DNNBuilder: an Automated Tool for Building High-Performance DNN Hardware Accelerators for FPGAs

Xiaofan Zhang¹, Junsong Wang², Chao Zhu², Yonghua Lin², JinJun Xiong³, Wen-mei Hwu¹, Deming Chen¹

¹UIUC, ²IBM Research-China, ³IBM T. J. Watson Research Center

ICCAD’18 Best Paper Award
Outline:

1. Background
2. Motivations
3. Automation Flow
4. Accelerator Architecture
5. Design Space Exploration
6. Experimental Results
7. Conclusions
Background

Deploying deep learning workloads in the cloud

Major requirements:
• Throughput performance
• Tail latency
• Power efficiency
Background

Deploying deep learning workloads at the edge

Major requirements:
- Real-time ability
- Energy efficiency design
- Area constraint
Outline:

1. Background
2. Motivations
3. Automation Flow
4. Accelerator Architecture
5. Design Space Exploration
6. Experimental Results
7. Conclusions
Motivation

Try FPGAs for both cloud- and edge-computing

😊 FPGAs deliver improved latency & energy efficiency (vs. CPUs, GPUs)

But..

🤔 FPGA have limited computation & memory resources (DSPs, BRAMs)

😢 Large design & test efforts (RTL programming, HW verification...)

😭 Challenges in resource allocation for unbalanced DNN layers

We need An end-to-end automated tool for mapping DNN to FPGAs
Outline:

1. Background
2. Motivations
3. Automation Flow
4. Accelerator Architecture
5. Design Space Exploration
6. Experimental Results
7. Conclusions
Automation Design Flow

3-step-solution as *Design, Generation, & Execution*

- Low latency
- High throughput
- Efficient use of FPGA on-chip memory
- Auto on-chip resource allocation

*To bridge the gap between fast DNN construction in software and slow hardware implementation*
Architecture

Overview of the proposed accelerator design

➢ A fine-grained layer-based pipeline structure
   1) Higher throughput
   2) Better support of streaming inputs
   3) Higher efficiency with dedicated design for each DNN layer

➢ A column-based cache scheme
   1) Lower latency, lower on-chip MEM demands
   2) Support HD input
   3) Real-time capability
Architecture

A fine-grained layer-based pipelined architecture

Proposed design

Higher throughput vs. recurrent structure

Lower latency vs. conventional pipeline structure

Reduce 7.7x latency for running YOLO
Architecture

Pipelined stages instantiated on FPGA

- **2-dim parallelism**
  - KPF - kernel parallel factor
  - CPF - channel parallel factor

- **Arbitrary quantization**
  - DW - bit-width for feature map
  - WW - bit-width for weight/bias
Architecture

RTL IPs for different DNN layers

- Adjustable parallel factor = CPF x KPF (more/less DSP utilization)
- On-chip buffers for sufficient data supply
Architecture

A column-based cache scheme

- Save on-chip memory
- Adjust data reuse factor

For example:
Kernel size = 3
Stride = 1

4 slices cache on-chip instead of keeping the whole feature maps
Architecture

Column-based cache scheme

Reduce 43x BRAM usage for running YOLO

BRAM usage reduction for keep feature maps
320x ~ 7x
43x on average
Design Space Exploration

An automatic resource allocator

Step 1: Computation allocation

Total BW 10 GB/S;  Total capability: 100 GOPS

\[ L_i = \alpha \frac{C_i}{R_i} \quad (1) \]

\[ TP = \frac{1}{\max \{L_i\}} = \frac{1}{\max \{\alpha C_i / R_i\}} \quad (2) \]

\[ L_{total} \geq \alpha \frac{C_1}{R_1} \Rightarrow \sum_i R_i L_{total} \geq \sum_i C_i \alpha \]

\[ L_{total} \geq \alpha \sum_i \frac{C_i}{R_i} \quad (3) \]

\[ \frac{C_1}{R_1} = \frac{C_2}{R_2} = \cdots = \frac{C_i}{R_i} \quad (4) \]

CTC: Computation to Communication Ratio (GOPS/Byte)

Mem. bound ↔ Comp. bound

Conv1 -> 15 GOPS
Conv2 -> 15 GOPS
Conv3 -> 21 GOPS
Conv4 -> 8 GOPS
Conv5 -> 5 GOPS
FC layer maximum usage 5 GOPS
Design Space Exploration

An automatic resource allocator

Step 2: memory bandwidth adjustment

To meet the BW constraint

- Total capability 100 GOPS
- Total BW 10 GB/S

CTC: Computation to Communication Ratio (GOPS/Byte)

Mem. bound ↔ Comp. bound

Required mem. BW drop

CTC increase
Outline:

1. Background
2. Motivations
3. Automation Flow
4. Accelerator Architecture
5. Design Space Exploration
6. Experimental Results
7. Conclusions
Experimental Results

Case study: real-time pedestrian/cyclist/car detection

Yolo9000 with HD input (1280x384, 20FPS) is mapped to Xilinx Zynq 706 FPGA @ 200MHz
Experimental Results

Case study: real-time pedestrian/cyclist/car detection
Experimental Results

Accuracy results after 16-bit & 8-bit quantization

Table 1: Top-1 Accuracy for image classification

<table>
<thead>
<tr>
<th>Network</th>
<th>Float32</th>
<th>Fix16</th>
<th>Fix16+f.-t. in Design</th>
<th>Fix8</th>
<th>Fix8+f.-t. in Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexnet</td>
<td>55.7%</td>
<td>53.3%</td>
<td>55.1% (0.6% ↓)</td>
<td>51.6%</td>
<td>53.4% (2.3% ↓)</td>
</tr>
<tr>
<td>ZF</td>
<td>58.0%</td>
<td>56.3%</td>
<td>57.6% (0.4% ↓)</td>
<td>54.2%</td>
<td>56.2% (1.8% ↓)</td>
</tr>
<tr>
<td>VGG16</td>
<td>68.3%</td>
<td>67.0%</td>
<td>69.3% (1.0% ↑)</td>
<td>63.7%</td>
<td>69.2% (0.9% ↑)</td>
</tr>
</tbody>
</table>

Table 2: Accuracy for object detection (AP@IOU=0.5)

<table>
<thead>
<tr>
<th>Network</th>
<th>Precision</th>
<th>Car</th>
<th>Pedestrian</th>
<th>Cyclist</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOLO (HD)</td>
<td>Float32</td>
<td>88.9%</td>
<td>64.9%</td>
<td>72.5%</td>
<td>75.5%</td>
</tr>
<tr>
<td></td>
<td>Fix16+f.-t. in Design</td>
<td>88.9%</td>
<td>65.0%</td>
<td>73.2%</td>
<td>75.7% (0.2% ↑)</td>
</tr>
<tr>
<td></td>
<td>Fix8+f.-t. in Design</td>
<td>88.9%</td>
<td>65.2%</td>
<td>72.6%</td>
<td>75.6% (0.1% ↑)</td>
</tr>
</tbody>
</table>

*f.-t. in Design represents the accuracy results are collected after retraining and fine-tuning
## Experimental Results

### Comparison: Embedded FPGAs for edge-devices

<table>
<thead>
<tr>
<th>Reference</th>
<th>[1]</th>
<th>[2]</th>
<th>DNNBuilder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categories</td>
<td>Edge-computing platforms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPGA chip</td>
<td>Zynq XC7Z045</td>
<td>Zynq XC7Z045</td>
<td>Zynq XC7Z045</td>
</tr>
<tr>
<td>Frequency</td>
<td>150 MHz</td>
<td>100 MHz</td>
<td>200 MHz</td>
</tr>
<tr>
<td>Network</td>
<td>VGG</td>
<td>VGG</td>
<td>VGG</td>
</tr>
<tr>
<td>Precision</td>
<td>Fix16</td>
<td>Fix16</td>
<td>Fix16 (Fix8)</td>
</tr>
<tr>
<td>DSPs (used/total)</td>
<td>780/900</td>
<td>824/900</td>
<td>680/900</td>
</tr>
<tr>
<td>DSP Efficiency</td>
<td>44.0%</td>
<td>69.6%</td>
<td>96.2%</td>
</tr>
<tr>
<td>Performance (GOPS)</td>
<td>137</td>
<td>230</td>
<td>262 (524)</td>
</tr>
<tr>
<td>Power Efficiency (GOPS/W)</td>
<td>14.2</td>
<td>24.4</td>
<td>36.4 (72.8)</td>
</tr>
</tbody>
</table>

**Zynq XC7Z045**
- LUT: 218,600
- FF: 437,200
- BRAM: 545
- DSP: 900

**KU115**
- LUT: 663,360
- FF: 1,326,720
- BRAM: 2160
- DSP: 5520

---

### Comparison: High-performance FPGAs for cloud computing

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Categories</td>
<td>Cloud-computing platforms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPGA chip</td>
<td>Arria10-1150</td>
<td>Arria10-1150</td>
<td>Stratix-V GXA7 + CPU</td>
<td>KU115</td>
</tr>
<tr>
<td>Frequency</td>
<td>303 MHz</td>
<td>385 MHz</td>
<td>200 MHz &amp; 2~3 GHz(CPU)</td>
<td>235 MHz</td>
</tr>
<tr>
<td>Network</td>
<td>Alexnet</td>
<td>VGG</td>
<td>Alexnet</td>
<td>VGG</td>
</tr>
<tr>
<td>Precision</td>
<td>Float16</td>
<td>Fix16</td>
<td>Fix16 in FPGA</td>
<td>Fix16 (Fix8)</td>
</tr>
<tr>
<td>DSPs (used/total)</td>
<td>2952/3036</td>
<td>2756/3036</td>
<td>512/512 in FPGA</td>
<td>4318/5520</td>
</tr>
<tr>
<td>DSP Efficiency</td>
<td>77.3%</td>
<td>84.3%</td>
<td>-</td>
<td>99.1%</td>
</tr>
<tr>
<td>Performance (GOPS)</td>
<td>1382</td>
<td>1790</td>
<td>781</td>
<td>2011 (4022)</td>
</tr>
<tr>
<td>Power Efficiency (GOPS/W)</td>
<td>30.7</td>
<td>47.8</td>
<td>-</td>
<td>90.2 (180.4)</td>
</tr>
</tbody>
</table>

**Zynq XC7Z045**
- Peaking at 526 GOPS

**KU115**
- Peaking at 4218 GOPS

---

## Experimental Results

**Comparison:** AlexNet inference performance GPU vs FPGA

<table>
<thead>
<tr>
<th>Platform</th>
<th>Precision</th>
<th>Batch</th>
<th>Throughput (img./S)</th>
<th>Power (W)</th>
<th>Efficiency (img./S/W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNNBuilder (ZC706)</td>
<td>Fix16, Fix8</td>
<td>1, 2</td>
<td>170, 340</td>
<td>7.2</td>
<td>23.6, 47.2</td>
</tr>
<tr>
<td>GPU-TX2[26]</td>
<td>Float16</td>
<td>2</td>
<td>250</td>
<td>10.7</td>
<td>23.3</td>
</tr>
<tr>
<td>DNNBuilder (KU115)</td>
<td>Fix16, Fix8</td>
<td>3, 6</td>
<td>1126, 2252</td>
<td>22.9</td>
<td>49.2, 98.3</td>
</tr>
<tr>
<td>GPU-TitanX</td>
<td>Float32</td>
<td>128</td>
<td>5120</td>
<td>227.0</td>
<td>22.6</td>
</tr>
</tbody>
</table>

[26] Nvidia. Nvidia Jetson TX2 delivers twice the intelligence to the edge.
Outline:

1. Background
2. Motivations
3. Automation Flow
4. Accelerator Architecture
5. Design Space Exploration
6. Experimental Results
7. Conclusions
Conclusions

- We presented DNNBuilder for building DNN accelerator on FPGAs
  1) an automation tool (*Design, Generation, and Execution*)
  2) a fine-grained layer-based pipeline architecture
  3) a column-based cache scheme
  4) a automatic resource allocation algorithm

- We delivered the state-of-the-art performance and power efficiency
  1) the best throughput: 4022 (KU115) and 526 GOPS (ZC706)
  2) the best efficiency: 180.2 (KU115) and 72.8 GOPS/W (ZC706)
Q & A

#AI Research Week
hosted by MIT-IBM Watson AI Lab