Heterogeneous Compute Architectures For Deep Learning In The Cloud

Ken O’Brien, Nicholas Fraser, Michaela Blott
27th March 2019
Outline

- Why FPGAs?
- Deep Learning: Challenges & Solutions
- FINN
- FPGAs to ACAPs
Mega-Trend: Explosion of Data

> Astronomically growing amounts of data
  >> More sensors
  >> More users
  >> More use cases: Genomics (DNA) “Genomical”

We need significantly more compute resources to process and extract patterns / insights from this data!

*Stephens, Zachary D., et al. "Big data: astronomical or genomical?"*
Technology:
End of Moore’s Law & Dennard Scaling

- Economics become questionable
- Power dissipation becomes problematic
Era of Heterogeneous Compute using Accelerators

> Diversification of increasingly heterogenous devices and system
  >> Moving away from standard van Neumann architectures

> True Architectural innovation & Unconventional Computing Systems
Deep Learning
- customized precision arithmetic
What’s the Challenge?
Example: Convolutional Neural Networks

*Forward Pass (Inference)*

For ResNet50:
- 70 Layers
- 7.7 Billion operations
- 25.5 millions of weights

*Basic arithmetic, incredible parallel but Huge Compute and Memory Requirements*
Compute and Memory for Inference

Spectrum of Neural Networks

Inference (1 input) GOPS
average

Inference (1 input) MBytes
average

Huge Compute and Memory Requirements & Variations

*architecture independent
**1 image forward
*** batch = 1
**** int8
Floating Point to Reduced Precision Neural Networks
Deliver Competitive Accuracy

ImageNet Classification Top-5 Error Over Time (ImageNet)

- Float point improvements are slowing down
- Reduced precision competitive accuracy

BNN
CNN
Reduced Precision
Internal
Reducing Precision
*Scales Performance & Reduces Memory*

> Reducing precision shrinks LUT cost
  >> Instantiate 100x more compute within the same fabric

> Potential to reduce memory footprint
  >> NN model can stay on-chip => no memory bottlenecks

<table>
<thead>
<tr>
<th>Precision</th>
<th>Modelsize [MB] (ResNet50)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1b</td>
<td>3.2</td>
</tr>
<tr>
<td>8b</td>
<td>25.5</td>
</tr>
<tr>
<td>32b</td>
<td>102.5</td>
</tr>
</tbody>
</table>

\[ C = \text{size of accumulator} \times \text{size of weight} \times \text{size of activation} \]
Reducing Precision Inherently Saves Power

FPGA:

LSTM - Test Error vs Power (W)

Target Device: ZU7EV
- Ambient temperature: 25 °C
- 12.5% of toggle rate
- 0.5 of Static Probability
- Power reported for PL accelerated block only

ASIC:

Relative Energy Cost

Source: Bill Dally (Stanford), Cadence Embedded Neural Network Summit, February 1, 2017

Design Space Trade-Offs

IMAGENET CLASSIFICATION TOP5% VS COMPUTE COST $F(\text{LUT, DSP})$

- 1b weights
- 2b weights
- 5bit weights
- 8bit weights
- FP weights
- minifloat
- ResNet-50
- Syq

Reduced Precision can:
- reduce cost / resources
- save power
- scale performance
Scaling with FINN
FINN – Tool for Exploration of NNs of FPGAs

- **Design Flow Tool for Quantized Neural Networks**
  - Rapid access to network structure and compute/memory footprint statistics
  - Performance prediction for target device
  - Automatic architecture scaling and generation for target device

- **Multi-stage tool-flow**
  - Frontend
  - Design Space Exploration
  - Backend

- **Binary Network Release Available**
  - [https://github.com/Xilinx/FINN](https://github.com/Xilinx/FINN)
HW Architecture – Dataflow

Input image → Layer 0 → Layer 1 → ... → Layer X-1 → Inference output

Weight buffer → Weight buffer → Weight buffer

Layer 0:
- 37 MOPS Conv.
- W=8 A=3

Layer 1:
- 797 MOPS Conv. + MaxPool
- W=1 A=3

Layer X-1:
- 797 MOPS Conv.
- W=1 A=3

Layer X:
- 21 MOPS Conv.
HW Architecture – Dataflow

Weight buffering in on-chip memory
- High operational intensity for inference
- Small intermediate buffer for feature maps
  - No data reordering between layers
  - Multi-line buffering for convolutions
  - Low latency, high throughput

Input image → Layer 0 → Layer 1 → ... → Layer X-1 → Inference output
HW Architecture – Dataflow

1 Compute engine per layer
- Ad-hoc arithmetic according to layer quantization
HW Architecture – Dataflow

1 Compute engine per each layer
- Adjust parallelism with compute requirements
Frontend Stage – Import and Network Statistics

Neural Network Description (Prototxt)

```prototxt
layer {
  name: "qt_inp"
  type: "Quant"
  bottom: "bn_inp"
  top: "qt_inp"
  quant_param {
    forward_func: "hwqg"
    backward_func: "relu"
    centers: 0.538 centers: clip_thr: 1.614
  }
}
layer {
  name: "ipl"
  type: "BinaryInnerProduct"
  bottom: "qt_inp"
  top: "ipl"
  param {
    lr_mult: 1
decay_mult: 1
  }
inner_product_param {
  num_output: 256
}
```

Per layer operations

<table>
<thead>
<tr>
<th>ops/layer</th>
<th>171320832</th>
<th>892296000</th>
<th>595745280</th>
<th>135364608</th>
<th>127401984</th>
<th>102760448</th>
<th>33554432</th>
<th>8192000</th>
</tr>
</thead>
</table>

Topology summary

<table>
<thead>
<tr>
<th>out_dim</th>
<th>filter_dim</th>
<th>in_chan</th>
<th>out_chan</th>
<th>parallel</th>
<th>in_dim</th>
</tr>
</thead>
<tbody>
<tr>
<td>212.0</td>
<td>12</td>
<td>3</td>
<td>68</td>
<td>1</td>
<td>224</td>
</tr>
<tr>
<td>54.0</td>
<td>5</td>
<td>34</td>
<td>90</td>
<td>2</td>
<td>54</td>
</tr>
<tr>
<td>26.0</td>
<td>3</td>
<td>180</td>
<td>272</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>12.0</td>
<td>3</td>
<td>136</td>
<td>192</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>12.0</td>
<td>3</td>
<td>192</td>
<td>128</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>5.0</td>
<td>1</td>
<td>12544</td>
<td>4096</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>1.0</td>
<td>1</td>
<td>4096</td>
<td>4096</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>1.0</td>
<td>1</td>
<td>4096</td>
<td>1000</td>
<td>1</td>
<td>1000</td>
</tr>
</tbody>
</table>
Design Space Exploration Stage: Balanced Dataflow

Neural Network Description

```json
{layer {
  name: "qt_inp"
  type: "Quant"
  bottom: "bn_inp"
  top: "qt_inp"
  quant_param {
    forward_func: "hwgg"
    backward_func: "relu"
    centers: 0.538 centers: 0.538
    clip_thr: 1.614
  }
}
```

Device Specification File

```json
{
  "name": "Xilinx:KU115",
  "type": "fgga",
  "frequency": 200,
  "resources": {
    "LUT": 1451000,
    "DSP": 5526,
    "BRAM": 75.9,
    "URAM": 0
  }
}
```

Folding Factor Calculation

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIMD</th>
<th>PE</th>
<th>MMV</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvolutionLayer</td>
<td>3</td>
<td>34</td>
<td>1</td>
</tr>
<tr>
<td>ConvolutionLayer</td>
<td>34</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>ConvolutionLayer</td>
<td>36</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>ConvolutionLayer</td>
<td>34</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ConvolutionLayer</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FullyConnectedLayer</td>
<td>32</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>FullyConnectedLayer</td>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FullyConnectedLayer</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Performance Prediction

Achieved FPS: 27400 with 15.83% LUT utilization and 48.73% BRAM utilization

LUTS: 141004/890829 BRAM: 787/1615
Convolutional Layer – Folding

Input Feature Map

Weights

Output Feature Map

Width

Channels

Height

SIMD

SIMD

PE

PE
Design Space Exploration Stage: Balanced Dataflow

Neural Network Description

```json
{  
    "layer": {  
        "name": "qt_inp",
        "type": "Quant",
        "bottom": "bn_inp",
        "top": "qt_inp",
        "quant_param": {  
            "forward_func": "hwgg",
            "backward_func": "relu",
            "clip_thr": 1.614
        }
    }
}
```

Device Specification File

```json
{
    "name": "XLNX:KU115",
    "type": "fgpa",
    "frequency": 200,
    "resources": {  
        "LUT": 1451000,
        "DSP": 5526,
        "BRAM": 75.9,
        "URAM": 0
    }
}
```

Folding Factor Calculation

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIMD</th>
<th>PE</th>
<th>MMV</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvolutionLayer</td>
<td>3</td>
<td>34</td>
<td>1</td>
</tr>
<tr>
<td>ConvolutionLayer</td>
<td>34</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>ConvolutionLayer</td>
<td>36</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>ConvolutionLayer</td>
<td>34</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ConvolutionLayer</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FullyConnectedLayer</td>
<td>32</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>FullyConnectedLayer</td>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FullyConnectedLayer</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Performance Prediction

1: Given a target FPS, what resources are required?
2: Given total resources, what FPS can be achieved?

Achieved FPS: 27400 with 15.83% LUT utilization and 48.73% BRAM utilization
LUTS: 141004/896829 BRAM: 787/1615
Vivado HLS – QNN Library

```
// convolution parameters
unsigned int ConvKernelDim,               // e.g. 3 for a 3x3 conv kernel (assumed square)
unsigned int IFMChannels,                // number of input feature maps
unsigned int IFMDim,                     // width of input feature map (assumed square)
unsigned int OFMChannels,                // number of output feature maps
unsigned int OFMDim,                     // IFMDim-ConvKernelDim+1 or less
unsigned int Stride,

// matrix-vector unit parameters
unsigned int SIMDWidth,                   // number of SIMD lanes
unsigned int PECount,                     // number of PEs
unsigned int WMemCount,                   // entries in each PEs weight memory
unsigned int TMemCount,                   // entries in each PEs threshold memory

// precision parameters
unsigned int WeightsPrecision,           // Number of bits in thresholds
unsigned int ThresholdsPrecision,        // Number of bits in thresholds
unsigned int MacPrecision,               // MAC bitwidth
unsigned int Input_precision,            // Input data bitwidth
unsigned int ActivationPrecision,        // Output data bitwidth
unsigned int ActivationType<8>,          // For first layer use int value

void ConvolutionLayer_Same_Batch(stream<ap_uint<IFMChannels * Input_precision>> & in,
```
Backend Stage - Hardware/ Runtime Generation

Neural Network Description

```json
{layer {
  name: "qt_inp"
  type: "Quant"
  bottom: "bn_inp"
  top: "qt_inp"
  quant_param {
    forward_func: "hwgg"
    backward_func: "relu"
    centers: 0.538 centers: 1.614
  }
}
```

Device Specification File

```json
{
  "name": "XLNX:KU15",
  "type": "fpga",
  "frequency": 200,
  "resources": {
    "LUT": 1451000,
    "DSP": 5526,
    "BRAM": 75.9,
    "URAM": 0
  }
}
```

Optimal Folding Factors

FINN QNN Library

FINN

Hardware Generation

```c
// definition for the streaming QNN accelerator
void DoCompute(stream<ap uint<32>> & inStream, stream<ap uint<32>> & outStream) {
  // variable declarations for internal streams
  stream<ap uint<128>> & FPGABufferLayer_1;
  #pragma HLS stream depth=2 variable=FPGABufferLayer_1
  stream<ap uint<128>> & FPGABufferLayer_3;
  #pragma HLS stream depth=2 variable=FPGABufferLayer_3

  // streaming compute engine for each layer
  #pragma HLS DATAFLOW

  MatrixVectorPrecisionBatch<16, 64, 1, 64, 784, 256, 0, FPGABufferLayer_1, weights_FPGABipolarMatrixThresholdLayer_8, MatrixVectorPrecisionBatch<64, 64, 1, 64, 256, 256, 0, FPGABufferLayer_3, weights_FPGABipolarMatrixThresholdLayer_8, ...
```
Hardware Generation – Network Dataflow Example

> top.cpp
  >> Sequence of layers, 1:1 with network topology

```c
// layer 0 (conv)
// layer config
/*
Using pcCount = 16 simdCount = 3 for engine 0
extracting conv-BN complex, OPM=64 IPV=3 k=3
Layer 0: 64 x 27
Width = 56 TMem = 4 */
#define LB_K 3
#define LB_IFM_CH 3
#define LB_OFM_DIM 32
#define LB_OFM_CH 64
#define LB_OFM_DIM 30
// hardware config
#define LB_SIZHD 3
#define LB_PE 16
#define LB_WMEM 36
#define LB_THEM 4
// layer 1 (conv)
// layer config
/*
Using pcCount = 32 simdCount = 32 for engine 1
extracting conv-BN complex, OPM=64 IPV=64 k=3
Layer 1: 64 x 576
Width = 56 TMem = 2 */
#define L1_K 3
#define L1_IFM_CH 64
#define L1_IFM_DIM 30
#define L1_OFM_CH 64
#define L1_OFM_DIM 28
// hardware config
#define L1_SIZHD 32
#define L1_PE 32
#define L1_WMEM 36
#define L1_THEM 2
```  

> config.h
  >> Finn-generated configuration, with network configuration values + parallelism-specific values

> (possible) params.h
  >> Finn-generated weights values to be hardcoded in the bitstream
Scaling Parallelism

> For each layer, set all SIMD, PE to 1
  - Single MAC

> Until hardware no longer fits on device or FPS target reached
  - Find slowest layer
    * Increase SIMD to next factor of IFM_CHANS or
    * Increase PE to next factor of OFM_CHANS

Goal: Calculate folding factors such that layers produce balanced dataflow
FINN
Performance Results

- Up to 50TOPS measured performance for BNNs
- Multiple precision types supported
  - 8-bit in DSPs, reduced precision in LUTs

<table>
<thead>
<tr>
<th>Network</th>
<th>Platform</th>
<th>Precision (W/A)</th>
<th>Performance (TOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>AWS-F1</td>
<td>1/1</td>
<td>50.8</td>
</tr>
<tr>
<td>CNV</td>
<td>AWS-F1</td>
<td>1/1</td>
<td>12.1</td>
</tr>
<tr>
<td>Tincy-YOLO</td>
<td>AWS-F1</td>
<td>1/3</td>
<td>5.3</td>
</tr>
<tr>
<td>DoReFa-Net/PF</td>
<td>AWS-F1</td>
<td>1/2</td>
<td>11.4</td>
</tr>
</tbody>
</table>

From FPGAs to ACAPs
New Heterogeneous Devices

> From the Xilinx World: Evolution of FPGAs to ACAPs

Up to ~147 TOPS of Int8 performance!
Conclusions

> As Moore's law has ended, heterogeneous accelerated systems have emerged

> High computational demand of machine learning applications is driving hardware development

> Customized dataflow architectures and memory subsystems, custom precisions
  • Dramatic performance scaling and energy efficiency benefits
    • Target Datacenter or Embedded devices
    • Enabling new exciting trade-offs within the design space

> New ACAP devices with AI engines
Thanks!

Adaptable.
Intelligent.