Braiding: a Scheme for Resolving Hazards in NORMA

Stephen Tridgell, Duncan J.M. Moss, Nicholas J. Fraser and Philip H.W. Leong
School of Electrical and Information Engineering,
The University of Sydney
How to beat other people to the money (latency)

› Low latency trading looks to trade in transient situations where market equilibrium disturbed

- 1ms reduction in latency can translate to $100M per year

› Latency also important to: prevent blackouts (cascading faults), turn off machine before it damages itself, etc

Information Week: Wall Street's Quest To Process Data At The Speed Of Light
Latency infrastructure already available

Exablaze Low-Latency Products

ExaLINK Fusion 48 SFP+ port layer
2 switch for replicating data typical 5
ns fanout, 95 ns aggregation, 110 ns
layer 2 switch

Xilinx Ultrascale FPGA, QDR SRAM,
ARM processor

ExaNIC X10 typical raw frame
latency 60 bytes 780 ns

What we can’t do: ML with this
type of latency

Source: exablaze.com
Examples are KLMS and KRLS

- Traditional ML algorithms batch based
  - Several passes through data
  - Requires storage of input data
  - 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
Obstacle to Pipelining

Dependency Problem

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

- This work
  - Implementation of an online machine learning scheme: NORMA
  - Resolve read-after-write dependencies through braiding
Datapath for NORMA

\[ f(x) = \sum_{i=1}^{D} \alpha_i \kappa(x, d_i) \]
Naive Online regularised Risk Minimization Algorithm

› Finds Dictionary $d_i$, and $\alpha_i$ (weights)

\[ f(x) = \sum_{i=1}^{D} \alpha_i \kappa(x, d_i) \]

› Minimise predictive error ($R_{\text{inst, } \lambda}$) by taking a step in direction of gradient

\[ f_{t+1} = f_t - \eta_t \partial f R_{\text{inst, } \lambda}[f, x_{t+1}, y_{t+1}] \bigg|_{f=f_t} \]

› Can be used for classification, regression, novelty detection

› Update for novelty detection

\[
(\alpha_i, \alpha_t, \rho) = \begin{cases} 
(\Omega \alpha_i, 0, \rho + \eta \nu) & \text{if } f(x_t) \geq \rho \quad \text{Add } x_{t+1} \text{ to dictionary} \\
(\Omega \alpha_i, \eta, \rho - \eta(1 - \nu)) & \text{otherwise}
\end{cases}
\]
\[(\alpha_i, \alpha_t, \rho) = \begin{cases} 
(\Omega \alpha_i, 0, \rho + \eta \nu) & \text{if } f(x_t) \geq \rho \quad \text{(Add } x_t \text{ to dictionary)} \\
(\Omega \alpha_i, \eta, \rho - \eta(1 - \nu)) & \text{otherwise}
\end{cases}\]
(α_i, α_t, ρ) = \begin{cases} 
(Ωα_i, 0, ρ + ην) & \text{if } f(x_t) ≥ ρ \quad \text{(Add } x_t \text{ to dictionary)} \\
(Ωα_i, η, ρ - η(1 - ν)) & \text{otherwise}
\end{cases}
Properties of NORMA

› NORMA is a sliding window algorithm
  - If new dictionary entry added \([d_1, \cdots d_D] \rightarrow [x_t, d_1, \cdots d_{D-1}]\)
  - Weight update is just a decay \(\alpha_i \rightarrow \Omega \alpha_i\)
  - Update cost is small compared to computing \(f(x_t)\)

› Is this really true?
Recall carry select adder
- implement both cases in parallel and select output

\[ f(x_{t+1}) = \sum_{i=1}^{D} \alpha_i \kappa(x_{t+1}, d_i) \]

Use the previous dictionary for \( x_t \) denoted \( \hat{d}_i \)

\[ f(x_{t+1}) = \sum_{i=1}^{D-1} \Omega \hat{\alpha}_i \kappa(x_{t+1}, \hat{d}_i) + \text{something} \]

if \( x_t \) is added then this term = \( \alpha_{x_t} \kappa(x_{t+1}, x_t) \)

if \( x_t \) is not added then this term = \( \Omega \hat{\alpha}_D \kappa(x_{t+1}, \hat{d}_D) \)
Braiding Datapath

\[
\begin{align*}
\kappa(x_{t+1}, x_t) & \\
\sum_{i=1}^{D-1} \alpha_i \kappa(x_{t+1}, d_i) & \\
\alpha_D \kappa(x_{t+1}, d_D) & \\
\end{align*}
\]

\[
\begin{align*}
\alpha_{x_t} & \\
\Omega & \\
\Omega & \\
\end{align*}
\]

\[
\begin{align*}
+ & \\
+ & \\
f_{t-1}(x_t) < \rho & \\
\end{align*}
\]

\[
\begin{align*}
\text{Mux} & \\
\end{align*}
\]

\[
\begin{align*}
f_t(x_{t+1}) & \\
\end{align*}
\]
Generalised to p cycles

\[ f_t(x_{t+1}) = \sum_{i=1}^{D-p} \Omega^p \hat{\alpha}_i \kappa(x_{t+1}, \hat{d}_i) \]

\[ q \begin{cases} 0 & \text{if } x_{t+1-p} \text{ is not added} \\ \Omega^{p-1} \alpha_{x_{t+1-p}} \kappa(x_{t+1}, x_{t+1-p}) & \text{otherwise} \\ 0 & \text{if } x_{t+2-p} \text{ is not added} \\ \Omega^{p-2} \alpha_{x_{t+2-p}} \kappa(x_{t+1}, x_{t+2-p}) & \text{otherwise} \\ \vdots \\ 0 & \text{if } x_t \text{ is not added} \\ \alpha_{x_t} \kappa(x_{t+1}, x_t) & \text{otherwise} \\ \sum_{i=D-p+1}^{D-q} \Omega^p \hat{\alpha}_i \kappa(x_{t+1}, \hat{d}_i) \end{cases} \]
› Implemented in Chisel
› On XC7VX485T-2FFG1761C achieves ~133 MHz
› Area $O(FD^2)$ ($F=$dimensionality of input vector), time complexity $O(FD)$
› Speedup 500x compared with single core CPU i7-4510U (8.10 fixed)

<table>
<thead>
<tr>
<th>F=8, D=</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (MHz)</td>
<td>133</td>
<td>138</td>
<td>137</td>
<td>131</td>
<td>127</td>
</tr>
<tr>
<td>DSPs (/2,800)</td>
<td>309</td>
<td>514</td>
<td>911</td>
<td>1,679</td>
<td>2,556</td>
</tr>
<tr>
<td>Slices (/759,000)</td>
<td>4,615</td>
<td>8,194</td>
<td>14,663</td>
<td>29,113</td>
<td>46,443</td>
</tr>
<tr>
<td>Latency (cycles)</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>Speedup (×)</td>
<td>47</td>
<td>91</td>
<td>178</td>
<td>344</td>
<td>509</td>
</tr>
<tr>
<td>Latency reduction (×)</td>
<td>4.69</td>
<td>8.30</td>
<td>14.9</td>
<td>28.7</td>
<td>39.2</td>
</tr>
</tbody>
</table>
Core with input vector $F=8$ and dictionary size $D=16$

<table>
<thead>
<tr>
<th>Design</th>
<th>Precision</th>
<th>Freq MHz</th>
<th>Latency Cycles</th>
<th>T.put Cycles</th>
<th>Latency nS</th>
<th>T.put nS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector KNLMS</td>
<td>Single</td>
<td>282</td>
<td>479</td>
<td>479</td>
<td>1,699</td>
<td>1,699</td>
</tr>
<tr>
<td>Pipelined KNLMS</td>
<td>Single</td>
<td>314</td>
<td>207</td>
<td>1</td>
<td>659</td>
<td>3.2</td>
</tr>
<tr>
<td>Braided NORMA</td>
<td>8.10</td>
<td>113</td>
<td>10</td>
<td>1</td>
<td>89</td>
<td>8.8</td>
</tr>
</tbody>
</table>
Braiding: rearrangement of a sliding window algorithm for hardware implementations
- NORMA used but other ML algorithms possible

Compared with pipelined KNLMS,
- 20x lower latency at 1/3 of the throughput

Open source (GPLv2): github.com/da-steve101/chisel-pipelined-olk