Acceleration of a CMS DNN based Algorithm

M. Barbone, W. Luk, T. Ourida, A. Tapper
Motivation

- Data rate set to increase with High-Luminosity LHC upgrade
- Current approach not scalable to meet requirements of additional processing due to increased luminosity

https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults
Contributions

• Novel parallel architecture for Long-Lived Particle (LLP) jet tagging
  – compute: both space and time
  – accelerated processing: reduce cost of storage for post-processing

• Explore an FPGA architecture using a highly computationally intensive algorithm which could benefit from acceleration
Long-Lived Particle Tagger

- Many models of new physics predict existence of exotic long-lived particles
- Lifetime frontier yet to be fully explored at the LHC
- Deep Neural Network trained on Monte Carlo to classify jets into Standard Model (light, b/c-quark gluon) and exotic long-lived particles
- Trained for lifetimes $c\tau \in [10 \mu m, 10 m]$ with lifetime as a parameter – 1mm benchmark
LLP Jet Tagging Algorithm

- Multiclass classification
- Jet class trained on 638 input features per labelled jet
  - Charged and neutral particles, properties of secondary vertices and global jet variables …
- Unsupervised domain adaptation via backpropagation
  - Trained on data and Monte Carlo in control region
- Architecture progressively compresses and extracts most discriminating features
- Resample jet $p_T$ and $\eta$ to have same distribution for all classes – discrimination only via correlations with other features
LLP Jet Tagging Algorithm

- Multiclass classification
- Jet class trained on 638 input features per labelled jet
  - Charged and neutral particles, properties of secondary vertices and global jet variables …
- Unsupervised domain adaptation via backpropagation
  - Trained on data and Monte Carlo in control region
- Architecture progressively compresses and extracts most discriminating features
- Resample jet $p_T$ and $\eta$ to have same distribution for all classes – discrimination only via correlations with other features
Serialised Cyclic RAM Multiply Accumulate

- Cyclic RAM block: RAM with read address connected to a counter
- Resource utilisation is heavily reduced as opposed to conventional convolution method
- Resource saving can be invested in instance duplication to increase throughput.
Instance Duplication Across Channels

- MAC units can be duplicated across the dimension common throughout the convolution layers
- MAC units can be duplicated across input channels
- Compute in space by taking advantage of computationally independent data streams
Elementwise Matrix Indexing for RAM Storage

- Matrices usually stored contiguously in memory
- Alternative storage mode: elements in same position in matrices stored in same RAM element
- Faster and simplified read mechanism compared to traditional storage mode
Dataflow and Maxeler

• Dataflow computing involves channelling data between bespoke compute units (kernels)

• Benefit of faster development as opposed to use of Verilog/VHDL and exposure to the rich ecosystem that Java provides

• MaxCompiler (Java based HLS tool) chosen for simplified pipelining and latency control

• Communication between host CPU and FPGA over PCIe interface
Results – Test Setup

- Hardware implementations running on a Xilinx VU9P @ 100MHz
- Resource utilisation comprised of usage of:
  - Look Up Tables (LUT)
  - Flip Flops (FF)
  - Digital Signal Processors (DSP)
  - Block Random Access Memory (BRAM)
- Resource utilisation:
  \[
  \frac{1}{4} \left( \frac{\text{Used LUT}}{\text{Total LUT}} + \frac{\text{Used FF}}{\text{Total FF}} + \frac{\text{Used DSP}}{\text{Total DSP}} + \frac{\text{Used BRAM}}{\text{Total BRAM}} \right) \times 100\%
  \]
## Results – Serialised Cyclic RAM Multiply Accumulate

<table>
<thead>
<tr>
<th>Hardware Design</th>
<th>Fully Unrolled</th>
<th>Partial Unroll</th>
<th>Single MAC Unit</th>
<th>Column MAC Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$ (MHz)</td>
<td>100</td>
<td>100</td>
<td>350</td>
<td>250</td>
</tr>
<tr>
<td>Latency (ns)</td>
<td>80.00</td>
<td>210.00</td>
<td>140.00</td>
<td>24.00</td>
</tr>
<tr>
<td>LUT (%)</td>
<td>26.86</td>
<td>13.24</td>
<td>1.08</td>
<td>17.31</td>
</tr>
<tr>
<td>FF (%)</td>
<td>21.61</td>
<td>16.88</td>
<td>1.07</td>
<td>19.22</td>
</tr>
<tr>
<td>DSP (%)</td>
<td>29.94</td>
<td>1.86</td>
<td>0.06</td>
<td>0.94</td>
</tr>
<tr>
<td>BRAM (%)</td>
<td>8.24</td>
<td>3.11</td>
<td>2.36</td>
<td>2.98</td>
</tr>
<tr>
<td>RU (%)</td>
<td>21.66</td>
<td>8.77</td>
<td>1.14</td>
<td>10.11</td>
</tr>
</tbody>
</table>
## Results – Elementwise Matrix Indexing for RAM Storage

<table>
<thead>
<tr>
<th>Hardware Design</th>
<th>Single Buffer</th>
<th>Double Buffer</th>
<th>Elementwise RAM Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$ (MHz)</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Latency (ns)</td>
<td>100.00</td>
<td>80.00</td>
<td>20.00</td>
</tr>
<tr>
<td>LUT (%)</td>
<td>1.65</td>
<td>3.38</td>
<td>0.71</td>
</tr>
<tr>
<td>FF (%)</td>
<td>6.43</td>
<td>13.22</td>
<td>2.90</td>
</tr>
<tr>
<td>DSP (%)</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>BRAM (%)</td>
<td>9.28</td>
<td>18.56</td>
<td>8.12</td>
</tr>
<tr>
<td>RU (%)</td>
<td>4.34</td>
<td>8.77</td>
<td>1.14</td>
</tr>
</tbody>
</table>
Comparisons to Prior Implementations

• C++ implementation ran on Intel Xeon Gold 6154 CPU @ 3.00 GHz

• Took 780ns to convolve two 16x4 input matrices

• Fastest FPGA implementation offers ~32.5x latency reduction

• FPGAs offer a viable route to online processing, especially after HL-LHC upgrade