PulseDL: A reconfigurable deep learning array processor dedicated to pulse characterization for high energy physics detectors

Pengcheng Ai
Pixel Laboratory at CCNU
Wuhan, China
Contents

• Background Information
• Architectural Research
• Hardware-Software Co-Design
• RTL Circuits and Pre-Layout Simulation
• Digital Flow and Post-Layout Simulation
• Conclusion and Future Work
• In the next surge of HEP detector upgrades, higher luminosity and event rates are desired
  – higher event rates pose a challenge to online pattern recognition
  – pile-up makes the problem even more complicated

• Data explosion in HEP tracking system
  – hundreds of millions of collisions per second
  – tens of petabytes of data per year

• If we can process the data at the front-end of electronics...
• Pulse shaping in HEP
  – ADC-based workflow (PHOS at ALICE as an example):
    – Pulse function after CR-RC2 shaper:
      \[
      f(t) = \begin{cases} 
      K \left( \frac{t-t_0}{\tau_p} \right)^2 \cdot e^{-2 \cdot \frac{t-t_0}{\tau_p}} + b, & \text{for } t \geq t_0 \\
      b & \text{for } t < t_0
      \end{cases}
      \]
    – Shaping time and peaking time ($\tau_p = n \cdot \tau_0$):
      • For PHOS-ALICE: 1 us and 2 us
      • For EMCal-ALICE: 100 ns and 200 ns
    – Sampling rate: 10 MSps
• Non-ideal characteristics of shaped pulses

- short-term change
- random noise (Gaussian)
- random noise (non-Gaussian and biased)

• Possible to use deep learning for regression
  - Cancel long-term drift
  - alleviate short-term change
  - work well under non-Gaussian and biased conditions

• Reference:
  2019 JINST 14 P03002, DOI: https://doi.org/10.1088/1748-0221/14/03/P03002
**Traditional Methods vs. Deep Learning**

<table>
<thead>
<tr>
<th></th>
<th>Traditional Methods</th>
<th>Deep Learning (Neural Network)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Examples</strong></td>
<td>Least square fitting (global)</td>
<td>Convolutional NN (local based)</td>
</tr>
<tr>
<td></td>
<td>Kalman filtering (recursive)</td>
<td>Recurrent/recursive NN</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>Linear model</td>
<td>Linear and nonlinear model</td>
</tr>
<tr>
<td><strong>Noise</strong></td>
<td>Gaussian</td>
<td>Gaussian and non-Gaussian</td>
</tr>
<tr>
<td><strong>Bias sensitive</strong></td>
<td>Yes</td>
<td>Usually no</td>
</tr>
<tr>
<td><strong>Theory</strong></td>
<td>Statistics-based</td>
<td>Approximation-based</td>
</tr>
<tr>
<td><strong>Priori Knowledge</strong></td>
<td>Need to design fitting functions</td>
<td>Need to design hyper-parameters</td>
</tr>
<tr>
<td><strong>Error analysis</strong></td>
<td>Possible</td>
<td>Usually not possible</td>
</tr>
<tr>
<td><strong>Overall performance</strong></td>
<td>Guaranteed only in statistically ideal conditions</td>
<td>Guaranteed in general conditions</td>
</tr>
</tbody>
</table>
• Overview of front-end ASICs
  – Most FEE parts (CSA, filter, shaper, ADC, buffer, multiplexer...) can be converted into integrated circuits
  – Pros: reduced system complexity, improved stability, less power, etc.
  – Cons: multi-channel uniformity, manufacturing cost, etc.
  – Pros > Cons in the long run

• “Information-centric digitization” (Murmann HEPIC’2017)
  – Reduce energy and data transmission through feature extraction
• Deep learning hardware accelerators are developing quickly in recent years

• In general, the operations in deep neural networks are highly parallel and can be performed concurrently
  – The array processor architecture is a good choice
  – The basic operating element is the processing engine (PE)

• Parameter sharing and near-memory computing become significant in DNN accelerators

*What can be shared?*

Input feature maps, filters, partial sums...

*How is the energy consuming?*

Register File(x1), Network-on-Chip(x2), On-chip Buffer(x6), DRAM(x200)...
• Three possible schemes for the overall architecture
  – CISC CPU with SIMD or MIMD
    • compatible like GPU (😊), Power consumption with cascaded buffers ( )))
  – RISC CPU with customized PE
    • best balance between performance and power (😊), generalize to NNs ( )))
  – Minimized RISC CPU with fully-powered PE
    • generalize to NNs (😊), Network-on-Chip to transfer data ( )))

• Three possible schemes for the PE
  – Weight stationary: reduce weight transmission
  – Output stationary: reduce partial sums transmission
  – Row stationary: reduce the transmission of weights, input feature map and partial data all together

• Scheme for the bus
  – Commercial (Ethernet, PCI-e...), On-chip bus (AXI, APB, TileLink...), Customized (ICB) -> Interfaced as a stand-alone IP core
Our major aim: design a digital integrated circuit to accelerate the one-dimensional DNN for HEP

- Pervasive in HEP: calorimeters, TPCs, hybrid pixel sensors...
- Moderate amount of computation
- Accessible with the digital process we use

The targeted network architecture

- Convolution (and transpose convolution) in dominance
- Fully-connected matrix multiplication
- Different kinds of activations (ReLU, leaky ReLU, Linear...)

Example: Denoising autoencoder + regression network

- Works well with the pulse shaping task in the previous study
- Scalable and reconfigurable with the hardware architectural scheme
• **PulseDL**: a deep learning array processor
## Network architecture in detail

<table>
<thead>
<tr>
<th>Name</th>
<th>Input Length</th>
<th>Input Channels</th>
<th>Kernel Length</th>
<th>Kernel Number / Output Channels</th>
<th>Output Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoder1</td>
<td>32</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Encoder2</td>
<td>16</td>
<td>16</td>
<td>4</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>Encoder3</td>
<td>8</td>
<td>32</td>
<td>4</td>
<td>64</td>
<td>4</td>
</tr>
<tr>
<td>Encoder4</td>
<td>4</td>
<td>64</td>
<td>4</td>
<td>128</td>
<td>2</td>
</tr>
<tr>
<td>Encoder5</td>
<td>2</td>
<td>128</td>
<td>4</td>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>Decoder5</td>
<td>1</td>
<td>128</td>
<td>4</td>
<td>128</td>
<td>2</td>
</tr>
<tr>
<td>Decoder4</td>
<td>2</td>
<td>128+128</td>
<td>4</td>
<td>64</td>
<td>4</td>
</tr>
<tr>
<td>Decoder3</td>
<td>4</td>
<td>64+64</td>
<td>4</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>Decoder2</td>
<td>8</td>
<td>32+32</td>
<td>4</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Decoder1</td>
<td>16</td>
<td>16+16</td>
<td>4</td>
<td>1</td>
<td>32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Input Length</th>
<th>Number of Weights and Biases</th>
<th>Output Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC1</td>
<td>32</td>
<td>W=32*256, B=256</td>
<td>256</td>
</tr>
<tr>
<td>FC2</td>
<td>256</td>
<td>W=256*256, B=256</td>
<td>256</td>
</tr>
<tr>
<td>Final Out</td>
<td>256</td>
<td>W=256*1, B=1</td>
<td>1</td>
</tr>
</tbody>
</table>
The parallelism of the convolution

- **Input Fmaps**: \(I[C][L]\)
- **Filter weights**: \(W[C][M][R]\)
- **Output Fmaps**: \(Ot[C][M][T] \rightarrow Od[C/NPE][M][T] \rightarrow Of[M][T]\)
- **PE number**: \(NPE\), **Stride**: \(S\)
The quantization effect on the precision

- We test the denoising autoencoder with different quantization schemes
  (Test dataset: 10000 examples, RMS is computed between output and label)
• **PE design**
  
  – PE ALU: perform multiply-and-accumulate operations
  
  – PE MEM CTL: interface between buffers and the PE arithmetic logic
  
  – PE CTL: implement different modes for various NN layers
• Adder tree design
  – Spatial adder tree (SA): sum up the results of different PEs
  – Temporal adder tree (TA): sum up the results of SA at different times
• **ICB Bus design**

Write transaction: return in the same cycle

Read transaction: return in the next cycle
• Validation method
  – We adopt a grey box validation method for the RTL design
  – Steps:
    • generate random input
    • calculate expected output
    • instantiate top level module
    • compare actual results in different stages of the pipeline with the expected
Pre-Layout Simulation

- Simulation results (at 20 MHz):

<table>
<thead>
<tr>
<th>Layer name</th>
<th>Bus type</th>
<th>Manual/Auto</th>
<th>Parallel</th>
<th>Time(μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv layer 2</td>
<td>Custom</td>
<td>Manual</td>
<td>No</td>
<td>323.600</td>
</tr>
<tr>
<td>conv layer 2</td>
<td>Custom</td>
<td>Manual</td>
<td>Yes</td>
<td>310.200</td>
</tr>
<tr>
<td>conv layer 5</td>
<td>Custom</td>
<td>Auto</td>
<td>No</td>
<td>4767.600</td>
</tr>
<tr>
<td>conv layer 5</td>
<td>ICB</td>
<td>Auto</td>
<td>No</td>
<td>1402.800</td>
</tr>
<tr>
<td>deconv layer 2</td>
<td>Custom</td>
<td>Auto</td>
<td>Yes</td>
<td>4597.400</td>
</tr>
<tr>
<td>deconv layer 2</td>
<td>ICB</td>
<td>Auto</td>
<td>Yes</td>
<td>1141.000</td>
</tr>
<tr>
<td>fc layer 2</td>
<td>Custom</td>
<td>Auto</td>
<td>Yes</td>
<td>5025.000</td>
</tr>
<tr>
<td>fc layer 2</td>
<td>ICB</td>
<td>Auto</td>
<td>Yes</td>
<td>1276.200</td>
</tr>
<tr>
<td>fc layer 1 (4x2)</td>
<td>Custom</td>
<td>Auto</td>
<td>No</td>
<td>842.000</td>
</tr>
<tr>
<td>fc layer 1 (2x4)</td>
<td>Custom</td>
<td>Auto</td>
<td>No</td>
<td>842.000</td>
</tr>
</tbody>
</table>
### Chip layout

<table>
<thead>
<tr>
<th></th>
<th><strong>In this design</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process</strong></td>
<td>GSMCR013（130 nm）</td>
</tr>
<tr>
<td><strong>Area</strong></td>
<td>24 mm²</td>
</tr>
<tr>
<td><strong>Design target</strong></td>
<td>General-purpose convolutional neural networks</td>
</tr>
<tr>
<td><strong>Voltage</strong></td>
<td>1.2V (core), 3.3V (PAD)</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>25 MHz (at least)</td>
</tr>
<tr>
<td><strong>Pads</strong></td>
<td>35 output PAD, 52 input PAD</td>
</tr>
<tr>
<td><strong>On-chip register</strong></td>
<td>69 kb</td>
</tr>
<tr>
<td><strong>On-chip RAM</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>Input: 8/16 bit fixed-point, Output: 32 bit fixed-point</td>
</tr>
</tbody>
</table>
Post-Layout Simulation

Power Heat map
Majority in the range: (1e-04 ~ 1e-03) mW, (1e-05 ~ 1e-04) mW, (1e-06 ~ 1e-05) mW

Run testbench to get the vcd file, and analyze in the EDA software (main freq. is 25MHz):

<table>
<thead>
<tr>
<th>Testbench name</th>
<th>Time (ms)</th>
<th>Internal power (mW)</th>
<th>Switching power (mW)</th>
<th>Total power (mW)</th>
<th>Power efficiency (GOPS/W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv_layer5_pow_t</td>
<td>3.816768</td>
<td>59.552</td>
<td>16.263</td>
<td>76.707</td>
<td>0.448</td>
</tr>
<tr>
<td>conv_layer5_icb_pow_t</td>
<td>1.132545</td>
<td>62.125</td>
<td>19.195</td>
<td>82.212</td>
<td>1.408</td>
</tr>
<tr>
<td>deconv_layer2_icb_ultra_pow_t</td>
<td>0.938185</td>
<td>62.840</td>
<td>19.236</td>
<td>82.989</td>
<td>6.734</td>
</tr>
</tbody>
</table>
Conclusion and Future work

• Conclusion
  – We designed a CNN ASIC for pulse shaping in HEP
  – Front-end and back-end design and validation were done
  – The chip has been taped out

• Contributions
  – Based on the theory, design a specific NN accelerator for HEP scenarios
  – Customized PE, combination of SA and TA, overall structure...
  – Achieve reasonable performance with the available digital process

• Future work
  – Hardware test when the chip is shipped back
  – Improve the PE (analog signals, in-memory operations...)
  – More applications (tracking, high-level triggering...)
Thank you!
And questions...