K-Means Implementation on FPGA for High-dimensional Data Using Triangle Inequality

Zhongduo Lin, Charles Lo, Paul Chow

High-Performance Reconfigurable Computing Group
University of Toronto

September-13-12
Why K-means?

• One of the most widely used unsupervised clustering algorithms in data mining and machine learning
Whats K-means

• **Unsupervised vs Supervised**
  – Classes are predetermined or not

• **A simple iterative clustering algorithm that partitions a given dataset into k clusters**
Basic K-means

1) K initial means selected
2) K clusters are created by assigning points to the nearest mean
3) The centroid of each clusters becomes the new mean
4) Repeat step 2 and 3 until convergence has been reached
Why Triangle Inequality

• **Big Data Era**
  – Data size
  – Number of dimensions
  – Number of clusters

• **Optimization**
  – Kd-tree with filter algorithm
  – Triangle inequality
Triangle Inequality

- $z < x + y$
- $x > z - y$
- If $x < z/2$ then $x < y$
Triangle Inequality

- \( z < x + y \)
  - \( x: pc3 \)
  - \( y: c1c3 \)
  - \( z: pc3 \)
  - \( x+y: \text{upper bound} \)

- \( x > z - y \)
  - \( x: pc4 \)
  - \( y: c2c4 \)
  - \( z: pc2 \)
  - \( z-y: \text{lower bound} \)
K-means with Triangle Inequality

- Keep the upper bound to the assigned center: n
- Keep the lower bound to all the centers: kn
K-means with Triangle Inequality

1. For points $x$ and centers $c$ such that $c \neq c(x)$ & $u(x) > l(x, c)$ & $u(x) > \frac{1}{2} d(c(x), c)$ compute $d(x, c)$ and $d_{min} = \min d(x, c)$, set $c_{min}$ to the cluster with distance $d_{min}$ to the point.

2. If any distance is computed in step 1, compute $d(x, c(x))$. if $d(x, c(x)) > d_{min}$ then assign $c(x) = c_{min}$

3. For each center $c$, let $m(c)$ be the mean of the points assigned to $c$.

4. For each point $x$ and center $c$, assign $l(x, c) = \max \{l(x, c) - d(c, m(c)) \leq 0\}$.

5. For each point $x$, assign $u(x) = u(x) + d(m(c(x)), c(x))$

6. Replace each center $c$ by $m(c)$.
Time Overhead of Triangle Inequality

• Distance between centers: \( d(c(x), c) \)
  – Not implemented

• Distance between new centers and the old ones: \( d(c, m(c)) \)
  – Parallel with updating bounds
Optimization for HW: square root

- **Square root elimination**
  - Distance squared
    
    \[
    u(x) = u(x) + d(m(c(x)), c(x))
    \]

- **bounds for**

    \[
    (x \pm y)^2 = x^2 \pm 2xy + y^2.
    \]
Optimization for HW: square root

1. Let $x y_{\text{min}} = \min \{x^2, y^2\}$ and $x y_{\text{max}} = \max \{x^2, y^2\}$

2. Rewrite $x y$ as $x y_{\text{min}} \times \sqrt{x y_{\text{max}} / x y_{\text{min}}}$

3. Let $i = \log_2(x y_{\text{min}})$ and $j = \log_2(x y_{\text{max}})$

4. $x y_{\text{approx}_n} = x y_{\text{min}} \ll (\frac{j-i}{2} + 1)$, where $\ll$ is a shift left operator
Optimization for HW: square root

1. Let $xy_{\text{min}} = \min \{x^2, y^2\}$ and $xy_{\text{max}} = \max \{x^2, y^2\}$

2. Rewrite $xy$ as $xy_{\text{min}} \times \sqrt{xy_{\text{max}} / xy_{\text{min}}}$

3. Let $i = \log_2(xy_{\text{min}})$ and $j = \log_2(xy_{\text{max}})$

4. $xy_{\text{approx} \_n} = xy_{\text{min}} \ll \left( \frac{j - i}{2} + 1 \right)$, where $\ll$ is a shift left operator

$x y_{\text{approx} \_a} = xy_{\text{min}} \ll \left( \frac{j - i + 1}{2} \right)$
Optimization for HW: square root

- Ratio: $x/y$
- $\text{sum: } (x + y)^2 \quad \text{diff: } (x - y)^2$

![Graph showing approximation error vs ratio](image)
Optimization for HW: square

• 8-bit square calculator for 6-LUT FPGA

\[
s^2 = (s_7s_6s_5s_4s_3s_2s_1s_0)^2 = \\
(s_7s_6 \ll 6 + s_5s_4s_3 \ll 3 + s_2s_1s_0)^2 = \\
(s_7s_6^2 \ll 12 \mid s_5s_4s_3^2 \ll 6 \mid s_2s_1s_0^2) + \\
((0s_7s_6 \times s_5s_4s_3) \ll 6 \mid (s_5s_4s_3 \times s_2s_1s_0)) \ll 4 + \\
(0s_7s_6 \times s_2s_1s_0) \ll 7
\]
Optimization for HW: square

• Comparison: 4-LUT

Table 1. Comparison between two square implementations

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>38</td>
<td>59</td>
<td>35.6 %</td>
</tr>
<tr>
<td>Logic delay (ns)</td>
<td>3.734</td>
<td>4.524</td>
<td>17.5 %</td>
</tr>
</tbody>
</table>
Hardware Platform

- ML605 Evaluation Board:
  - XC6VLX240T
  - 512 MB DDR3 (Max BW: 6.4GB)
  - PCIe interface (8-lane Gen 1)
Interface Overview

Fig. 2. HW Interface Overview
Interface Overview

Fig. 2. HW Interface Overview
Fig. 3. HA Architecture
HW Architecture

Fig. 3. HA Architecture
HW Architecture

Fig. 3. HA Architecture
Benchmarks: $k=10$, $d=1024$

- **Mnist**: gray scale picture of digits 0-9  
  - 28*28 to 32*32  
  - Initial centers: manually picked up

- **Uniform Random (UR)**  
  - No seed is set  
  - Initial centers: first 10 points
Result: approximation

- Distance calculation ratio:

  \[
  \frac{\text{number of distance calculations with optimization}}{\text{number of distance calculations without optimization}}
  \]

- ORI: original triangle inequality optimization
- APT: naïve approximation
- AAPT: aggressive approximation
Result: approximation

Fig. 4. Optimization performance for MNIST

Fig. 5. Optimization performance for UR
HW experiment: cost

- 10% more slice LUTs
- 3.4% more registers
- BRAM!!
HW experiment: speed

- Total time approximation
  \[(50 + 32 + \frac{k \times Dim}{N_{div}} + k \times n) \times I + N_D \times \left(\frac{Dim}{N_{dist}} - 1\right)\]

- When \(n\) is big enough
  \[(R_d + (1 - R_d) \frac{N_{dist}}{Dim}) T\]

- For 32000 MNIST data, \(R_d = 12\%\), processing time: 0.23T, saving 77\%
HW experiment: speed

- **SW platform:**
  - Intel Quad-core i5-2500 CPU, 1 thread
  - 3.3GHz, 4GB DDR2

- **HW platform:**
  - 100MHz

### Table 4. Execution time of different implementations

<table>
<thead>
<tr>
<th>data size</th>
<th>baseline sw</th>
<th>optimized sw</th>
<th>optimized hw</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>807 ms</td>
<td>294 ms</td>
<td>5 ms</td>
</tr>
</tbody>
</table>
Future Work

• Store the bounds in external memory
• Parallelism between different centers
• Better comparison with software implementation
Q & A