Kernel Normalised Least Mean Square
› Microcoded KRLS had good programmability but minimal overlap of instructions
  - Datapath idle much of the time

› Ideas
  - Use a simpler algorithm
  - Hardwire datapath and pipeline

› How do we deal with dependencies?
KNLMS Algorithm

› Requires
  - A dictionary D which is a subset of the training set
  - A vector of weights $\alpha$, one for each entry in D
  - A kernel function $\kappa(X_i, X_j)$
› $\kappa$ chosen at design time, D and $\alpha$ are learned from the data
› Prediction is
  $$\tilde{y}_i = \sum_{n=1}^{N} \alpha_n \kappa(x_i, \tilde{x}_n)$$
› We use Gaussian kernel which is a universal approximator
  $$\kappa(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$$
1. Evaluate $\kappa(X_i, X_j)$ between new input $x_n$ and each $D_{n-1}$ to form kernel vector $k$

2. If $\max(k) < \mu_0$ add $x_n$ to dictionary to form $D_n$

3. Update weights using

$$\alpha_n = \alpha_{n-1} + \frac{\eta}{\epsilon + k^T k} (y_n - k^T \alpha_{n-1}) k$$

How do we find $\kappa, \mu_0, \epsilon$ and $\eta$? Using a parameter search
If we add $L$ pipeline registers, we have to wait $L \sim 200$ cycles before we can update

$$\alpha_n = \alpha_{n-1} + \frac{\eta}{\epsilon + k^T k} (y_n - k^T \alpha_{n-1}) k.$$
Removing Dependencies

- Weight update is
  \[ \alpha_n = \alpha_{n-1} + \frac{\eta}{\epsilon + k^T k} (y_n - k^T \alpha_{n-1})k \]

- Training is usually:
  - Instead we run L independent problems (different parameters) in the pipeline
    - Updates ready after L subproblems
    - Less data transfer

for (hyperparameters)
  for (inputs)
    learn_model()
High Throughput KRLS

\[ \kappa(x_i, \tilde{x}_i) = e^{-\gamma \|x_i - \tilde{x}_i\|} \]

\[ \forall i \in \{1, \ldots, N\} \]

\[ D_i = \begin{cases} [D_{i-1}; x] & \text{if } \mu < \mu_0 \\ D_{i-1} & \text{otherwise} \end{cases} \]

\[ \tilde{y} = k^T \alpha \]

\[ ||k||^2 = k^T k \]

\[ \alpha_i = \alpha_{i-1} + \frac{\eta}{\epsilon + ||k||^2} (y_i - \tilde{y}_i) k \]
RIFFA 2.0 requires a PCIe enabled workstation and a FPGA on a board with PCIe

- Simple software interface
- Provides a first word fall through FIFO interface for reading/writing data (no knowledge of bus addresses, buffer sizes, or PCIe packet formats is required)
- Communicates using direct memory access (DMA) transfers and interrupt signaling
- Supports AVNet Spartan LX150T, Xilinx ML605, Xilinx VC707, Altera DE5-Net, DE4 and DE2i boards.
- Supports Ubuntu, Fedora, Windows 7
## Hardware Interface

<table>
<thead>
<tr>
<th>Name</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHNL_RX_CLK</td>
<td>O</td>
<td>Provide the clock signal to read data from the incoming FIFO.</td>
</tr>
<tr>
<td>CHNL_RX</td>
<td>I</td>
<td>Goes high to signal incoming data. Will remain high until all incoming data is written to the FIFO.</td>
</tr>
<tr>
<td>CHNL_RX_ACK</td>
<td>O</td>
<td>Must be pulsed high for at least 1 cycle to acknowledge the incoming data transaction.</td>
</tr>
<tr>
<td>CHNL_RX_LAST</td>
<td>I</td>
<td>High indicates this is the last receive transaction in a sequence.</td>
</tr>
<tr>
<td>CHNL_RX_LEN[31:0]</td>
<td>I</td>
<td>Length of receive transaction in 4 byte words.</td>
</tr>
<tr>
<td>CHNL_RX_OFF[30:0]</td>
<td>I</td>
<td>Offset in 4 byte words indicating where to start storing received data (if applicable in design).</td>
</tr>
<tr>
<td>CHNL_RX_DATA[DWIDTH-1:0]</td>
<td>I</td>
<td>Receive data.</td>
</tr>
<tr>
<td>CHNL_RX_DATA_VALID</td>
<td>I</td>
<td>High if the data on CHNL_RX_DATA is valid.</td>
</tr>
<tr>
<td>CHNL_RX_DATA_REN</td>
<td>O</td>
<td>When high and CHNL_RX_DATA_VALID is high, consumes the data currently available on CHNL_RX_DATA.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHNL_TX_CLK</td>
<td>O</td>
<td>Provide the clock signal to write data to the outgoing FIFO.</td>
</tr>
<tr>
<td>CHNL_TX</td>
<td>O</td>
<td>Set high to signal a transaction. Keep high until all outgoing data is written to the FIFO.</td>
</tr>
<tr>
<td>CHNL_TX_ACK</td>
<td>I</td>
<td>Will be pulsed high for at least 1 cycle to acknowledge the transaction.</td>
</tr>
<tr>
<td>CHNL_TX_LAST</td>
<td>O</td>
<td>High indicates this is the last send transaction in a sequence.</td>
</tr>
<tr>
<td>CHNL_TX_LEN[31:0]</td>
<td>O</td>
<td>Length of send transaction in 4 byte words.</td>
</tr>
<tr>
<td>CHNL_TX_OFF[30:0]</td>
<td>O</td>
<td>Offset in 4 byte words indicating where to start storing sent data in the PC thread's receive buffer.</td>
</tr>
<tr>
<td>CHNL_TX_DATA[DWIDTH-1:0]</td>
<td>O</td>
<td>Send data.</td>
</tr>
<tr>
<td>CHNL_TX_DATA_VALID</td>
<td>O</td>
<td>Set high when the data on CHNL_TX_DATA valid. Update when CHNL_TX_DATA is consumed.</td>
</tr>
<tr>
<td>CHNL_TX_DATA_REN</td>
<td>I</td>
<td>When high and CHNL_TX_DATA_VALID is high, consumes the data currently available on CHNL_TX_DATA.</td>
</tr>
</tbody>
</table>

The value of DWIDTH will be either 32, 64, or 128.
Downstream and Upstream Transfer Sequence

Source: Matthew Jacobsen UCSD
Receiving Data

RX Timing Diagram

- CHNL_RX_CLK
- CHNL_RX
- CHNL_RX_ACK
- CHNL_RX_LAST
- CHNL_RX_LEN[31:0]: 16
- CHNL_RX_OFF[30:0]: 0
- CHNL_RX_DATA[63:0]: D0, D1, D2, D3, D4, D5, D6, D7
- CHNL_RX_DATA_VALID
- CHNL_RX_DATA_REN

Source: Matthew Jacobsen UCSD
Transmitting Data

**TX Timing Diagram**

- **CHNL_TXCLK**
- **CHNL_TX**
- **CHNL_TX_ACK**
- **CHNL_TX_LAST**
- **CHNL_TX_LEN[31:0]** 16
- **CHNL_TX_OFF[30:0]** 0
- **CHNL_TX_DATA[63:0]** D0, D1, D2, D3, D4, D5, D6, D7
- **CHNL_TX_DATA_VALID**
- **CHNL_TX_DATA_REN**

Source: Matthew Jacobsen UCSD
RIFFA 2.0 Performance

Source: Matthew Jacobsen UCSD
System Integration
### Performance

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Algorithm</th>
<th>DSPs</th>
<th>Freq MHz</th>
<th>Time ns</th>
<th>Slowdown rel. to Float</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve</td>
<td>KNLMS</td>
<td>12</td>
<td>96.7</td>
<td>7,829</td>
<td>2,462</td>
</tr>
<tr>
<td>Float</td>
<td>KNLMS</td>
<td>1267</td>
<td>314</td>
<td>3.18</td>
<td>1</td>
</tr>
<tr>
<td>Fused</td>
<td>KNLMS</td>
<td>787</td>
<td>289</td>
<td>3.46</td>
<td>1.1</td>
</tr>
<tr>
<td>System</td>
<td>KNLMS</td>
<td>691</td>
<td>250</td>
<td>13.6</td>
<td>4.3</td>
</tr>
<tr>
<td>CPU (C)</td>
<td>KNLMS</td>
<td>-</td>
<td>3,600</td>
<td>940</td>
<td>296</td>
</tr>
<tr>
<td>CPU (KAFBOX)</td>
<td>KNLMS</td>
<td>-</td>
<td>3,600</td>
<td>73,655</td>
<td>23,162</td>
</tr>
<tr>
<td>Pang et al. [2013]</td>
<td>SW-KRLS</td>
<td>30</td>
<td>237</td>
<td>9,000</td>
<td>2,830</td>
</tr>
</tbody>
</table>
Achieved a 2830x speedup over a microcoded FPGA implementation
- by reducing flexibility, and
- fully pipelining the design

Despite using a good library, the PCIe interface reduced performance by a factor of 4.3