MERCURY: Accelerating DNN Training By Exploiting Input Similarity

Vahid Janfaza, Kevin Weston, Moein Razavi, Shantanu Mandal, Farabi Mahmud, Alex Hilty, Abdullah Muzahid

2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

Review by Dong Hyun Kim, Yeungnam University, Department of Computer Science and Engineering

Overview

- DNN are computationally intensive to train, but it has a lot of similarities among input vectors.
- Thus, the computation of one input vector can be skipped by reusing the already-computed results.
- MERCURY exploits input similarity during DNN training in a hardware acceleartor using RPQ.

Main Idea

- MERCURY uses RPQ to detect similarity among input vectors.
- Input vector is converted into a bit-sequence, called Signature. If two input vector has same signature, it could be skipped.
- Those signatures are stored in special cache, called MCACHE.
- MERCURY also keeps the dataflow and computations regular for the current hardware accelerator.

Background

- Input similarity
  - How to know that the two vectors have similarity?
- Random Projection Quantization
  - What is RPQ?
- Dataflow for DNN accelerator
  - How does accelerator process DNN workloads?

Input Similarity

$$v_2 \cdot w = v_1 \cdot w + \varepsilon \cdot w$$

$$\longrightarrow \quad \varepsilon_i \approx 0$$

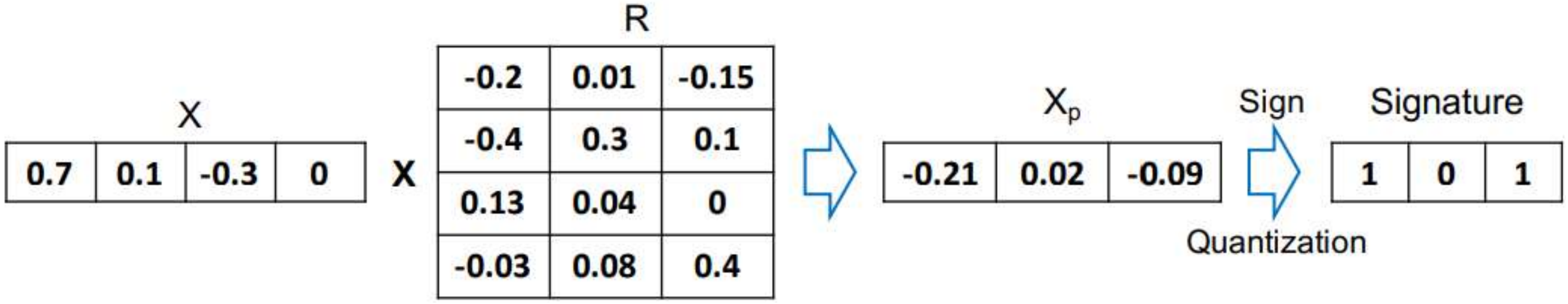
$$\longrightarrow \quad \varepsilon_i \cdot w \approx 0$$

$$\longrightarrow \quad v_2 \cdot w \approx v_1 \cdot w$$

- weight vector w and two input vectors v1, v2.
  - $v_1 = [v_{1,1}, v_{1,2}, v_{1,3}]$
  - $v_2 = [v_{1,1}+\varepsilon_1, v_{1,2}+\varepsilon_2, v_{1,3}+\varepsilon_3]$

- If ε_i represents insignificant difference, v_1 and v_2 have similarity and the computation of v_2 can be skipped.

Random Projection Quantization



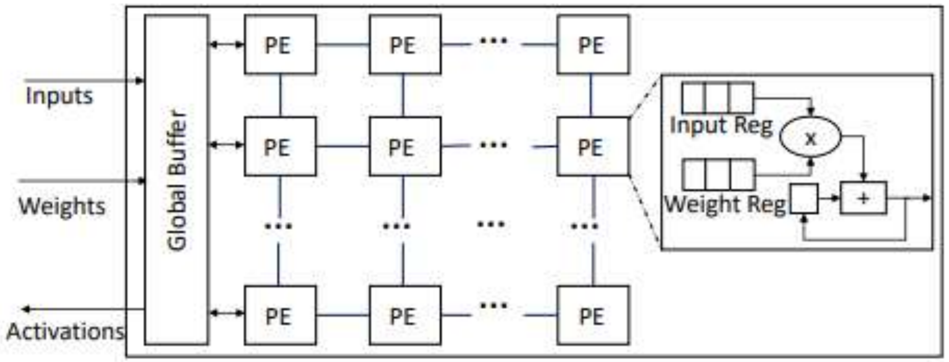
- X (1 x m) x R (m x n) = X_p (1 x n)
  - X: input vector
  - R: randomly populated from normal distribution
  - Xp: projected vector

- RPQ is dimension reduction technique.

- It converts one vector to another with lower dimension and also quantized them further.

- Finally, a bit-sequence called signature is made by RPQ, and if two vectors have same signature, they are treated as similar.

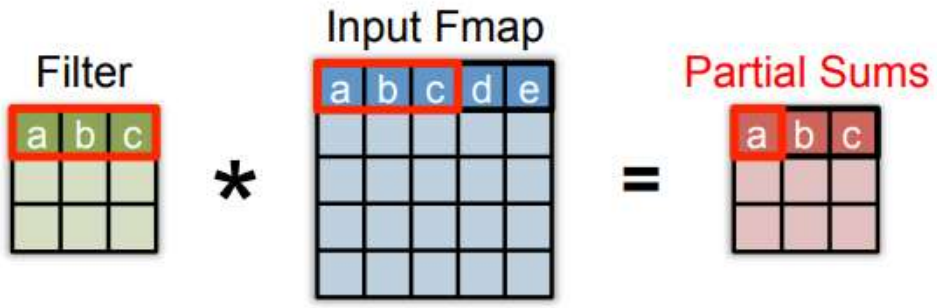
Overview of DNN Accelerator

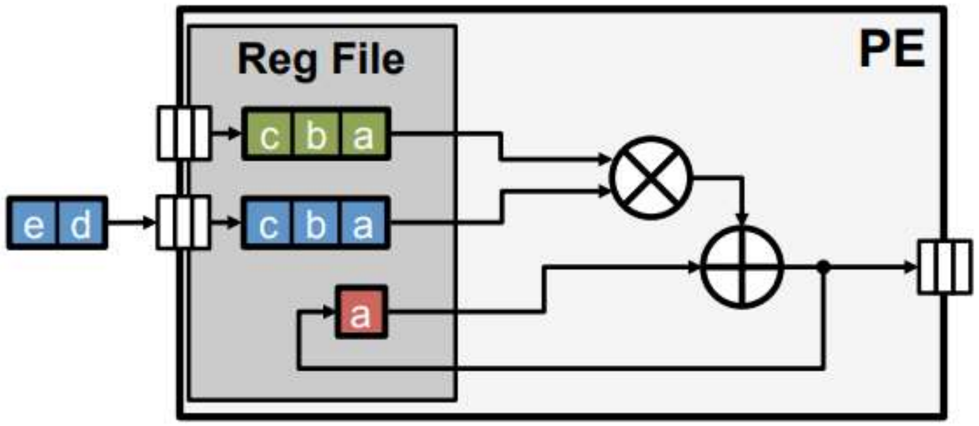


- Typical DNN accelerator has a number of hardware PEs
- They are connected vertically and horizontally using on-chip networks.
- There is a global buffer to hold inputs, weights, and partial sums.
- The chip is connected to off-chip memory to receive inputs and store outputs.

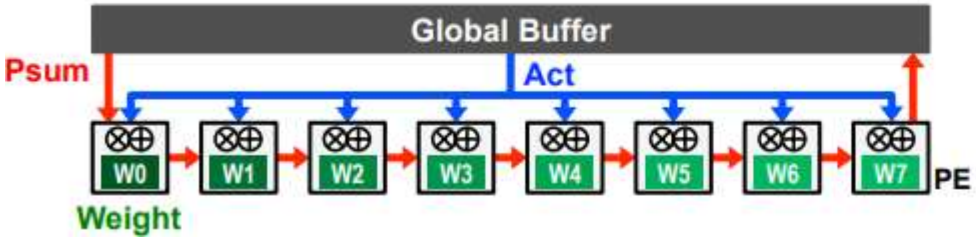
Processing Element

- Each PE has registers to hold inputs, weights and partial sums.
- Each PE also has multiplier and adder units.

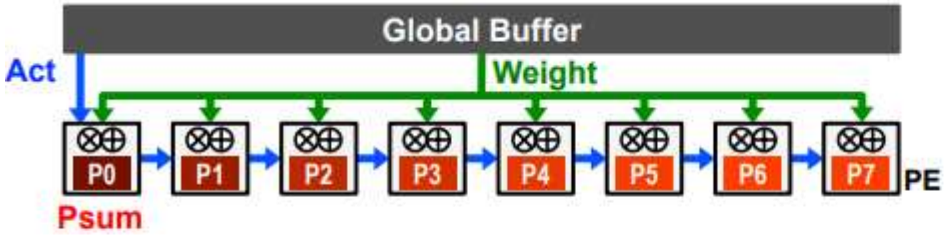




Dataflow of Accelerator



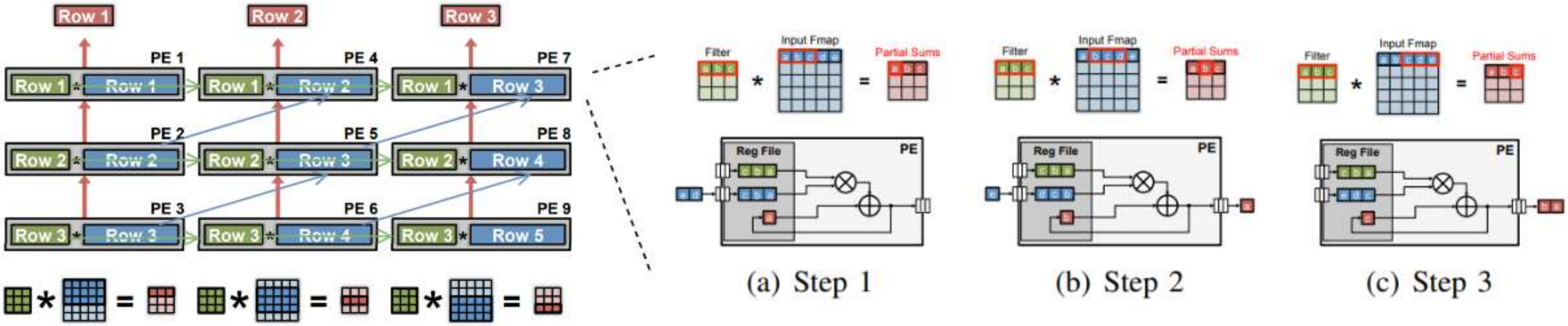
Weight Stationary



Output Stationary

- Each PE distributes inputs and weights and generates partial sums based on a dataflow.
- There are dataflows based on which data is kept unchanged in the PE unit.
  - Weight stationary: each PE holds weight inside its register file.
  - Output stationary: each PE holds partial result accumulation.
  - Row stationary: each PE processes one row of the input.

Row Stationary



- How datas are flow in Row-Stationary?
  - Weights stream horizontally.
  - Input rows stream diagonally.
  - Partial sums are accumulated vertically.

- Row stationary is considered one of the most efficient dataflows for reuse.

- Thus, this MERCURY choose row-stationary as a baseline.

Prior Works for Data Reuse

- UCNN
  - Exploits weight repetitions to reuse CNN sub-computations.

- DeepReuse and Adaptive Deep Reuse
  - Both uses Locality Sensitive Hashing to find similarity among input vectors.
  - But LSH requires computationally expensive pre-processing.
  - LSH also interferes original dataflows.

- Diffy
  - Uses element-wise comparison to detect similarity.
  - This is inefficient since comparison need to be done serially.

MERCURY vs Prior Works

- Unlike prior approaches, MERCURY uses a well-established hashing technique, RPQ.
- With RPQ, MERCURY checks similarity between the vectors.
- RPQ does not require new dataflow. Thus it can be easily used in both forward and backward pass.

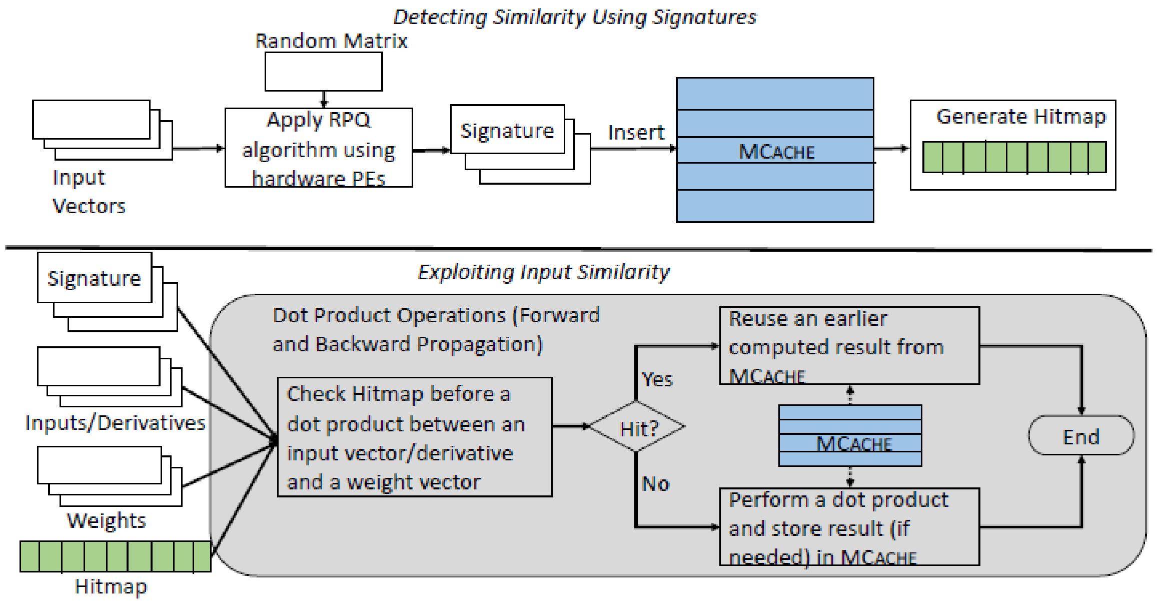
Overview of MERCURY

1. Signature generation & update of structures

2. Exploit input similarity with signature

3. Adaptation for accuracy & efficiency

4. Other dataflows



Detecting Similarity Using Signature

- Signature as a convolution operation
- Dataflow of signature calculation
- Signature management

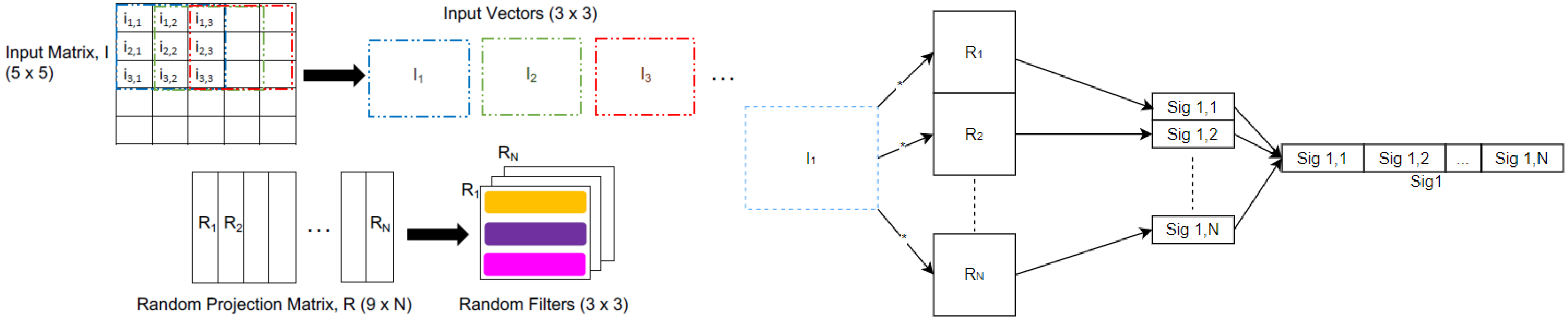
Signature as a Convolution Operation

- Assume that
  - 5x5 input
  - 3x3 kernel
  - 3x3 output

- Matrix R for RPQ
  - Each input vector has 9 elements, so R is 9xN.
  - We have R_1, R_2 .... R_N.
  - Finally, re-organize each column vector of R to shape 3x3.

- Signature can be formulated as 2D convolutions.

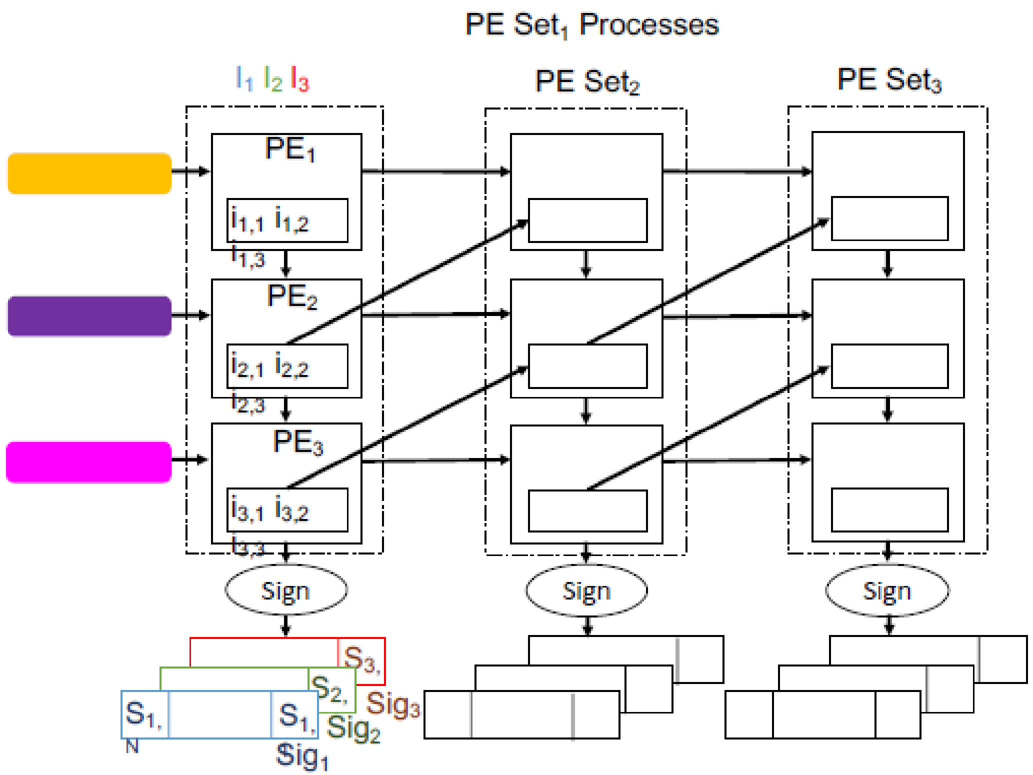
Signature as a Convolution Operation



- For example, input vector I_1 is converted into a signature Sig1, consisting of N-bits.
- It means Sig1,1 is generated with R_1, Sig1,2 is with R_2, and so on. Other signatures (Sig2 to SigN) can be generated as same way.
- Those processes can be easily mapped to row-stationary accelerator since it is same as convolution operation.

Signature Calcuation Process

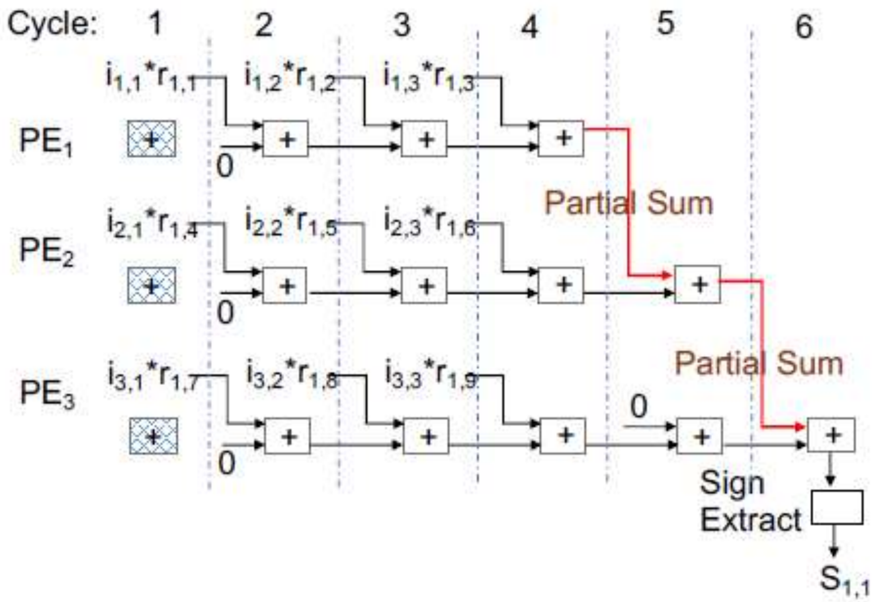
- Filter rows stream horizontally while input rows stream diagonally.
- PE_1 to PE_3 perform three 2D convolutions in a streaming fashion.
- Starting with R_1, PE Set_1 calculates S1,1, S2,1 and S3,1.
- After that, S1,2, S2,2 and S3,2 (second bit) is calculated.
- Finally, N bits of all signatures are calculated.



Signatures are generated through convolution operations

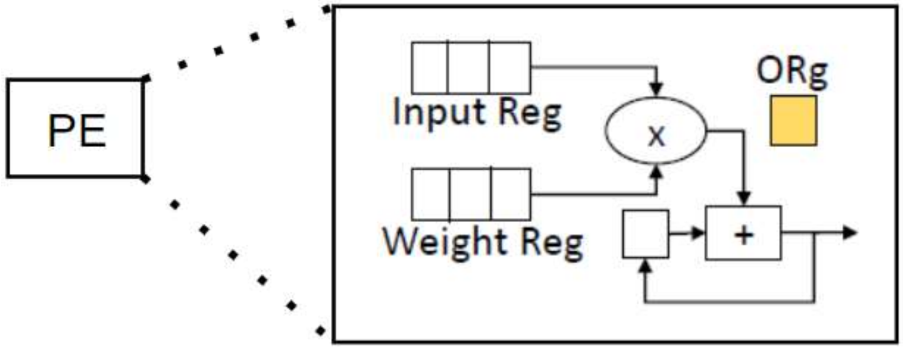
Dataflow of Signature Calculation

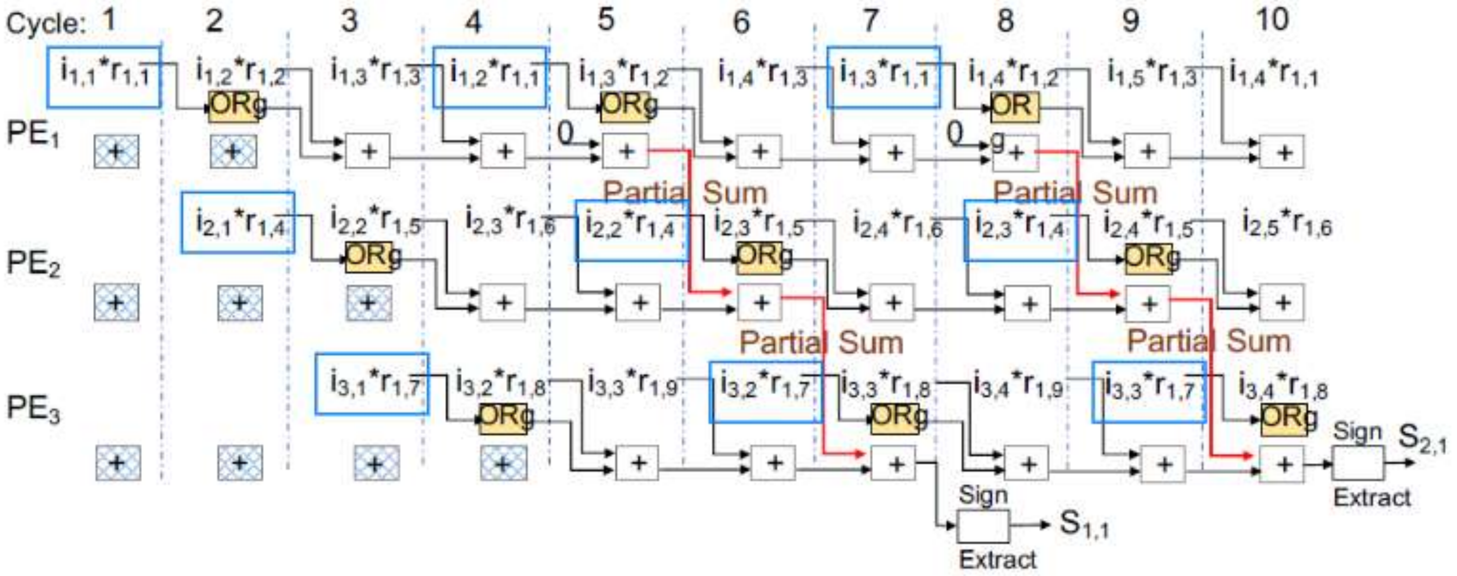
$$\begin{bmatrix} i_{1,1} & i_{1,2} & i_{1,3} \\ i_{2,1} & i_{2,2} & i_{2,3} \\ i_{3,1} & i_{3,2} & i_{3,3} \end{bmatrix} * \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{1,4} & r_{1,5} & r_{1,6} \\ r_{1,7} & r_{1,8} & r_{1,9} \end{bmatrix} = \boxed{\text{Sig } 1,1} \Rightarrow$$



- Input I_1 : $i_{1,1}$ to $i_{3,3}$
- R_1 : $r_{1,1}$ to $r_{1,9}$
- Non-overlapping fashion
  - Generating single bit of a signature requires six cycles.
  - Calculation of one bit of one signature does not overlap with another signature.

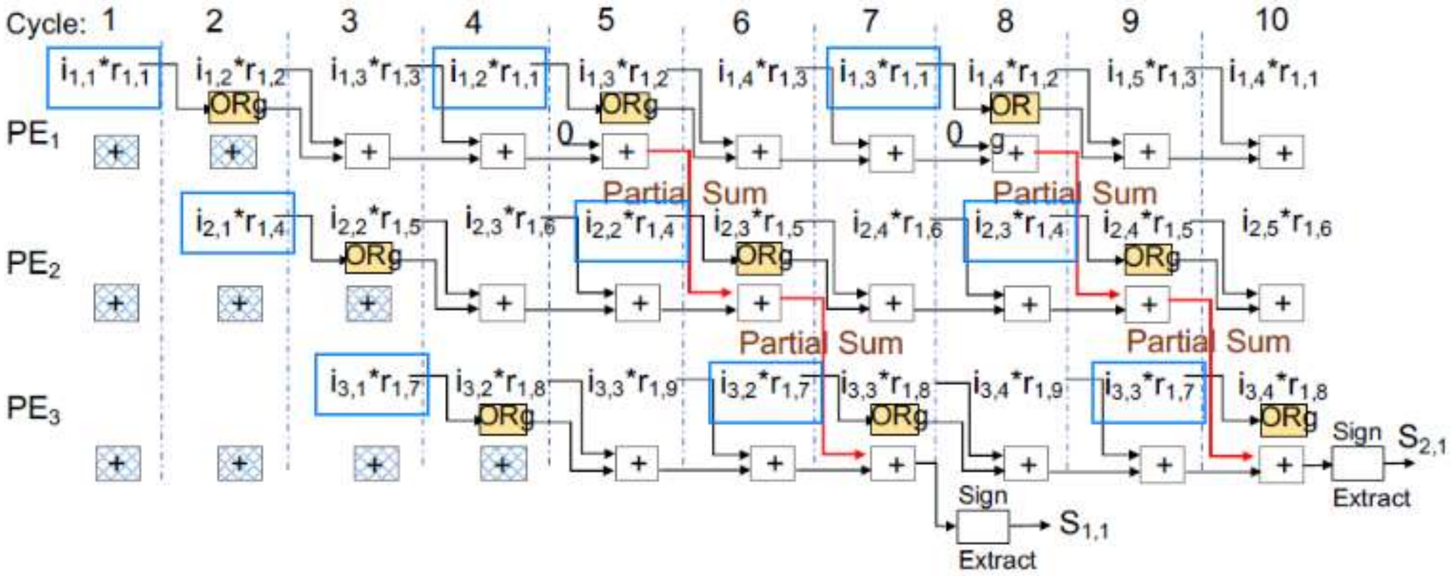
Overlapped Fashion

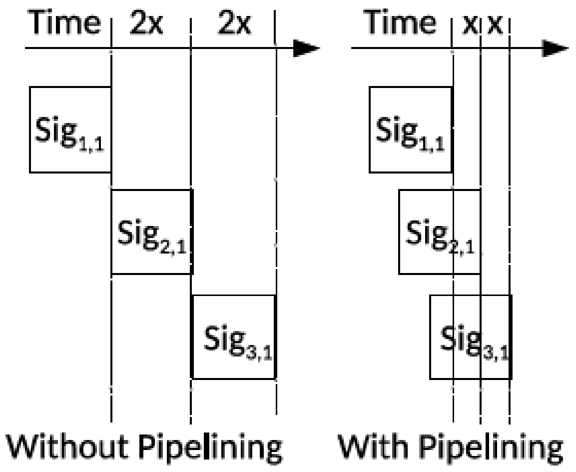




- MERCURY propose to pipeline the calculation of one signature with another.
- For this, register named Overlapped register (ORg) is added.
- Calculation starting time of PE_2 and PE_3 is intentionally delayed.

Overlapped Fashion





- ORg holds the result of multiplying the first element of each row of input and random vectors.

- Sig1,1 takes seven cycles but calculation of Sig2,1 has already started.

- In this manner, for (X x X) input vectors, first bit of signature could be generated in 2X+1 cycles, but other bits take X cycles to finish.

Signature Management

- MERCURY manages signatures using three structures.
- Signature Table stores the signatures
- MCACHE keeps dot product results computed between different input and weight vectors.
- Hitmap keeps track of which signature causes a hit in MCACHE.

Signature Table

- The signature table is indexed by the input vector number.
- MERCURY can easily find signature of each input vector.

Hitmap

- Hitmap is indexed by input vector number as same as signature table.

- MERCURY stores certain signature is HIT or not.

MCACHE

- MCACHE is indexed and tagged with the signature.

- When a signature is calculated by the PEs, MERCURY stores it in the signature table and then accesses MCACHE.

- MCACHE keeps computed dot product results so that input vectors with similar signature can reuse them.

MCACHE

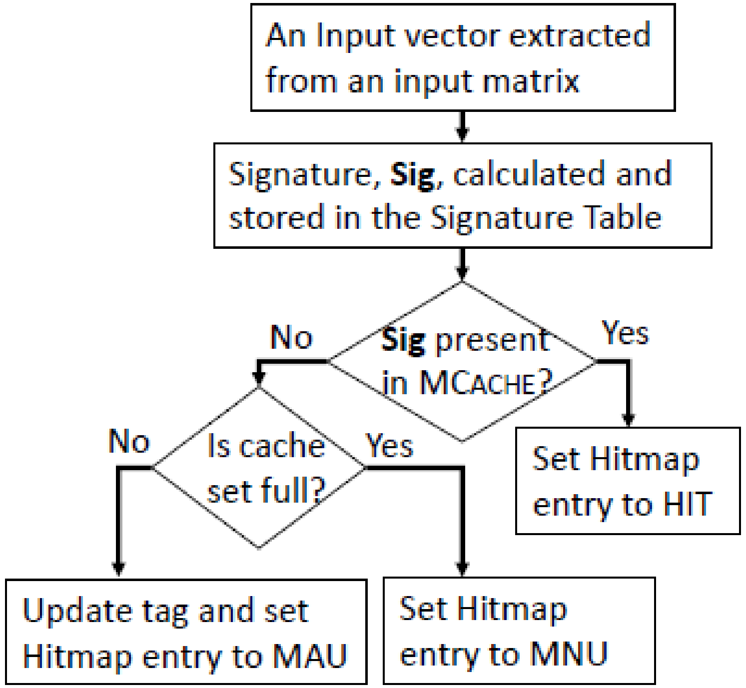
- MCACHE is different from normal cache in two ways

1. Tag and data are separate
    - Tag (i.e., signature) is produced before data (i.e., computed results), cache tag and data are not updated together.
    - Thus, each line has two valid bits, Valid Tag (VT) and Valid Data (VD).

2. There is no replacement in MCACHE.
    - When a set is full, no new entries are inserted into MCACHE.
    - Why? Just to simplify the design of MCACHE.

Update of Structures

- There is Sig from the input vector.
  - Sig is in MCACHE: this is hit, and Hitmap entry is set to HIT.
  - Sig is not in MCACHE: that is new Sig
    - MCACHE is not full
      - Update tag of entry
      - Hitmap entry is makred as Miss And Update (MAU).
    - MCACHE is full
      - Sig is not be inserted.
      - Hitmap entry is makred as Miss No Update (MNU).



- Hitmap and signatures are calculated before the convolution operations for a channel begin.

Exploiting Input Similarity

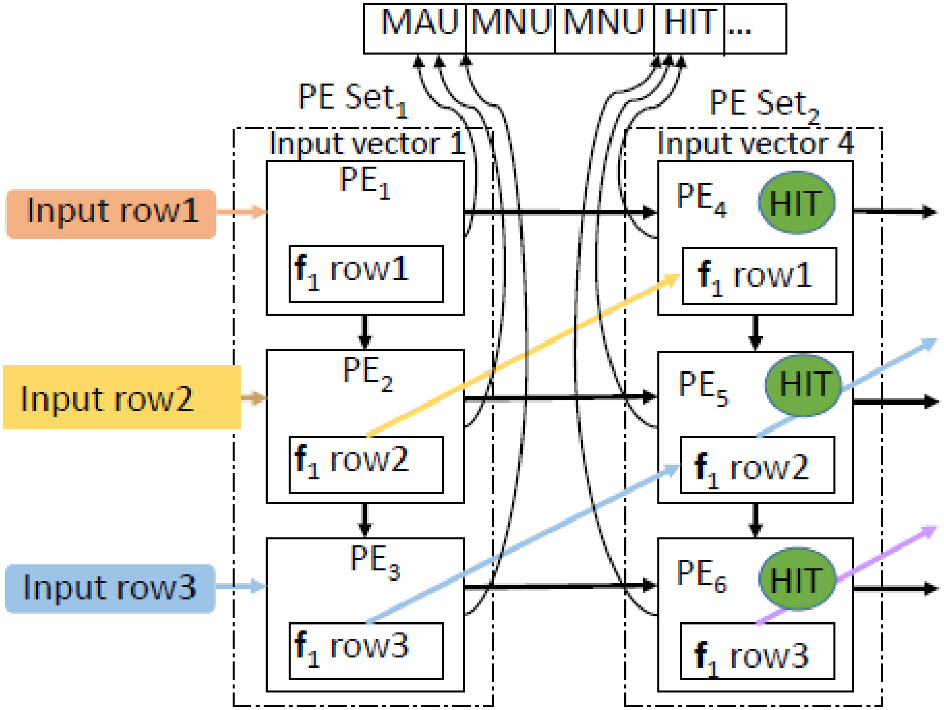
- MERCURY on convolutional layer
  - Forward Propagation
  - Backward Propagation

- MERCURY on fully connected layer

- MERCURY on attention layer

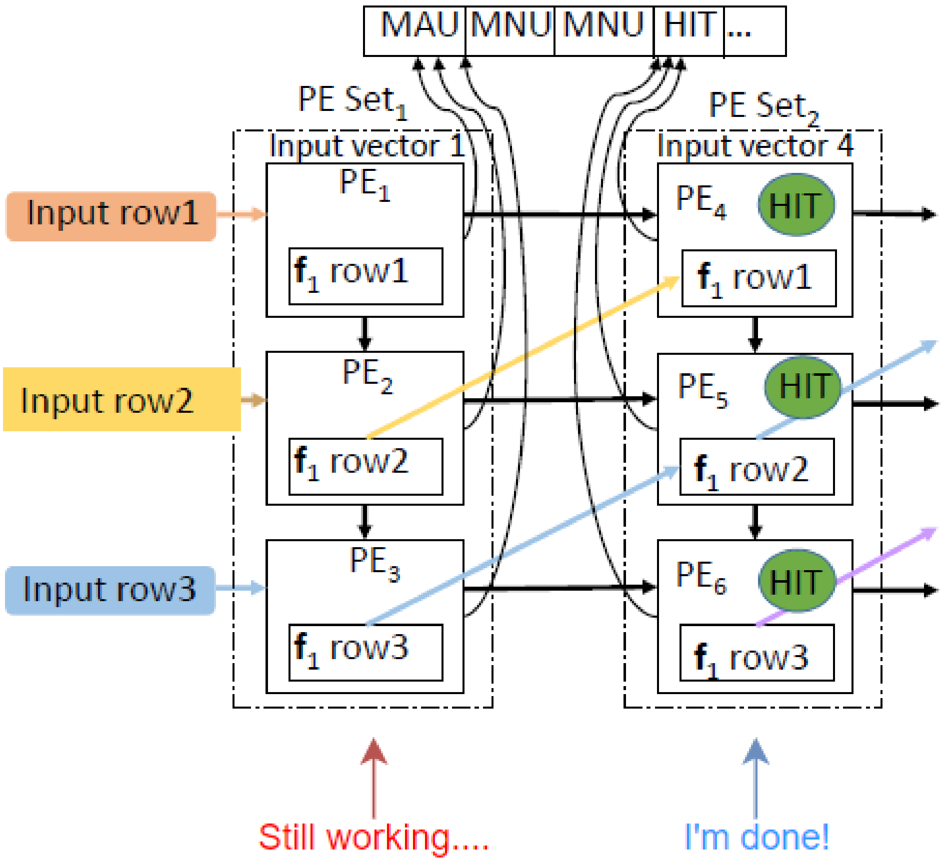
Input Similarity in Forward Pass

- PE Set1
  - Check entry 1 of the Hitmap
  - MAU: compute dot product and update data portion.
- PE Set2
  - Check entry 4 in the Hitmap
  - Hit: reuse previous result.



Problem

- Each PE set acts independent of other PE sets.
- One PE set might reuse a lot of computed results and finish its work.
- But, another PE set might not done its work yet.
- Thus, MERCURY has two design for this.
  - Synchronous Design
  - Asynchronous Design



Synchronous Design

- PE set always wait for other all PE sets.
  - Each PE set maintains a busy bit (B).
  - A controller checks all B bits.
  - If none of them are busy, controller loads next filter and input.
- Processing the next filter
  - VD flags of MCACHE is all invalidated since weight filter is different from the previous one.
  - VT flags and Hitmap are still valid and kept since input vectors are remain the same.
- Processing the next channel
  - When MERCURY processes next channel, all of the structures need to be recalculated and reinitialized.
- Pros and Cons
  - Intuitive and simpler design
  - But limits performance improvement because faster PE sets idle until slower one completes.

Asynchronous Design

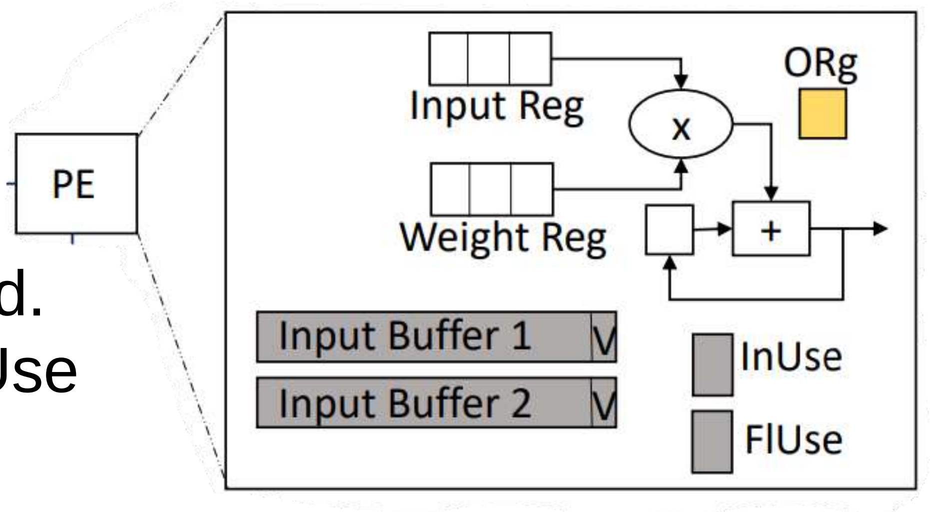
- On asynchronous design,
  - Faster PE sets can work on the next filter and input vectors.
  - Slower ones work on the previous filter and input vectors.

- But this requires additional buffer and coordination shceme.

- Three major changes
  1. Each PE is extended to have two input buffers to store new input vectors.
  2. The accelerator stores multiple filters in a shared buffer so that each PE can access it.
  3. Make MCACHE a multi-version cache since each input vector and filter produces new dot product result.

(1) Two Input Buffers

- Each buffer has an associated valid (V) bit.
- InUse register
  - Indicates which of the two buffer is currently used.
  - All PEs in PE set will have the same value in InUse register.
- In extra buffer,
  - The next input vector is stored in a streaming fashion.
  - Thus, when those PE sets finish computations, new input vector is already available.

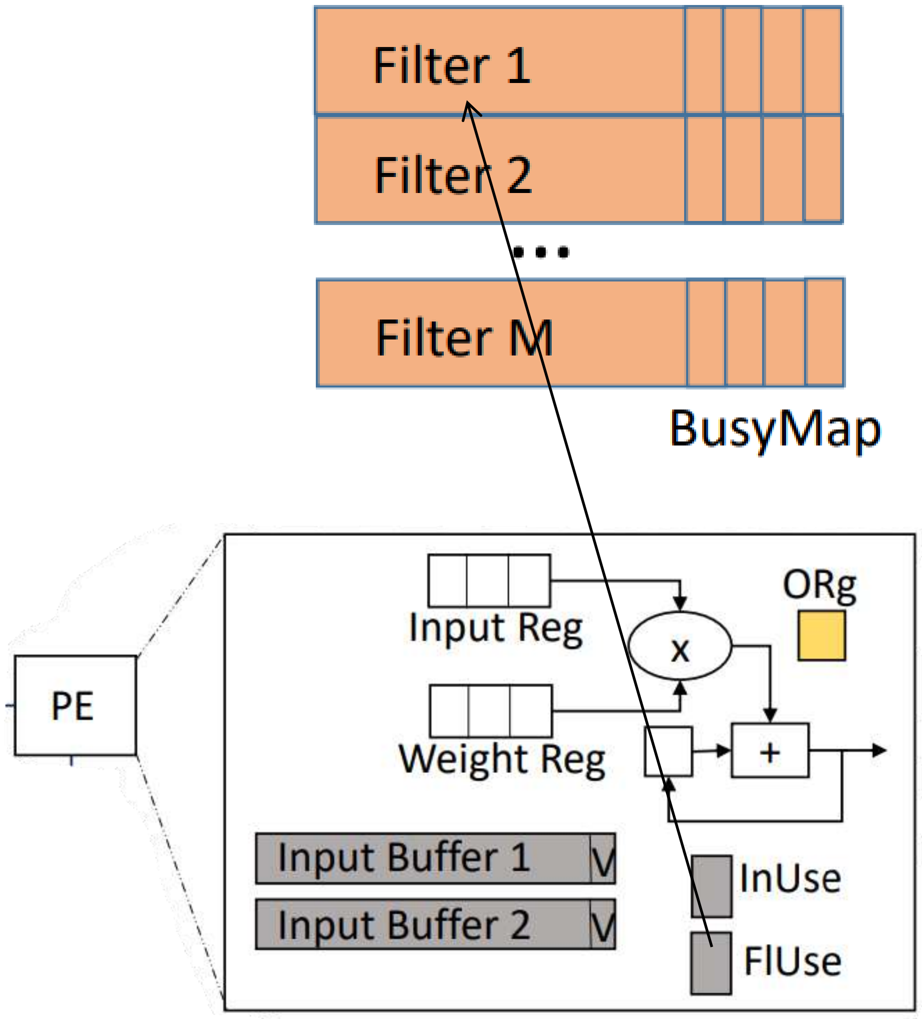


(2) Multiple Filter

- BusyMap
  - Each filter has an associated BusyMap.
  - This indicates which PE sets are currently busy with that filter.

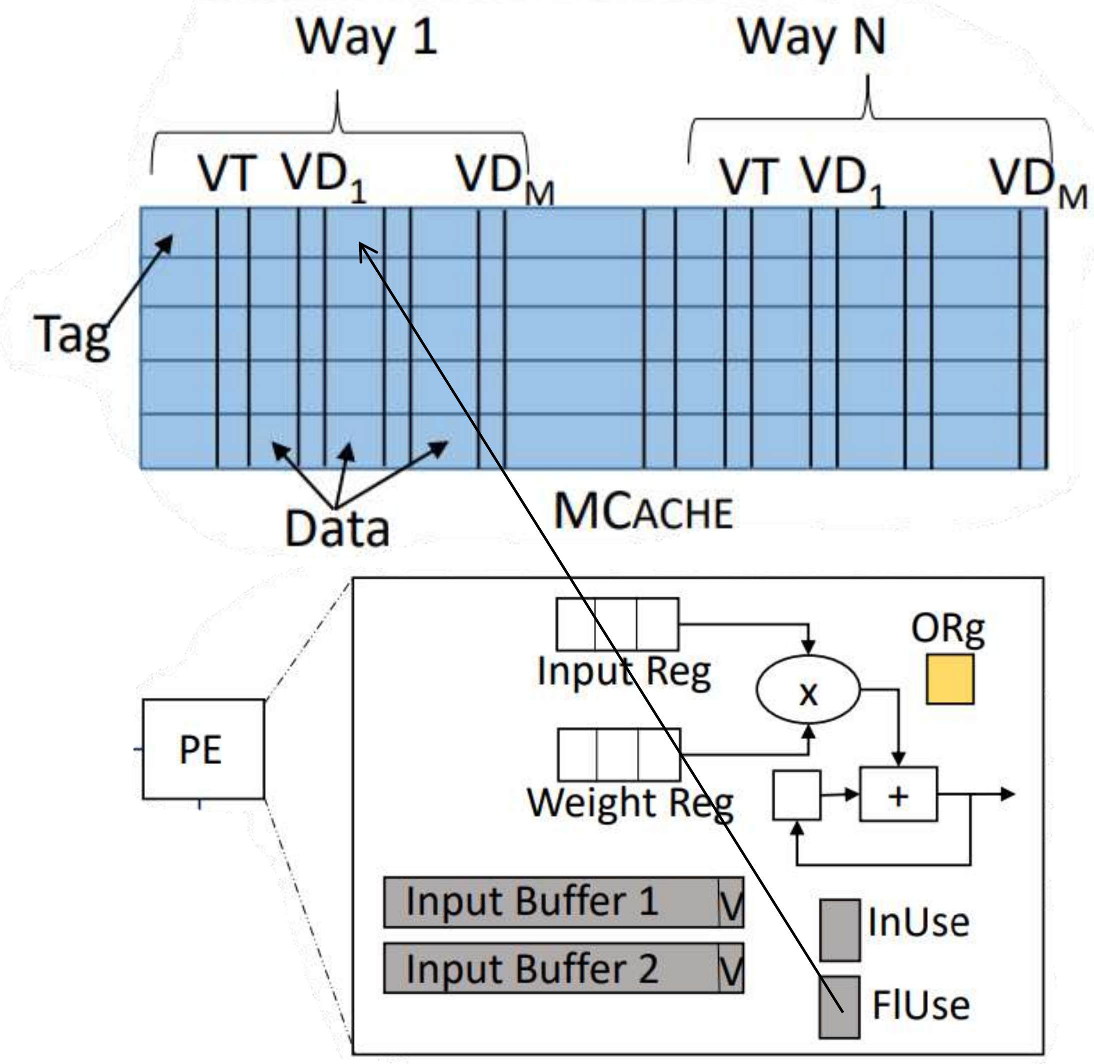
- FlUse
  - Each PE has this register.
  - This indicates which filter is using.
  - All PEs in PE set has same value.

- When all PE sets finish using a filter, new filter is loaded and the BusyMap is initialized.



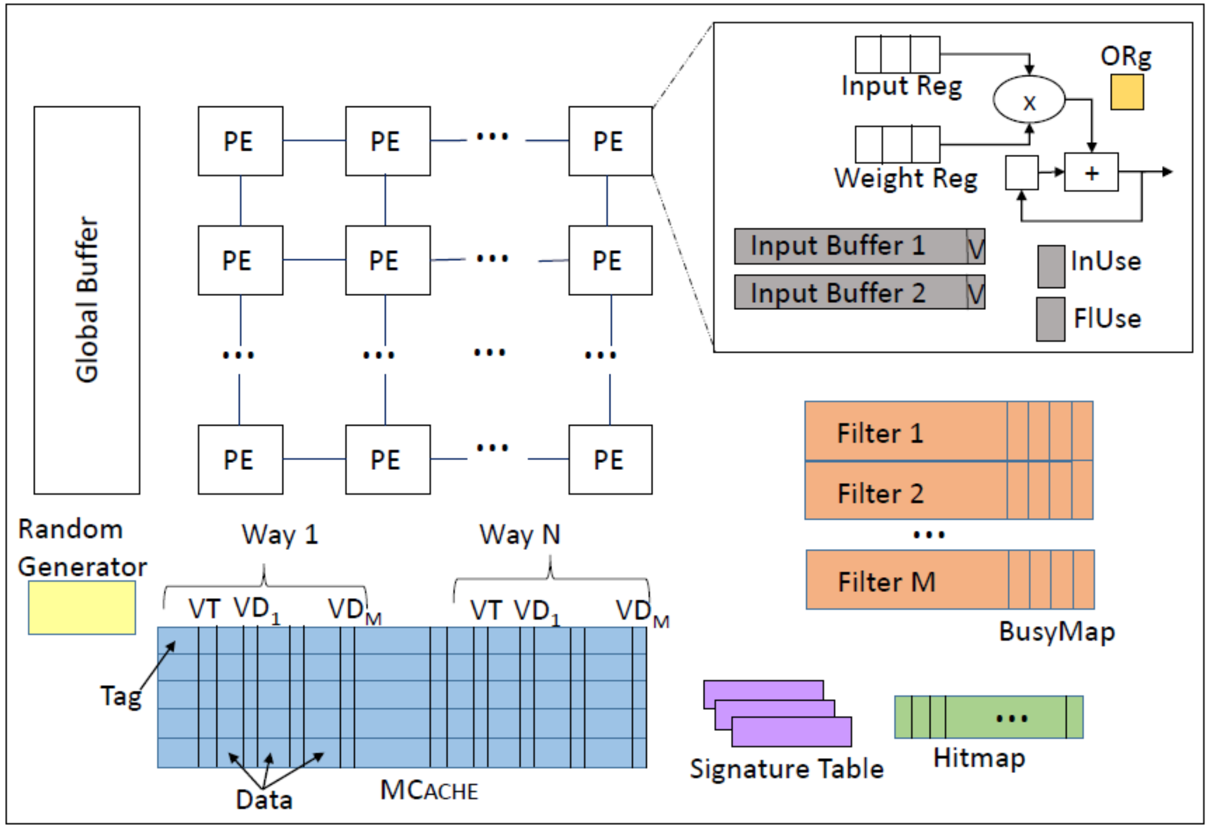
(3) Multi-version MCACHE

- Each cache line has multiple versions of data.
- Each data portion has its own VD bit.
- There are as many version as the number of filters.
- If no space to store new filter, PEs in the PE set remain idle.
- PE uses FlUse register when it accesses MCACHE to specify the version of data.



Summary of Asynchronous Design

- Two Input Buffer
  - InUse Register
- BusyMap
  - FlUse Register
- Multi-version MCACHE



Input Similarity in Backward Pass

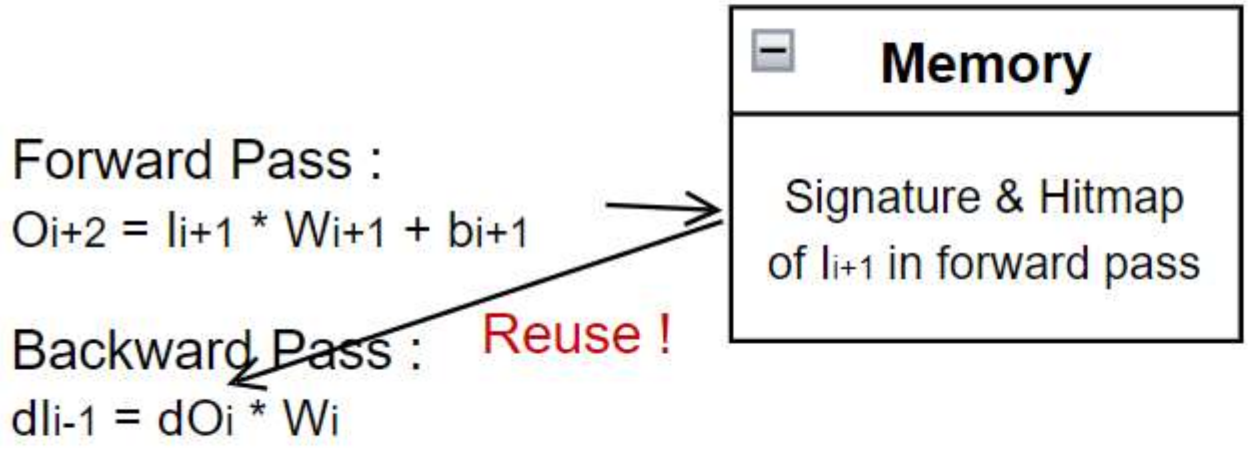
- Two computations in backward pass
  - dW_i: Calculation of weight derivatives
  - dI_i: Calculation of input derivatives

- Let's focus on dI_i.
  - If the filters of layer i+1 have same dimension as those of layer i,
  - Signatures and Hitmap produced by layer i+1 for I_{i+1} can be applied to calculation of dI_i.

- Therefore, save signatures and the Hitmap of each layer during the forward pass and reload them on backward pass.

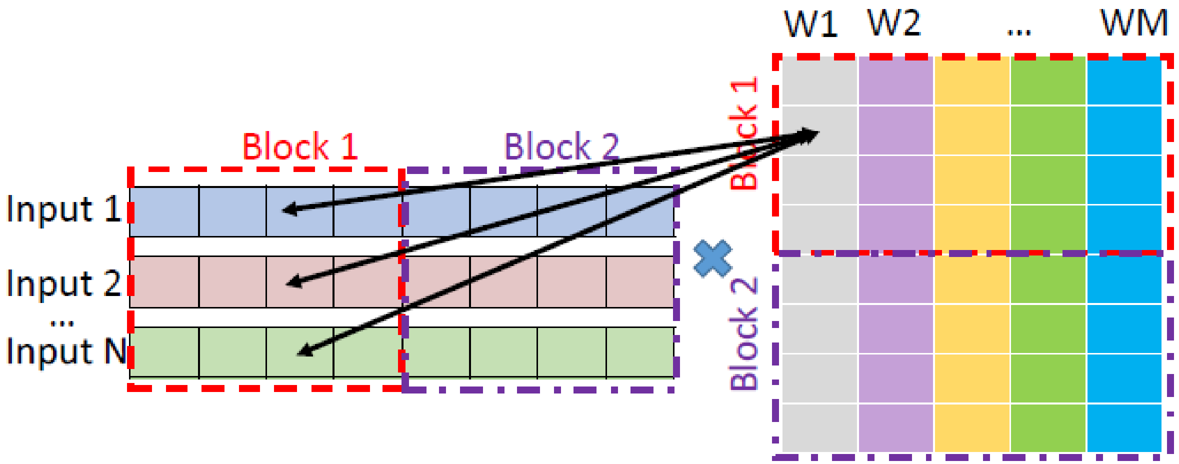
- Note that if filter's dimensions don't match, MERCURY can't reuse previous results.

Input Similarity in Backward Pass



- Input in layer i+1 is same as output in layer i.
- Also, same calculation, input vector * weight filter operation in forward & backward pass.
- Thus, if dimension of filter on layer i and i+1 is match, those signature and hitmap in layer i+1 could be reused for dI_i.

Input Similarity in Fully Connected Layer



- Inputs and weights in minibatch are divided into blocks base on the number of PEs.
- One PE multiplies Input 1 of block 1 with W1 of weight block 1 followed by W2, W3, ... WM.
- Concurrently, another PE multiplies Input 2 and W1 followed by W2 to WM.
- This continues up to Input N.

Input Similarity in Fully Connected Layer

- Similar as convolutional layer, if one input is similar to another, calculation could be skipped.

- This is Hit and those PEs can start the operation of the next input or weight (same as asynchronous processing).

- Also, structures such as MCACHE, Hitmap and signature table are handled samely.

Input Similarity in Attention Layer

- In an attention layer, assume that
  - Input vectors: $\mathbf{X^{t*k}} = \mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_t}$
  - Output vectors: $\mathbf{Y^{t*k}} = \mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_t}$.
- To produce output vector y_i, attention layer takes a weighted average over all input vectors, $\mathbf{Y_j} = \sum_j W_{ij} X_j$
- This can be represented as $\mathbf{Y^{t*k}} = \mathbf{W^{t*t}} * \mathbf{X^{t*k}}$
- Thus, exploiting the similarity among x_i vectors is possible.

Adaptation

- DNN models become more sensitive to computation reuse.

- Thus, MERCURY use adaptive way.
  1. Increase in Signature Length
  2. Stoppage of Similarity Detection

Increase in Signature Length

- Definition of input vector similarity
  - $v_1 - v_2 = \varepsilon$
  - Small ε: two vectors are similar.

- Larger signature
  - Less impact on model accuracy
  - But reduce computation reuse

- Therefore, MERCURY starts with smaller signature size and gradually increases signature length. How?

- If loss is not reduced for k consecutive iterations, MERCURY increments signature length by 1.

Stoppage of Similarity Detection

- MERCURY analytically determines if detecting similarity can save computations or not.

- MERCURY recorded the total computation cost (i.e., cycles) C_S for signature generation when some computation are reused.

- This cost is compared with the total computation cost C_B of the baseline system without any computation reuse.

- If $C_S > C_B$ => MERCURY stops generating signatures.

Other Dataflows

- MERCURY can be easily implemented in other dataflow accelerators.

1. Weight Stationary

2. Input Stationary

Weight Stationary

- Weights are stationary in the PEs, and input vectors are broadcasted to PEs.

- Signature calculation
  - Load random vectors (i.e. $R_1 \sim R_N$) to PEs and one input vector will be broadcasted to PEs.
  - A signature is made by several PEs
  - Finally, signature table is updated.

- Exploiting similarity
  - Signature table, Hitmap and MCACHE is updated by specifying the hit/miss for them.
  - Weights are loaded, similar vectors skip the dot product and reuse the previous results.

Input Stationary

- Inputs are stationary in PEs and weights will be broadcasted.

- Signature calculation
  - Load input vectors (e.g. $I_{1,1} \sim I_{3,3}$) and one random vector will be broadcasted to PEs.
  - Similar to weight-stationary dataflow, signature is made by several PEs and signature table is updated.

- Exploiting similarity
  - Weights are loaded and streamed.
  - But if there is a hit for an input vector, MERCURY skips broadcasting weights and loads next input vector.
  - If if there is a miss, MERCURY streams weight vector to the PEs.

Setup for Evaluation

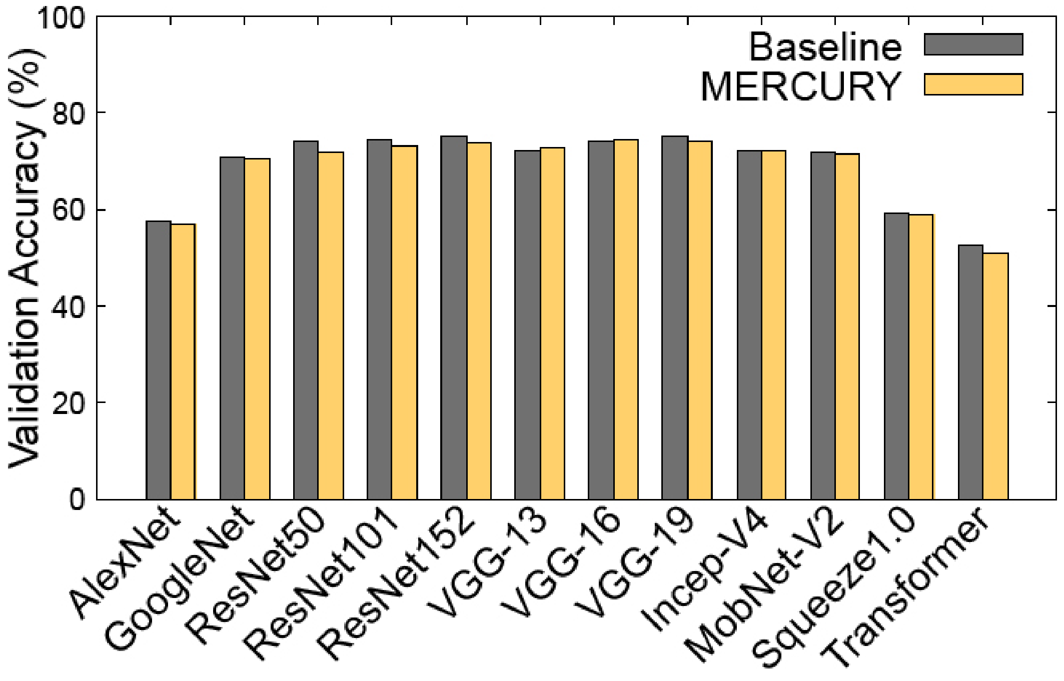
- Hardware
  - Virtex 7 FPGA board
  - Eyeriss-style row stationary accelerator
  - MCACHE: 1024 entries & 16 associativity
- Twelve DNN models
  - AlexNet, GoogleNet, VGG13 etc...
- Dataset
  - 80 Image classes from ImageNet / report the top 1% accuracy.
  - Multi30k for evaluating transformer model / evaluated with Bleu score.
- Comparison with:
  - UCNN
  - Unlimited Zero-Pruning
  - Unlimited Similarity Detection

Evaluation

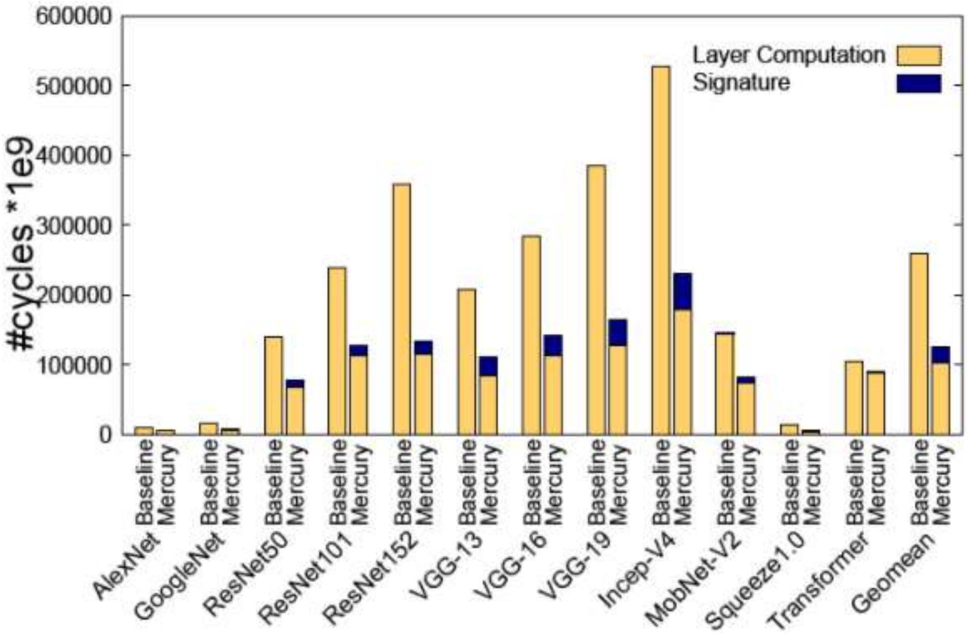
- Accuracy and Performance Comparison
- Case Study of VGG13
- Comparative Analysis
  - vs UCNN in inference mode
  - vs Unlimited Zero Pruning
  - vs Unlimited Similarity Detection
- Results with Other Dataflows

Accuracy between Models

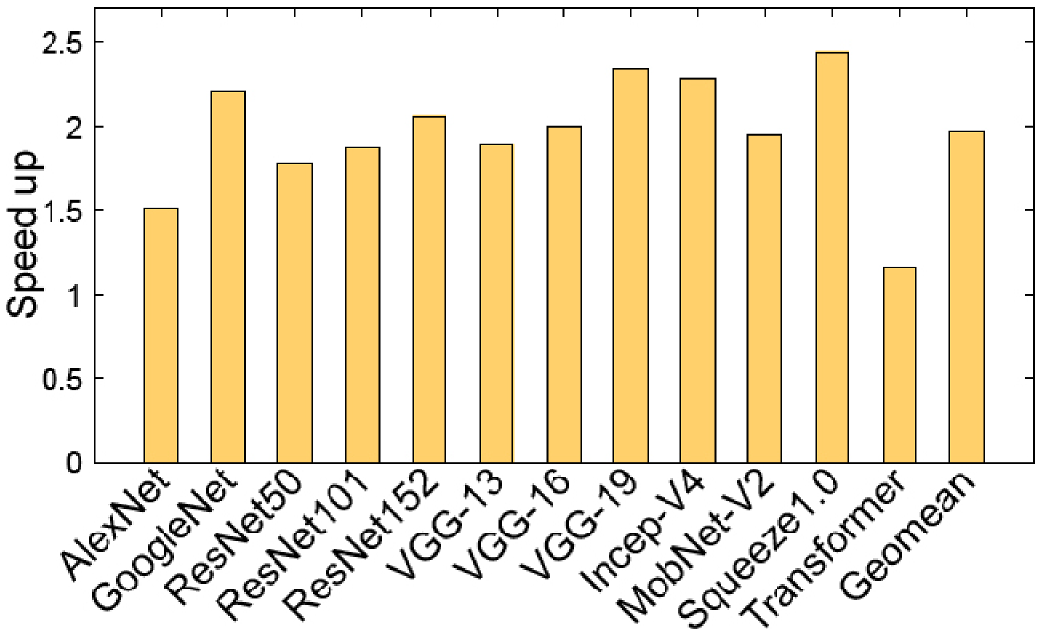
- This figure shows the impact of MERCURY on model's accuracy.
- Overall, there is a 0.7% reduction in accuracy.
- MERCURY's accuracy is comparable to the baseline system.



Performance between Models



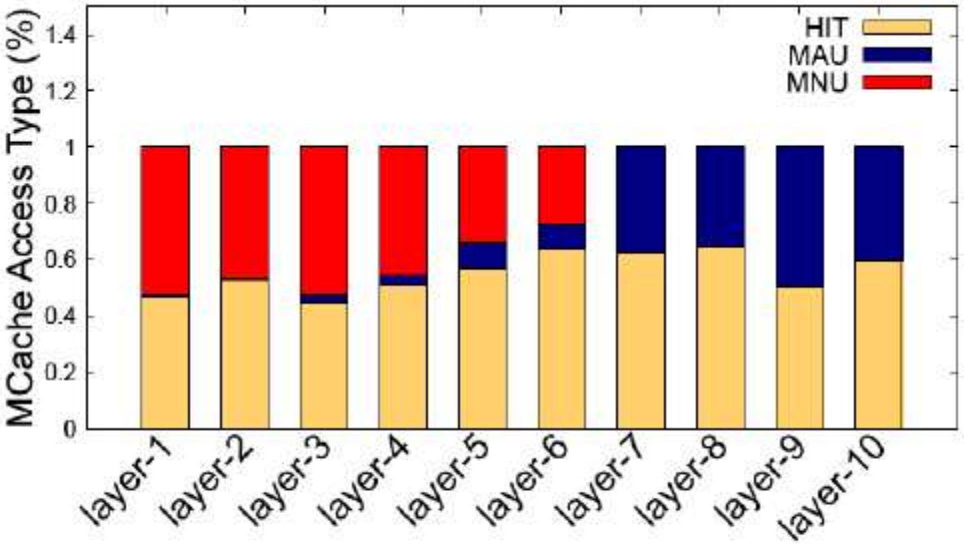
- This figure shows the computational cycle breakdown of MERCURY vs baseline.
- Most of the cycles belong to the layer computations.
- Signature computation accounts for only a small fraction of total cycles.



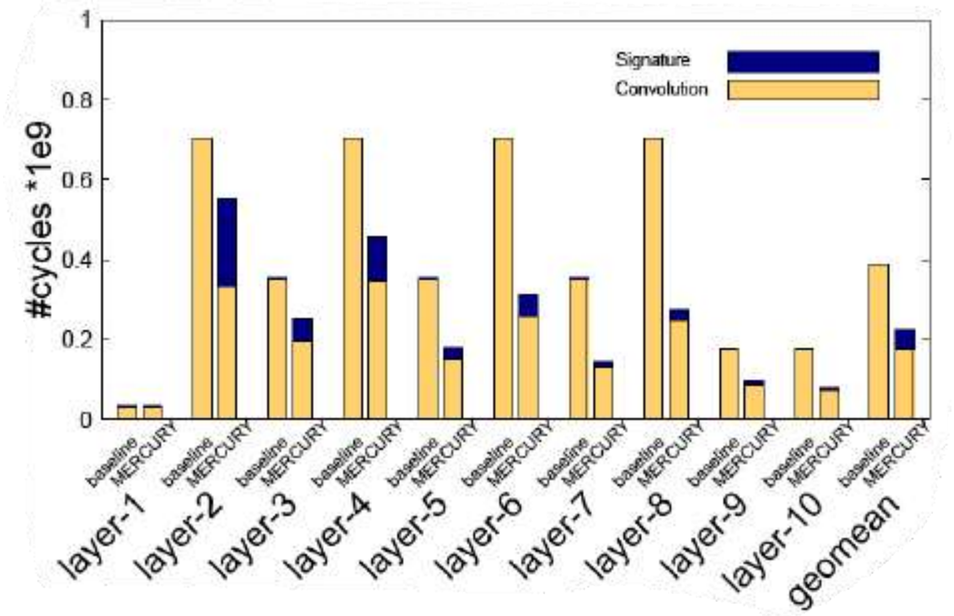
- This figure shows total speed up per model.
- Overall, MERCURY can reduce total computation by about 50%.
- This results in an average speedup of 1.97x.

Case Study of VGG13

This is a detailed analysis of VGG13 to show how MERCURY works.

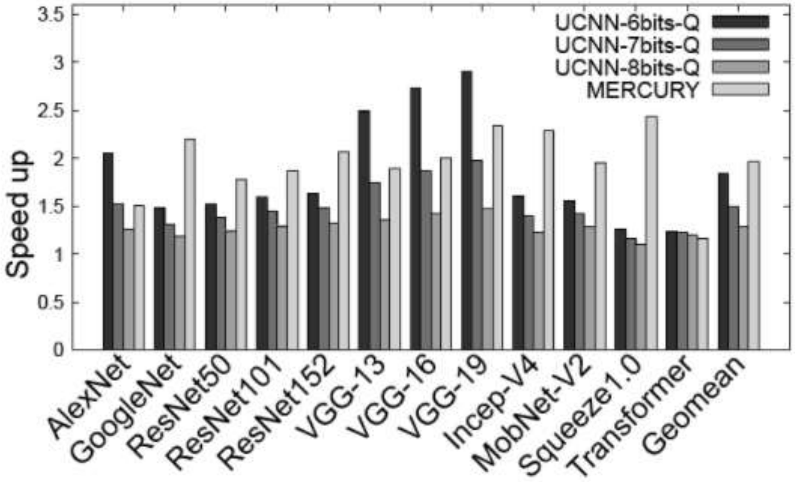


- We can see a gradual increase in MCACHE Hit and MAU percentage.

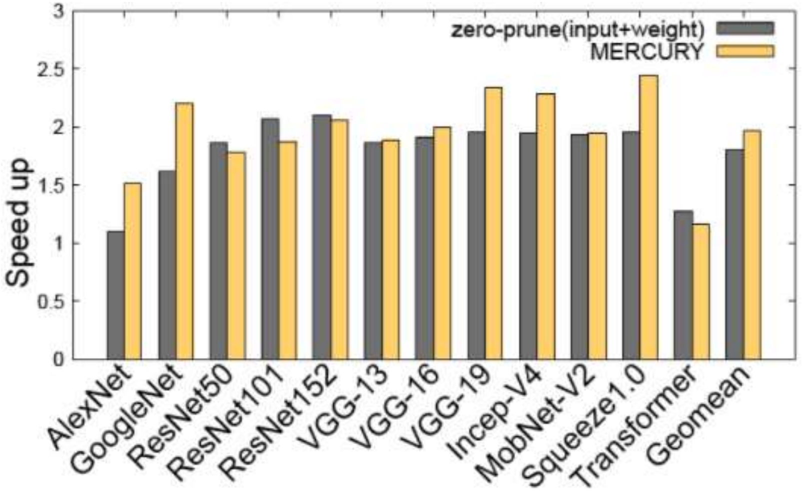


- Computational cycles vary across layers of VGG13.
- As we mentioned before, the cycles for signature calculation do not account for much.

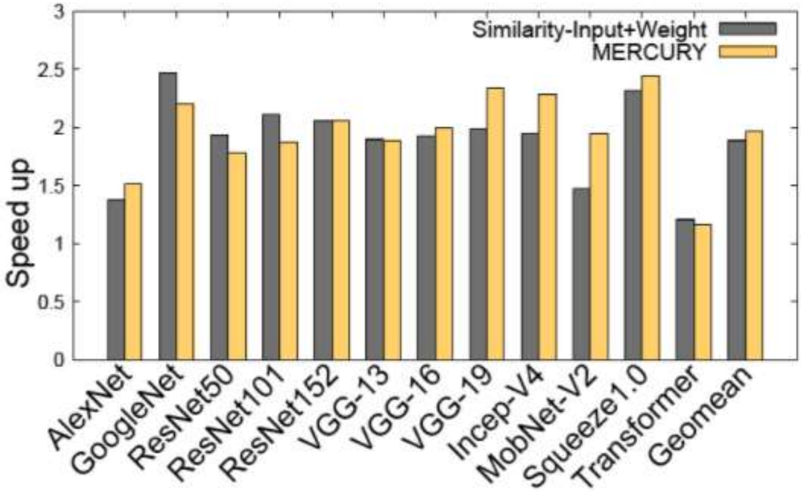
Comparative Analysis



- Comparison with UCNN in inference
- UCNN has different quantization policies: 6,7,8-bit
- MERCURY outperforms UCNN

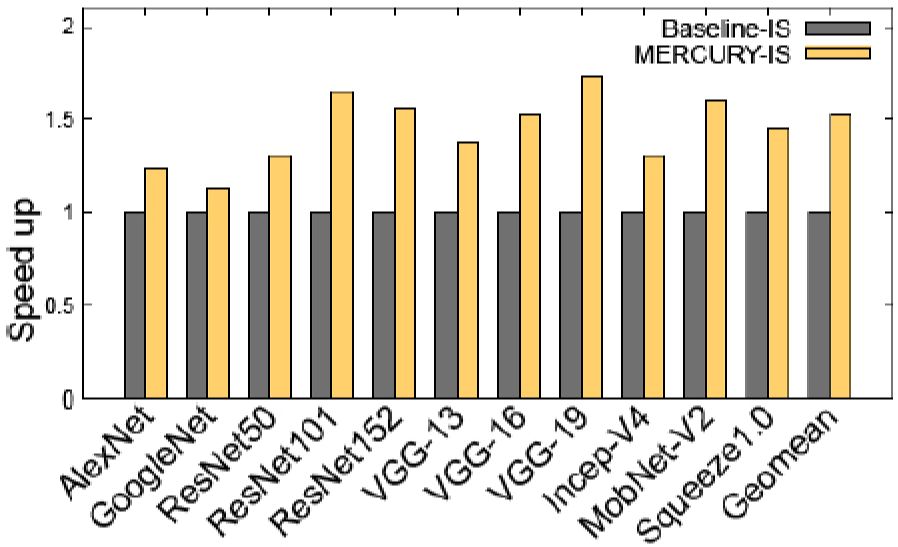


- Comparison with Unlimited Zero Pruning
- Zero pruning here assumes that the accelerator can detect and save all zero-related computations.
- On average, MERCURY outperforms by 4%.



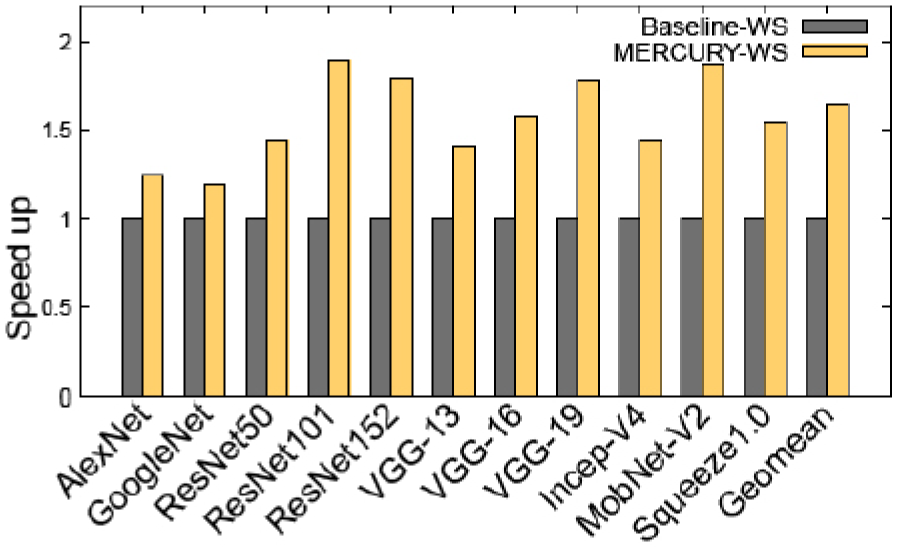
- Comparison with Unlimited Similarity Detection
- This assumes that it can find and save all similar elements.
- On average, MERCURY performs 2% better than the Unlimited Similarity technique.

Results with Other Dataflows



- Baseline-IS vs MERCURY-IS

- MERCURY with input stationary gives an average performance gain of 1.55x over the baseline



- Baseline-WS vs MERCURY-WS

- MERCURY with weight stationary gives an average performance gain of 1.66x over the baseline.

Summary

- MERCURY exploits the similarity among input vectors during DNN training.
- Key points
  - Use RPQ to make signature of the input vector and use signature to check if there exists similarity.
  - Use MCACHE, Signature table, Hitmap to store signature and to reuse the computed results.
  - Keeps the dataflow as the baseline.
- Adaptation
  - Gradually increasing the signature length long to prevent accuracy degradation.
  - Compare the computation costs of MERCURY vs baseline.
- Experimental evaluation with twelve different DNN model:
  - MERCURY speeds up the training by 1.97x.
  - Similar accuracy to the baseline system.