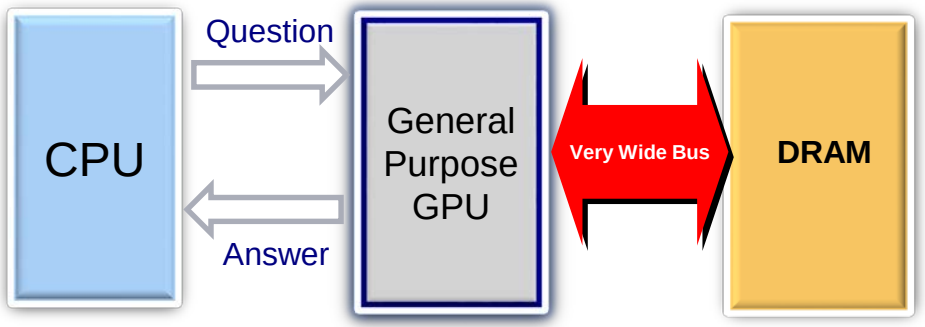
In-Place Deep Learning

Dr. Avidan Akerib
VP Associative Computing BU

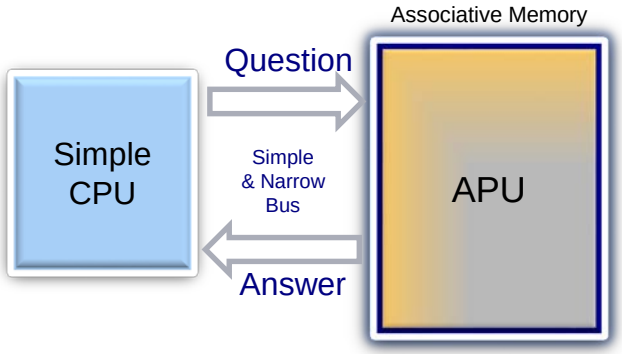
CPU-GPGPU Solution



- Creates I/O bottleneck between the processor and memory
  - Limits performance
  - Increases power consumption
- Does not scale with the search, sort, and rank requirements of applications like recommender systems and data mining

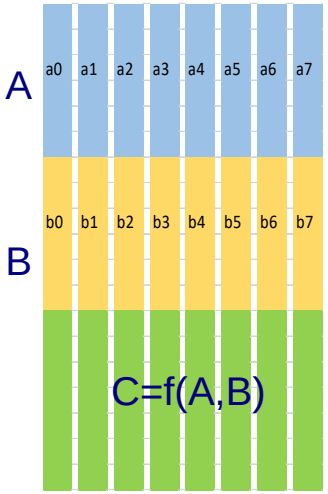
Brain Computes in Place

GSI's Solution
APU—Associative Processing Unit



- Computes in-place directly in the memory array—removes the I/O bottleneck
- Significantly increases performance
- Reduces power

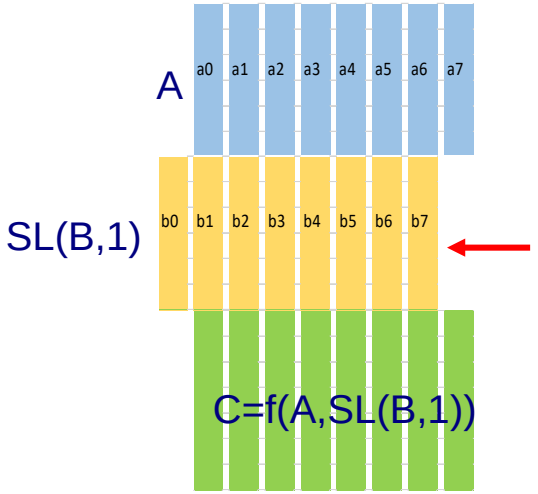
Computing in the Bit Lines



Each bit line becomes a processor and storage

Millions of bit lines = millions of processors

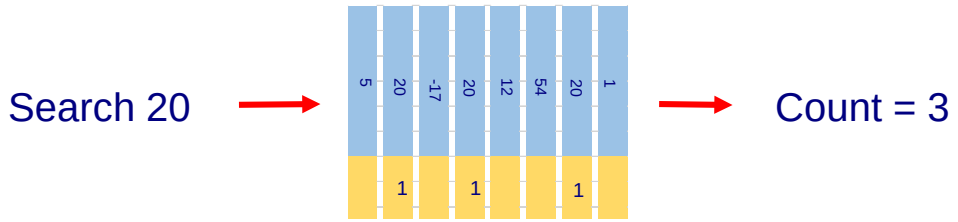
Neighborhood Computing



Parallel shift of bit lines @ 1 cycle sections

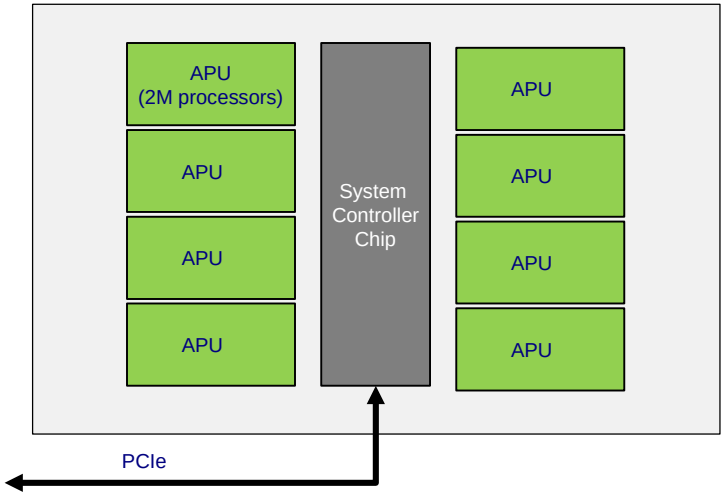
Enables neighborhood operations such as convolutions

Search & Count



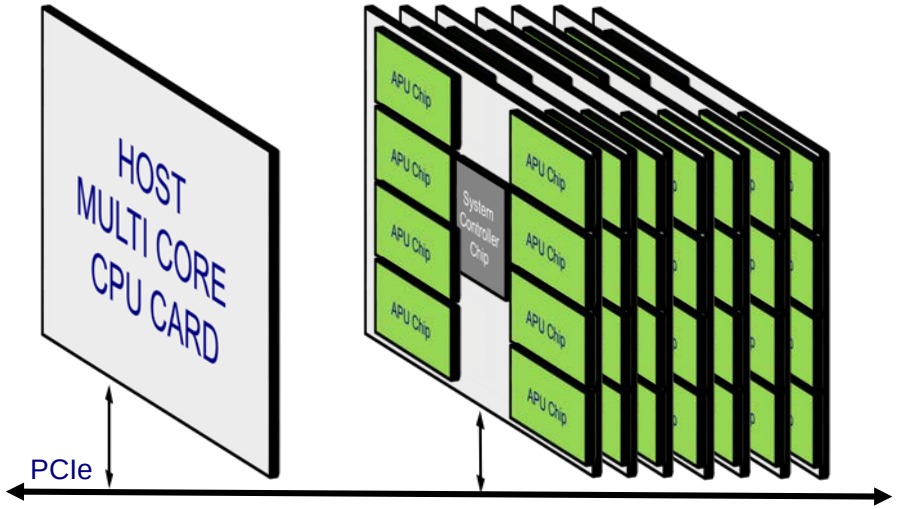
- Search (binary or ternary) all bit lines in 1 cycle
  - 128 M bit lines => 128 Peta search/sec
- Key Applications for search and count:
  - Recommender systems
  - K-Nearest Neighbors (using cosine similarity search)
  - Random forest
  - Image histogram
  - Regular expression

APU Card



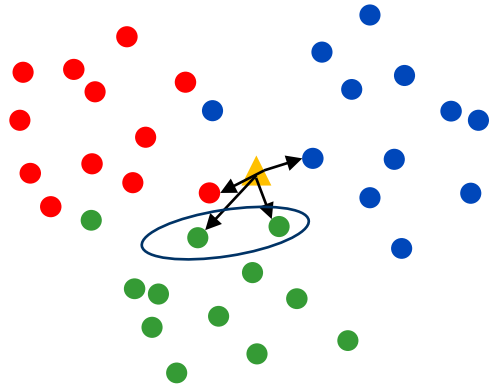
- PCI card size holds up to 8 APUs
- 125 Tera OPS @ 25 Watts (average of search, add, sub, FP mul, etc)
- 5 Tera OPS/Watt

APU Server



8 APU Cards : 1 Peta OPS/sec @ 200 W

K-Nearest Neighbors (k-NN)



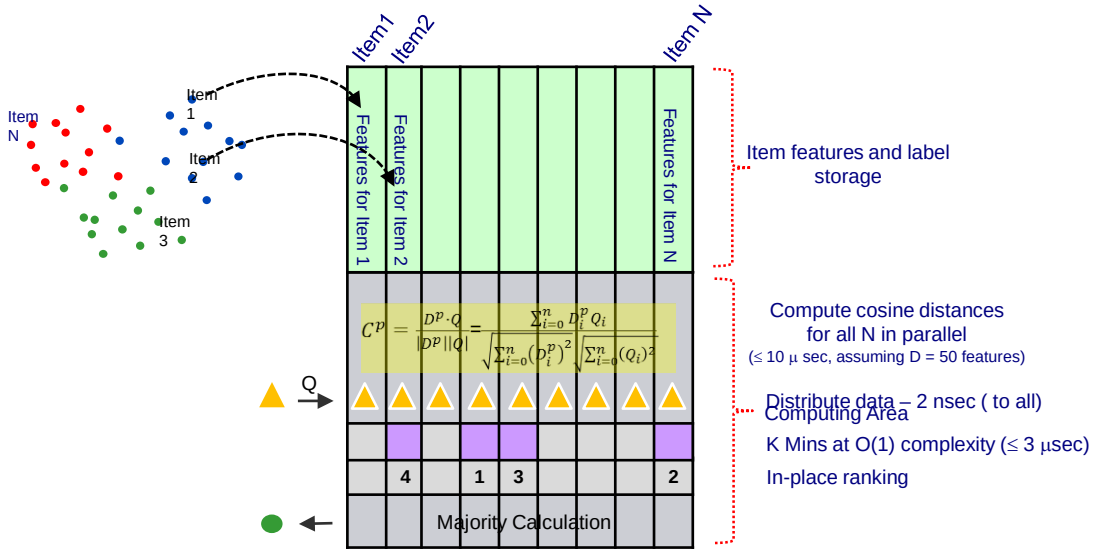
Simple example: N = 36, 3 Groups, 2 dimensions (D = 2) for X and Y

K = 4

Group **Green** selected as the majority

For actual applications: N = Billions, D = Tens, K = Tens of thousands

k-NN use case in an APU



With the data base in an APU, computation for all N items done in

≤ 0.05 ms, independent of K

Large Database Example Using APU Servers

- Number of items: Billions
- Features per item: tens to hundreds
- Latency: ≤ 1 msec
- Throughput: Scales to 1M similarity searches/sec
- k-NN: Top 100,000 nearest neighbors

Example k-NN for Recognition

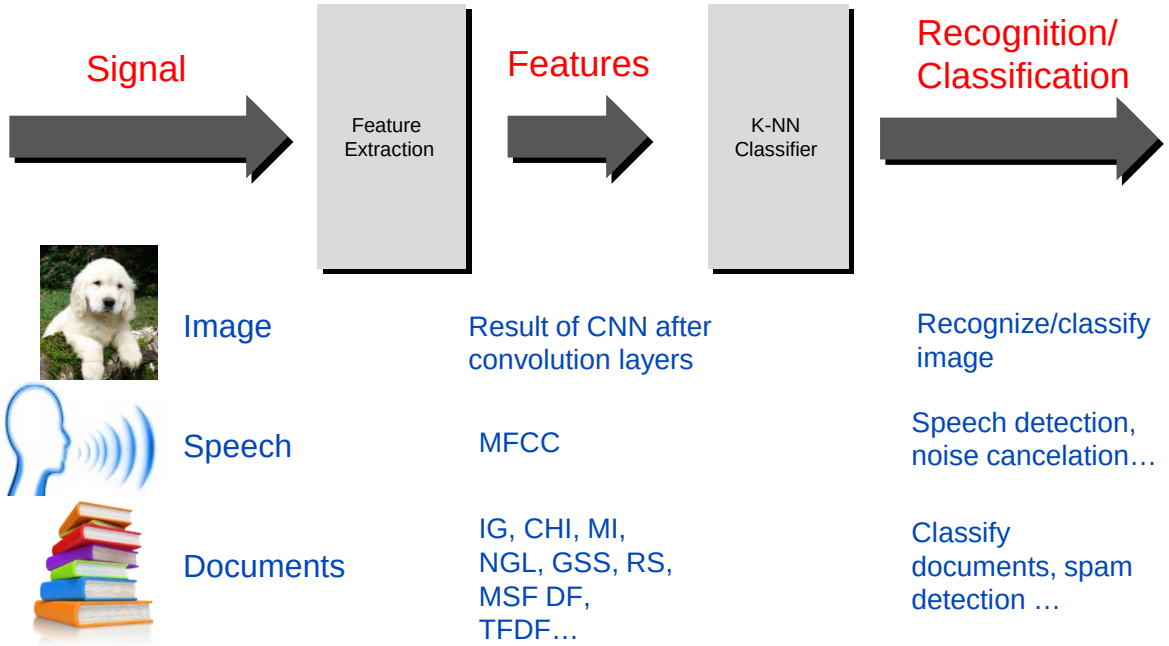
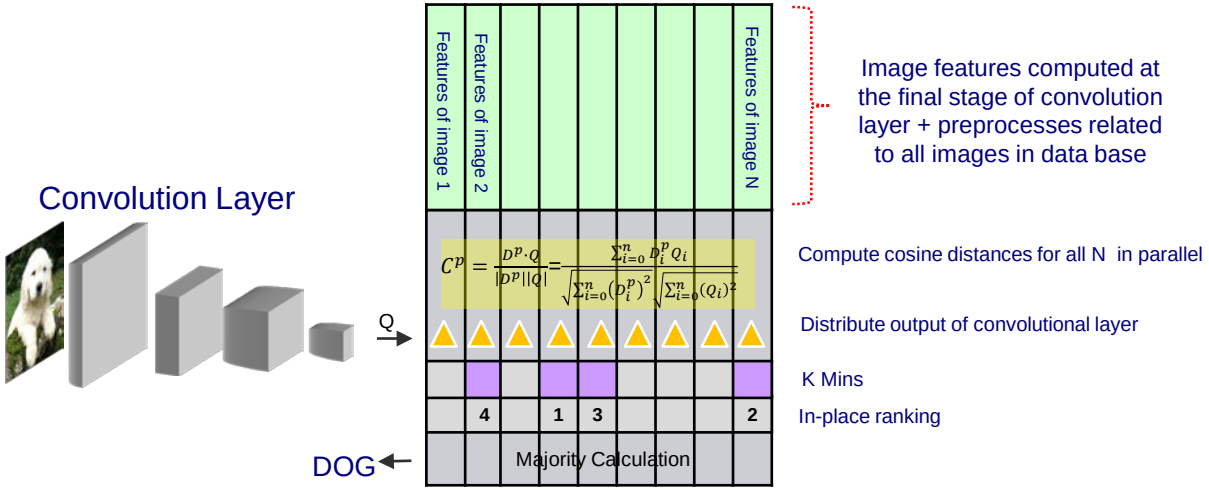


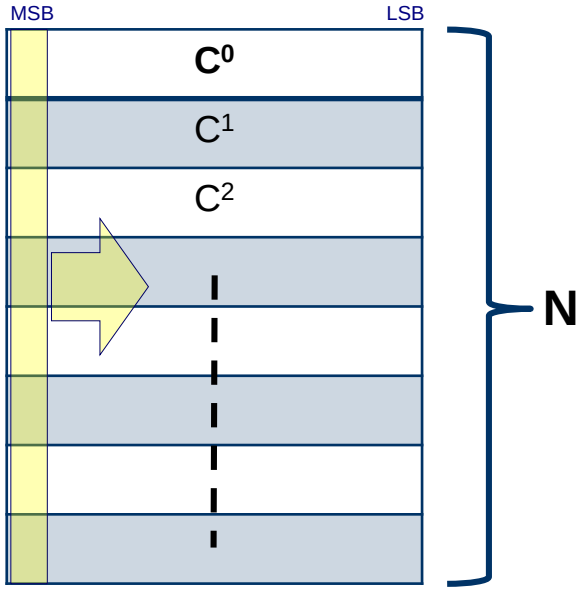
Image Recognition using k-NN



Computation with k-NN replacing fully connected layer ≤ 0.05 ms
compared to 2-3 ms with today's FC solution

K-MINS: O(1) Algorithm

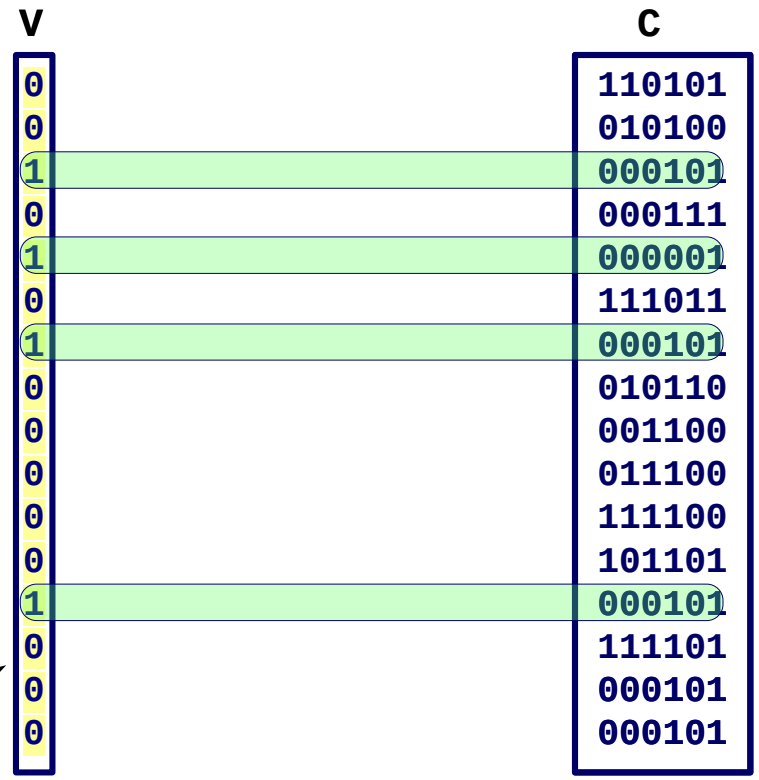
```
KMINS(int K, vector C){
  M := 1, V := 0;
  FOR b = msb to b = lsb:
    D := not(C[b]);
    N := M & D;
    cnt = COUNT(N|V)
    IF cnt > K:
      M := N;
    ELIF cnt < K:
      V := N|V;
    ELSE: // cnt == K
      V := N|V;
      EXIT;
    ENDIF
  ENDFOR
}
```



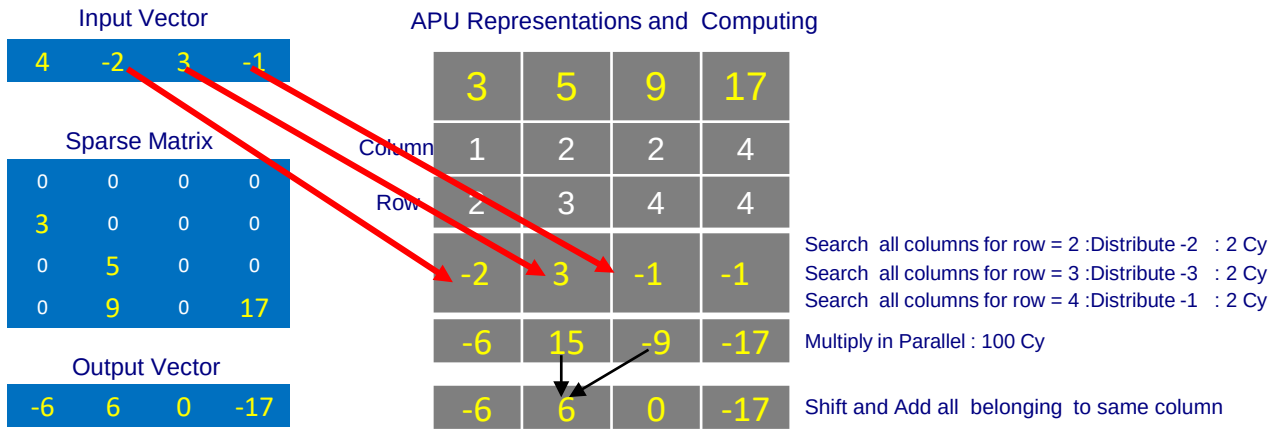
K-MINS: The Algorithm

```
KMINS(int K, vector C){
  M := 1, V := 0;
  FOR b = msb to b = lsb:
    D := not(C[b]);
    N := M & D;
    cnt = COUNT(N|V)
    IF cnt > K:
      M := N;
    ELIF cnt < K:
      V := N|V;
    ELSE: // cnt == K
      V := N|V;
      EXIT;
    ENDIF
  ENDFOR
}
```

final output



Matrix Multiplication
Dense (1XN) Vector by Sparse NxM



Complexity including IO : O (N +logβ)
where β is the number of nonzero elements in the sparse matrix
N << M in general for recommender systems

Sparse Matrix Multiplication Performance Analysis

- G3 circuit matrix
  - 1.5M X 1.5M sparse matrix
  - Roughly 8M nonzero elements
- 20–100 GFLOPS with GPGPU solution
- APU solution provides 64 TFLOPS using the same amount of power as the GPGPU solution above
- > 500x improvement with APU solution

Summary

- Access data by content not by address
  - In-place computation of randomly distributed data using millions of parallel processors
- >500x performance-over-power improvement for recommender systems and data mining
- >500x performance improvement for sparse matrix multiplication compared to GPGPU
- Scales to 1M queries of cosine similarity searches/second for billions of records
- 5 Terra OPS/Watt