
Philip Leong (梁恆惠) | Computer Engineering Laboratory
School of Electrical and Information Engineering,
The University of Sydney
Focuses on how to use parallelism to solve demanding problems
- Novel architectures, applications and design techniques using VLSI, FPGA and parallel computing technology

Research
- Reconfigurable computing
- Machine learning
- Nanoscale interfaces
What is an FPGA?

User-customisable integrated circuit

- Dedicated blocks: memory, transceivers and MAC, PLLs, DSPs, ARM cores

Source: Xilinx
Instrumentation of the future can use FPGA-based ML to interpret and process data adaptively in real-time

- Offer new capabilities
- e.g. trigger oscilloscope or spectrum analyzer on an anomaly
- e.g. battery power, high data rates, supercomputer performance in small form factor

There are many applications that could benefit

- test equipment
- network monitors
- prognostics and health management
Challenges in measurement and control are becoming feasible

- Significant improvements in ML algorithms but cannot keep up with sources e.g. hyperspectral imager or wireless transceiver
- Need extremely high throughput
- In control applications we need low latency e.g. triggering data collection in Large Hadron Collider
- Need very low latency

Improvements in throughput and latency enable new applications!
FPGAs offer an opportunity to provide ML algorithms with higher throughput and lower latency through
- Exploration – easily try different ideas to arrive at a good solution
- Parallelism – so we can arrive at an answer faster
- Integration – so interfaces are not a bottleneck
- Customisation – problem-specific designs to improve efficiency

Describe our work on implementations of ML that use these ideas (and can be applied to instrumentation)
Exploration (Online kernel methods)
Parallelisation
Integration
Customisation
Examples are KLMS and KRLS

- Traditional ML algorithms are batch based
  - Make several passes through data
  - Requires storage of the input data
  - Not all data may be available initially
  - Not suitable for massive datasets

- Our approach: online algorithms
  - Incremental, inexpensive state update based on new data
  - Single pass through the data
  - Can be high throughput, low latency
Mapping to a Feature Space

\[ k(x, x') = \exp \left( -\frac{||x - x'||^2}{2\sigma^2} \right) \]

Input Space

\[ f(x_i) = \sum_{j=1}^{N} \alpha_j \kappa(x_i, v_j) \]

Feature Space

› Choose high dimensional feature space (so easily separable)
› Use kernel trick to avoid computing the mapping (fast)
› Do regression/classification using
<table>
<thead>
<tr>
<th>Impl.</th>
<th>M</th>
<th>N</th>
<th>Latency (cycles)</th>
<th>Fmax (MHz)</th>
<th>Time (ns)</th>
<th>CPU (ns)</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector - Flexible (Stratix 5)</td>
<td>8</td>
<td>127</td>
<td>4396</td>
<td>157</td>
<td>28000</td>
<td>141000</td>
<td>5.1x</td>
</tr>
<tr>
<td>Pipeline - Throughput (Virtex 7)</td>
<td>8</td>
<td>16</td>
<td>207</td>
<td>314</td>
<td>3.18</td>
<td>940</td>
<td>296x</td>
</tr>
<tr>
<td>Braided - Latency (Virtex 7)</td>
<td>8</td>
<td>200</td>
<td>13</td>
<td>127</td>
<td>7.87</td>
<td>4025.8</td>
<td>511x</td>
</tr>
<tr>
<td>FASTFOOD - Capacity (Kintex Ultrascale)</td>
<td>1k</td>
<td>16k</td>
<td>1694</td>
<td>500</td>
<td>3388</td>
<td>580000</td>
<td>171x</td>
</tr>
</tbody>
</table>

- M – Input dimension
- N – Dictionary Size (or Sliding Window Size for KRLS)

![Diagram](image-url)
Exploration

Parallelisation (pipelined KNLMS)

Integration

Customisation
Dependency Problem

Cannot process $x_i$ until we update weights from $\{x_{i-1}, y_{i-1}\}$
Removing Dependencies

› Training is usually:
  for (hyperparameters)
    for (inputs)
      learn_model()

› Alternative is to find L independent problems
  - E.g. monitor L different things

› Our approach: run L independent problems (different parameters) in the pipeline
  - Updates ready after L subproblems
  - Less data transfer

  for (inputs)
    for (hyperparameters)
      learn_model()

• Similar approach for multiclass classification (train C(C-1)/2 binary classifiers)
High Throughput KNLMS

\[ \kappa(x, \hat{x}_t) = e^{-\gamma \|x - \hat{x}_t\|^2} \]
\[ \forall i \in \{1, \ldots, N\} \]
\[ D_t = \begin{cases} [D_{t-1}; x] & \text{if } \mu < \mu_0 \\ D_{t-1} & \text{otherwise} \end{cases} \]
\[ \tilde{y} = k^T \alpha \]
\[ \|k\|^2 = k^T k \]
\[ \alpha_t = \alpha_{t-1} + \frac{\eta}{\epsilon + \|k\|^2} (y_t - \tilde{y}_t) k \]
Core with input vector M=8 and dictionary size N=16 (KNLMS)

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Freq (MHz)</th>
<th>Time (ns)</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Pipelined</td>
<td>250</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Non-pipelined</td>
<td>250</td>
<td>800</td>
<td>200</td>
</tr>
<tr>
<td>CPU (C)</td>
<td>3,600</td>
<td>940</td>
<td>235</td>
</tr>
</tbody>
</table>

- Can do online learning from 200 independent data streams at 70 Gbps (160 GFLOPS)
Exploration

Parallelisation

Integration (radio frequency machine learning)

Customisation
Radio Frequency Machine Learning

- Processing radio frequency signals remains a challenge
  - high bandwidth and low latency difficult to achieve
- Autoencoder to do anomaly detection
Train so \( \tilde{x} \sim x \) (done in an unsupervised manner)
Autoencoder learns “normal” representation

- Anomaly if distance between autoencoder output and input large
- FPGA has sufficiently high performance to process each sample of waveform at 200 MHz!
  - This minimises latency and maximises throughput
  - Weights trained on uP and updated on FPGA without affecting inference
Software Defined Radio Architecture

Implemented on Ettus X310 platform

- **Autoencoder training**
- **Autoencoder Parameters** ($W, b$)
- **Hardware Driver**
- **Ingress/Egress Interface**
- **Crossbar**
- **Radio Core**
- **Autoencoder (Optional FFT)**

- From **I/Q samples**
- To **Anomaly/Normal** (can be used by FPGA or PC)
Example
Typical SDR latency >> 1 ms

<table>
<thead>
<tr>
<th>Module</th>
<th>II</th>
<th>Latency (cycles)</th>
<th>BRAM</th>
<th>DSP</th>
<th>FF</th>
<th>LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windower</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1511</td>
<td>996</td>
</tr>
<tr>
<td>FFT</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>48</td>
<td>4698</td>
<td>2796</td>
</tr>
<tr>
<td>NN</td>
<td>1</td>
<td>17</td>
<td>4</td>
<td>1280</td>
<td>213436</td>
<td>13044</td>
</tr>
<tr>
<td>$L_2$-Norm</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>32</td>
<td>1482</td>
<td>873</td>
</tr>
<tr>
<td>Thres</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>Weight Update</td>
<td>258</td>
<td>257</td>
<td>0</td>
<td>0</td>
<td>21955</td>
<td>4528</td>
</tr>
<tr>
<td>Inference (FFT+NN)</td>
<td>1</td>
<td>37</td>
<td>1068</td>
<td>1360</td>
<td>241522</td>
<td>45448</td>
</tr>
<tr>
<td>Inference (NN)</td>
<td>1</td>
<td>29</td>
<td>1068</td>
<td>1312</td>
<td>236824</td>
<td>42652</td>
</tr>
<tr>
<td>Total</td>
<td>N/A</td>
<td>N/A</td>
<td>1068</td>
<td>1360</td>
<td>263477</td>
<td>49976</td>
</tr>
<tr>
<td>Total Util.</td>
<td>N/A</td>
<td>N/A</td>
<td>67%</td>
<td>88%</td>
<td>51%</td>
<td>19%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Throughput</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inference(FFT+NN)</td>
<td>5ns</td>
<td>185ns</td>
</tr>
<tr>
<td>Inference(NN)</td>
<td>5ns</td>
<td>105ns</td>
</tr>
<tr>
<td>Weight Update</td>
<td>1290ns</td>
<td>1285ns</td>
</tr>
</tbody>
</table>
› Exploration
› Parallelisation
› Integration
› Customisation (binarised neural networks)
Inference with Convolutional Neural Networks

Slides from Yaman Umuroglu et. al., “FINN: A framework for fast, scalable binarized neural network inference,” FPGA’17

- tens of megabytes of floating point weight data (from training)

- image to be classified

- billions of floating point multiply-accumulate ops

  (several joules of energy)

- «cat»
The extreme case of quantization
- Permit only two values: +1 and -1
- Binary weights, binary activations
- Trained from scratch, not truncated FP

Courbariaux and Hubara et al. (NIPS 2016)
- Competitive results on three smaller benchmarks
- Open source training flow
- Standard “deep learning” layers
  - Convolutions, max pooling, batch norm, fully connected…

<table>
<thead>
<tr>
<th></th>
<th>MNIST</th>
<th>SVHN</th>
<th>CIFAR-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary weights &amp; activations</td>
<td>0.96%</td>
<td>2.53%</td>
<td>10.15%</td>
</tr>
<tr>
<td>FP weights &amp; activations</td>
<td>0.94%</td>
<td>1.69%</td>
<td>7.62%</td>
</tr>
<tr>
<td>BNN accuracy loss</td>
<td>-0.2%</td>
<td>-0.84%</td>
<td>-2.53%</td>
</tr>
</tbody>
</table>

% classification error (lower is better)
Advantages of BNNs

Vivado HLS estimates on Xilinx UltraScale+ MPSoC ZU19EG

› Much smaller datapaths
  - Multiply becomes XNOR, addition becomes popcount
  - No DSPs needed, everything in LUTs
  - Lower cost per op = more ops every cycle

› Much smaller weights
  - Large networks can fit entirely into on-chip memory (OCM)
  - More bandwidth, less energy compared to off-chip

<table>
<thead>
<tr>
<th>Precision</th>
<th>Peak TOPS</th>
<th>On-chip weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1b</td>
<td>~66</td>
<td>~70 M</td>
</tr>
<tr>
<td>8b</td>
<td>~4</td>
<td>~10 M</td>
</tr>
<tr>
<td>16b</td>
<td>~1</td>
<td>~5 M</td>
</tr>
<tr>
<td>32b</td>
<td>~0.3</td>
<td>~2 M</td>
</tr>
</tbody>
</table>

› fast inference with large BNNs
## Comparison

<table>
<thead>
<tr>
<th>Prior Work</th>
<th>Accuracy</th>
<th>FPS</th>
<th>Power (chip)</th>
<th>Power (wall)</th>
<th>kFPS / Watt (chip)</th>
<th>kFPS / Watt (wall)</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST, SFC-max</td>
<td>95.8%</td>
<td>12.3 M</td>
<td>7.3 W</td>
<td>21.2 W</td>
<td>1693</td>
<td>583</td>
<td>1</td>
</tr>
<tr>
<td>MNIST, LFC-max</td>
<td>98.4%</td>
<td>1.5 M</td>
<td>8.8 W</td>
<td>22.6 W</td>
<td>177</td>
<td>269</td>
<td>1</td>
</tr>
<tr>
<td>CIFAR-10, CNV-max</td>
<td>80.1%</td>
<td>21.9 k</td>
<td>3.6 W</td>
<td>11.7 W</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SVHN, CNV-max</td>
<td>94.9%</td>
<td>21.9 k</td>
<td>3.6 W</td>
<td>11.7 W</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MNIST, Alemdar et al.</td>
<td>97.8%</td>
<td>255.1 k</td>
<td>0.3 W</td>
<td>-</td>
<td>806</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>CIFAR-10, TrueNorth</td>
<td>83.4%</td>
<td>1.2 k</td>
<td>0.2 W</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>SVHN, TrueNorth</td>
<td>96.7%</td>
<td>2.5 k</td>
<td>0.3 W</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

Max accuracy loss: ~3%  
10 – 100x better performance  
CIFAR-10/SVHN energy efficiency comparable to TrueNorth ASIC
› Exploration (Online kernel methods)
› Parallelisation
› Integration
› Customisation
LSTM using HLS tutorial
  - https://github.com/phwl/hlslstm

Kernel methods code e.g. braiding
  - https://github.com/da-steve101/chisel-pipelined-olk

FINN - can do trillions of binary operations per second
  - https://github.com/Xilinx/BNN-PYNQ
Conclusion

- **Exploration**
  - Kernel methods optimised using different algorithms, mathematical techniques, computer architectures, arithmetic

- **Parallelism**
  - Increased by removing dependencies to ensure all stages do work every cycle

- **Integration**
  - In radio frequency, this allows latency to be reduced by 4 orders of magnitude

- **Customisation**
  - Increase parallelism by reducing precision
  - Keep weights on-chip to devote more hardware to arithmetic

- **FPGAs can greatly assist with the implementation of ICEMI’s theme of “measurement and intelligent sensing”**
  - Learning & inference at 70 Gbps
  - Learning & inference with 100 ns latency
  - Image processing @ 12.3 Mfps
  - Multimodal measurements

- **Radio frequency anomaly detector**
  - We are using this to predict physical and media access layer protocols
  - Could also be used as a novel diagnostic instrument - monitor RF output of electronic equipment, detect anomalies
Thank you!

Philip Leong (philip.leong@sydney.edu.au)
http://www.sydney.edu.au/people/philip.leong