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

Zhongduo Lin, Charles Lo, Paul Chow

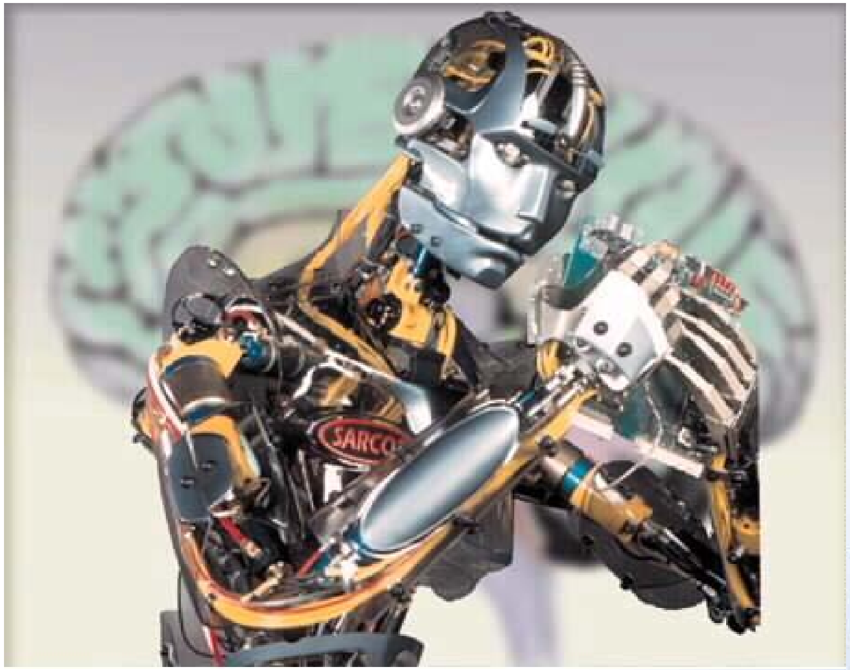
High-Performance Reconfigurable Computing Group

University of Toronto

September-13-12

Why K-means?

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

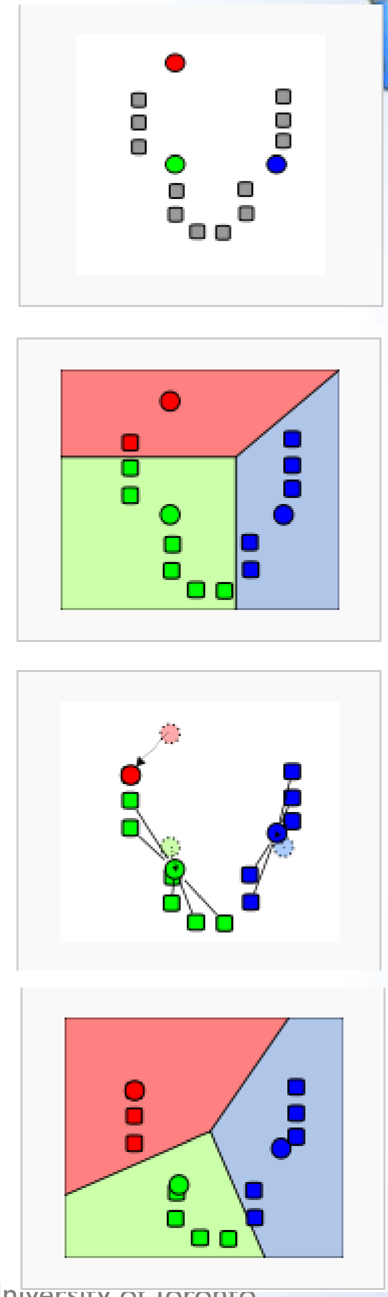


Whats K-means

- 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

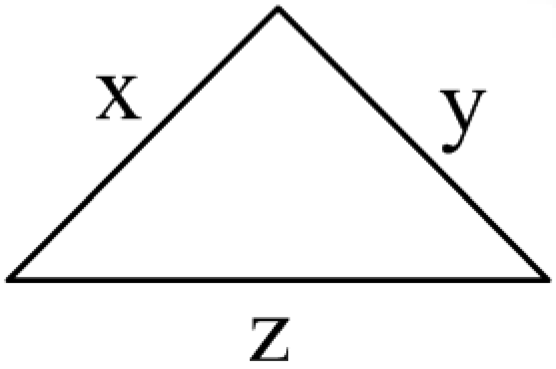


Why Triangle Inequality

- Big Data Era
  - – Data size
  - – Number of dimensions
  - – Number of clusters
- Optimization
  - – Kd-tree with filter algorithm
  - – Triangle inequality

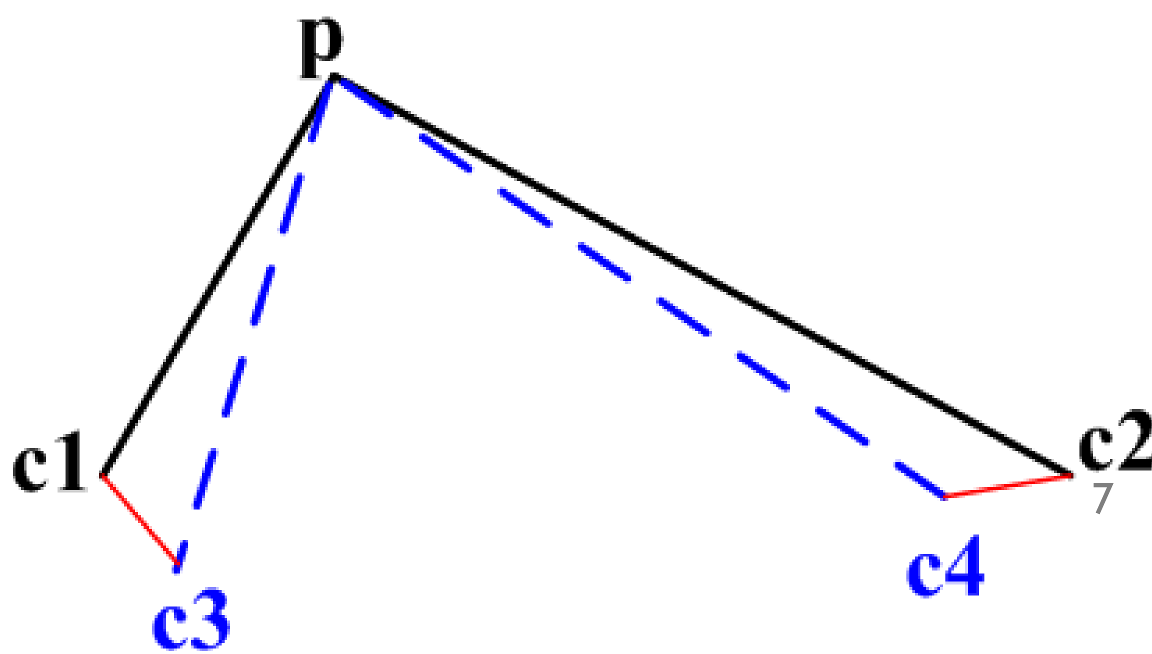
Triangle Inequality

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



Triangle Inequality

- z < x + y
  - x: pc3
  - y: c1c3
  - z: pc3
  - x+y: upper bound
- x > z – y
  - x: pc4
  - y: c2c4
  - z: pc2
  - z-y: 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
   $c \neq c(x)$ && $u(x) > l(x,c)$ && $\boxed{u(x) > \frac{1}{2}d(c(x),c)}$
   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
   $l(x,c) = \max\{l(x,c) - \boxed{d(c,m(c))}, 0\}$.

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
  - $u(x) = \bar{u}(x) + d(m(\bar{c}(x)), c(x))$
- bounds for $(x \pm y)^2$

$$(x \pm y)^2 = x^2 \pm 2xy + y^2.$$

Optimization for HW: square root

1. Let $xy_{min} = \min\{x^2, y^2\}$ and $xy_{max} = \max\{x^2, y^2\}$
2. Rewrite xy as $xy_{min} \times \sqrt{xy_{max}/xy_{min}}$
3. Let $i = \log_2(xy_{min})$ and $j = \log_2(xy_{max})$
4. $xy_{approx\_n} = xy_{min} \ll (\frac{(j-i)}{2} + 1)$,
   where $\ll$ is a shift left operator

Optimization for HW: square root

1. Let $xy_{min} = \min\{x^2, y^2\}$ and $xy_{max} = \max\{x^2, y^2\}$
2. Rewrite xy as $xy_{min} \times \sqrt{xy_{max}/xy_{min}}$
3. Let $i = \log_2(xy_{min})$ and $j = \log_2(xy_{max})$
4. $\boxed{xy_{approx\_n} = xy_{min} \ll (\frac{(j-i)}{2} + 1),}$ where $\ll$ is a shift left operator

$$xy_{approx\_a} = xy_{min} \ll (\frac{(j-i+1)}{2})$$

Optimization for HW: square root

- Ratio: x/y
- sum: $(x+y)^2$ diff: $(x-y)^2$

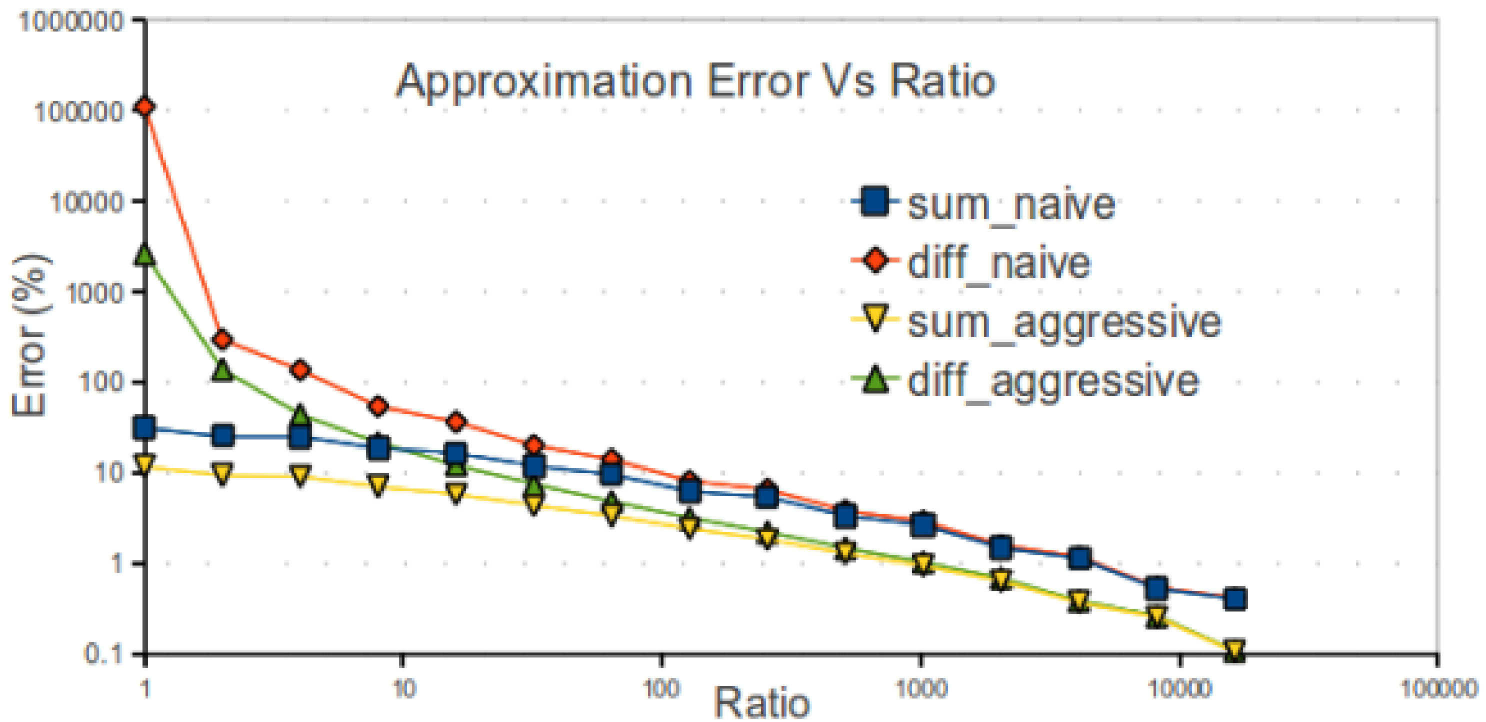


Fig. 1. Approximation Error

Optimization for HW: square

- 8-bit square calculator for 6-LUT FPGA

$$
\begin{aligned}
&s^2 = (s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0)^2 = \\
&(s_7 s_6 \ll 6 + s_5 s_4 s_3 \ll 3 + s_2 s_1 s_0)^2 = \\
&(s_7 s_6{}^2 \ll 12 \mid s_5 s_4 s_3{}^2 \ll 6 \mid s_2 s_1 s_0{}^2) + \\
&((0 s_7 s_6 \times s_5 s_4 s_3) \ll 6 \mid (s_5 s_4 s_3 \times s_2 s_1 s_0)) \ll 4 + \\
&(0 s_7 s_6 \times s_2 s_1 s_0) \ll 7
\end{aligned}
$$

Optimization for HW: square

- Comparison: 4-LUT

Table 1. Comparison between two square implementations

	V6 Opt.	[4]	improvement
LUTs	38	59	35.6 %
Logic delay (ns)	3.734	4.524	17.5 %

Hardware Platform

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

Interface Overview

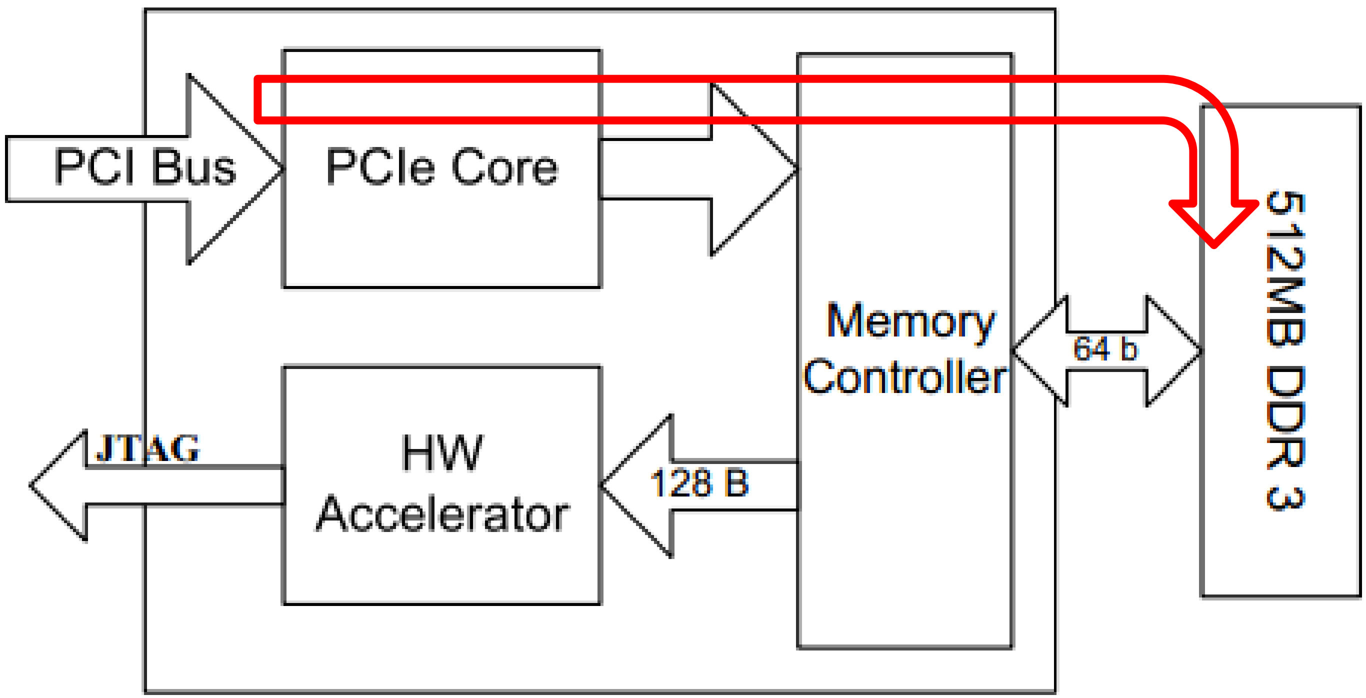


Fig. 2. HW Interface Overview

Interface Overview

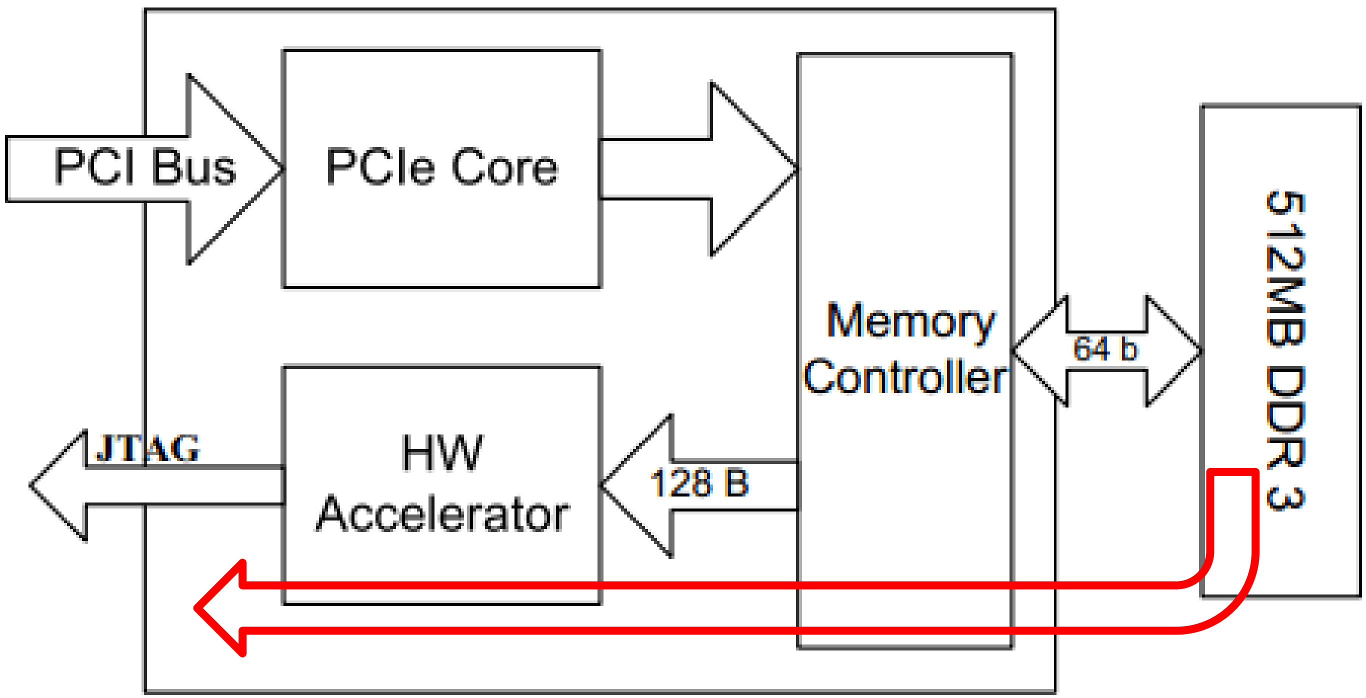


Fig. 2. HW Interface Overview

HW Architecture

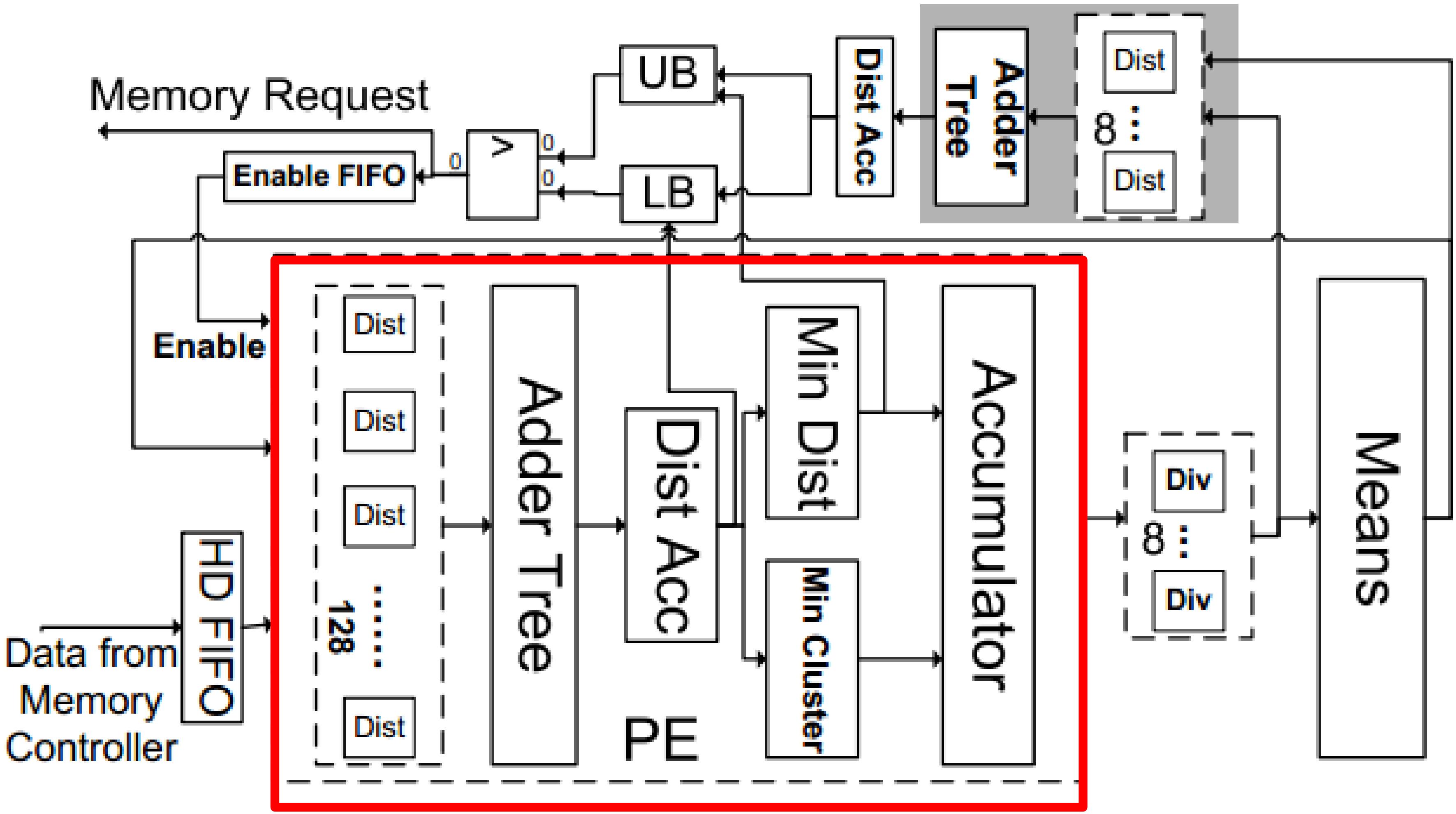


Fig. 3. HA Architecture

HW Architecture

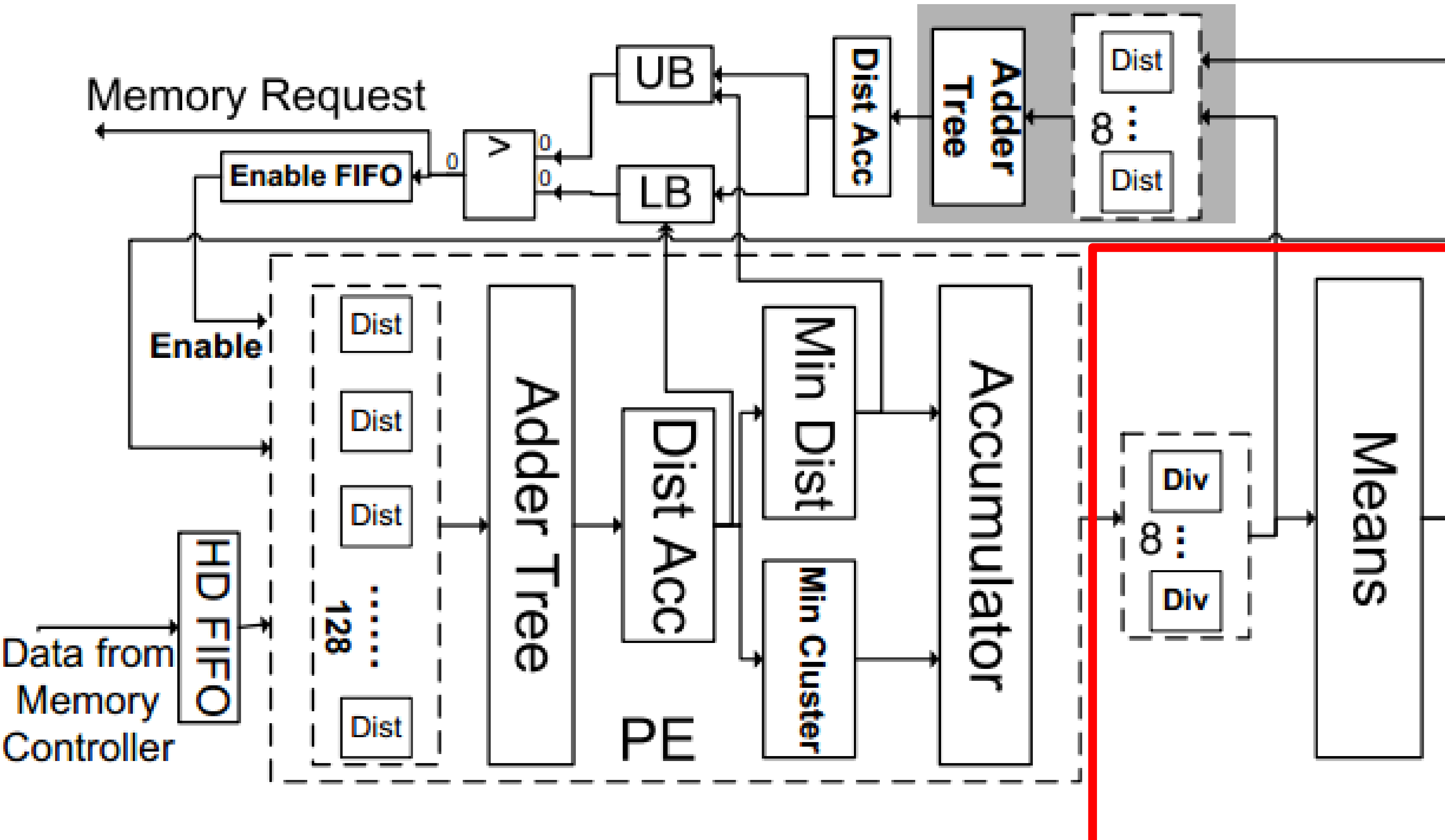


Fig. 3. HA Architecture

HW Architecture

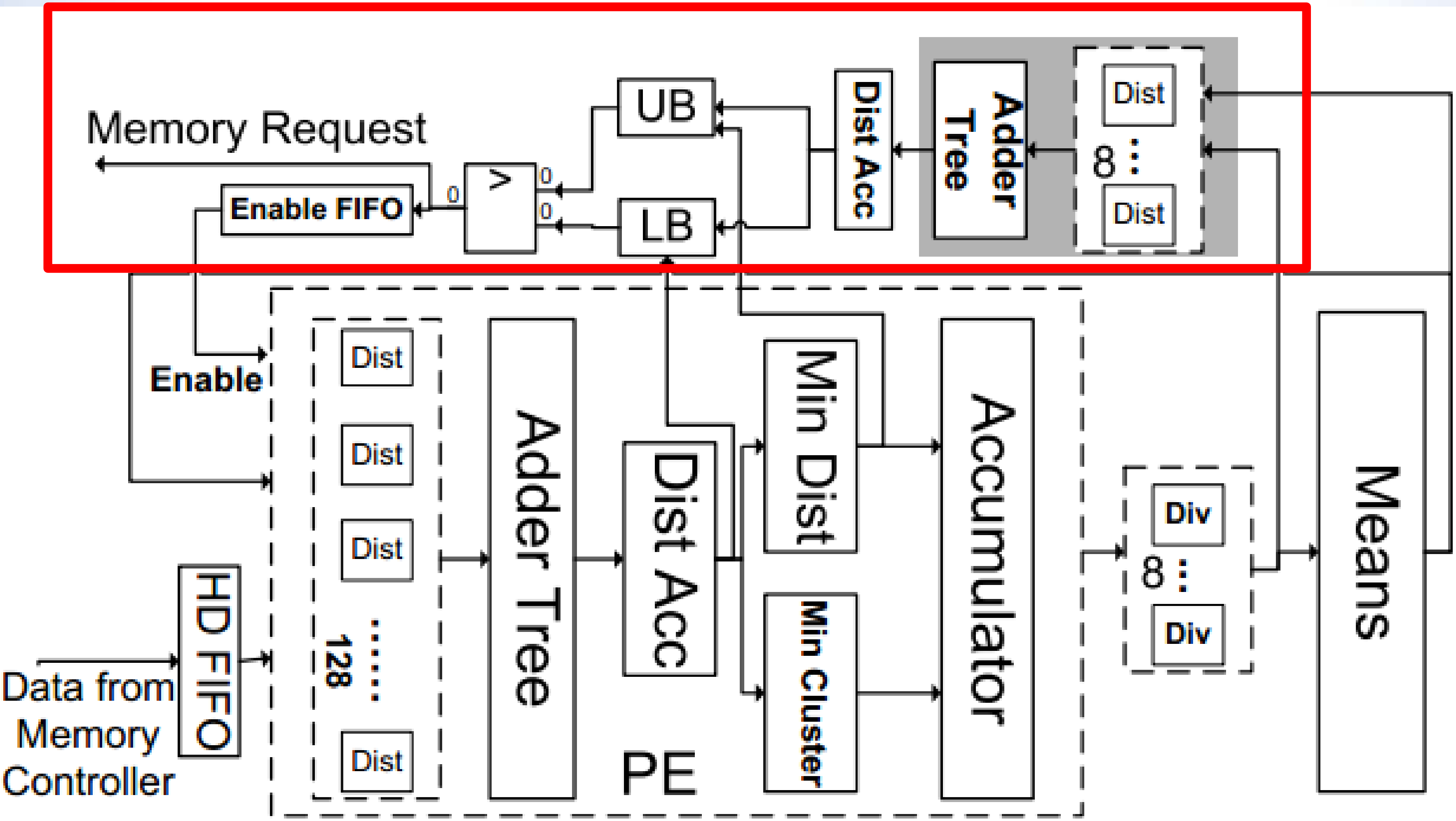


Fig. 3. HA Architecture

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

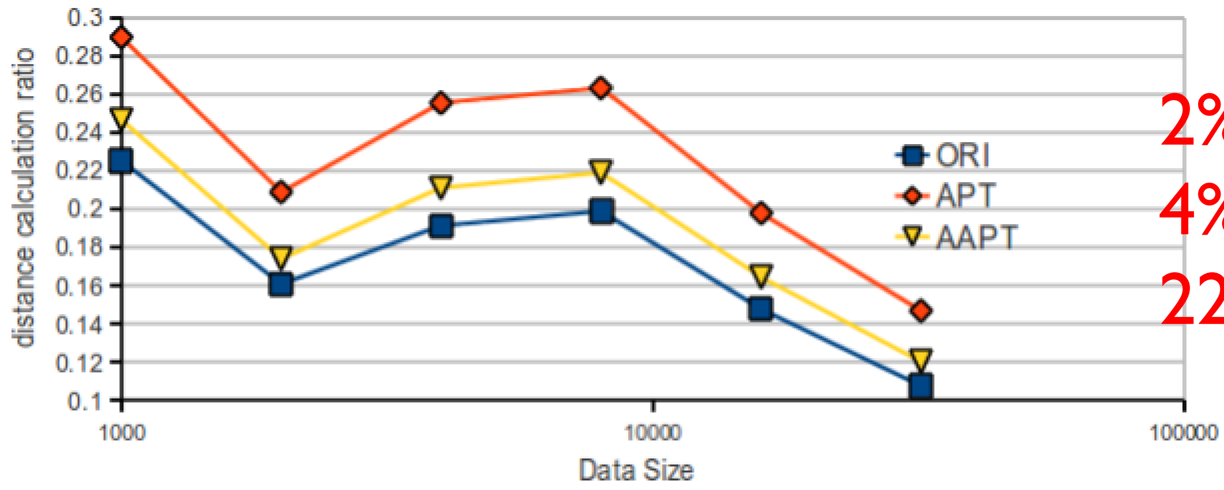
Result: approximation

- Distance calculation ratio:

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

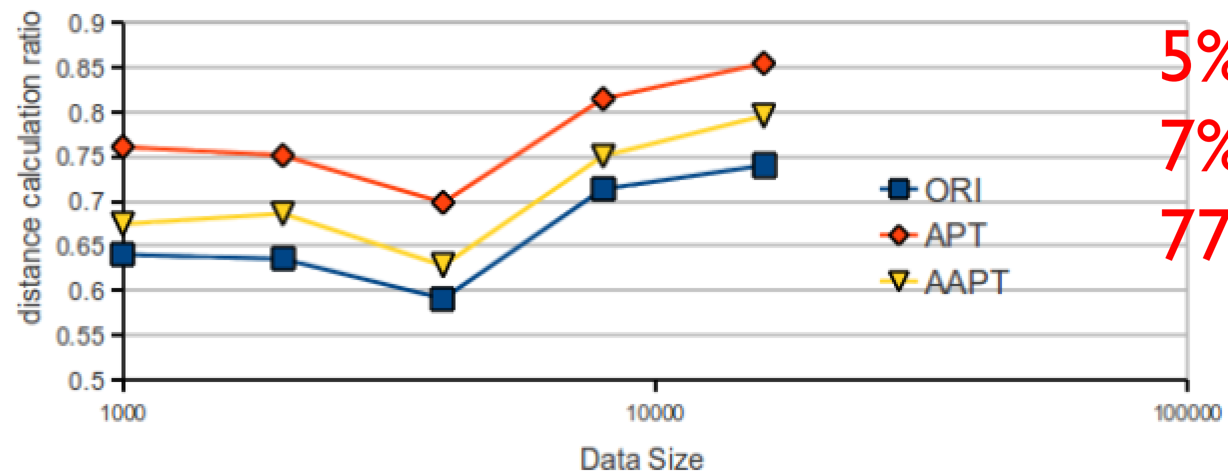
- ORI: original triangle inequality optimization
- APT: naïve approximation
- AAPT: aggressive approximation

Result: approximation



2%
4%
22.7%

Fig. 4. Optimization performance for MNIST



5%
7%
77.6%

Fig. 5. Optimization performance for UR

HW experiment: cost

- 10% more slice LUTs
- 3.4% more registers
- BRAM!!

Table 2. Implementation result with optimization

Data Size	1024	4096	16384
Slice LUTs	44194	45021	43269
Registers	22521	22600	22453
RAM	198	287	403

Table 3. Implementation result of baseline system

Data Size	1024	4096	16384
Slice LUTs	40466	40399	40455
Slice Registers	21630	21640	21645
RAM	170	172	172

HW experiment: speed

- Total time approximation

$$(50 + 32 + \frac{k \times Dim}{N_{div}} + kn) \times I + N_D \times (\frac{Dim}{N_{dist}} - 1)$$

- 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%

27

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

data size	baseline sw	optimized sw	optimized hw
1024	807 ms	294 ms	5 ms

Future Work

- Store the bounds in external memory
- Parallelism between different centers
- Better comparison with software implementation

Q&A