HybridDNN: A Framework for High-Performance Hybrid DNN Accelerator Design and Implementation

Hanchen Ye¹, Xiaofan Zhang¹, Zhize Huang², Gengsheng Chen², Deming Chen¹
¹University of Illinois at Urbana-Champaign,
²Fudan University
Hanchen Ye

- I’m a PhD student in University of Illinois at Urbana-Champaign (UIUC), advised by Prof. Deming Chen. I obtained my Bachelor and Master degree in Fudan University in 2017 and 2019, respectively. My research interests lie in the area of Hardware Acceleration, High-Level Synthesis (HLS), and Deep Learning.
- Personal website: hanchenye.com/about/
Outline

• Motivation
• HybridDNN Framework
• Accelerator Design
• Design Space Exploration
• Experimental Results
• Conclusion
Motivation (1)

Goals

• Easy and fast deployment
• Flexibility regarding different applications / scenarios
• High performance & efficiency

Solution?

• Complete and automated design flow which is flexible for various DNNs and FPGAs
• Winograd fast algorithm
Motivation (1)

**Goals**

- Easy and fast deployment
- Flexibility regarding different applications / scenarios
- High performance & efficiency

**Solution?**

- Complete and automated design flow which is flexible for various DNNs and FPGAs
- Winograd fast algorithm

Cloud FPGAs

Embedded FPGAs
Motivation (2)

• Winograd Algorithm $F(m \times m, r \times r)$:
  • $Y = A^T \left[ [GgG^T] \odot [B^T dB] \right] A$
  • $G, B, A$: Winograd transformation matrices
  • $\odot$: Element-Wise Matrix Multiply (EWMM)

• Pros:
  • Reduce MAC number of convolution by $2.25 \times (m = 2, r = 3)$ to $4 \times (m = 4, r = 3)$

• Cons:
  • Not friendly to fully-connected (FC) layers, $1 \times 1$ convolutional (CONV) layers, and $>1$ stride size => Low flexibility for various DNNs
  • High memory bandwidth demand => Low flexibility for various FPGAs
Motivation (2)

• Winograd Algorithm $F(m \times m, r \times r)$:
  - $Y = A^T \left[ [GgG^T] \odot [B^T dB] \right] A$
  - $G, B, A$: Winograd transformation matrices
  - $\odot$: Element-Wise Matrix Multiply (EWMM)

• Pros:
  - Reduce MAC number of convolution by $2.25 \times (m = 2, r = 3)$ to $4 \times (m = 4, r = 3)$

• Cons:
  - Not friendly to fully-connected (FC) layers, $1 \times 1$ convolutional (CONV) layers, and $>1$ stride size
    - $\Rightarrow$ Low flexibility for various DNNs
  - High memory bandwidth demand
    - $\Rightarrow$ Low flexibility for various FPGAs
Motivation (3)

- Winograd Algorithm $F(m \times m, r \times r)$:
  - $Y = A^T[(GgG^T)\odot(B^TdB)]A$
  - $G, B, A$: Winograd transformation matrices
  - $\odot$: Element-Wise Matrix Multiply (EWMM)

- Pros:
  - Reduce MAC number of convolution by $2.25 \times (m = 2, r = 3)$ to $4 \times (m = 4, r = 3)$

- Cons:
  - Not friendly to fully-connected (FC) layers, $1\times1$ convolutional (CONV) layers, and $>1$ stride size
    => Low flexibility for various DNNs
  - High memory bandwidth demand
    => Low flexibility for various FPGAs

**Architecture?**

× Homogeneous Winograd
✓ Hybrid Spatial / Winograd
Motivation (3)

• Winograd Algorithm $F(m \times m, r \times r)$:
  • $Y = A^T[[GgG^T] \odot [B^TdB]]A$
  • $G, B, A$: Winograd transformation matrices
  • $\odot$: Element-Wise Matrix Multiply (EWMM)

• Pros:
  • Reduce MAC number of convolution by $2.25 \times (m=2, r=3)$ to $4 \times (m=4, r=3)$

• Cons:
  • Not friendly to fully-connected (FC) layers, $1 \times 1$ convolutional (CONV) layers, and $>1$ stride size
    => Low flexibility for various DNNs
  • High memory bandwidth demand
    => Low flexibility for various FPGAs

Architecture?

× Homogeneous Winograd
✓ Hybrid Spatial / Winograd

Challenges

• Accelerator design:
  • Computation resource reuse
  • Memory management
  • Inter-layer context switch
  • … …

• Large design space
  • Modeling
  • Exploration
HybridDNN Framework (1)
HybridDNN Framework (2)

Input of Step 1 & 2

- FPGA Specification:
  - On-chip resources (LUT, DSP, and BRAM)
  - External memory bandwidth
  - Number of dies (for cloud FPGA)

- DNN Model:
  - DNN architecture description
  - Pre-trained weights
Output of Step1 & 2

- **DNN Mapping Strategy:**
  - Dataflow, CONV mode of each layer
  - Partition strategy of each layer

- **HLS Template Configuration:**
  - Parallel factors $PI$, $PO$, and $PT$
  - Number of instances $NI$
HybridDNN Framework (4)

Output of Step 3
- Instructions & Data Files:
  - Instructions (.bin)
  - Reordered weights data (.bin)
  - Host executable file
- FPGA Bitstream

Step 3
- Compiler
- HLS Synth.
- RTL Impl.

Step 4
- Inst. & Data Files
- FPGA DNN App.
- FPGA Runtime

Instructions & Data Files:
- Instructions (.bin)
- Reordered weights data (.bin)
- Host executable file

FPGA Bitstream
Accelerator Design (1)
Accelerator Design (1)

Efficient and Flexible

- Instruction-based accelerator with customized instruction set
- CTRL Module:
  - Load, decode, and distribute instructions to functional modules
Accelerator Design (1)

Efficient and Flexible

- Instruction-based accelerator with customized instruction set
- CTRL Module:
  - Load, decode, and distribute instructions to functional modules
Accelerator Design (1)

- Instruction-based accelerator with customized instruction set
- CTRL Module:
  - Load, decode, and distribute instructions to functional modules

Efficient and Flexible
Accelerator Design (2)
Accelerator Design (2)

• **LOAD_INP & _WGT Module:**
  - Load input feature maps and weights from external memory

• **COMP Module:**
  - Carry out the computation

• **SAVE Module:**
  - Write back output feature maps
Accelerator Design (2)

• **LOAD_INP & _WGT Module:**
  - Load input feature maps and weights from external memory

• **COMP Module:**
  - Carry out the computation

• **SAVE Module:**
  - Write back output feature maps
Accelerator Design (2)

• LOAD_INP & _WGT Module:
  • Load input feature maps and weights from external memory

• COMP Module:
  • Carry out the computation

• SAVE Module:
  • Write back output feature maps
Accelerator Design (2)

- **LOAD_INP & _WGT Module:**
  - Load input feature maps and weights from external memory

- **COMP Module:**
  - Carry out the computation

- **SAVE Module:**
  - Write back output feature maps

*Module-level Pipeline*
Accelerator Design (2)
Accelerator Design (3)

- **Processing Engine (PE):**
  - Reused by Winograd and Spatial CONV
  - \( PT \times PT \) GEMM Cores

- **GEMM Cores:**
  - MAC broadcast-array paralleled along input (\( P_I \)) and output channels (\( P_O \))

- **GEMM Cores Organization:**
  - Spatial: A large broadcast array
  - Winograd: Compute independently
Accelerator Design (3)

- **Processing Engine (PE):**
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores

- **GEMM Cores:**
  - MAC broadcast-array paralleled along input ($PI$) and output channels ($PO$)

- **GEMM Cores Organization:**
  - Spatial: A large broadcast array
  - Winograd: Compute independently
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores

- GEMM Cores:
  - MAC broadcast-array paralleled along input ($PI$) and output channels ($PO$)

- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently
Accelerator Design (3)

- **Processing Engine (PE):**
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores

- **GEMM Cores:**
  - MAC broadcast-array paralleled along input ($PI$) and output channels ($PO$)

- **GEMM Cores Organization:**
  - Spatial: A large broadcast array
  - Winograd: Compute independently
Accelerator Design (3)

• Processing Engine (PE):
  • Reused by Winograd and Spatial CONV
  • $PT \times PT$ GEMM Cores

• GEMM Cores:
  • MAC broadcast-array paralleled along input ($PI$) and output channels ($PO$)

• GEMM Cores Organization:
  • Spatial: A large broadcast array
  • Winograd: Compute independently
Accelerator Design (3)

- **Processing Engine (PE):**
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores

- **GEMM Cores:**
  - MAC broadcast-array paralleled along input ($PI$) and output channels ($PO$)

- **GEMM Cores Organization:**
  - Spatial: A large broadcast array
  - Winograd: Compute independently
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores

- GEMM Cores:
  - MAC broadcast-array paralleled along input ($PI$) and output channels ($PO$)

- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently
Accelerator Design (3)

- **Processing Engine (PE):**
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores

- **GEMM Cores:**
  - MAC broadcast-array paralleled along input ($PI$) and output channels ($PO$)

- **GEMM Cores Organization:**
  - Spatial: A large broadcast array
  - Winograd: Compute independently
Accelerator Design (3)

- **Processing Engine (PE):**
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores

- **GEMM Cores:**
  - MAC broadcast-array paralleled along input ($PI$) and output channels ($PO$)

- **GEMM Cores Organization:**
  - Spatial: A large broadcast array
  - Winograd: Compute independently
Accelerator Design (4)

- **Load & Save Manager:**
  - Can switch between Winograd and Spatial CONV mode

- **Spatial Mode**

- **Winograd Mode:**
  - LUT-based Winograd transformation between two domains
Accelerator Design (4)

• **Load & Save Manager:**
  • Can switch between Winograd and Spatial CONV mode

• **Spatial Mode**
  • **Winograd Mode:**
    • LUT-based Winograd transformation between two domains
Accelerator Design (4)

- **Load & Save Manager:**
  - Can switch between Winograd and Spatial CONV mode

- **Spatial Mode**
  - **Winograd Mode:**
    - LUT-based Winograd transformation between two domains
Accelerator Design (4)

• Load & Save Manager:
  • Can switch between Winograd and Spatial CONV mode

• Spatial Mode
  • Winograd Mode:
    • LUT-based Winograd transformation between two domains
Accelerator Design (4)

- **Load & Save Manager:**
  - Can switch between Winograd and Spatial CONV mode

- **Spatial Mode**

- **Winograd Mode:**
  - LUT-based Winograd transformation between two domains
Accelerator Design (4)

- **Load & Save Manager:**
  - Can switch between Winograd and Spatial CONV mode

- **Spatial Mode**

- **Winograd Mode:**
  - LUT-based Winograd transformation between two domains
Accelerator Design (4)

• Load & Save Manager:
  • Can switch between Winograd and Spatial CONV mode

• Spatial Mode

• Winograd Mode:
  • LUT-based Winograd transformation between two domains

Resource Overhead:
LUTs for Winograd transformation
**Design Space Exploration**

<table>
<thead>
<tr>
<th>HW Parameters</th>
<th>PI, PO, PT, NI</th>
</tr>
</thead>
</table>
| SW Parameters | \{mode_1, mode_2, ...mode_L\},  
\{dataflow_1, dataflow_2, ...dataflow_L\} |
| Constraints   | PI ≥ PO ≥ 1, PT ∈ \{4, 6\},  
N_{LUT} < LUT, N_{DSP} < DSP, N_{BRAM} < BRAM,  
mode_l ∈ \{"spat", "wino"\}, dataflow_l ∈ \{"is", "ws"\} |
| Objective     | \[\sum_{l=1}^{L} T_l\] |

**Presume**
- Totally $L$ CONV / FC layers
- $is$ and $ws$ means input and weight stationary
- $T_l$: latency of the $l$-th layer

**Step0:** Latency and FPGA resource (LUT, DSP, and BRAM) modeling

**Step1:** Search for design candidates with different HW parameters

**Step2:** Search for optimal SW parameters combination for each candidate

**Step3:** Select the design candidate with the lowest latency
# Design Space Exploration

<table>
<thead>
<tr>
<th>HW Parameters</th>
<th>PI, PO, PT, NI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW Parameters</td>
<td>{mode_1, mode_2, ..., mode_L}, {dataflow_1, dataflow_2, ..., dataflow_L}</td>
</tr>
<tr>
<td>Constraints</td>
<td>PI \geq PO \geq 1, PT \in {4, 6}, N_{LUT} &lt; LUT, N_{DSP} &lt; DSP, N_{BRAM} &lt; BRAM, mode_l \in {&quot;spat&quot;, &quot;wino&quot;}, dataflow_l \in {&quot;is&quot;, &quot;ws&quot;}</td>
</tr>
<tr>
<td>Objective</td>
<td>\sum_{l=1}^{L} T_l</td>
</tr>
</tbody>
</table>

**Presume**
- Totally $L$ CONV / FC layers
- $is$ and $ws$ means input and weight stationary
- $T_l$: latency of the $l$-th layer

- **Step0**: Latency and FPGA resource (LUT, DSP, and BRAM) modeling
- **Step1**: Search for design candidates with different HW parameters
- **Step2**: Search for optimal SW parameters combination for each candidate
- **Step3**: Select the design candidate with the lowest latency
Design Space Exploration

<table>
<thead>
<tr>
<th>HW Parameters</th>
<th>PI, PO, PT, NI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW Parameters</td>
<td>{mode_1, mode_2, …mode_L}, {dataflow_1, dataflow_2, …dataflow_L}</td>
</tr>
<tr>
<td>Constraints</td>
<td>PI ≥ PO ≥ 1, PT ∈ {4, 6}, N_{LUT} &lt; LUT, N_{DSP} &lt; DSP, N_{BRAM} &lt; BRAM, mode_i ∈ {&quot;spat&quot;, &quot;wino&quot;}, dataflow_i ∈ {&quot;is&quot;, &quot;ws&quot;}</td>
</tr>
<tr>
<td>Objective</td>
<td>(\sum_{l=1}^{L} T_l)</td>
</tr>
</tbody>
</table>

**Presume**
- Totally \(L\) CONV / FC layers
- \(is\) and \(ws\) means input and weight stationary
- \(T_l\): latency of the \(l\)-th layer

**Step0:** Latency and FPGA resource (LUT, DSP, and BRAM) modeling
**Step1:** Search for design candidates with different HW parameters
**Step2:** Search for optimal SW parameters combination for each candidate
**Step3:** Select the design candidate with the lowest latency
Design Space Exploration

<table>
<thead>
<tr>
<th>HW Parameters</th>
<th>PI, PO, PT, NI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW Parameters</td>
<td>{mode_1, mode_2, …mode_L}, {dataflow_1, dataflow_2, …dataflow_L}</td>
</tr>
<tr>
<td>Constraints</td>
<td>PI ≥ PO ≥ 1, PT ∈ {4, 6}, N_{LUT} &lt; LUT, N_{DSP} &lt; DSP, N_{BRAM} &lt; BRAM, mode_l ∈ {&quot;spat&quot;, &quot;wino&quot;}, dataflow_l ∈ {&quot;is&quot;, &quot;ws&quot;}</td>
</tr>
<tr>
<td>Objective</td>
<td>(\sum_{l=1}^{L} T_l)</td>
</tr>
</tbody>
</table>

**Presume**

- Totally \(L\) CONV / FC layers
- \(is\) and \(ws\) means input and weight stationary
- \(T_l\): latency of the \(l\)-th layer

**Step0**: Latency and FPGA resource (LUT, DSP, and BRAM) modeling
**Step1**: Search for design candidates with different HW parameters
**Step2**: Search for optimal SW parameters combination for each candidate
**Step3**: Select the design candidate with the lowest latency
Design Space Exploration

<table>
<thead>
<tr>
<th>HW Parameters</th>
<th>( PI, PO, PT, NI )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW Parameters</td>
<td>{mode_1, mode_2, \ldots mode_L}, {dataflow_1, dataflow_2, \ldots dataflow_L}</td>
</tr>
<tr>
<td>Constraints</td>
<td>( PI \geq PO \geq 1, PT \in {4, 6}, N_{LUT} &lt; LUT, N_{DSP} &lt; DSP, N_{BRAM} &lt; BRAM, ) ( mode_l \in {&quot;spat&quot;, &quot;wino&quot;}, dataflow_l \in {&quot;is&quot;, &quot;ws&quot;} )</td>
</tr>
<tr>
<td>Objective</td>
<td>( \sum_{l=1}^{L} T_l )</td>
</tr>
</tbody>
</table>

**Presume**
- Totally \( L \) CONV / FC layers
- \( is \) and \( ws \) means input and weight stationary
- \( T_l \): latency of the \( l \)-th layer

- **Step0**: Latency and FPGA resource (LUT, DSP, and BRAM) modeling
- **Step1**: Search for design candidates with different HW parameters
- **Step2**: Search for optimal SW parameters combination for each candidate
- **Step3**: Select the design candidate with the lowest latency
Experimental Results (1)

- **Xilinx VU9P Configuration:**
  - \( PI = 4, PO = 4, PT = 6, NI = 6 \)
- **PYNQ-Z1 Configuration:**
  - \( PI = 4, PO = 4, PT = 4, NI = 1 \)

<table>
<thead>
<tr>
<th>Device</th>
<th>[26]</th>
<th>[4]</th>
<th>[6]</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Xilinx VU9P</td>
<td>Arria10</td>
<td>Xilinx VU9P</td>
<td>Xilinx VU9P</td>
</tr>
<tr>
<td>Precision</td>
<td>VGG16</td>
<td>VGG16</td>
<td>VGG16</td>
<td>VGG16</td>
</tr>
<tr>
<td>Freq.(MHz)</td>
<td>16-bit</td>
<td>16-bit</td>
<td>16-bit</td>
<td>12-bit*</td>
</tr>
<tr>
<td>DSPs</td>
<td>210</td>
<td>385</td>
<td>214</td>
<td>167</td>
</tr>
<tr>
<td>CNN Perf.(GOPS)</td>
<td>4096</td>
<td>2756</td>
<td>5349</td>
<td>5163</td>
</tr>
<tr>
<td>Power(W)</td>
<td>1510</td>
<td>1790</td>
<td>1828.6</td>
<td>3375.7</td>
</tr>
<tr>
<td>LUTs</td>
<td>37.5</td>
<td>49.3</td>
<td>45.9</td>
<td>2.6</td>
</tr>
<tr>
<td>DSP Effi. (GOPS/DSP)</td>
<td>0.37</td>
<td>0.65</td>
<td>0.34</td>
<td>0.65</td>
</tr>
<tr>
<td>Energy Effi. (GOPS/W)</td>
<td>NA</td>
<td>47.78</td>
<td>37.1</td>
<td>73.5</td>
</tr>
</tbody>
</table>

*DNN parameters are quantized to 8-bit; input feature maps are set to 12-bit in PE due to the Winograd matrix transformation*
Experimental Results (1)

<table>
<thead>
<tr>
<th>Device</th>
<th>[26]</th>
<th>[4]</th>
<th>[6]</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xilinx VU9P</td>
<td>Xilinx</td>
<td>Arria10</td>
<td>Xilinx</td>
<td>Xilinx</td>
</tr>
<tr>
<td>Model</td>
<td>VGG16</td>
<td>VGG16</td>
<td>VGG16</td>
<td>VGG16</td>
</tr>
<tr>
<td>Precision</td>
<td>16-bit</td>
<td>16-bit</td>
<td>16-bit</td>
<td>12-bit*</td>
</tr>
<tr>
<td>Freq.(MHz)</td>
<td>210</td>
<td>385</td>
<td>214</td>
<td>167</td>
</tr>
<tr>
<td>DSPs</td>
<td>4096</td>
<td>2756</td>
<td>5349</td>
<td>5163</td>
</tr>
<tr>
<td>CNN Perf.(GOPS)</td>
<td>1510</td>
<td>1790</td>
<td>1828.6</td>
<td>3375.7</td>
</tr>
<tr>
<td>Power(W)</td>
<td>NA</td>
<td>37.5</td>
<td>49.3</td>
<td>45.9</td>
</tr>
<tr>
<td>DSP Effi. (GOPS/DSP)</td>
<td>0.37</td>
<td>0.65</td>
<td>0.34</td>
<td>0.65</td>
</tr>
<tr>
<td>Energy Effi. (GOPS/W)</td>
<td>NA</td>
<td>47.78</td>
<td>37.1</td>
<td>73.5</td>
</tr>
</tbody>
</table>

*DNN parameters are quantized to 8-bit; input feature maps are set to 12-bit in PE due to the Winograd matrix transformation

- Xilinx VU9P Configuration:
  - $PI = 4, PO = 4, PT = 6, NI = 6$
- PYNQ-Z1 Configuration:
  - $PI = 4, PO = 4, PT = 4, NI = 1$

<table>
<thead>
<tr>
<th></th>
<th>LUTs</th>
<th>DSPs</th>
<th>18Kb BRAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>VU9P</td>
<td>706353 (59.8%)</td>
<td>5163 (75.5%)</td>
<td>3169 (73.4%)</td>
</tr>
<tr>
<td>PYNQ-Z1</td>
<td>37034 (69.61%)</td>
<td>220 (100%)</td>
<td>277 (98.93%)</td>
</tr>
</tbody>
</table>
## Experimental Results (2)

- **CONV Layer Test Cases:**
  - Kernel Size: 1x1, 3x3, 5x5, 7x7
  - Feature Size: 224, 112, 56, 28, 14
  - Channel Size: 512, 256, 128, 64

- **Spatial CONV:**
  - Support all kernel sizes
  - Stable and close to peak perf.

- **Winograd CONV:**
  - Higher peak perf. than Spatial CONV
  - Small feature size => Low weight reuse rate => Bounded by memory BW
Experimental Results (2)

• CONV Layer Test Cases:
  • Kernel Size: 1x1, 3x3, 5x5, 7x7
  • Feature Size: 224, 112, 56, 28, 14
  • Channel Size: 512, 256, 128, 64

• Spatial CONV:
  • Support all kernel sizes
  • Stable and close to peak perf.

• Winograd CONV:
  • Higher peak perf. than Spatial CONV
  • Small feature size =>
    Low weight reuse rate =>
    Bounded by memory BW
Experimental Results (2)

- CONV Layer Test Cases:
  - Kernel Size: 1x1, 3x3, 5x5, 7x7
  - Feature Size: 224, 112, 56, 28, 14
  - Channel Size: 512, 256, 128, 64

- Spatial CONV:
  - Support all kernel sizes
  - Stable and close to peak perf.

- Winograd CONV:
  - Higher peak perf. than Spatial CONV
  - Small feature size => Low weight reuse rate => Bounded by memory BW
Conclusion

• HybridDNN Framework: generate highly optimized accelerators for the latest generation of cloud and embedded FPGAs

• Instruction-based Architecture:
  • Hybrid CONV Processing Engine (Spatial and Winograd CONV)
  • Support multiple dataflow (input and weight stationary)
  • Scalable parallel factors ($P_I$, $P_O$, $P_T$, and $N_I$)

• Design Space Exploration:
  • Performance estimation model (4.27% and 4.03% error for VU9P and PYNQ-Z1)
  • Efficient algorithm for optimizing HW & SW parameters

• On-board Experimental Results:
  • 3375.3 (VU9P) and 83.3 (PYNQ-Z1) GOPS performance on VGG-16
Questions

Thank you!
July 22, 2020