Scaling the Cascades
Interconnect-aware FPGA implementation of Machine Learning problems

Anand Samajdar, Tushar Garg, Tushar Krishna, Nachiket Kapre
nachiket@uwaterloo.ca
Claim

• Hard FPGA interconnect (cascades) efficiently supports nearest neighbour communication + reuse in ML workloads

• Three kinds of UltraScale+ cascades [DSP, BRAM, URAM]
  • Combination of (1) pixel, (2) row, (3) map reuse

• Deliverables:
  • 650 MHz full-chip operation
  • 7x better latency, 30% lower throughput than the formidable Xilinx SuperTile design for GoogLeNet v1
Landscape of FPGA+ML accelerators

![Graph showing the frequency of various FPGA+ML accelerators with SuperTile[29] and BrainWave[6] highlighted]
Communication Requirements of 3x3 Convolution

Input Row k

Input Row k+1

Input Row k+2

Weights

Output Row k

Output Map J

Input Map I
Communication Requirements of 3x3 Convolution

- **Input Map I**
- **Output Map J**
- **Weights**

1. **Pixel Streaming**
2. **Row Streaming**
3. **Channel Streaming**

Input Rows: k, k+1, k+2
Output Rows: k

Input Map I: 3 rows
Output Map J: 1 row
Weights: 3 channels
Reuse Patterns

Input Row $k$
Input Row $k+1$
Input Row $k+2$

Output Row $k$

P cascade for summation
Reuse Patterns

Input Row $k$
Input Row $k+1$
Input Row $k+2$

+$\rightarrow$
Output Row $k$

$\times$

Input Map $I$

Output Map $J$

Input Row $k+2$
Weight Row 2
Input Row $k+1$
Weight Row 1
Input Row $k$
Weight Row 0

Exploit Data Reuse

B cascade for weight streaming

A cascade for pixel streaming

P cascade for summation

$\times$

pixel streaming

row streaming

1

2
Reuse Patterns

Input Row $k$
Input Row $k+1$
Input Row $k+2$

Output Row $k$

Input Map I
Output Map J

Input Map I

Input Row $k+2$
Input Row $k+1$
Input Row $k$

Weight Row 2
Weight Row 1
Weight Row 0

Exploit Data Reuse

B cascade for weight streaming
A cascade for pixel streaming
P cascade for summation

pixel streaming
row streaming

1
2
Reuse Patterns

3x3 Convolution Tile
Reuse Patterns

Input Map I → 3x3 Convolution Tile → Output Map J

Input Map I+1 → 3x3 Convolution Tile

Input Map I++ → 3x3 Convolution Tile
Reuse Patterns

Input Map I → 3x3 Convolution Tile → Output Map J

Input Map I+1 → 3x3 Convolution Tile

Input Map I+.. → 3x3 Convolution Tile

Weights
channel streaming

3
Reuse Patterns

Weights
Input Map I

Weights
Input Map I+1

Weights
Input Map I+..

3x3 Convolution Tile

Output Map J

channel streaming

3
Xilinx UltraScale+ FPGA Cascades

- BRAM18 support A/B cascades 2x72b-wide links
- DSP48 supports A, B, P cascades (systolic input and summation)
- URAM288 supports A, B cascades
Outline

• Understanding Cascades

• Assembling the FPGA accelerator + FPGA Layout

• MLPerf Evaluation

• Conclusions + Discussion
Promise of Cascades

- Absorb data movement onto dedicated interconnect vs. General-purpose wiring
- Higher clock frequency operation, layout-friendly architecture
Our approach

- Exploit cascades aggressively!

- **DSP48**
  - For 3x3 convolution, length-9 cascades
  - P cascade for summation (like INT8 paper)
  - A cascade for systolic retiming (like DSP48E2 user guide)
  - B cascades for weights (our contribution)

(a) DSP48 cascade (891 MHz)
Our approach

- Exploit cascades aggressively!

- **RAMB18E2** *(our contribution)*
  - For 3x3 convolution, only need 3 BRAM-long chains
  - A/B cascade for shift operation
  - Swap between A and B to keep one read port available.

(b) RAMB18 cascade (825 MHz)
Our approach

• Exploit cascades aggressively!

• **URAM288** *(our contribution)*

  • Alternating A/B cascades of length-2

  • Both data + addresses cascades

  • Shift operation tricky!

• Due to 72b width, and resource ratios, **idle cycles** available for realizing shifts

(c) URAM288 cascade (650 MHz)
Putting it together
Putting it together
Putting it together

Weights (initial shift)
Putting it together

Weights (initial shift)

Pixel streaming
Putting it together
Putting it together

Row streaming
Putting it together

RAMB 18 (C) → RAMB 18 (B) → RAMB 18 (A)

URA-288 (Input) to next RAMB

from previous RAMB 18 (Kern)
Putting it together

Map streaming
Putting it together
Putting it together

1. **Pixel streaming**
   - RAMB 18 (Kern)
   - DSP48
   - +

2. **Row streaming**
   - RAMB 18 (C)
   - RAMB 18 (B)
   - RAMB 18 (A)
   - +

3. **Channel streaming**
   - URAM 288 (Input)
   - (Weights)
   - +

- URAM 288 (Output)

- From previous URAM to next URAM
- URAM streaming
- to next URAM

Diagram shows the flow of data from inputs to outputs through DSP48 elements, highlighting pixel, row, and channel streaming.
A 3x3 tile layout
A 3x3 tile layout
A 3x3 tile layout
Tiling the design

- VU37P device has specific resource mix
  - For each URAM, you get 4.2 BRAMs and 9.4 DSP48s
- Repeating pattern must conform to this ratio
  - One tile: 2 URAMs, 8 BRAMs, 18 DSPs
- Physical layout XDC constraints must account for irregular column arrangement of hard resources
Tiling the design

- VU37P device has specific resource mix
  - For each URAM, you get 4.2 BRAMs and 9.4 DSP48s
- Repeating pattern must conform to this ratio
  - One tile: 2 URAMs, 8 BRAMs, 18 DSPs
- Physical layout XDC constraints must account for irregular column arrangement of hard resources

(a) 3×3 Convolution Tiles
Matrix-Matrix Multiplication

- Limited Reuse opportunities

- Split large matrix across URAMs
  - Each URAM stores a set of complete rows —> allows result vector to be independently processed.

- Partial vector results then circulated across the chip in a ring-like fashion —> using BRAM cascades

- URAM cascades only used for loading matrix at start
VU37P Layout
VU37P FPGA Mapping

CONVOLUTION

MATRIX-MULTIPLY
VU37P FPGA Mapping

80% 20%

CONVOLUTION  MATRIX-MULTIPLY
VU37P FPGA Mapping

80%  20%
Effect of using cascades

• Registers in hard interconnect save us fabric registers for other pipelining needs

• Clock period marginally better

• Obvious reduction in interconnect utilization

<table>
<thead>
<tr>
<th>Design</th>
<th>Size</th>
<th>LUTs</th>
<th>FFs</th>
<th>Clk (ns)</th>
<th>Net Util. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Fabric</td>
<td>Cascade</td>
<td>%</td>
<td>Fabric</td>
</tr>
<tr>
<td>Convolution</td>
<td>Block</td>
<td>325</td>
<td>327</td>
<td>0%</td>
<td>1.3K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tile (2 blocks)</td>
<td>424</td>
<td>435</td>
<td>-2%</td>
<td>1.9K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Full-Chip</td>
<td>20.9K</td>
<td>21.1K</td>
<td>-1%</td>
<td>95.3K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrix-Vector</td>
<td>Block</td>
<td>98</td>
<td>98</td>
<td>0%</td>
<td>775</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tile (4 blocks)</td>
<td>375</td>
<td>374</td>
<td>0%</td>
<td>2.3K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Full-Chip</td>
<td>90.2K</td>
<td>90.2K</td>
<td>0%</td>
<td>56.8K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE II: Resource and Frequency Trends for Convolution and Matrix Vector Multiplication blocks, tiles and full-chip layouts.
Effect of using cascades

• Registers in hard interconnect save us fabric registers for other pipelining needs

• Clock period marginally better

• Obvious reduction in interconnect utilization

<table>
<thead>
<tr>
<th>Design</th>
<th>Size</th>
<th>LUTs</th>
<th>FFs</th>
<th>Clk (ns)</th>
<th>Net Util. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Fabric</td>
<td>Cascade</td>
<td>%</td>
<td>Fabric</td>
</tr>
<tr>
<td>Convolution</td>
<td>Block</td>
<td>325</td>
<td>327</td>
<td>0%</td>
<td>1.3K</td>
</tr>
<tr>
<td></td>
<td>Tile (2 blocks)</td>
<td>424</td>
<td>435</td>
<td>-2%</td>
<td>1.9K</td>
</tr>
<tr>
<td></td>
<td>Full-Chip</td>
<td>20.9K</td>
<td>21.1K</td>
<td>-1%</td>
<td>95.3K</td>
</tr>
<tr>
<td>Matrix-Vector</td>
<td>Block</td>
<td>98</td>
<td>98</td>
<td>0%</td>
<td>775</td>
</tr>
<tr>
<td></td>
<td>Tile (4 blocks)</td>
<td>375</td>
<td>374</td>
<td>0%</td>
<td>2.2K</td>
</tr>
<tr>
<td></td>
<td>Full-Chip</td>
<td>90.2K</td>
<td>90.2K</td>
<td>0%</td>
<td>56.8K</td>
</tr>
</tbody>
</table>

TABLE II: Resource and Frequency Trends for Convolution and Matrix Vector Multiplication blocks, tiles and full-chip layouts.
Evaluation Methodology

• We use the SCALE-Sim cycle-accurate simulator
  
  • [https://github.com/ARM-software/SCALE-Sim](https://github.com/ARM-software/SCALE-Sim)
  
• Map URAMs -> IFMAP/OFMAP SRAMs
  
• BRAMs and DSP cascades
  => systolic array links
  
• VU37P can fit systolic array of size 960x9 (conv), 480x9 (mm)
Xilinx SuperTile

- GoogLeNet v1 mapped to VU9P FPGA (Amazon F1)
- 3046 images/s + 3.3ms latency
- Scorching 720 MHz operation!
- Mind-numbing 88% overall efficiency

Why is SuperTile so good?

- **Base Design**
  - High-frequency layout using DSP cascades,
  - Systolic data movement in fabric
  - Throughput boost
  - Decompose the systolic array into sub-arrays
  - Perform pipelining across CNN layers
  - Sacrifice some latency to significantly boost throughput!

https://dl.acm.org/citation.cfm?id=3293925
MLPerf Benchmarks

- **Caveat**: Result not verified by MLPerf. MLPerf name and logo are trademarks. See [www.mlperf.org](http://www.mlperf.org) for more information.

- Different ML workloads

- Various domains

- Different compute complexity

<table>
<thead>
<tr>
<th>Topology</th>
<th>Operation Count</th>
<th>Storage (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(MLPerf)</td>
<td>All</td>
<td>Conv</td>
</tr>
<tr>
<td>AlphaGoZero</td>
<td>352M</td>
<td>352M</td>
</tr>
<tr>
<td>DeepSpeech2</td>
<td>1.7G</td>
<td>1.7G</td>
</tr>
<tr>
<td>FasterRCNN</td>
<td>3.5G</td>
<td>1.6G</td>
</tr>
<tr>
<td>NCF</td>
<td>11M</td>
<td>0</td>
</tr>
<tr>
<td>Resnet50</td>
<td>3.4G</td>
<td>1.6G</td>
</tr>
<tr>
<td>Sentimental</td>
<td>210M</td>
<td>0</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>1.3G</td>
<td>1.3G</td>
</tr>
</tbody>
</table>
MLPerf Benchmarks

- **Caveat**: Result not verified by MLPerf. MLPerf name and logo are trademarks. See [www.mlperf.org](http://www.mlperf.org) for more information.

- Different ML workloads
- Various domains
- Different compute complexity

### TABLE I: MLPerf and GoogLeNet benchmark characteristics.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Operation Count</th>
<th>Storage (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(MLPerf)</td>
<td>All</td>
</tr>
<tr>
<td>AlphaGoZero</td>
<td>352M</td>
<td>352M</td>
</tr>
<tr>
<td>DeepSpeech2</td>
<td>1.7G</td>
<td>1.7G</td>
</tr>
<tr>
<td>FasterRCNN</td>
<td>3.5G</td>
<td>1.6G</td>
</tr>
<tr>
<td>NCF</td>
<td>11M</td>
<td>0</td>
</tr>
<tr>
<td>Resnet50</td>
<td>3.4G</td>
<td>1.6G</td>
</tr>
<tr>
<td>Sentimental</td>
<td>210M</td>
<td>0</td>
</tr>
<tr>
<td>Transformer</td>
<td>113M</td>
<td>35M</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>1.3G</td>
<td>1.3G</td>
</tr>
</tbody>
</table>
MLPerf Benchmarks

- **Caveat**: Result not verified by MLPerf. MLPerf name and logo are trademarks. See [www.mlperf.org](http://www.mlperf.org) for more information.

- Different ML workloads
- Various domains
- Different compute complexity

### TABLE I: MLPerf and GoogleSuite benchmark characteristics

<table>
<thead>
<tr>
<th>Topology</th>
<th>Operation Count</th>
<th>Storage (bytes)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(MLPerf)</td>
<td>All</td>
<td>Conv</td>
</tr>
<tr>
<td>AlphaGoZero</td>
<td>352M</td>
<td>352M</td>
<td>353K</td>
</tr>
<tr>
<td>DeepSpeech2</td>
<td>1.7G</td>
<td>1.7G</td>
<td>74K</td>
</tr>
<tr>
<td>FasterRCNN</td>
<td>3.5G</td>
<td>1.6G</td>
<td>1.8G</td>
</tr>
<tr>
<td>NCF</td>
<td>11M</td>
<td>0</td>
<td>11M</td>
</tr>
<tr>
<td>Resnet50</td>
<td>3.4G</td>
<td>1.6G</td>
<td>1.8G</td>
</tr>
<tr>
<td>Sentimental</td>
<td>210M</td>
<td>0</td>
<td>210M</td>
</tr>
<tr>
<td>Transformer</td>
<td>113M</td>
<td>35M</td>
<td>78M</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>1.3G</td>
<td>1.3G</td>
<td>46M</td>
</tr>
</tbody>
</table>
## Performance Results

<table>
<thead>
<tr>
<th>Topology</th>
<th>Ratio (Conv:MM)</th>
<th>Cycles</th>
<th>Time (ms)</th>
<th>Tput. (inference/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlphaGoZero</td>
<td>90:10</td>
<td>60K</td>
<td>0.09</td>
<td>10K</td>
</tr>
<tr>
<td>DeepSpeech2</td>
<td>60:40</td>
<td>1.2M</td>
<td>1.89</td>
<td>528</td>
</tr>
<tr>
<td>FasterRCNN</td>
<td>30:70</td>
<td>903K</td>
<td>1.38</td>
<td>719</td>
</tr>
<tr>
<td>NCF</td>
<td>0:100</td>
<td>2.4K</td>
<td>0.003</td>
<td>260K</td>
</tr>
<tr>
<td>Resnet50</td>
<td>30:70</td>
<td>848K</td>
<td>1.3</td>
<td>766</td>
</tr>
<tr>
<td>Sentimental</td>
<td>100:0</td>
<td>24K</td>
<td>0.037</td>
<td>27K</td>
</tr>
<tr>
<td>GoogLeNet (Us)</td>
<td>70:30</td>
<td>261K</td>
<td>0.40</td>
<td>2.4K</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>-</td>
<td>-</td>
<td>3.3</td>
<td>3K</td>
</tr>
<tr>
<td>(SuperTile [30])</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Performance Results

<table>
<thead>
<tr>
<th>Topology (MLPerf)</th>
<th>Ratio (Conv:MM)</th>
<th>Cycles</th>
<th>Time (ms)</th>
<th>Tput. (inference/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlphaGoZero</td>
<td>90:10</td>
<td>60K</td>
<td>0.09</td>
<td>10K</td>
</tr>
<tr>
<td>DeepSpeech2</td>
<td>60:40</td>
<td>1.2M</td>
<td>1.89</td>
<td>528</td>
</tr>
<tr>
<td>FasterRCNN</td>
<td>30:70</td>
<td>903K</td>
<td>1.38</td>
<td>719</td>
</tr>
<tr>
<td>NCF</td>
<td>0:100</td>
<td>2.4K</td>
<td>0.003</td>
<td>260K</td>
</tr>
<tr>
<td>Resnet50</td>
<td>30:70</td>
<td>848K</td>
<td>1.3</td>
<td>766</td>
</tr>
<tr>
<td>Sentimental</td>
<td>100:0</td>
<td>24K</td>
<td>0.037</td>
<td>27K</td>
</tr>
<tr>
<td>GoogLeNet (Us)</td>
<td>70:30</td>
<td>261K</td>
<td>0.40</td>
<td>2.4K</td>
</tr>
<tr>
<td>GoogLeNet (SuperTile [30])</td>
<td>-</td>
<td>-</td>
<td>3.3</td>
<td>3K</td>
</tr>
</tbody>
</table>
## Performance Results

<table>
<thead>
<tr>
<th>Topology</th>
<th>Ratio (Conv:MM)</th>
<th>Cycles</th>
<th>Time (ms)</th>
<th>Tput. (inference/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlphaGoZero</td>
<td>90:10</td>
<td>60K</td>
<td>0.09</td>
<td>10K</td>
</tr>
<tr>
<td>DeepSpeech2</td>
<td>60:40</td>
<td>1.2M</td>
<td>1.89</td>
<td>528</td>
</tr>
<tr>
<td>FasterRCNN</td>
<td>30:70</td>
<td></td>
<td>1.38</td>
<td>719</td>
</tr>
<tr>
<td>NCF</td>
<td>0:100</td>
<td></td>
<td>0.003</td>
<td>260K</td>
</tr>
<tr>
<td>Resnet50</td>
<td>30:70</td>
<td></td>
<td>1.3</td>
<td>766</td>
</tr>
<tr>
<td>Sentimental</td>
<td>100:0</td>
<td></td>
<td>0.037</td>
<td>27K</td>
</tr>
<tr>
<td><strong>GoogLeNet (Us)</strong></td>
<td>70:30</td>
<td>261K</td>
<td>0.40</td>
<td>2.4K</td>
</tr>
<tr>
<td><strong>GoogLeNet</strong></td>
<td></td>
<td></td>
<td>3.3</td>
<td>3K</td>
</tr>
<tr>
<td><strong>(SuperTile [30])</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7x Lower Latency
# Performance Results

<table>
<thead>
<tr>
<th>Topology (MLPerf)</th>
<th>Ratio (Conv:MM)</th>
<th>Cycles</th>
<th>Time (ms)</th>
<th>Tput. (inference/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlphaGoZero</td>
<td>90:10</td>
<td>60K</td>
<td>0.09</td>
<td>10K</td>
</tr>
<tr>
<td>DeepSpeech2</td>
<td>60:40</td>
<td>1.2M</td>
<td>1.89</td>
<td>528</td>
</tr>
<tr>
<td>FasterRCNN</td>
<td>30:70</td>
<td>903K</td>
<td>1.38</td>
<td>710</td>
</tr>
<tr>
<td>NCF</td>
<td>0:100</td>
<td>2.4K</td>
<td>0.003</td>
<td>202</td>
</tr>
<tr>
<td>Resnet50</td>
<td>30:70</td>
<td>848K</td>
<td>1.3</td>
<td>760</td>
</tr>
<tr>
<td>Sentimental</td>
<td>100:0</td>
<td>24K</td>
<td>0.037</td>
<td>271</td>
</tr>
<tr>
<td><strong>GoogLeNet (Us)</strong></td>
<td>70:30</td>
<td>261K</td>
<td>0.40</td>
<td>2.4K</td>
</tr>
<tr>
<td><strong>GoogLeNet</strong></td>
<td>-</td>
<td>-</td>
<td>3.3</td>
<td>3K</td>
</tr>
</tbody>
</table>

*Note: 30% Lower Throughput*
Performance Results

<table>
<thead>
<tr>
<th>Io:MM</th>
<th>Cycles</th>
<th>Time (ms)</th>
<th>Tput. (inference/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>60K</td>
<td>0.09</td>
<td>10K</td>
</tr>
<tr>
<td>0</td>
<td>1.2M</td>
<td>1.89</td>
<td>528</td>
</tr>
<tr>
<td>0</td>
<td>903K</td>
<td>1.38</td>
<td>719</td>
</tr>
<tr>
<td>0</td>
<td>2.4K</td>
<td>0.003</td>
<td>207</td>
</tr>
<tr>
<td>0</td>
<td>848K</td>
<td>1.3</td>
<td>767</td>
</tr>
<tr>
<td>0</td>
<td>24K</td>
<td>0.037</td>
<td>27K</td>
</tr>
</tbody>
</table>

- GoogLeNet (Us) 70:30 261K 0.40 2.4K
- GoogLeNet (SuperTile [30]) - 3.3 3K

30% Lower Throughput
Performance Results

**Layer Pipelining**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Tput. (inference/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10K</td>
</tr>
<tr>
<td>P2</td>
<td>528</td>
</tr>
<tr>
<td>P3</td>
<td>716</td>
</tr>
<tr>
<td>P4</td>
<td>20</td>
</tr>
</tbody>
</table>

**Sentimental**

<table>
<thead>
<tr>
<th>Model</th>
<th>Tput. (inference/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoogLeNet (Us)</td>
<td>261K</td>
</tr>
<tr>
<td>GoogLeNet (SuperTile [30])</td>
<td>3K</td>
</tr>
</tbody>
</table>

30% Lower Throughput
Performance Results

<table>
<thead>
<tr>
<th>Topology (MLPerf)</th>
<th>Ratio (Conv:MM)</th>
<th>Cycles</th>
<th>Time (ms)</th>
<th>Tput. (inference/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlphaGoZero</td>
<td>90:10</td>
<td>60K</td>
<td>0.09</td>
<td>10K</td>
</tr>
<tr>
<td>DeepSpeech2</td>
<td>60:40</td>
<td>1.2M</td>
<td>1.89</td>
<td>528</td>
</tr>
<tr>
<td>FasterRCNN</td>
<td>30:70</td>
<td>903K</td>
<td>1.38</td>
<td>719</td>
</tr>
<tr>
<td>NCF</td>
<td>0:100</td>
<td>2.4K</td>
<td>0.003</td>
<td>260K</td>
</tr>
<tr>
<td>Resnet50</td>
<td>30:70</td>
<td>848K</td>
<td>1.3</td>
<td>766</td>
</tr>
<tr>
<td>Sentimental</td>
<td>100:0</td>
<td>24K</td>
<td>0.037</td>
<td>27K</td>
</tr>
<tr>
<td>GoogLeNet (Us)</td>
<td>70:30</td>
<td>261K</td>
<td>0.40</td>
<td>2.4K</td>
</tr>
<tr>
<td>GoogLeNet (SuperTile [30])</td>
<td>-</td>
<td>-</td>
<td>3.3</td>
<td>3K</td>
</tr>
</tbody>
</table>
# Performance Results

<table>
<thead>
<tr>
<th>Topology</th>
<th>Ratio (Conv:MM)</th>
<th>Cycles</th>
<th>Time (ms)</th>
<th>Tput. (inference/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlphaGoZero</td>
<td>90:10</td>
<td>60K</td>
<td>0.09</td>
<td>10K</td>
</tr>
<tr>
<td>DeepSpeech2</td>
<td>60:40</td>
<td>1.2M</td>
<td>1.89</td>
<td>528</td>
</tr>
<tr>
<td>FasterRCNN</td>
<td>30:70</td>
<td>903K</td>
<td>1.38</td>
<td>719</td>
</tr>
<tr>
<td>NCF</td>
<td>0:100</td>
<td>2.4K</td>
<td>0.003</td>
<td>260K</td>
</tr>
<tr>
<td>Resnet50</td>
<td>30:70</td>
<td>848K</td>
<td>1.3</td>
<td>766</td>
</tr>
<tr>
<td>Sentimental</td>
<td>100:0</td>
<td>24K</td>
<td>0.037</td>
<td>27K</td>
</tr>
<tr>
<td>GoogLeNet (Us)</td>
<td>70:30</td>
<td>261K</td>
<td>0.40</td>
<td>2.4K</td>
</tr>
<tr>
<td>GoogLeNet (SuperTile [30])</td>
<td>-</td>
<td>-</td>
<td>3.3</td>
<td>3K</td>
</tr>
</tbody>
</table>
Optimizing the mapping

Relative Inference Latency

Space Division Multiplexing (Conv:MM)
Compute Characteristics

![Graph showing the relationship between operation count ratio and space division multiplexing (Conv:MM).]
Conclusions

- 650+ MHz operation for FPGA ML accelerator tailored for Xilinx UltraScale+
- 7x better latency, 30% poorer throughput vs. Xilinx SuperTile
- Hard interconnect cascades save us 30% registers + 12% on clock period vs. Fabric interconnect
Discussion

• **URAM Bandwidth balance** — Matrix-Multiplication performance suffers due to missing bandwidth from URAM —> *Give us 144b ports vs 72b ports!*

• **Dynamic Control** — Can build unified Conv + MM blocks if data flow in cascades even more programmable —> *Give us more control!*

• **Abandon Versal** — Versal architecture is not an FPGA. Improve DSPs, BRAMs, URAMs + hard interconnect instead —> *Stay true to your roots, Xilinx!*
Discussion

• **URAM Bandwidth balance** — Matrix-Multiplication performance suffers due to missing bandwidth from URAM —> *Give us 144b ports vs 72b ports!*

• **Dynamic Control** — Can build unified Conv + MM blocks if data flow in cascades even more programmable —> *Give us more control!*

• **Abandon Versal** — Versal architecture is not an FPGA. Improve DSPs, BRAMs, URAMs + hard interconnect instead —> *Stay true to your roots, Xilinx!*
Communication Requirements of Matrix Multiplication

- Vector $i$ + Vector $i+(k-2)$ + Vector $i+(k-1)$ + Vector $i+k$ = Partial result vector

Matrix
Communication Requirements of Matrix Multiplication

- Vector \( i \)
- Vector \( i+(k-2) \)
- Vector \( i+(k-1) \)
- Vector \( i+k \)

\[ x \times \]

\[ + \] Partial result vector

Matrix
Communication Requirements of Matrix Multiplication

Vector $i+(k-2)$

Vector $i+(k-1)$

Vector $i+k$

$\times$

$\times$

$\times$

$\times$

Matrix

$\times$

Accum stream

+ Partial result vector

$\times$

Matrix initialize

Vector fanout

1

2

3
Communication Pattern

Vector i

Vector i+(k-2)

Vector i+(k-1)

Vector i+k

Matrix

Partial result vector

P cascade for summation
Communication Pattern

Vector i

Vector i+(k-2)

Vector i+(k-1)

Vector i+k

Partial result vector

Matrix

Matrix

Vector i+k

Vector i+(k-1)

Vector i+(k-2)

Vector i

Partial result vector

Matrix

Accum stream

1

P cascade for summation

67
Communication Pattern

1 Accum stream

P cascade for summation

Vector $i$

Matrix

Vector $i + (k-2)$

Vector $i + (k-1)$

Vector $i + k$

Partial result vector

Partial Result

82x633
Communication Pattern

Matrix initialize

Matrix

Vector i+k

Vector i+(k-1)

Vector i+(k-2)

Vector i

Partial result vector

P cascade for summation

Partial Result

Vector fanout

Accum stream

Vector i+k

Vector i+(k-1)

Vector i+(k-2)

Vector i

Matrix initialize

Matrix

Vector i+k

Vector i+(k-1)

Vector i+(k-2)

Vector i

Partial result vector

P cascade for summation

Partial Result

Vector fanout

Accum stream

Vector i+k

Vector i+(k-1)

Vector i+(k-2)

Vector i

Partial result vector

P cascade for summation

Partial Result

Vector fanout

Accum stream
Matrix-Matrix Multiplication
Matrix-Matrix Multiplication
Matrix-Matrix Multiplication

Vector fanout
Matrix-Matrix Multiplication
Matrix-Matrix Multiplication
Matrix-Matrix Multiplication

Initial loading
MLPerf Benchmarks

• **Caveat:** Result not verified by MLPerf. MLPerf name and logo are trademarks. See [www.mlperf.org](http://www.mlperf.org) for more information.

• Different ML workloads

• Various domains

• Different compute complexity

• SoftMax not implemented in hardware

### TABLE I: MLPerf and GoogLeNet benchmark characteristics.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Operation Count</th>
<th>Storage (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(MLPerf)</td>
<td>All</td>
<td>Conv</td>
</tr>
<tr>
<td>AlphaGoZero</td>
<td>352M</td>
<td>352M</td>
</tr>
<tr>
<td>DeepSpeech2</td>
<td>1.7G</td>
<td>1.7G</td>
</tr>
<tr>
<td>FasterRCNN</td>
<td>3.5G</td>
<td>1.6G</td>
</tr>
<tr>
<td>NCF</td>
<td>11M</td>
<td>0</td>
</tr>
<tr>
<td>Resnet50</td>
<td>3.4G</td>
<td>1.6G</td>
</tr>
<tr>
<td>Sentimental</td>
<td>210M</td>
<td>0</td>
</tr>
<tr>
<td>Transformer</td>
<td>115M</td>
<td>39M</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>1.3G</td>
<td>1.3G</td>
</tr>
</tbody>
</table>
Why is SuperTile so good?

- Base Design
  - High-frequency layout using DSP cascades,
  - LUT RAMs for weights,
  - Systolic data movement in fabric

- Throughput boost
  - Decompose the systolic array into sub-arrays
  - Perform pipelining across CNN layers
  - Sacrifice some latency to significantly boost throughput!
**Xilinx INT8 optimization**

- Soft-fracture a DSP48 27x18 multiplier to compute two 8x8 multiplications with a common operand
- A*B and D*B computed in together when A, B, D are 8 inputs

*Figure 7: Two Products in a Packed Word*

Xilinx INT8 optimization

• Create DSP cascades of length 7 to accumulate multiple product terms

• Enough precision headroom to avoid overflow for this length of the chain

• Some fabric operation needed for final accumulations, or if 3x3 convolution support required

Xilinx DSP48 Systolic Mode

Figure 4-5: Adder Cascade