Machine-Made



Hand-designed

## Compute

Graphs based on M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," ICML 2019
Single crop, single model

## 2015-2019: Faster, Smaller, More Accurate, and Machine-Made



## Compute

Graphs based on M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," ICML 2019
Single crop, single model

## 2015-2019: Faster, Smaller, More Accurate, and Machine-Made



## Compute

## 2015-2019: Faster, Smaller, More Accurate, and Machine-Made




Compute
Model Size

Graphs based on M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," ICML 2019
Single crop, single model

## High-Throughput Linear Layers

>High clock rates

## High-Throughput Linear Layers

>High clock rates
>> Build fast (peak DSP clock rate) matrix multipliers on FPGAs

## High-Throughput Linear Layers

>High clock rates
>> Build fast (peak DSP clock rate) matrix multipliers on FPGAs
>> Keep interconnects between memory and compute simple

## High-Throughput Linear Layers

>High clock rates
>> Build fast (peak DSP clock rate) matrix multipliers on FPGAs
>> Keep interconnects between memory and compute simple
>> Can still get 90\% peak clock rate in overlay processor
>> Similar principle to RISC

## High-Throughput Linear Layers

>High clock rates
>> Build fast (peak DSP clock rate) matrix multipliers on FPGAs
>> Keep interconnects between memory and compute simple
>Can still get 90\% peak clock rate in overlay processor
>> Similar principle to RISC
>High compute efficiency

## High-Throughput Linear Layers

> High clock rates
>> Build fast (peak DSP clock rate) matrix multipliers on FPGAs
>> Keep interconnects between memory and compute simple
>> Can still get 90\% peak clock rate in overlay processor
>S Similar principle to RISC
>High compute efficiency
>> Tensors reshaped into matrices
>>Matrices blocked for FPGA matrix multipliers

## High-Throughput Linear Layers

> High clock rates
>>Build fast (peak DSP clock rate) matrix multipliers on FPGAs
>> Keep interconnects between memory and compute simple
>Can still get 90\% peak clock rate in overlay processor
>> Similar principle to RISC
>High compute efficiency
>> Tensors reshaped into matrices
>>Matrices blocked for FPGA matrix multipliers
>> Memory allocation, address generation, and non-linear layers key to success

## A Domain-Specific Architecture for Neural Networks



## A Domain-Specific Architecture for Neural Networks


looks the same?

## A Domain-Specific Architecture for Neural Networks



## A Domain-Specific Architecture for Neural Networks



Thank you, dataflow graphs.

## An Example Command

$$
X=\max \left(A B+c 1^{T}, 0\right)
$$

## X = relu(add(matmul(A, B), broadcast(c))

No explicit nested loops.
Dimensions in tensors.
One tensor memory roundtrip.

## A Domain-Specific Architecture for Neural Networks



## MxV: Don't Leave Performance on the Table



## MxV: Don't Leave Performance on the Table



Compute-Efficient Neural Processor Overlay [FPGA 2019]

> Platform: VCU1525 board with VU9P-2 FPGA
> Case Study: GoogLeNet v1 Inference
>> 3 parallel GoogLeNets with independent weights
>> Each network runs with batch size 1
>> Aggregate 3046 images/sec, 3.3 ms latency
> Compute
>> DSP supertile arrays running at 720 MHz
>> 56\% DSP48 tiles consumed, DSP cycles 95\% utilized
>> Per-tensor block floating-point, 8-/16-bit significands
$>$ Memory
>> No external DRAM on accelerator card used
>> All tensors stored in UltraRAM \& BRAM
>> 1/2 DSP clock rate to simplify timing convergence

## A Domain-Specific Architecture for Neural Networks



## Uniform Quantization

## Original dist. (e.g. in fp32)

## Uniform Quantization

$x$

## Threshold scalar $x$ to be learned



## Original dist.

Scale

Uniform Quantization
$x$


Original dist.

Scale

## Round

Uniform Quantization
$x$

$\left.\operatorname{clip}\left(\frac{z}{s}\right],-Z, Q-Z-1\right)$

Original dist.

Scale

## Round

Clip
§. XILINX.

Uniform Quantization
$x$
$\frac{x}{s}$
$\left\lfloor\frac{x}{s}\right\rceil$
$\left.\operatorname{clip}\left(\left\lvert\, \frac{x}{s}\right.\right\rceil,-Z, Q-Z-1\right)$
Zero point


Uniform Quantization
$x$
$\frac{x}{S}$
$\left\lfloor\frac{x}{s}\right\rceil$
$\operatorname{clip}\left(\left\lfloor\frac{x}{s}\right\rceil,-Z, Q-Z-1\right)$
Zero point
Bins


Original dist.

Scale

## Round

Clip

Exact zero

Uniform Quantization

## $x$ <br> $$
\left\lfloor\frac{x}{s}\right\rceil
$$



# Original dist. 

Scale

## Round

## Clip

§ XILINX.

## INT-to-Float

$$
q=\operatorname{clip}\left(\left\lfloor\frac{x}{s}\right\rfloor,-Z, Q-Z-1\right)
$$



Numbers that enter integer matrix multiplier hardware

## INT-to-Float

$$
q=\operatorname{clip}\left(\left\lfloor\frac{x}{s}\right\rfloor,-Z, Q-Z-1\right)
$$



## Which int8?

| Example | Zero Point Z | Bins Q | Integer Values |
| :--- | :---: | :---: | :---: |
| int8 | 128 | 256 | $[-128,127]$ |

## Which int8?

| Example | Zero Point Z | Bins Q | Integer Values |
| :--- | :---: | :---: | :---: |
| int8 | 128 | 256 | $[-128,127]$ |
| Symmetric int8 | 127 | 255 | $[-127,127]$ |

## Which int8?

| Example | Zero Point Z | Bins Q | Integer Values |
| :--- | :---: | :---: | :---: |
| int8 | 128 | 256 | $[-128,127]$ |
| Symmetric int8 | 127 | 255 | $[-127,127]$ |
| uint8 | 0 | 256 | $[0,255]$ |

## In general...

|  | Zero Point $Z$ | Bins $\mathbf{Q}$ | Integer Values |
| :--- | :---: | :---: | :---: |
| $\mathbf{2}^{b}$ Ievels with zero <br> point $Z$ | $Z \in\left[0,2^{b}-1\right]$ | $2^{\text {b }}$ | $\left[-Z, 2^{\text {b }}-1-Z\right]$ |

## A Domain-Specific Architecture for Neural Networks



## UltraScale+ $+^{\text {TM }}$ Memory

## 

Distributed RAM
(bits to kilobits)

## UltraScale+TM Memory



External DDR DRAM
(10s of gigabits)
Capacity
天 XILINX

## Use FPGA URAM for Capacity and Bandwidth



## A Domain-Specific Architecture for Neural Networks



## A Domain-Specific Architecture for Neural Networks

 and address generation

## A Domain-Specific Architecture for Neural Networks



## 2D Convolution: Tensor View



## 2D Convolution: Tensor View

Output Tensor $\mathcal{Y} \in \mathbb{R}^{H_{2} \times W_{2} \times C_{\text {out }}}$
Output channel 0 Output channel 1 Output channel 2 Output channel 3

Input Tensor $\mathcal{X} \in \mathbb{R}^{H_{1} \times W_{1} \times C_{\text {in }}}$


## 2D Convolution: Tensor View

Output Tensor $y \in \mathbb{R}^{H_{2} \times W_{2} \times C_{\text {out }}}$
Output channel 0 Output channel 1 Output channel 2 Output channel 3


Input Tensor $\mathcal{X} \in \mathbb{R}^{H_{1} \times W_{1} \times C_{\text {in }}}$

$W_{1}$

Although input and output tensors have three axes, convolution is 2 D , not 3 D .
> There's a 2D convolution filter mask for every input-output channel pair ( $3 \times 4=12$ in this example).

## 2D Convolution: Tensor View

Output Tensor $\mathcal{Y} \in \mathbb{R}^{H_{2} \times W_{2} \times C_{\text {out }}}$
Output channel 0 Output channel 1 Output channel 2 Output channel 3

Input Tensor $\mathcal{X} \in \mathbb{R}^{H_{1} \times W_{1} \times C_{\text {in }}}$


Filter Weight Tensor $\mathcal{W} \in \mathbb{R}^{F_{y} \times F_{x} \times C_{i n} \times C_{\text {out }}}$
© Copyright 2019 Xilinx
K. XILINX.

## 2D Convolution: Tensor View

Output Tensor $y \in \mathbb{R}^{H_{2} \times W_{2} \times C_{\text {out }}}$
Output channel 0 Output channel 1 Output channel 2 Output channel 3

Input Tensor $\mathcal{X} \in \mathbb{R}^{H_{1} \times W_{1} \times c_{\text {in }}}$


Filter Weight Tensor $\mathcal{W} \in \mathbb{R}^{F_{y} \times F_{x} \times C_{\text {in }} \times C_{\text {out }}}$
© Copyright 2019 Xilinx
§: XILINX.

## 2D Convolution: From Tensor to Flat View

 one per input channel

## 2D Convolution: Flat View



Parallel convolution sliding windows, one per input channel

## 2D Convolution: Flat View



Parallel convolution sliding windows, one per input channel

## 2D Convolution: Flat Silicon View



- Input channel broadcast
- Per-channel element-wise multiplication with filter weights

Reduction per output channel


## Interpreting Weights (Sort of)



## Reshaping Tensors to Matrices

## 3-Mode Output Tensor

$Y \in \mathbb{R}^{H_{2} \times W_{2} \times C_{\text {out }}}$


Elements per Output Channel

$$
N=28 \times 28=784
$$

## Reshaping Tensors to Matrices

## 3-Mode Output Tensor

$Y \in \mathbb{R}^{H_{2} \times W_{2} \times C_{\text {out }}}$


Elements per Output Channel

$$
N=28 \times 28=784
$$

Output channels

$\square$

Output Matrix
$Y \in \mathbb{R}^{C_{o u t} \times H_{2} W_{2}}$

## Reshaping Tensors to Matrices

3-Mode Output Tensor
$Y \in \mathbb{R}^{H_{2} \times W_{2} \times C_{o u t}}$


Elements per Output Channel

$$
N=28 \times 28=784
$$



4-Mode Weight Tensor
$\mathcal{V} \in \mathbb{R}^{F_{y} \times F_{x} \times C_{i n} \times C_{\text {out }}}$


Input Channels $\times$ Weights per Filter


## Reshaping Tensors to Matrices

3-Mode Output Tensor
$y \in \mathbb{R}^{H_{2} \times W_{2} \times C_{o u t}}$


Elements per Output Channel


## Reshaping Tensors to Matrices for Matrix Multipliers

3-Mode Output Tensor
$y \in \mathbb{R}^{H_{2} \times W_{2} \times C_{o u t}}$


Elements per Output Channel


4-Mode Weight Tensor
$\mathcal{W} \in \mathbb{R}^{F_{y} \times F_{x} \times C_{i n} \times C_{\text {out }}}$


Input Channels $\times$ Weights per Filter


3-Mode Input Tensor $\mathcal{X} \in \mathbb{R}^{H_{1} \times W_{1} \times C_{i n}}$

$N=28 \times 28=784$

Input Matrix $X \in \mathbb{R}^{F_{y} F_{x} C_{i n} \times H_{2} W_{2}}$

## Reshaping Tensors to Matrices for Matrix Multipliers

3-Mode Output Tensor
$y \in \mathbb{R}^{H_{2} \times W_{2} \times C_{o u t}}$


Elements per Output Channel


4-Mode Weight Tensor
$\mathcal{W} \in \mathbb{R}^{F_{y} \times F_{x} \times C_{i n} \times C_{\text {out }}}$


3-Mode Input Tensor $\mathcal{X} \in \mathbb{R}^{\boldsymbol{H}_{1} \times W_{1} \times C_{i n}}$


Input Channels $\times$ Weights per Filter

$N=28 \times 28=784$

Input Matrix $X \in \mathbb{R}^{F_{y} F_{x} C_{i n} \times H_{2} W_{2}}$

## Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection

## Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection Conv2D: Image classification, object detection

## Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection Conv2D: Image classification, object detection Conv3D: Medical imaging, video analytics

## Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Conv2D: Image classification, object detection Conv3D: Medical imaging, video analytics
Conv4D: Light-field imaging for material recognition

## Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Conv2D: Image classification, object detection
Conv3D: Medical imaging, video analytics
Conv4D: Light-field imaging for material recognition
$N$-Mode Output Tensor

$$
y \in \mathbb{R}^{\boldsymbol{o}_{1} \times \cdots \times \boldsymbol{o}_{N} \times C_{\text {out }}}
$$

( $N+1$ )-Mode Weight Tensor
$\mathcal{W} \in \mathbb{R}^{F_{1} \times \cdots \times F_{N} \times C_{i n} \times C_{\text {out }}}$

$$
y=w x
$$

$N$-Mode Input Tensor
$X \in \mathbb{R}^{I_{1} \times \cdots \times I_{N} \times C_{i n}}$

## Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Conv2D: Image classification, object detection
Conv3D: Medical imaging, video analytics
Conv4D: Light-field imaging for material recognition
$N$-Mode Output Tensor

$$
y \in \mathbb{R}^{\boldsymbol{o}_{1} \times \cdots \times \boldsymbol{o}_{N} \times C_{\text {out }}}
$$

( $N+1$ )-Mode Weight Tensor
$\mathcal{W} \in \mathbb{R}^{F_{1} \times \cdots \times F_{N} \times C_{i n} \times C_{\text {out }}}$

## $y=\mathcal{W} x$ <br> $$
Y=W X
$$

Output Matrix
$Y \in \mathbb{R}^{C_{\text {out }} \times\left(0_{1} \cdots 0_{N}\right)}$

Weight Matrix
$W \in \mathbb{R}^{C_{\text {out }} \times\left(F_{1} \cdots F_{N}\right) C_{i n}}$
$N$-Mode Input Tensor
$X \in \mathbb{R}^{I_{1} \times \cdots \times I_{N} \times C_{\text {in }}}$

## Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Conv2D: Image classification, object detection
Conv3D: Medical imaging, video analytics
Conv4D: Light-field imaging for material recognition
$N$-Mode Output Tensor

$$
y \in \mathbb{R}^{\boldsymbol{o}_{1} \times \cdots \times \boldsymbol{o}_{N} \times C_{\text {out }}}
$$

( $N+1$ )-Mode Weight Tensor
$\mathcal{W} \in \mathbb{R}^{F_{1} \times \cdots \times F_{N} \times C_{\text {in }} \times C_{\text {out }}}$
$N$-Mode Input Tensor
$\mathcal{X} \in \mathbb{R}^{I_{1} \times \cdots \times I_{N} \times C_{\text {in }}}$


Weight Matrix
$W \in \mathbb{R}^{C_{\text {out }} \times\left(F_{1} \cdots F_{N}\right) C_{i n}}$

Basically, parenthesize axes to make both hardware and software happy

Output Matrix
$Y \in \mathbb{R}^{C_{\text {out }} \times\left(\boldsymbol{O}_{1} \cdots \boldsymbol{O}_{N}\right)}$

Input Matrix

$$
X \in \mathbb{R}^{\left(F_{1} \cdots F_{N}\right) c_{i n} \times\left(I_{1} \cdots I_{N}\right)}
$$

## Decide What You Need to Optimize in Hardware

> Convolution parameters and flavors
>> Padding, strides, number of axes
>> Dilated, transposed, depthwise-separable,...
>> See Session M2, FPL 19.
> Winograd algorithm: patch size vs. speed-up vs. numerical stability
$>$ Balance ease of use vs. energy efficiency using mixed data types
> Hardware-software co-design
>> Good data movement instructions keep compute fed and software stable
>> FPGAs provides adaptability, especially for non-linear layers

# Natural-Language Processing 

₹. XLINX

## Reading Comprehension (SQuAD Dataset)

Tesla theorized that the application of electricity to the brain enhanced intelligence. In 1912, he crafted "a plan to make dull students bright by saturating them unconsciously with electricity," wiring the walls of a schoolroom and, "saturating [the schoolroom] with infinitesimal electric waves vibrating at high frequency. The whole room will thus, Mr. Tesla claims, be converted into a health-giving and stimulating electromagnetic field or 'bath.'" The plan was, at least provisionally approved by then superintendent of New York City schools, William H. Maxwell.

## Natural Language Processing: Reading Comprehension

Stanford Question and Answering Dataset (SQuAD) 1.1


## Stanford Question and Answering Dataset (SQuAD) 1.1



## Stanford Question and Answering Dataset (SQuAD) 1.1



## SQuAD 2.0 (with Unanswerable Questions)

## SQuAD 2.0 Monthly High Scores




## Convolution vs. Attention



## Self-Attention



## Attention Mechanism

## Associative Array or Content-Addressable Memory



## Attention Mechanism

What does this query match?

## Query



Key


Blue

## Attention Mechanism

## Exact match between query and one of the keys

Query


## Attention Mechanism

Returns associated value. Each query is a one-hot vector.

Query
Key

Value



## Attention in Neural Networks

Attention: Soft Associative Array


## Attention in Neural Networks

What does this query match?


## Attention in Neural Networks

Can we "soft-match?" Dot product measures similarity. Dot

| Query | Key | Product | Value |
| :---: | :---: | :---: | :---: |
| $[-0.2,1.1,-0.9]$ | $[-0.2,1.1,-0.9]$ | 2.1 | $[235,28,35]$ |
|  | $[0.4,0.7,-1.1]$ | 1.7 | $[30,100,14]$ |
|  | $[-0.1,-1.2,1.1]$ | -2.3 | $[74,126,207]$ |

## Attention in Neural Networks

Exponentiate


## Attention in Neural Networks

Normalize. Softmax!

| Query |  | Key | Weight |
| :---: | :---: | :---: | :---: |
| Value |  |  |  |
| $[-0.2,1.1,-0.9]$ | $[-0.2,1.1,-0.9]$ | $58.9 \%$ | $[235,28,35]$ |
|  | $[0.4,0.7,-1.1]$ | $40.3 \%$ | $[30,100,14]$ |
|  | $[-0.1,-1.2,1.1]$ | $0.76 \%$ | $[74,126,207]$ |

## Attention in Neural Networks

Weighted sum of values:

## [151, 58, 28]

| Query | Key | Weight | Value |
| :---: | :---: | :---: | :---: |
| $[-0.2,1.1,-0.9]$ | $[-0.2,1.1,-0.9]$ | $58.9 \%$ | $[235,28,35]$ |
|  | $[0.4,0.7,-1.1]$ | $40.3 \%$ | $[30,100,14]$ |
|  | $[-0.1,-1.2,1.1]$ | $0.76 \%$ | $[74,126,207]$ |

## Attention in Neural Networks

In matrix form for multiple queries in parallel hardware:


Need more softmax than typical vision neural networks.
Can add in ACAP/FPGA.

## Word Embeddings

Words represented as vectors (>500 components)
"You shall know a word by the company it keeps."

Word analogy
is : was :: ? : were


## Word Embeddings

## Word analogy

is : was :: ? : were


## Word Embeddings

## Word analogy

is : was :: ? : were


## Word Embeddings

## Word analogy

## is : was :: are : were



## Word Embeddings

## Word analogy

is : was :: are : were

## Embedding analogy

is - was $\approx$ are - were


## Word Embeddings

## Embeddings from unsupervised training

## Word analogy

is : was :: are : were

Embedding analogy
is - was $\approx$ are - were


## Model Sizes

>Embedding storage in Translation Networks
> Transformer Base: ~32K $\times 512$ fp32
>> OpenAI GPT-2: ~50K $\times 1600$ fp32
>> Above examples considered small
>> Can convert fp32 to 8-bit or 16-bit data types
$>$ Model sizes (Number of parameters)
>> Transformer Base: 61M
>> Open AI GPT-2: 1.6B
$>$ Instruction set and tensor SRAM tuned for random \& burst HBM
> Beam search

# Combining Vision and Language 

§ XILINX

## From Image Classification to Object Detection



## From Reading Comprehension and Object Detection to Visual Reasoning



## Era of Easy Scaling Over

## - Moore's Law

Transistor improvement slows

$\Sigma$
Amdah's Law
Multicore not enough

## Dennard Scaling

Power density rises

40 Years of Processor Performance


[^0]
## Built to Last

>Model commands after NN framework function calls
>> Linear layers: matrix multiplication \& convolution
>> Attention is powerful and good for parallelism
>> Beam search in NLP: Amdahl's Law
>> Activation functions, N-D pooling, layer normalization, NMS, LSTM, RNN, GRU, ...
> Compute is easy. Memory is hard and is key to a adaptable design.
> Neural networks do merge, e.g. captioning, visual reasoning
$>$ ML moves fast. Plan instruction superset upfront for adaptability.
> Implement what you need today. Reconfigure HW to adapt to changes.

Thank you
₹. XILINX

## Adaptable. Intelligent.

§. XILINX


[^0]:    Source: John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018

