Processing-in-Storage for Machine Learning and Bioinformatics

Roman Kaplan, Leonid Yavits and Ran Ginosar

Department of Electrical Engineering

Technion

Outline

Hardware

1. Motivation and main concepts
2. Our hardware proposal (PRINS)

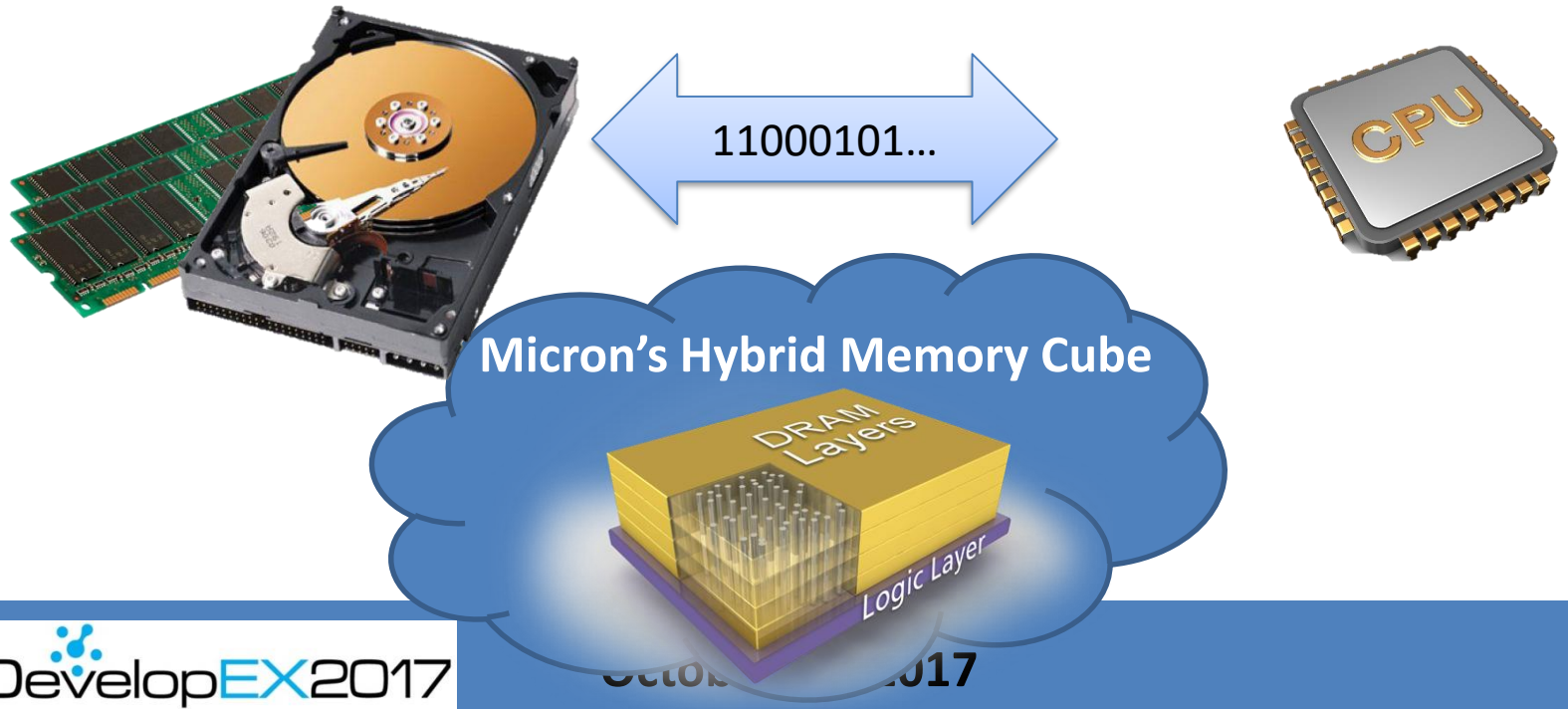
Machine Learning

3. K-Means
4. K-Nearest Neighbors (KNN)

5. Conclusions

Motivation: Overcome the von Neumann Bottleneck

- Traditional processing: Mem ↔ CPU / GPU
- Major bottleneck for big data workloads
- Even worse in a datacenter



Old Concept: Associative Processing

- Also called “Lookup table” computing
  - Invented in the 70s
- Shortly resurrected in 90s due to proliferation of digital video
- Since then – good as gone, until 3-4 years ago (with the rise of the near data processing)
- Rely on Content Addressable Memories (CAM)

Toy Example: Associative Vector Addition

$$A_0 + B_0 \rightarrow C_{out}, S$$

Memory map

A			B			C_{out}	S
0		3	4		7	8	11
0	1	0	1	1	0		
1	0	1	0	1	0		
1	1	0	1	0	1		
0	0	1	1	1	1		
1	1	0	0	1	0		
0	1	0	1	0	1		
1	0	1	1	1	0		
1	1	1	0	0	1		

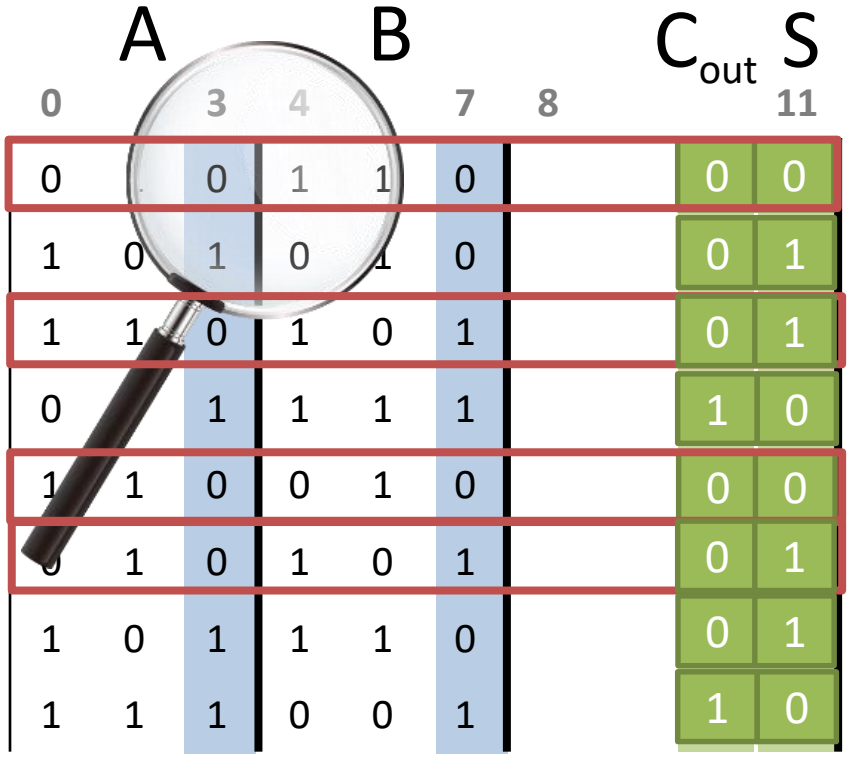
Half Adder Truth table

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Example: Associative Vector Addition

Compare → Write

Memory map



A			B			C_{out}	S
0		0	1	1	0	0	0
1	0	1	0	1	0	0	1
1	1	0	1	0	1	0	1
0		1	1	1	1	1	0
1	1	0	0	1	0	0	0
0	1	0	1	0	1	0	1
1	0	1	1	1	0	0	1
1	1	1	0	0	1	1	0

Half Adder Truth table

A	B	C_{out}	S	
0	0	0	0	✓
0	1	0	1	✓
1	0	0	1	✓
1	1	1	0	✓

Example: Associative Vector Addition

Memory map

A			B			C_{out}	S
0		3	4		7	8	11
0	1	0	1	1	0	1	0	0
1	0	1	0	1	0	0	0	1
1	1	0	1	0	1	0	0	1
0	0	1	1	1	1	1	0	0
1	1	0	0	1	0	1	0	0
0	1	0	1	0	1	0	0	1
1	0	1	1	1	0	0	0	1
1	1	1	0	0	1	1	0	0

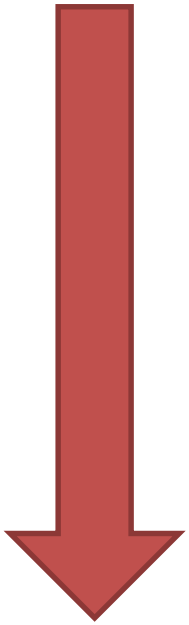
Full Adder Truth table

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

Associative Processing – Main Takeaway

- Processing in a fixed number of cycles, regardless of dataset size

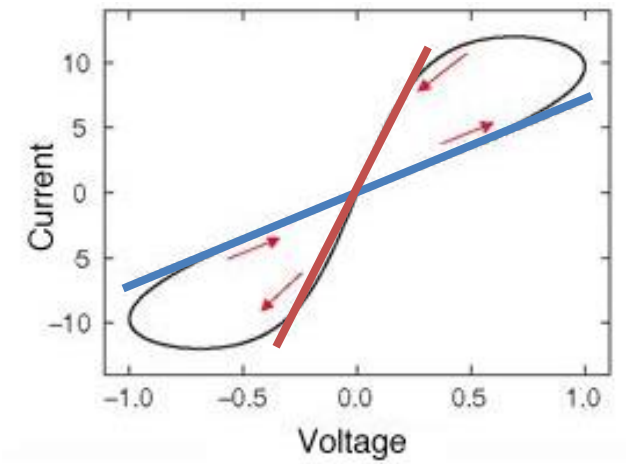
A			B			S			
0		3	4		7	8			11
0	1	0	1	1	0	0	1	0	0
1	0	1	0	1	0	0	0	1	1
1	1	0	1	0	1	1	0	1	1
0	0	1	1	1	1	1	0	0	0
1	1	0	0	1	0	1	1	0	0
0	1	0	1	0	1	0	0	1	1
1	0	1	1	1	0	1	0	1	1
1	1	1	0	0	1	1	1	0	1

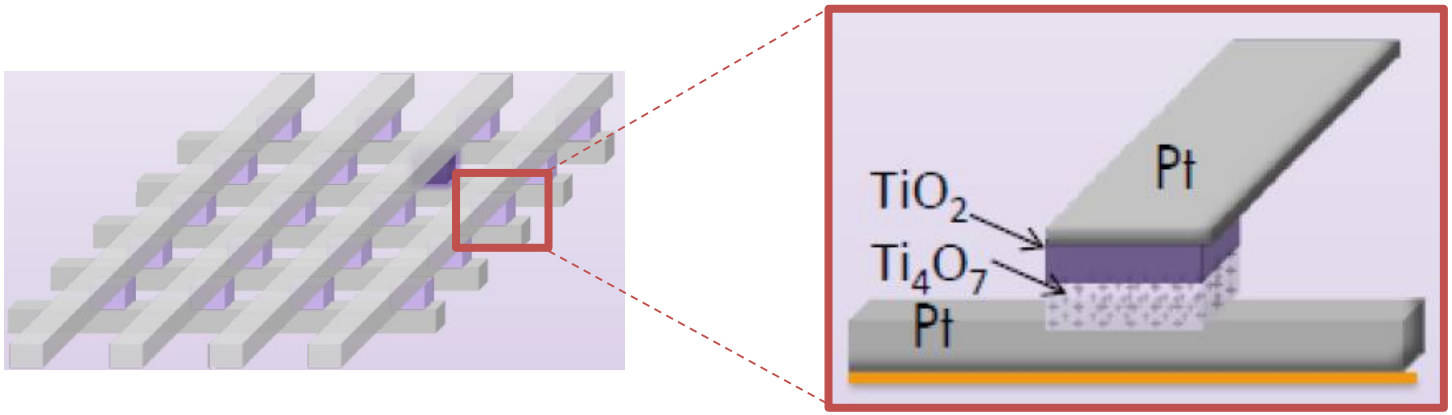


Vector length does not matter

So What's New? Memristors

- Devices that change resistance with applied voltage
- Can be placed in metal layers above silicon
- Small area footprint: $4F^2$
- Non volatile

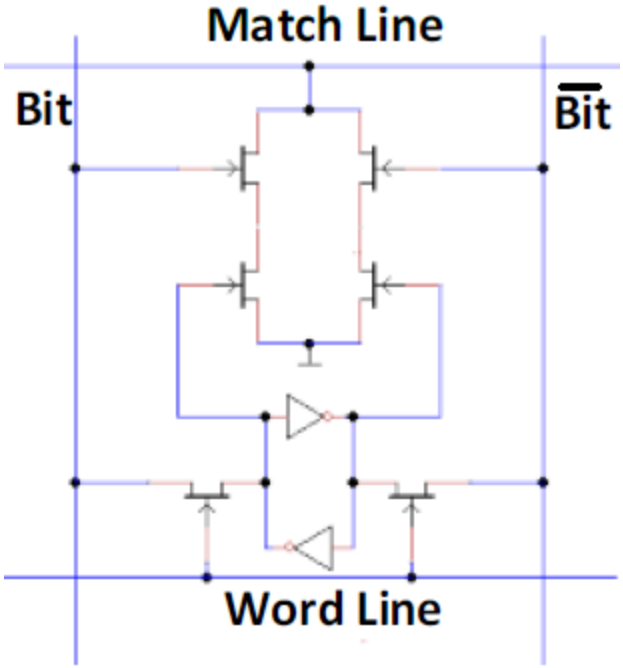




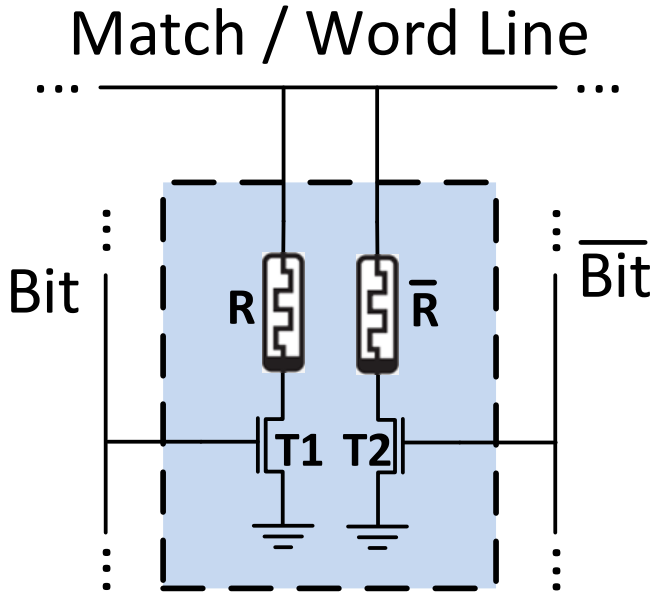
Resistive CAM Bitcell

CAM bitcell: 6 transistors ➔ 2 transistors + 2 memristors

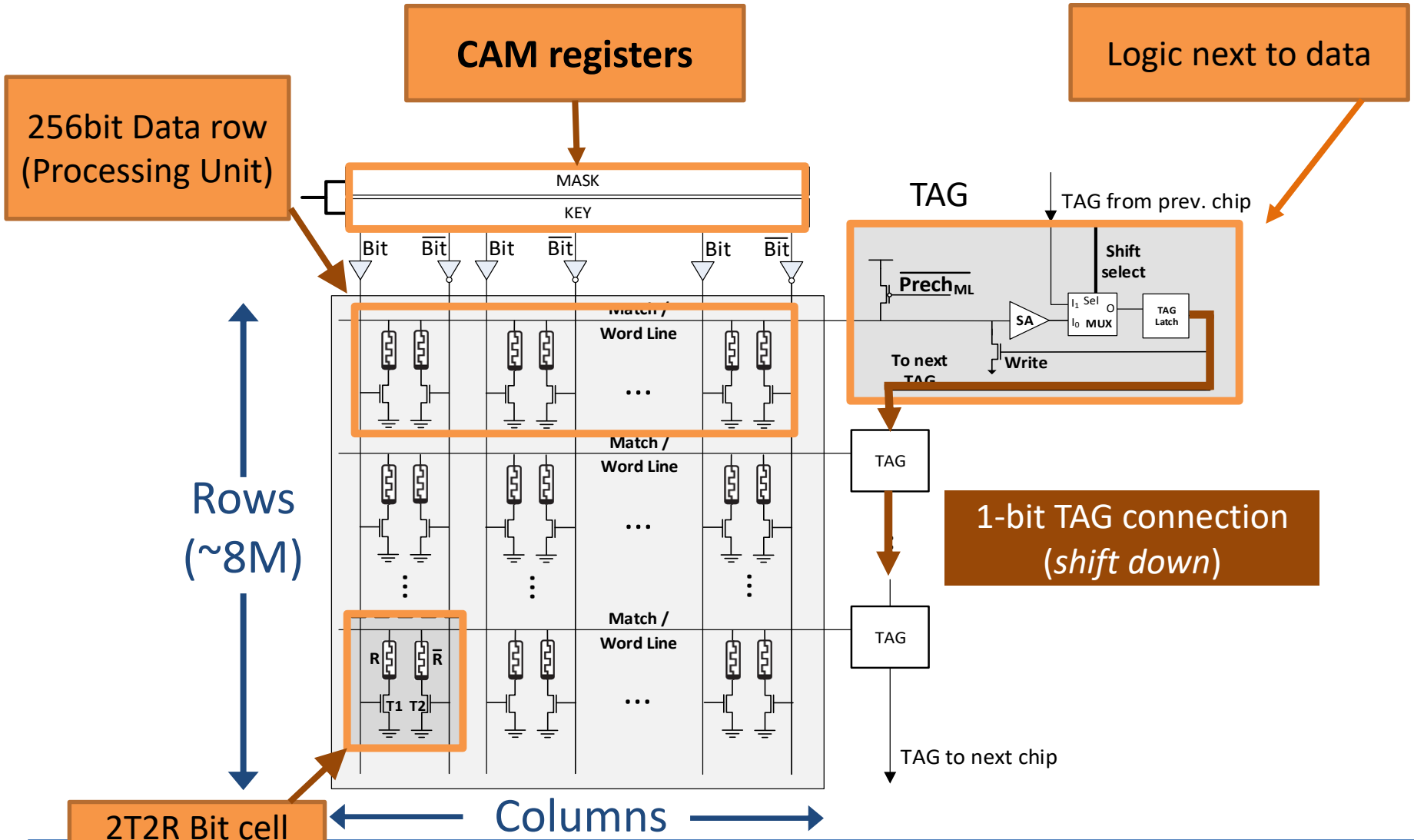
CMOS CAM bitcell (**6T**)



Resistive CAM bitcell (**2T2R**)

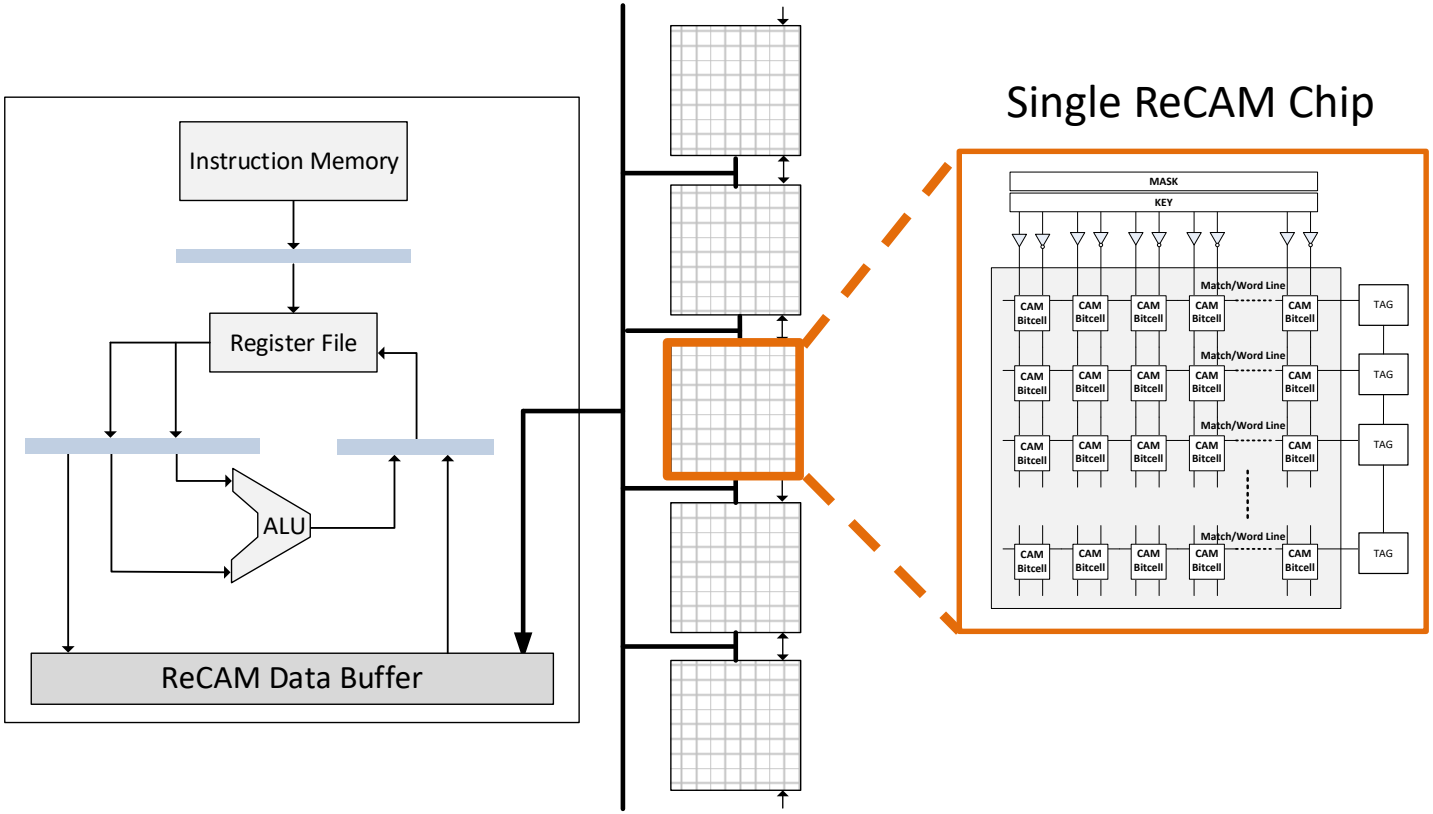


ReCAM Crossbar – Single Chip



PRINS System

- Conceptually, one long array
  - Divided into separate integrated circuits
- Associative processing = SIMD ➔ Linear scalability



ReCAM Operations

- Two instruction types:
  - In-row (e.g., A+B→C)
  - Multi-row reduction (e.g., Max among several rows)
- Multiplication is implemented as a combination of '+'
- Cycles per operation:

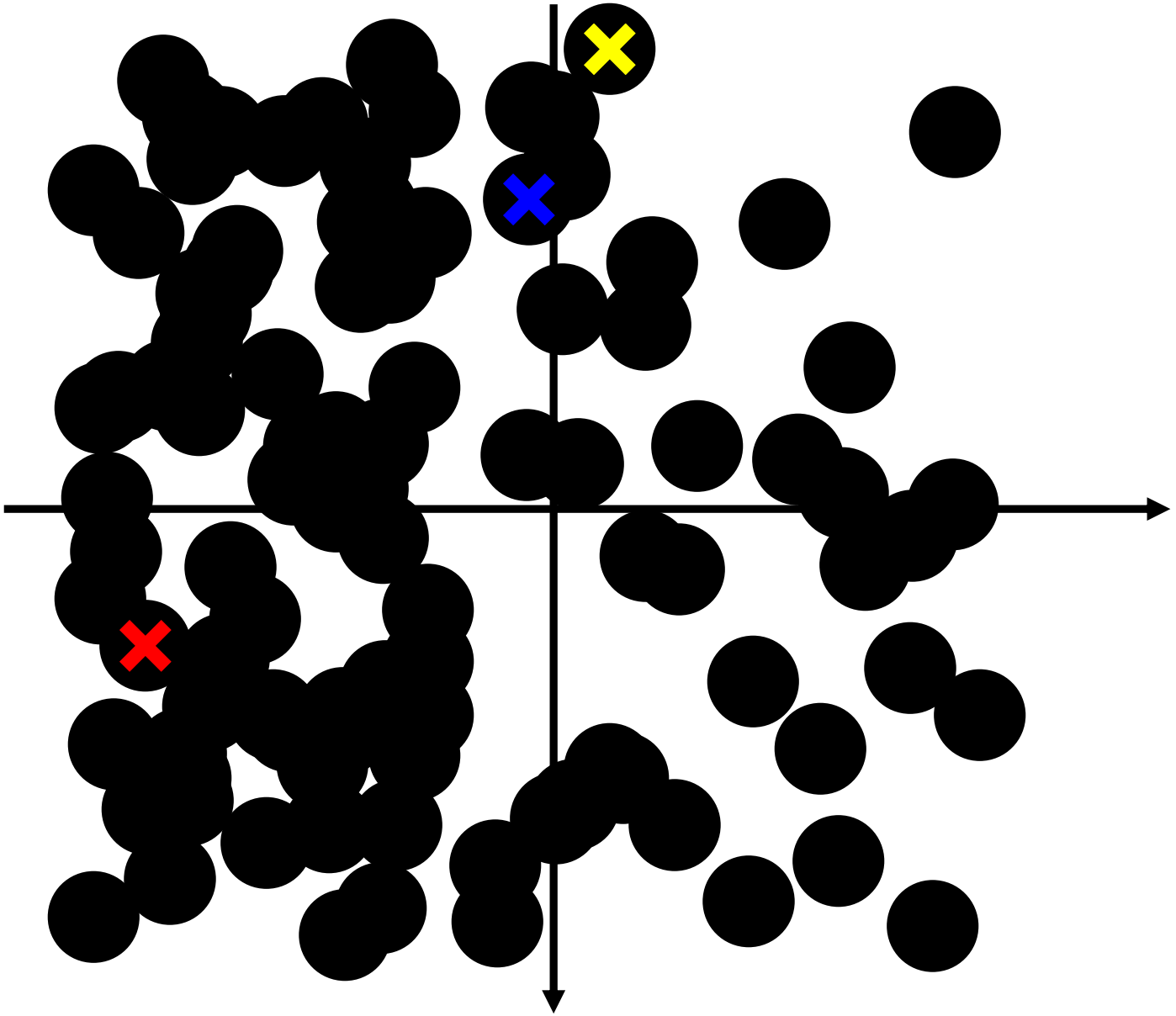
Instruction	Cycles
32 bit	
B ← A + B	256
C ← A + B	512
Row-wise Max (A, B)	64
Max Scalar (A)	128
Shift down by one row	128

Machine Learning on ReCAM:

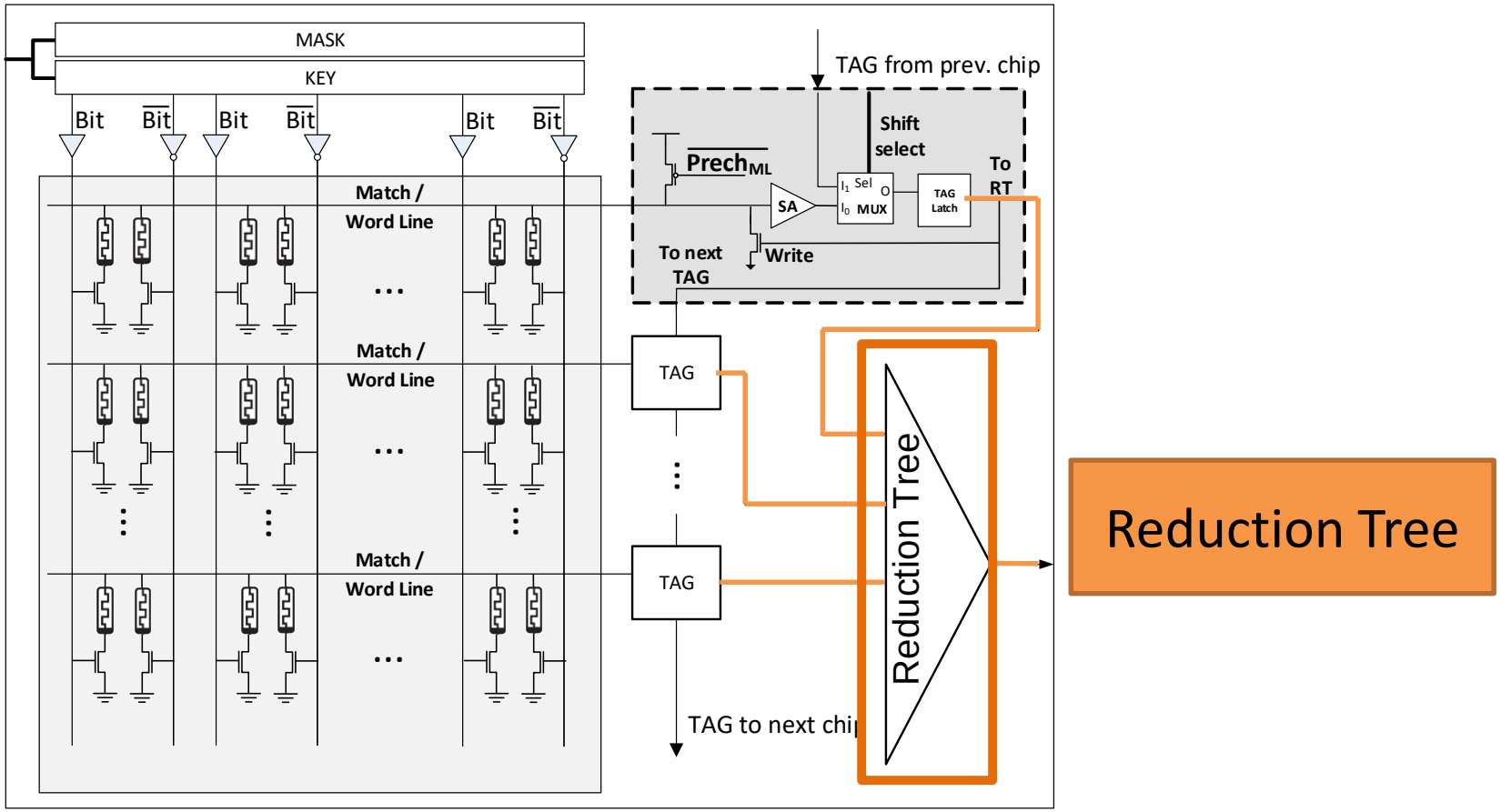
K-Means &

K-Nearest Neighbors (KNN)

K-Means Algorithm: Demo



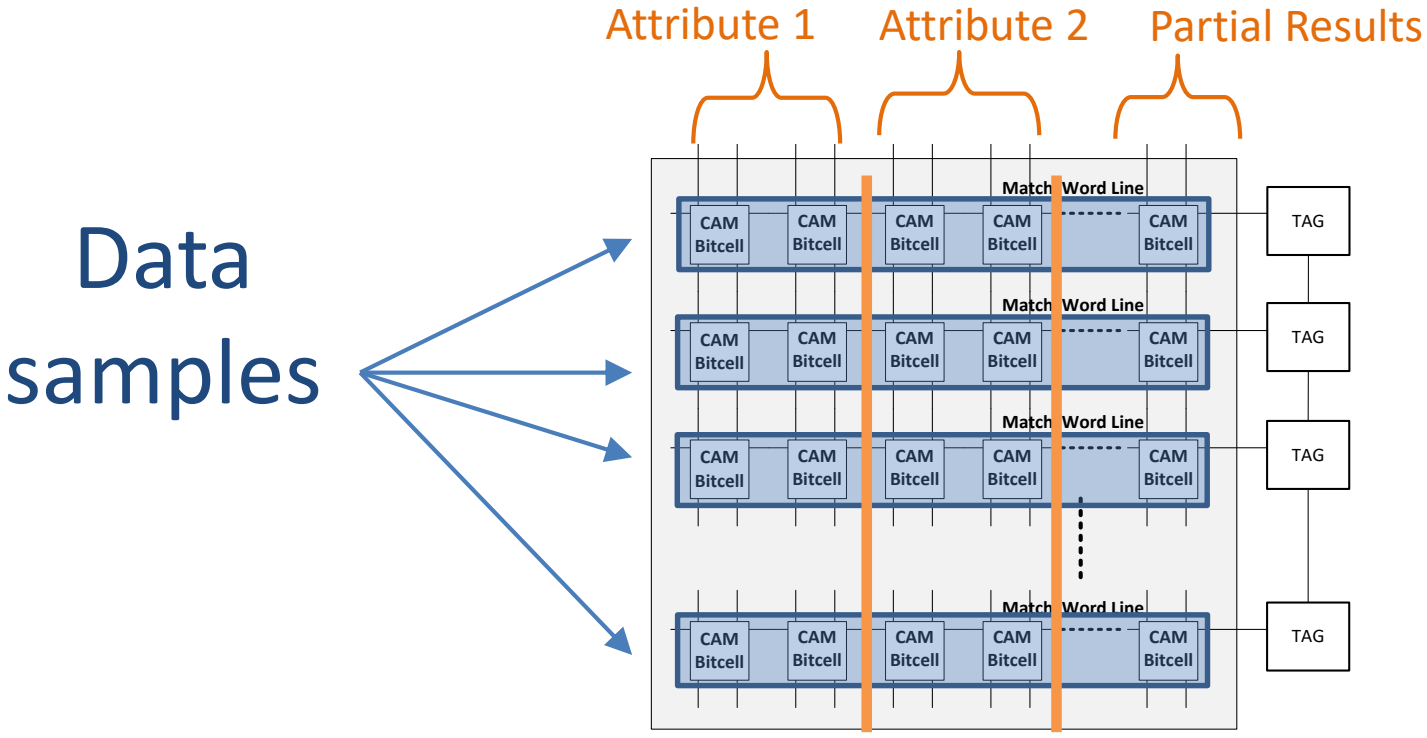
ReCAM Crossbar with a Reduction Tree



Mapping the Dataset to ReCAM

- Every sample has multiple attributes (dimensions)
- The goal: maximize parallelism

➔ Data sample per ReCAM row



K-Means Algorithm

Algorithm 2 *K*-Means Implementation in RCAM

// X: the group of samples
// Every $x \in X$ is stored in a separate RCAM row

//*Assignment:* assign each sample with a cluster
// Each of the k means is a tuple: (i_{mean}, $Mean$)

1: For each $i_{mean} \in [1, K]$:
   Do-all $x \in X$:
      Write *mean* coordinates to *temp* columns
      For each $attr \in \{sample\ attributes\}$:
4:       $dist_{attr} \leftarrow x_{attr} - mean_{attr}$
5:       $sqDist_{attr} \leftarrow (dist_{attr})^2$
6:       $sqDist_{to\_mean} \leftarrow sqDist_{to\_mean} + sqDist_{attr}$
7:    Tag rows with $sqDist_{to\_mean} < min\_sqDist$
8:    Write $min\_sqDist \leftarrow sqDist_{to\_mean}$
9:    Write $i_{assigned\_mean} \leftarrow i_{mean}$

Assignment Loop

...: calculate new mean coordinates
10: ...ch $i_{mean} \in [1, K]$:
11: ...with $i_{assigned\_mean} == i_{mean}$
12: For ...$ttr \in \{sample\ attributes\}$:
13: $Sum_{attr} \leftarrow Reduction(x_{attr})$
14: $Cluster\_size \leftarrow Reduction(match\_lines)$
15: $mean_{attr} \leftarrow Sum_{attr}/Cluster\_size$

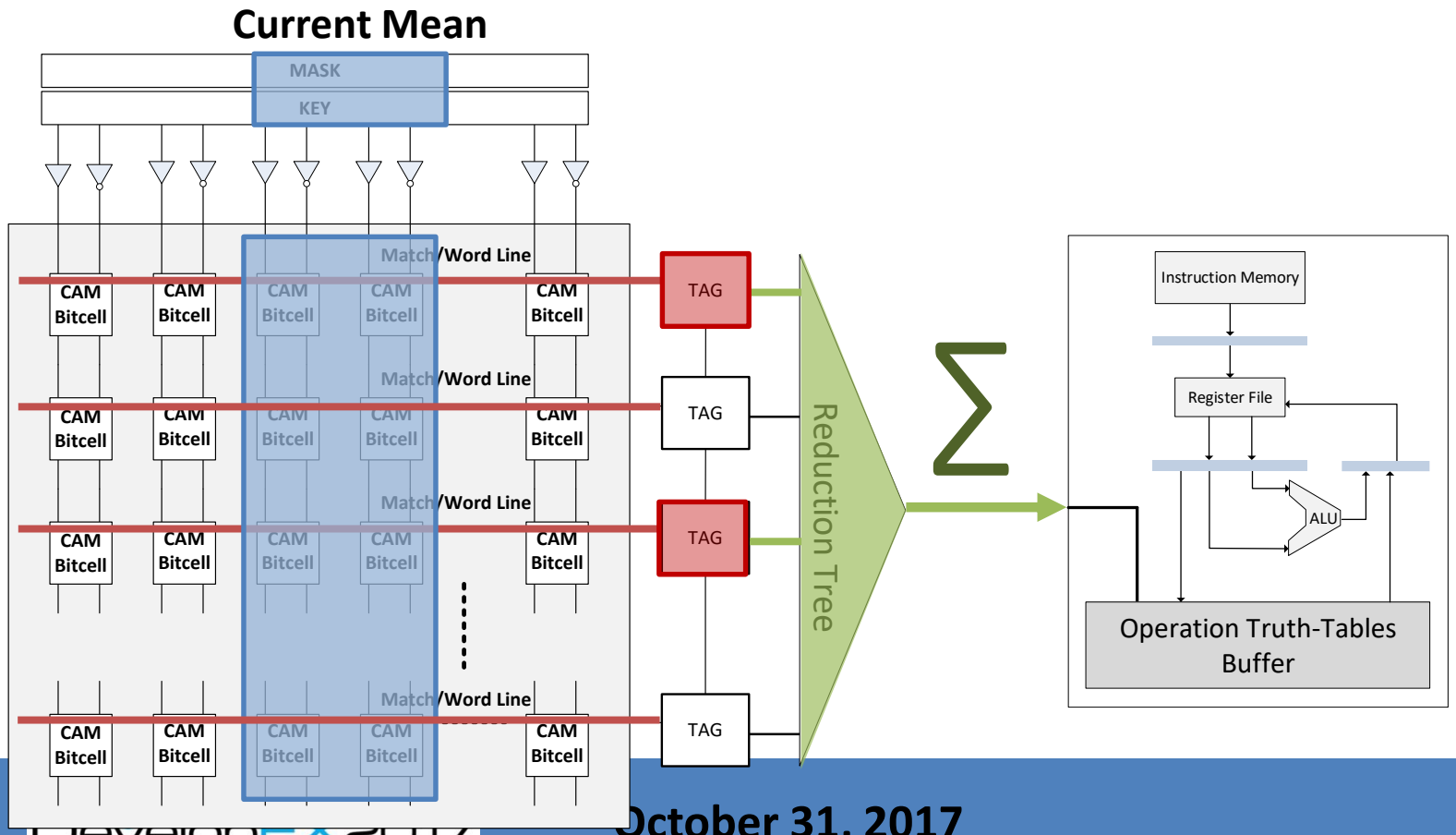
Update Loop

Parallel sections

K-Means Algorithm – Update Loop

For each Mean:

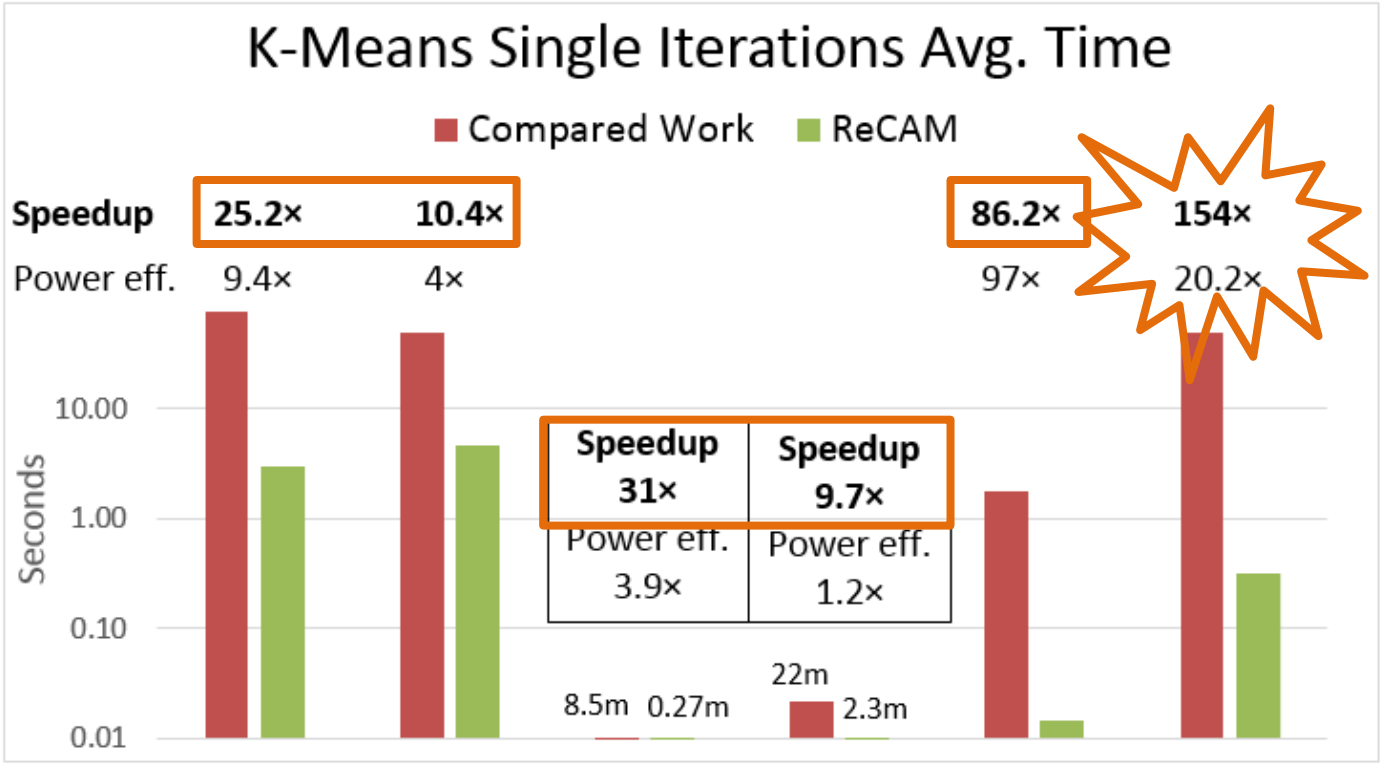
1. Tag samples ∈ Mean:
2. Reduce coordinates



Simulation and Comparison

- Cycle-accurate simulator
  - Frequency: 500Mhz
  - Memrisotr read/write energy = 1fJ/100fJ
  - Power: 200W for full chip
- Compared with different implementations: CPU, FPGA, GPU, 10-GPU Cluster
- Various datasets: 10-100MB up to 150GB
  - 10s-100s of attributes per data sample

Performance Comparisons

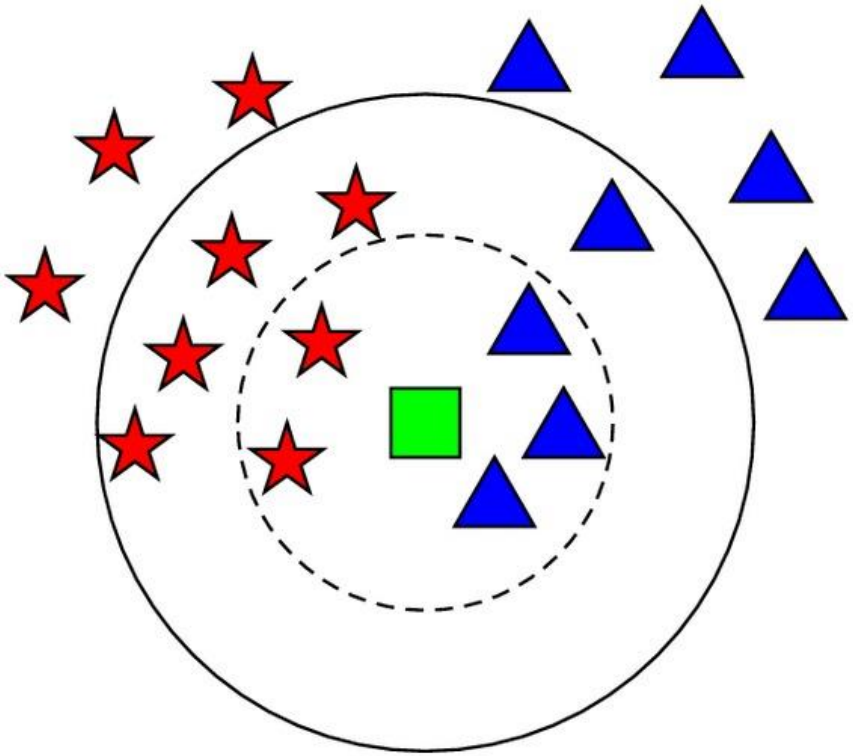


Platform	Intel Xeon CPU	Xilinx ZC706	Altera Stratix V FPGA	K20M GPU	10-K20M GPUs
Dataset Samples	1M	1M	2M	1.4M	1B
Reference	[1]	[2]	[3]	[4]	[5]

[1] Ding, Yufei, *et al*. "*Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup*." ICML 2015.
[2] Li, Zhehao, *et al*. "*High-performance K-means Implementation based on a Coarse-grained Map-Reduce Architecture*." arXiv:1610.05601 (2016).
[3] Ramanathan, Nadesh, *et al*. "*A Case for Work-stealing on FPGAs with OpenCL Atomics*." SIGDA 2016.
[4] Bhimani, Janki, *et al*. "*Accelerating K-Means clustering with parallel implementations and GPU computing*." HPEC 2015.
[5] Rossbach, Christopher J. *et al.* "Dandelion: a compiler and runtime for heterogeneous systems.", SOSP 2013.

K-Nearest Neighbors

Classification and regression algorithm



K Nearest Neighbors

Algorithm 2 KNN Implementation in RCAM

//K denotes the number of nearest neighbors.
//Every sample $x \in X$ may be stored in several consecutive RCAM rows; the code assumes one row per sample for simplicity.
//Each sample is characterized by M attributes
//Calculate distance of each dataset sample from query
For each $attr \in [1, M]$
  Do-all $x \in X$:
    $dist_{attr} \leftarrow Query_{attr} - x_{attr}$
    $sqDist_{attr} \leftarrow (dist_{attr})^2$
    $sqDist \leftarrow sqDist_{partial} + sqDist_{attr}$

//Find K closest samples
//Histogram of all classes maintained by microcontroller
//Start with all samples unmarked
Loop K times
  Tag all unmarked samples
  Tag and mark first row with min value of $sqDist$
  Retrieve $class$ of tagged row to microcontroller
  On microcontroller: Histogram[$class$]++
//Classification: Class with highest histogram

Parallel sections

Calculate distance

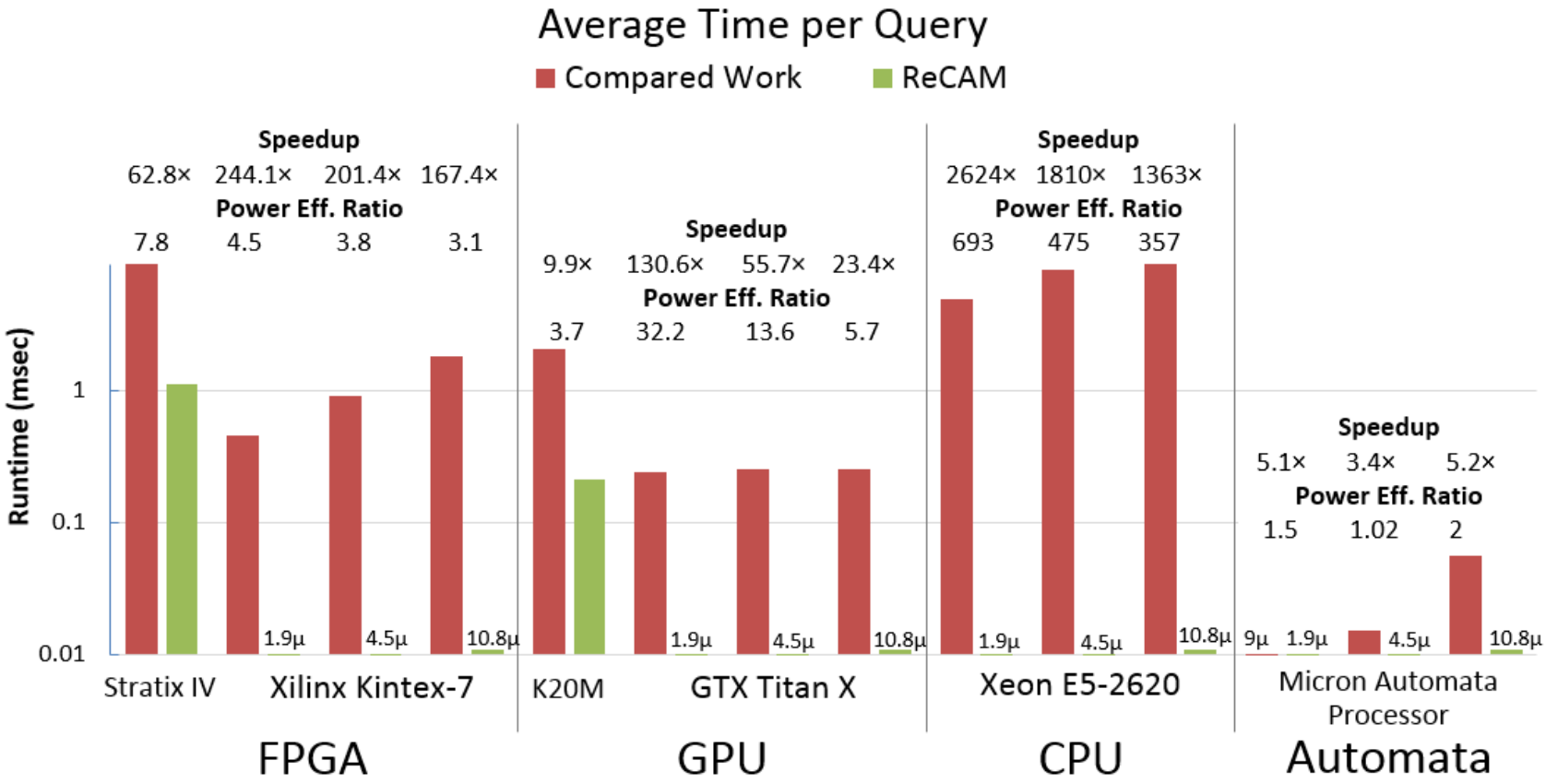
Find K-nearest

Simulation and Comparison

- Cycle-accurate simulator
  - Frequency: 500Mhz
  - Memrisotr read/write energy = 1fJ/100fJ
  - Power: 200W for full chip
- Compared with different implementations: CPU, FPGA, GPU, Automata
- Various datasets: 10 - 800MB
  - 10s-100s of attributes per data sample

Performance Results

3 - 2000× speedup



[1] Pu, Y., et al. "An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl". FCCM, pp. 167-170, 2015.
[2] Gutiérrez, P. D., et al. "GPU-SME-kNN: Scalable and memory efficient kNN and lazy learning using GPUs." Information Sciences, 373, 165-182, 2016.
[3] Lee, V. T., et al. "Similarity Search on Automata Processors." In IPDPS, 2017.

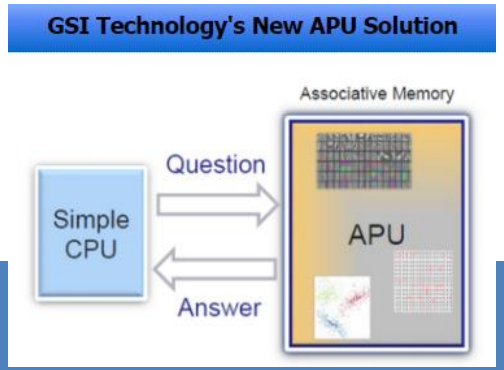
Conclusions

- Resurrected approach + new materials

  ➔ Architecture for next-generation big data HPC

- Might be useful for various problems
  - We also simulated bioinformatics alg.

- Industry interest: GSI Technology is developing an Associative Processing Unit (APU)



Thank you

Questions?