SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation

2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)

Authors:
Yang Zhao, Xiaohan Chen, Yue Wang, Chaojian Li, Haoran You,
Yonggan Fu, Yuan Xie, Zhangyang Wang, Yingyan Lin

Presenters:
Vishal Vijay Devadiga (CS21BTECH11061)
Shiv Tikoo (CS23BTNRK11001)

Table of Contents

Introduction

Background and Motivation

- Growing Demand to bring DNN-powered intelligence to resource constrained devices having limited energy and storage resources
- Since DNN inference mainly comprises of MAC operations, high performance can be achieved through parallelism, but this incurs heavy data movement.
- Data movement between the DRAM and the PEs is costly
  - In DianNao, 95% of the inference energy is consumed by data movements associated with DRAM.
- Modern DNNs have a huge number of parameters
  - In ResNet50, a medium-scale DNN, there are ≥ 23 million parameters.

Motivation

- On a commercial 28nm chip
  - Data access from DRAM takes about 700x more energy than MAC operations
  - Data access from SRAM takes about 10x - 17x more energy than MAC operations

TABLE I: Unit energy cost per 8-bit extracted from a commercial 28nm technology.

	DRAM	SRAM	MAC	multiplier	adder
Energy (pJ/8bit)	100	1.36−2.45	0.143	0.124	0.019

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

SmartExchange Introduction

- “How to alleviate heavy data movement?” is the question that SmartExchange paper tries to answer.
- The paper first proposes the SmartExchange Algorithm which aggressively reduces both:
  - Energy consumption of Data Movement
  - Storage Size associated with weights of a DNN
- To take advantage of the SmartExchange algorithm, the paper also proposes a dedicated accelerator which makes use of the much reduced weight storage and readily-quantized weights.

SmartExchange Algorithm

Formulation of the Problem

Let the weight matrix be $W \in R^{M \times N}$. This weight matrix is decomposed into two matrices:

- Basis Matrix $B \in R^{r \times N}$
- Coefficient Matrix $C_e \in R^{M \times r}$

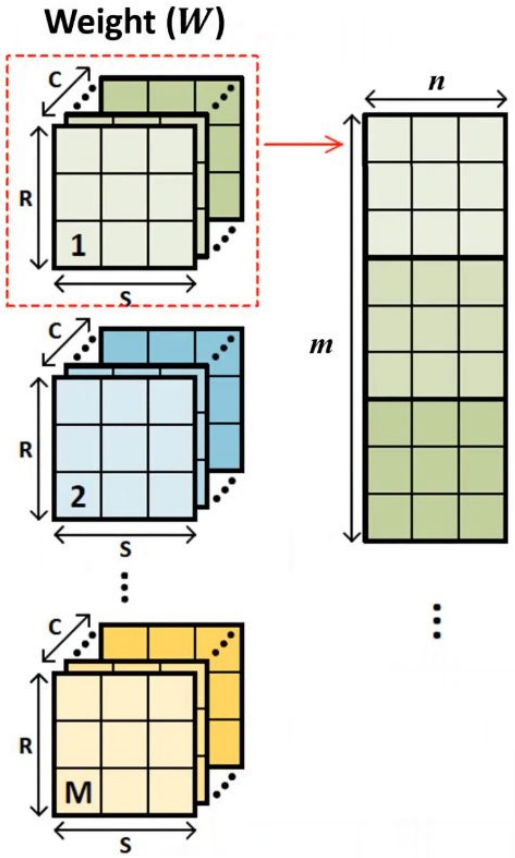
such that

$$W \cong C_e B$$

The reconstruction error is defined by

$$|| W - C_e B ||^2_F$$

Where $|| M ||_F$ is the square root of sum of squares of individual elements in the matrix M



Stack the weight filter into a tall matrix, that is, reshape R×S×C matrices into (R×C)×S matrices.

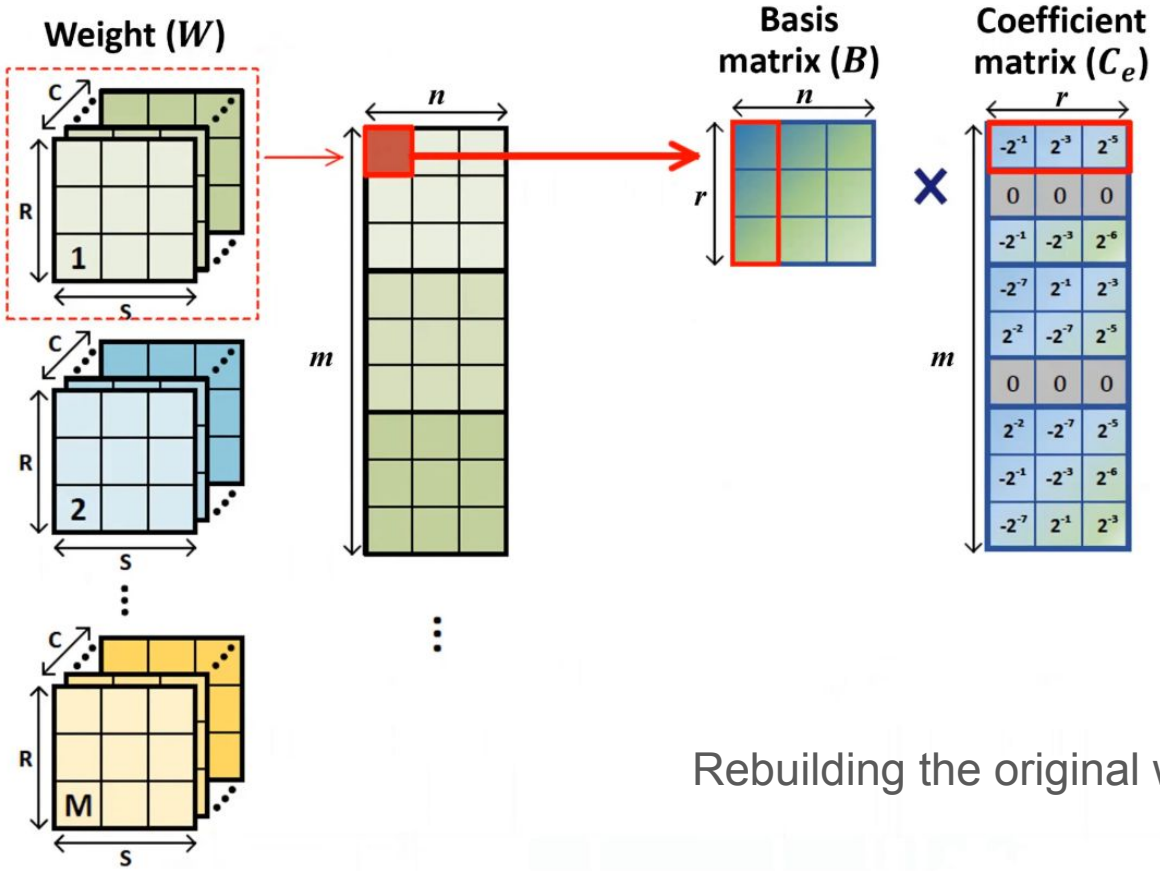
Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Properties of the Decomposed Matrices

- Basis Matrix B
  - Very small matrix
- Coefficient Matrix C_e
  - Large yet highly Sparse Structure
  - Highly Quantized: values of non zero elements are exactly powers of 2

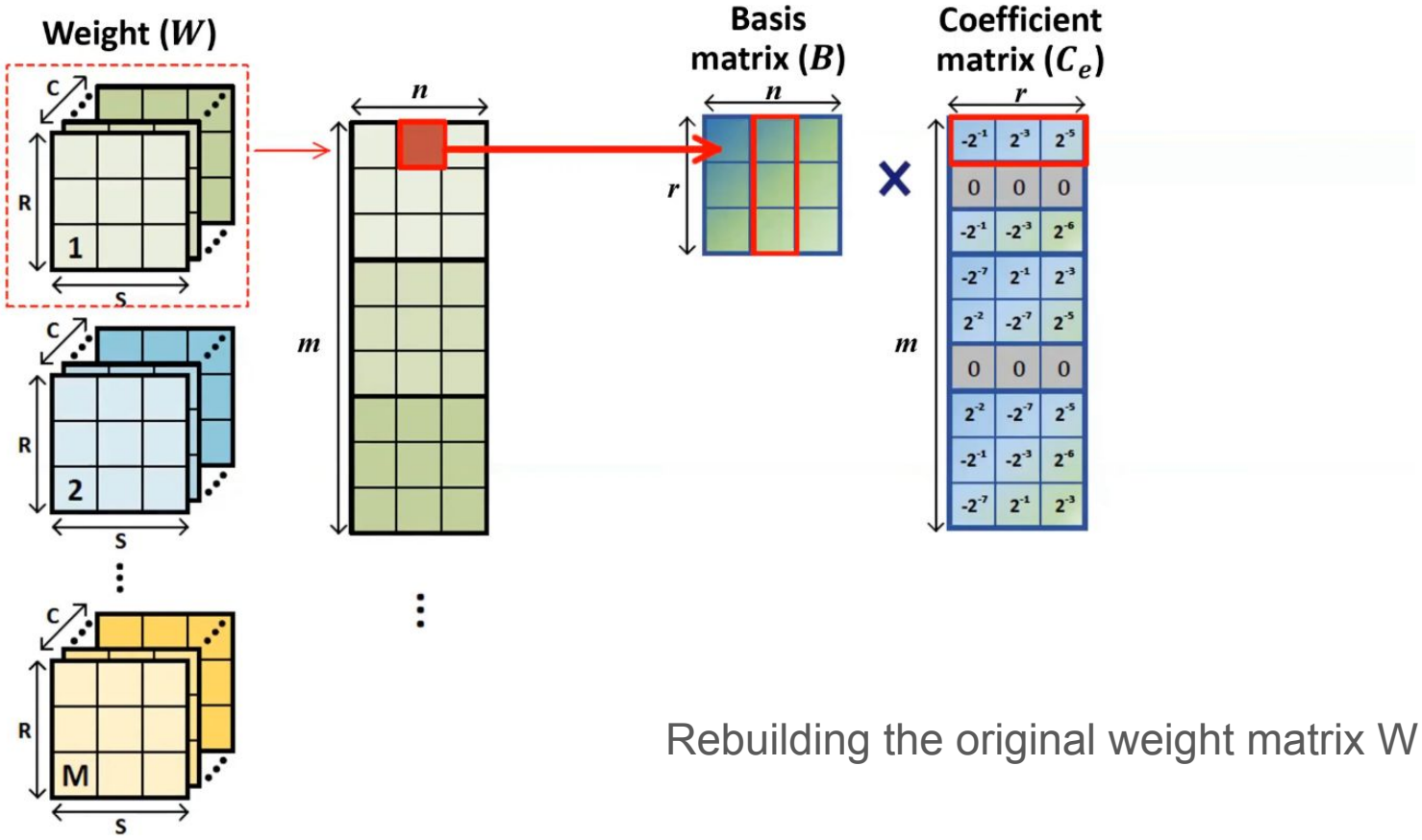
Bit representation of C_e is very compact.

Rebuilding the original weight matrix W from B and C_e is simplified into lower-cost shift-and-add operations.



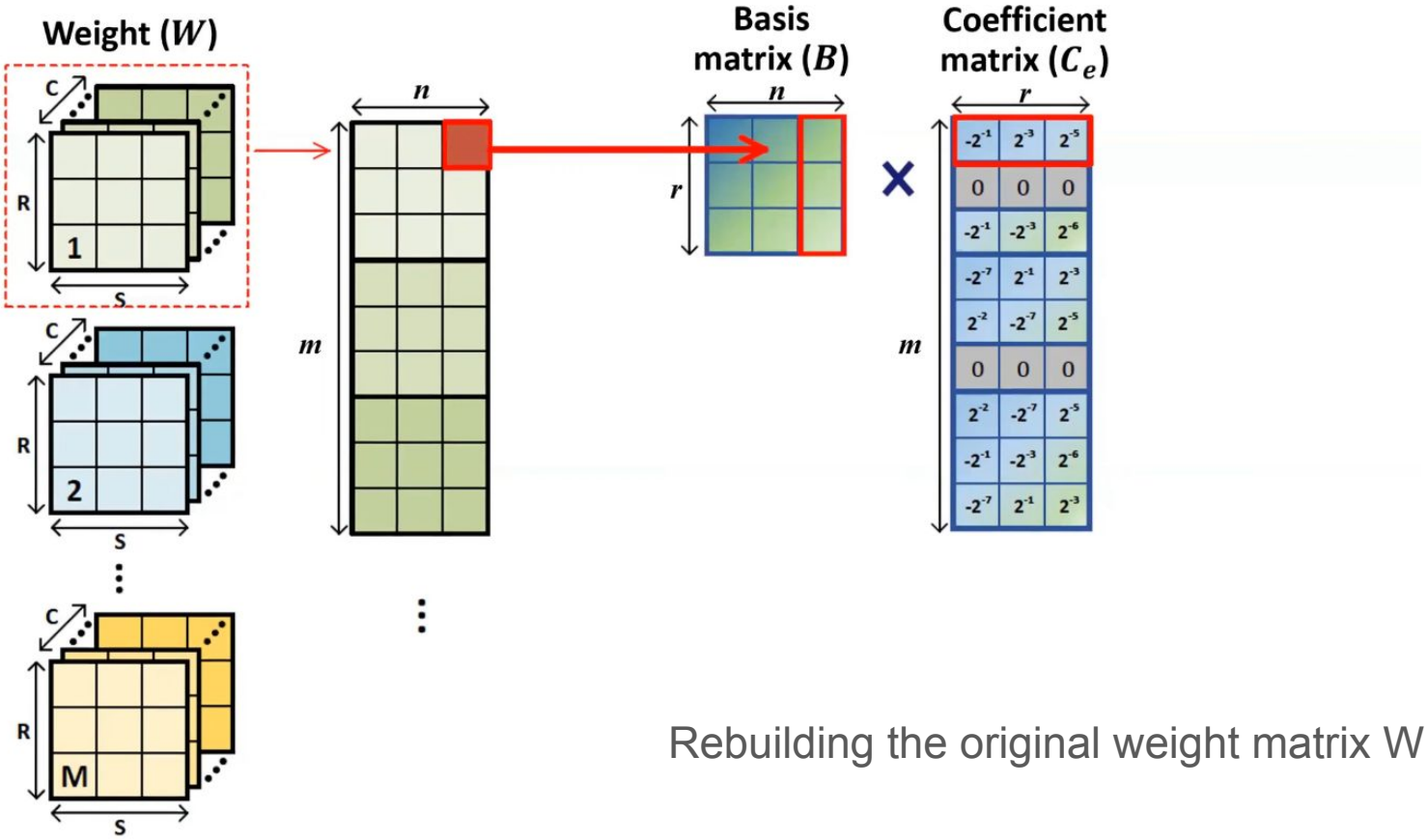
Rebuilding the original weight matrix W

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation



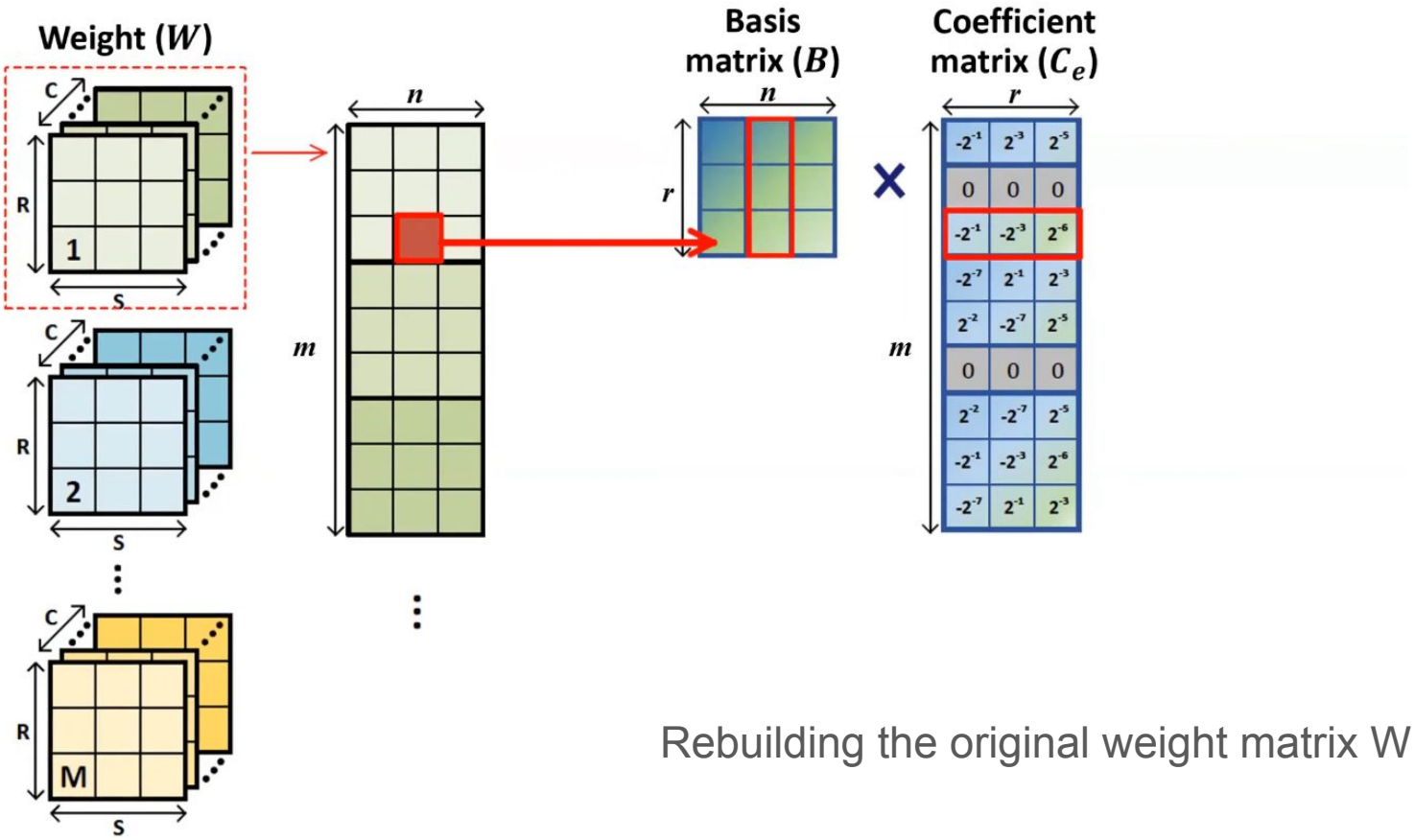
Rebuilding the original weight matrix W

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation



Rebuilding the original weight matrix W

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation



Rebuilding the original weight matrix W

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Constrained Optimization

$$\text{argmin} \; || W - C_e B ||^2_F$$

such that:

- $\Sigma_j \; || C_e [:, j] ||_0 \leq S_c$
- $\forall \; i,j : | P | \leq N_p$
- $C_e [i, j] \in \Omega_p$

S_c is the Sparsity Constant

N_P controls the bit-width required to represent elements in C_e

P is the integer set whose cardinality is less than N_p

Ω_p is the set of power-of-2 values in set P

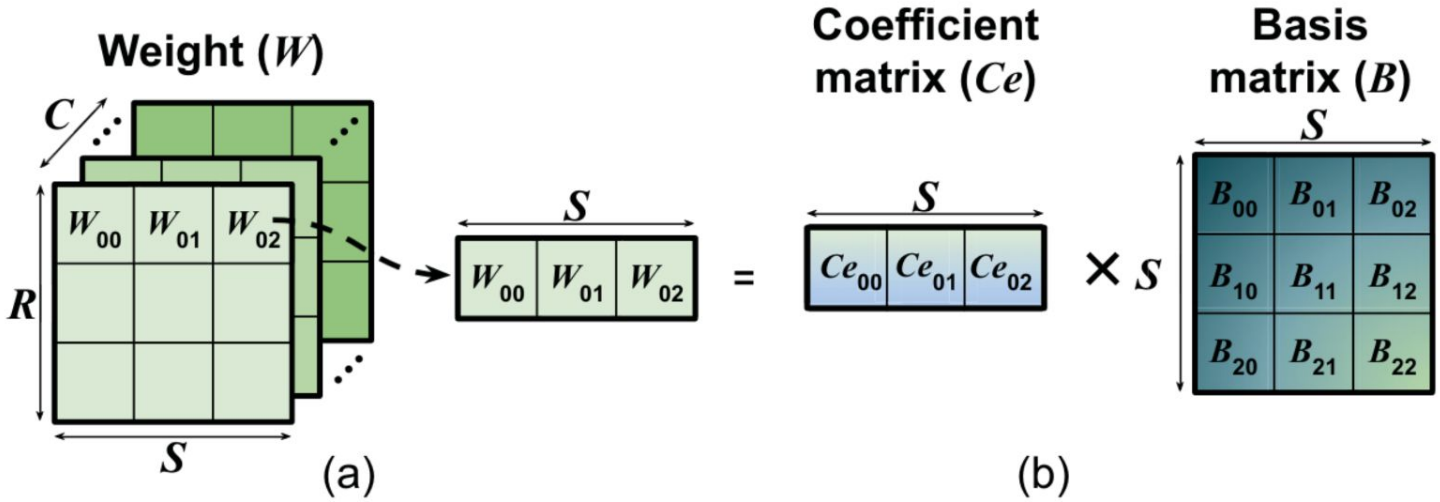


Fig. 2: An illustration of (a) a 3D weight filter and its parameters' notations, and (b) rebuilding one row of weights using the corresponding basis and coefficient matrix.

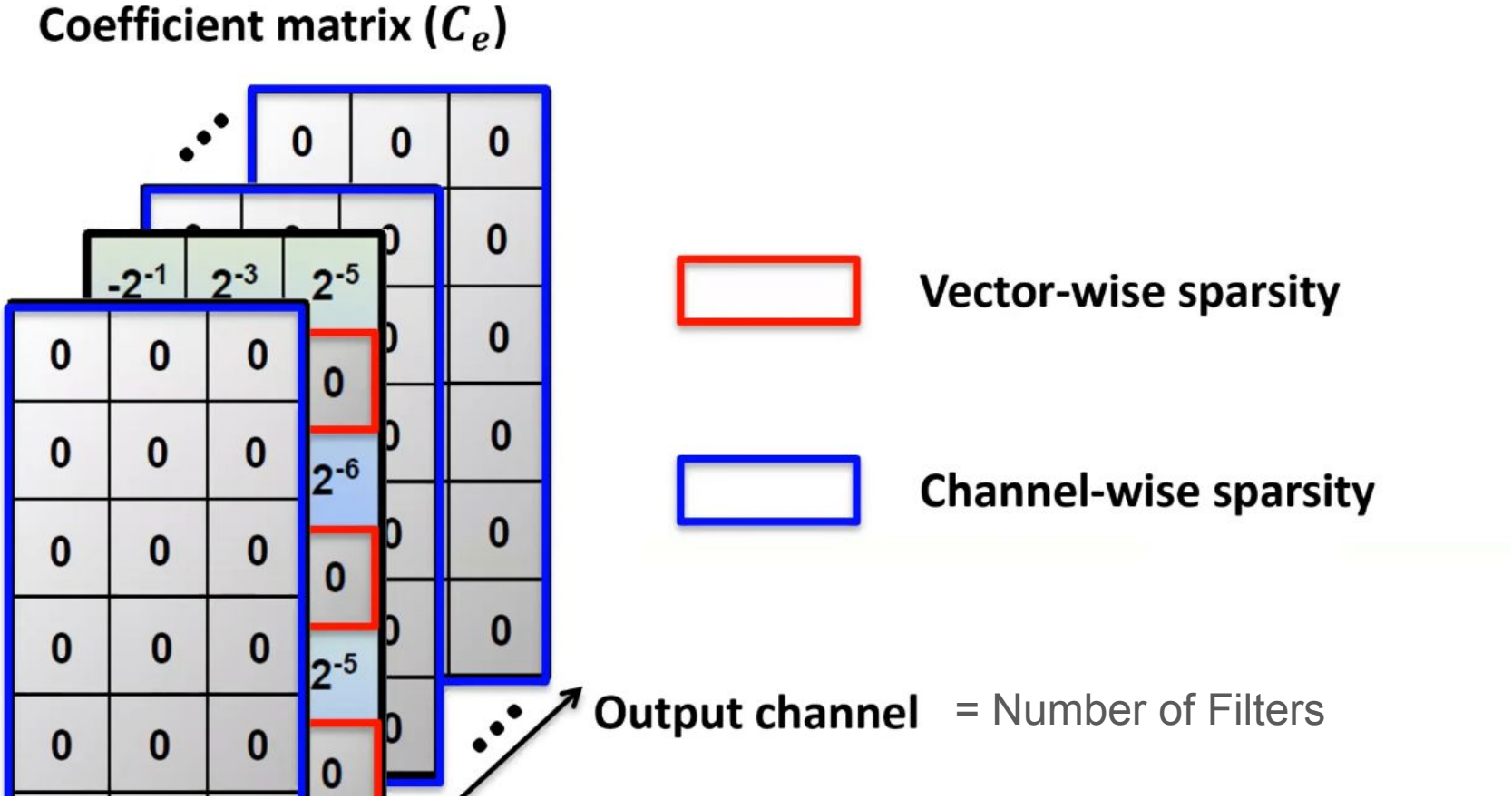
Reference: "SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation"

Coefficient matrix (C_e)

-2^{-1}	2^{-3}	2^{-5}
0	0	0
-2^{-1}	-2^{-3}	2^{-6}
0	0	0
2^{-2}	-2^{-7}	2^{-5}
0	0	0

Vector-wise sparsity

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation



Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Algorithm 1 *SmartExchange* Algorithm.

Sparsify C_e in a channel-wise manner;
Initialize C_e and B; Iteration = 0;
while $\|\delta(C_e)\| \geq tol$ or $iteration < tol\_maximum$:
    Step 1: Quantize C_e to powers of 2;
    Step 2: Fitting B and C_e;
    iteration = iteration + 1;
    Step 3: Sparsity C_e in a vector-wise manner;
Re-quantize C_e and re-fit B.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Steps of the Algorithm

Initialize C_e and B

- In the experiments, C_e is initialized to W and B is initialized to I.

Sparsify C_e in channel wise manner

- Each channel has a scaling factor associated with it in batch normalization
- Prune Channels whose corresponding scaling factor in batch normalization layers is lower than a manually controlled threshold for each layer.
- In practice, only done in the first training epoch once given the observation that pruned channel structure does not change much

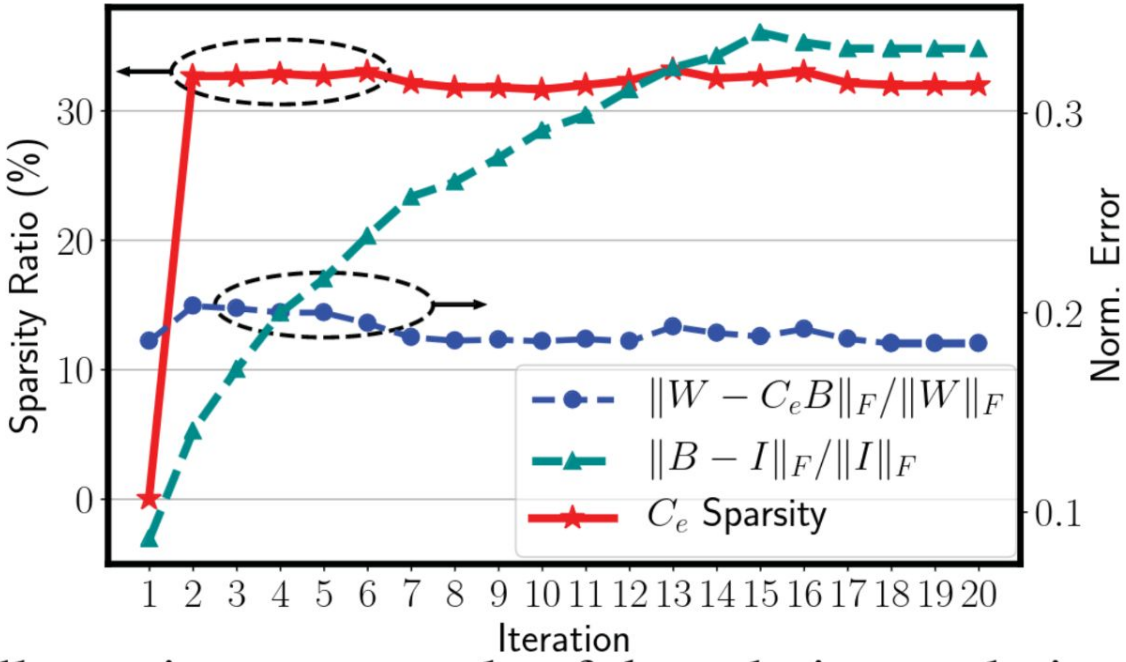


Fig. 9: Illustrating an example of the solution evolution during the *SmartExchange* algorithm training.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Steps of the Algorithm

LOOP STEP

Quantizing C_e

- Normalize each column in C_e such that each column has a unit norm
- Project Non Zero elements of C_e to Ω_p such that each non zero element is rounded to its nearest power-of-2 value
- Define □(C_e) to be the quantization difference(error) of C_e

Fitting B and C_e

- Fit B by solving $\text{argmin}_B \| W - C_e B \|^2_F$
- Fit C_e by solving $\text{argmin}_{Ce} \| W - C_e B \|^2_F$

Sparsifying C_e in vector-wise manner

- Zero out elements in C_e based on the magnitudes to meet vector-wise sparsity constraint S_c, where S_c is manually controlled per layer

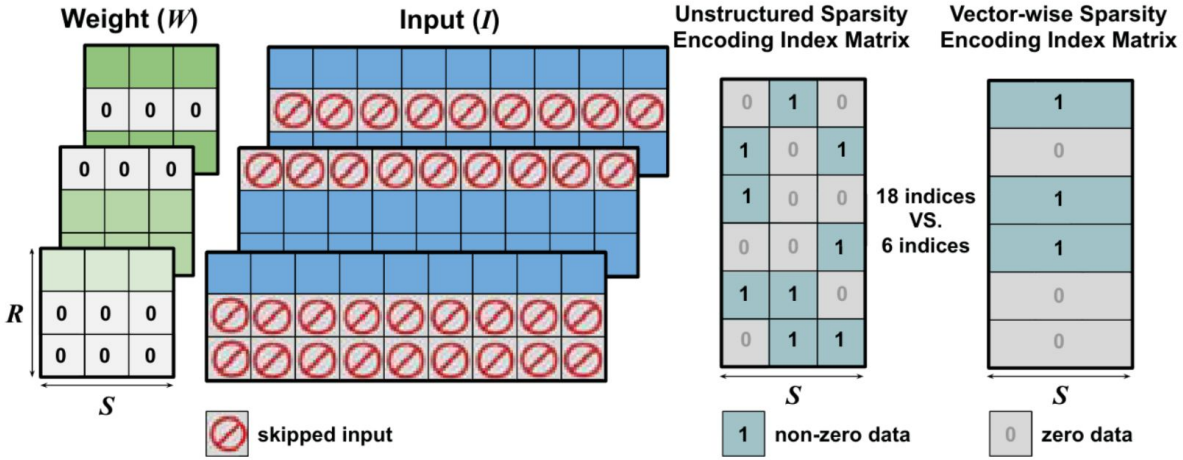
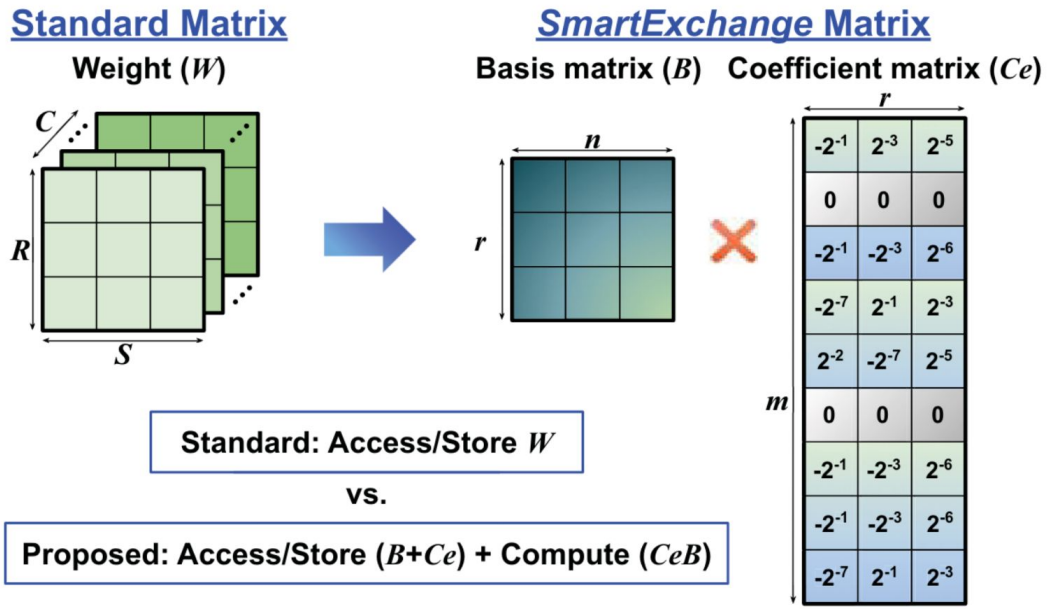


Fig. 3: An illustration of (a) vector-wise skipping the corresponding activations, and (b) the reduced indexing overhead, thanks to the enforced vector-wise weight sparsity of the *SmartExchange* algorithm.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”



- **Memory consumption of W >> Size of $(B + Ce)$**
- **Data movement cost >> computation for large DNNs**

Fig. 1: The proposed *SmartExchange*'s weight representation.

Reference: "SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation"

Applying SmartExchange algorithm to DNNs

The value of “r” is a design knob of Smart Exchange for trading-off achieved compression rate and model accuracy. A smaller value of “r” would compress the model more, but would incur a higher accuracy loss.

In all experiments, we will initialize C_e with W and B with I

Remember that Basis Matrix $B \in R^{r \times n}$, Coefficient Matrix $C_e \in R^{m \times r}$

For CONV layers, let a kernel be of size (C, R, S).

In practice, R = S, thus, n = S. We keep r to be equal to S, since kernels are usually small.

Also, $m = C \times R$.

Applying SmartExchange algorithm to DNNs

FC Layer

- Let the FC layer be $W \in R^{M \times C}$
- Reshape each row of W into a new matrix W_i of size (C/r , r). If C is not divisible by r, then the matrix is padded with zeros
- If C >> r, then the matrices are split into smaller matrices

CONV Layer

- Let W be of size (M, C, R, S)
- Case 1: R = S > 1
  - Reshape the M filters in W into matrices of size (R × C, S).
  - Can be split into smaller matrices if R × C >> S.
- Case 2: R = S = 1
  - W is reshaped into a matrix of size (M × C)
  - This is treated the same as FC layer

Splitting of matrices is done along the first dimension. Reconstruction error might tend to be large with imbalanced dimensions.

Post-processing and Re-training

- On applying SmartExchange Algorithm on VGG19 pretrained on CIFAR-10 dataset with tol = 10^{-10} and maximum iterations = 30, $\theta = 4 \times 10^{-3}$
  - Compression rate is over 10x
  - Accuracy Loss is about 3.21%
  - Time taken to apply SmartExchange was only 30 seconds
- A re-training step can remedy the accuracy problem while maintaining the desired properties of the matrices
  - An empirical approach is taken where:
    - Retraining the model for 1 epoch
    - Reapplying the SmartExchange algorithm to ensure the matrices structure
  - Experiments showed that this process further increases the accuracy while maintaining the structure of the matrices

SmartExchange Accelerator

Motivation

- The paper proposes an algorithm-hardware codesign framework. Let's first understand why can't existing accelerators fully utilise SmartExchange benefits:
  - Lack support for rebuilding weights, because of the decomposition step of the algorithm.
  - Lack dedicated design to fully utilise vector wise sparsity in C_e because of the special structure of C_e.

- SmartExchange accelerator is designed targeting these very pain points.

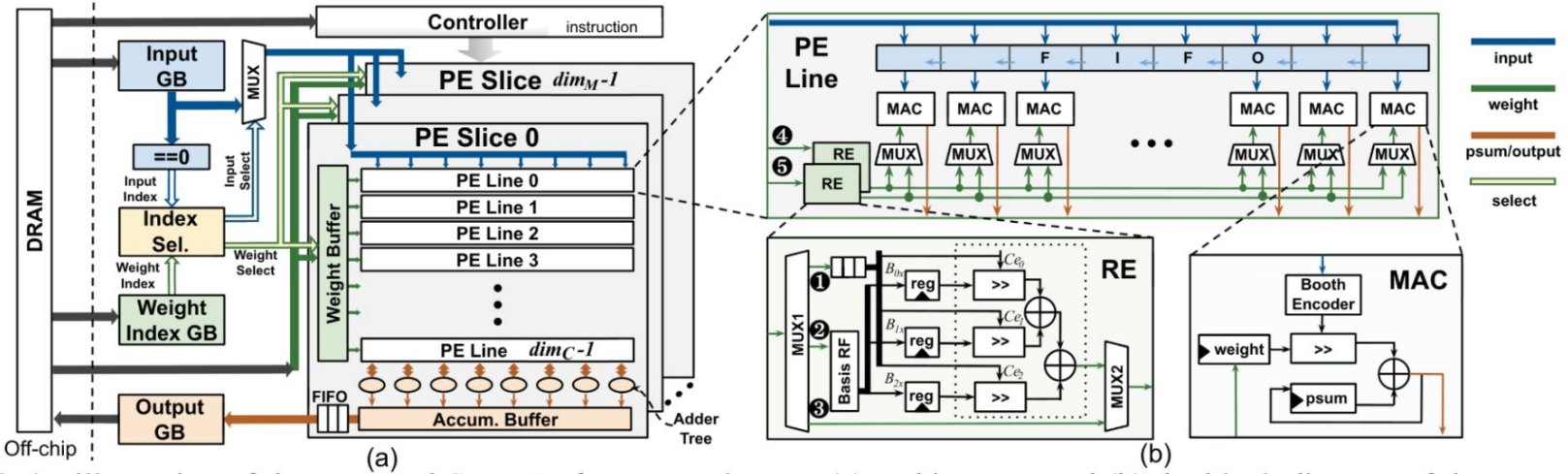


Fig. 5: An illustration of the proposed *SmartExchange* accelerator: (a) architecture, and (b) the block diagram of the processing element (PE) line, each of which includes two rebuilding engines (REs) and eight multiply-and-accumulate (MAC) units.

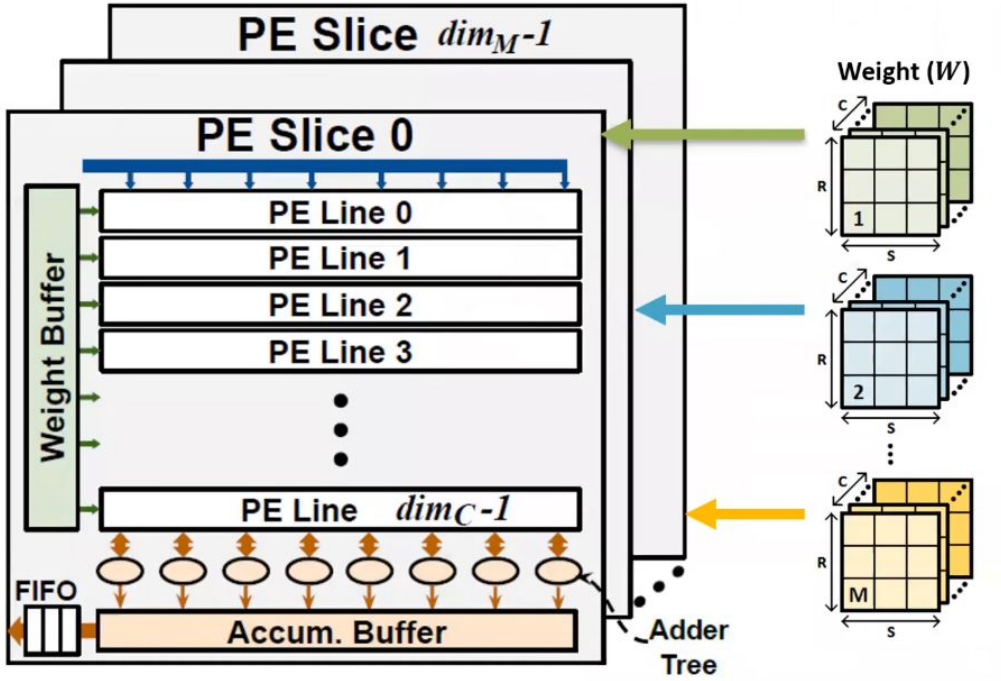
Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Design Principles & Considerations

- The design has four main aims which it is focused on:
  - Hybrid dataflow to maximise data reuses.
  - To minimise the overhead of rebuilding weights.
  - To leverage the structured sparsity in C_e and activations.
  - To support new age compact models with depthwise convolution and squeeze and excite layers.

Hybrid Dataflow

- Operation overview of the 3D PE array:
  - The PE array is divided into dim_m PE slices. Each PE slice corresponds to one output channel.
  - Inside each PE slice there are dim_c PE lines. Each PE line computes the partial sum of one output row.
  - The PE lines within one PE slice compute the partial sum of the same output row.



Each PE slice corresponds to one output channel

Fig. M PE slices, M filter kernels

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

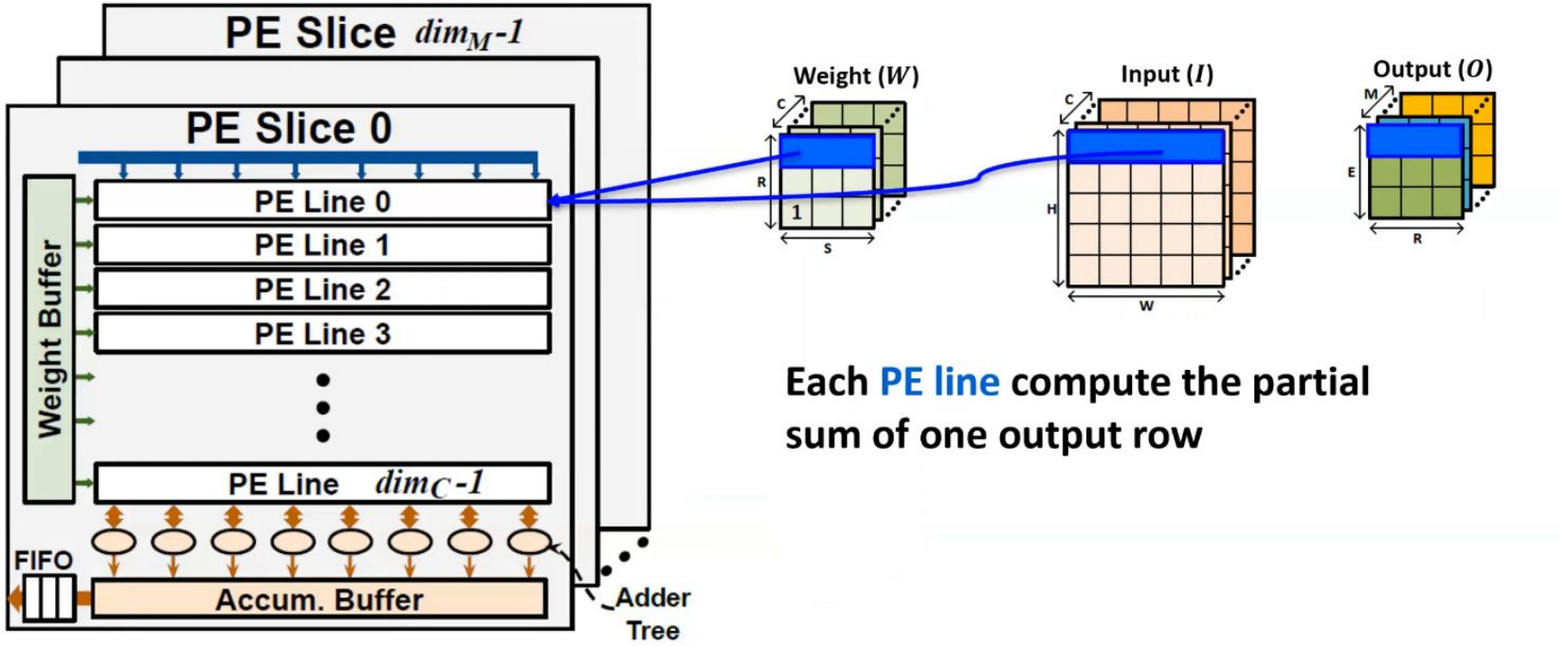


Fig. C PE lines, C input channels

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

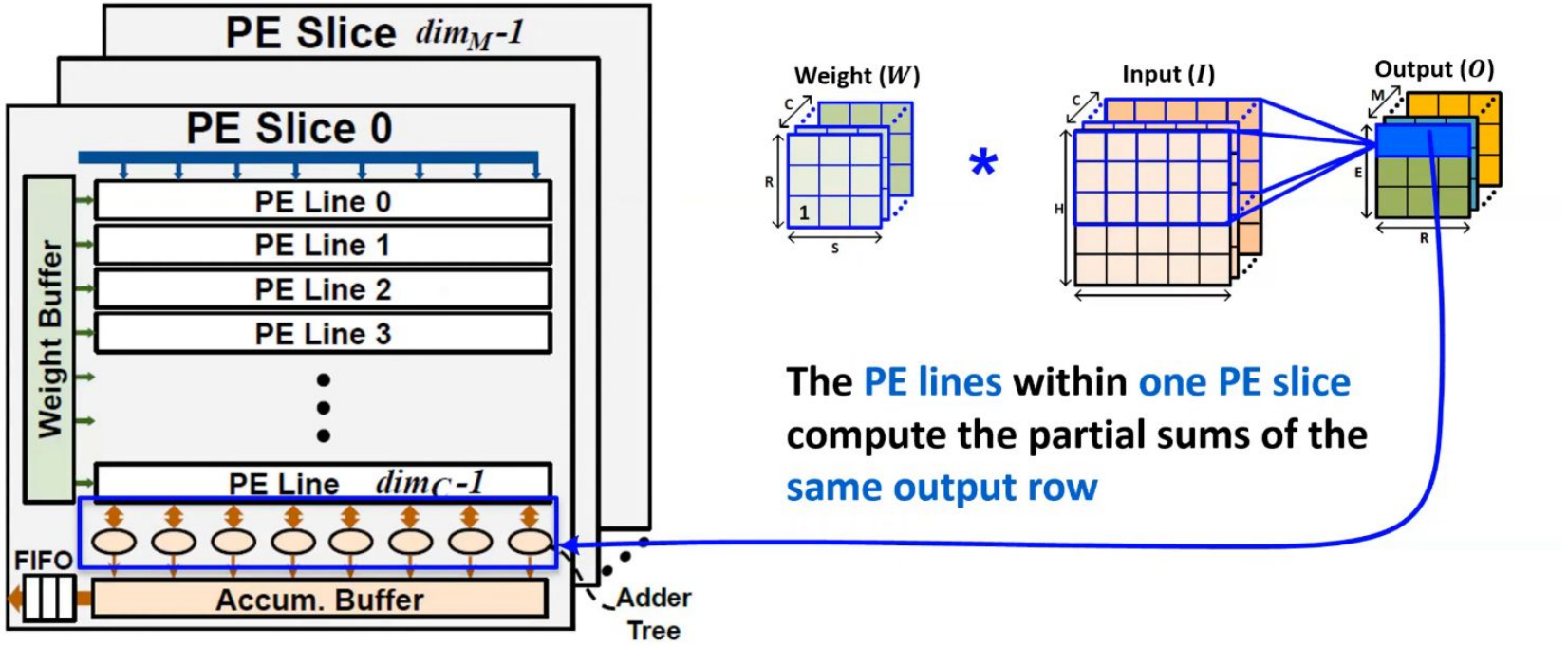


Fig. PE slice corresponds to output row

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Hybrid Dataflow

- Data Flow is row stationary within PE lines; to maximise input, weight and psum reuses.

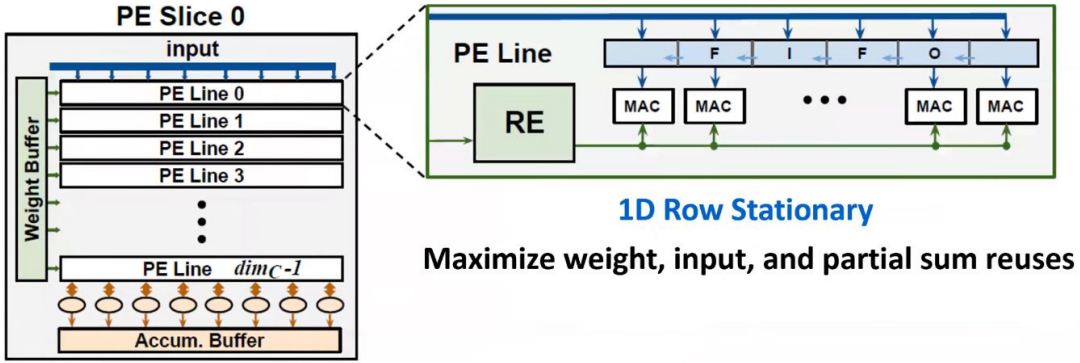


Fig. PE line

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Row Stationary Dataflow in PE line

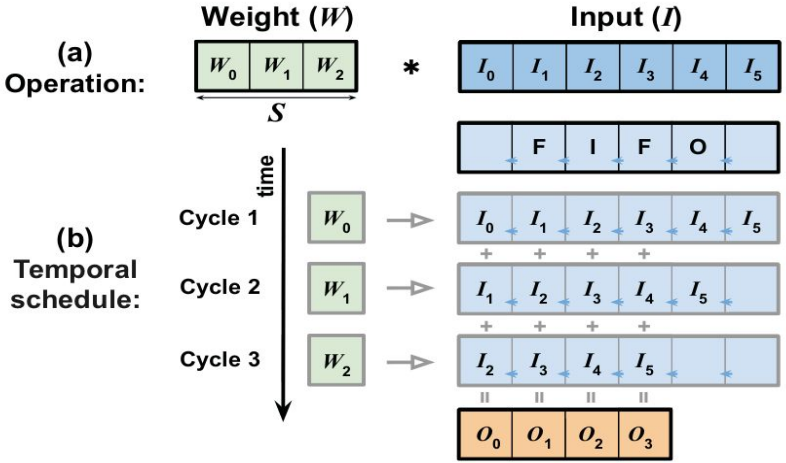


Fig. 6: An illustration of the proposed 1D row stationary along each PE slice (in this particular example, FIFO size is 5, and in general it should be $dim_F + S - 1$): (a) 1D CONV and (b) processing flow of 1D row stationary.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Hybrid Dataflow

- Data Flow is output stationary across PE lines; to maximise psum reuses.

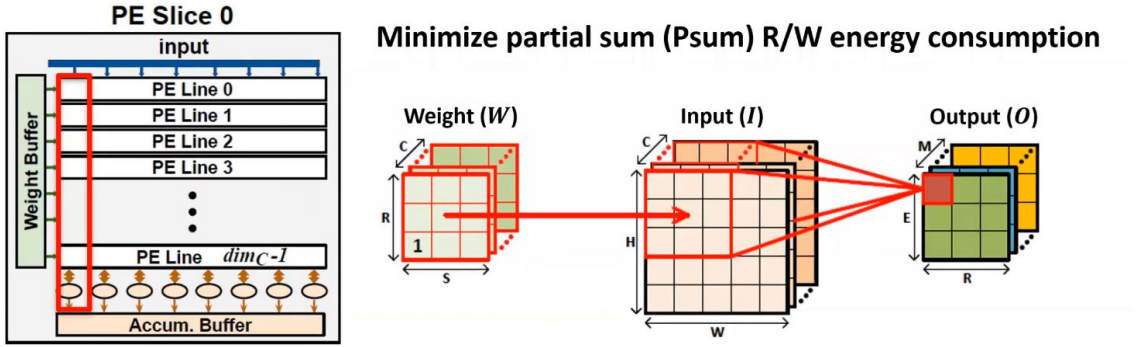


Fig. PE slice

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Minimising Overhead of Rebuilding Weights

- The SmartExchange algorithm decomposes the original weight matrix into a small basis matrix and a large sparse coefficient matrix with power-of-2 values.
- This reduces the memory storage and data movement costs associated with the weights, but also introduces the need to rebuild the weights from the decomposed matrices before performing the convolution operations.
- To minimize the overhead of this rebuilding process, we design the accelerator with the following principles:
  - Store the basis matrix close to the rebuild engine.
  - Place rebuild engine to be close to or within processing elements.
  - Use a weight stationary dataflow for the basis matrix.

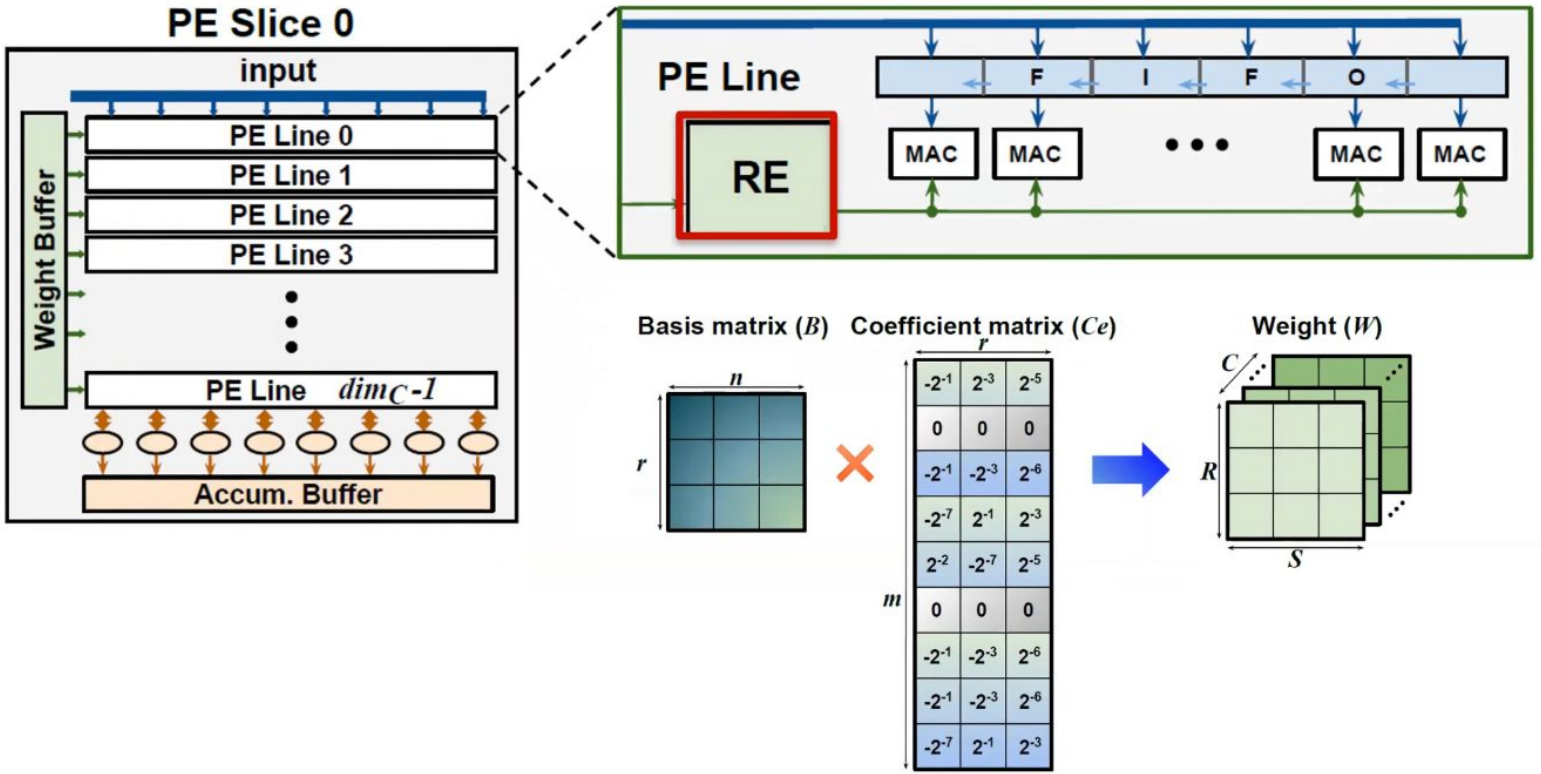


Fig. Visualisation of various elements of SmartExchange Accelerator

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Storing the Basis Matrix

- The rebuild engine is a dedicated hardware unit that restores the weights by multiplying the basis matrix and the corresponding coefficient matrix.
- Since the basis matrix is reused many times for each weight filter, we store it in a local register file within the RE to avoid fetching it repeatedly from the global buffer or the off-chip memory.
- For weight matrix ((C × R)) × S it is decomposed into coefficient matrix of size ((C × R)) × S and a basis matrix of S × S. The basis matrix has ‘C × R’ reuses whereas coefficient matrix has just ‘S’ reuses.
- In turn they set the basis matrix close to the rebuild engine and PEs.

Storing the Rebuild Engine

- The rebuild engine should be placed close to or within the processing elements (PEs).
- The PEs are the main computation units that perform the multiply-and-accumulate (MAC) operations on the input activations and the rebuilt weights.
- To reduce the data movement cost of the rebuilt weights, the REs are placed close to the PEs, or even within the PE lines.
- This way, the rebuilt weights can be directly fed into the PEs without going through the global buffer or the off-chip memory.

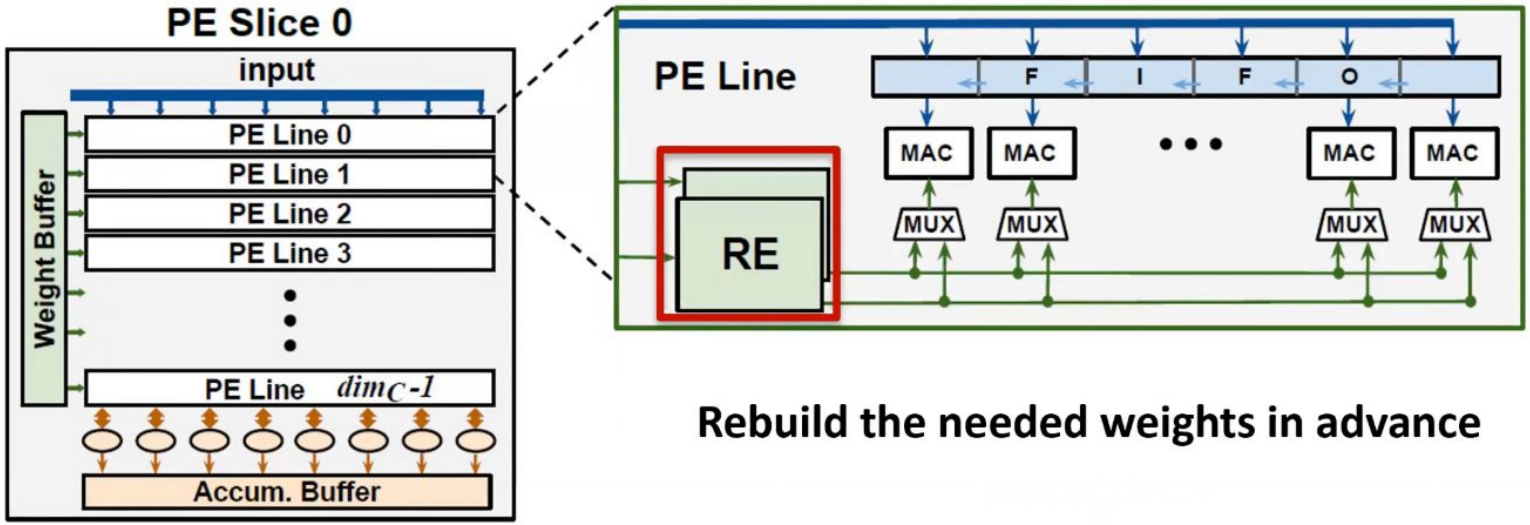


Fig. 2 RE modules operate in a ping pong manner

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Dataflow

- The idea is to use a weight-stationary dataflow for the basis matrix. The dataflow determines how the data is distributed and reused among the PEs.
- Since the basis matrix is the most frequently reused data in the rebuilding process, we adopt a weight-stationary dataflow for it.
- So once being fetched from the memory, it stays in the RE until all the corresponding weights are rebuilt. This avoids unnecessary data movements and maximizes the data reuse efficiency.

Leveraging Structured Sparsity

- The vector wise sparsity enforced by the algorithm on the coefficient matrices offered benefits of:
  - Vector wise skipping both the memory accesses and computations. The SmartExchange algorithm enforces a structured sparsity pattern on the coefficient matrices. This means that many rows in the coefficient matrix are all zeros.

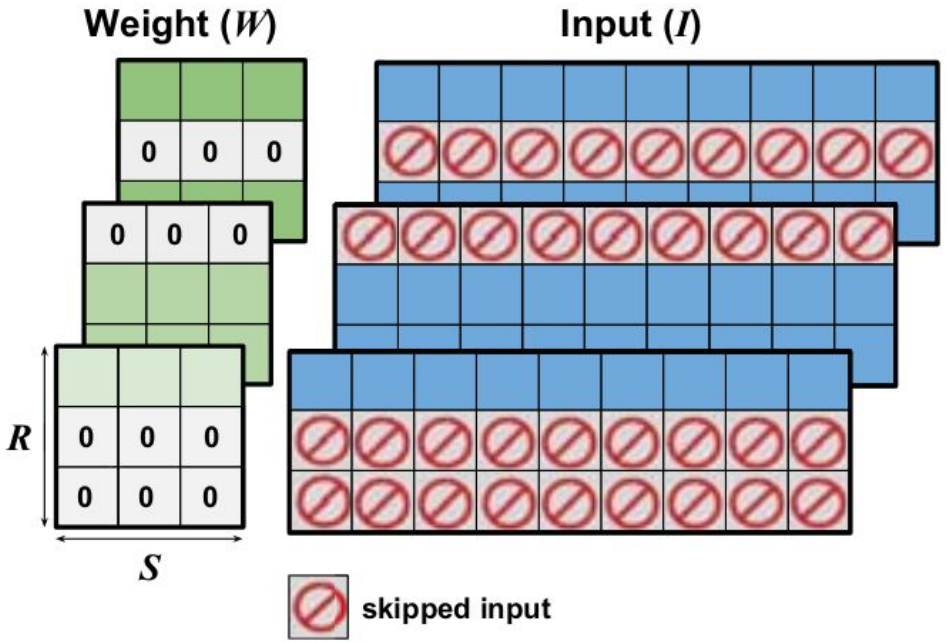


Fig. vector wise skipping of corresponding activations

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Leveraging Structured Sparsity

REDUCED COEFFICIENT MATRIX ENCODING OVERHEAD:

The coefficient matrix in SmartExchange is not only sparse but also quantized to power-of-2 values.

- This means that the non-zero elements in the coefficient matrix can be represented by their binary exponents, which require much fewer bits than the original floating-point values.
- Moreover, the vector-wise sparsity pattern of the coefficient matrix can be encoded by a compact index map, which indicates the locations of the non-zero vectors.
- This way, the encoding overhead of the coefficient matrix is greatly reduced, which further saves the memory storage and access cost.

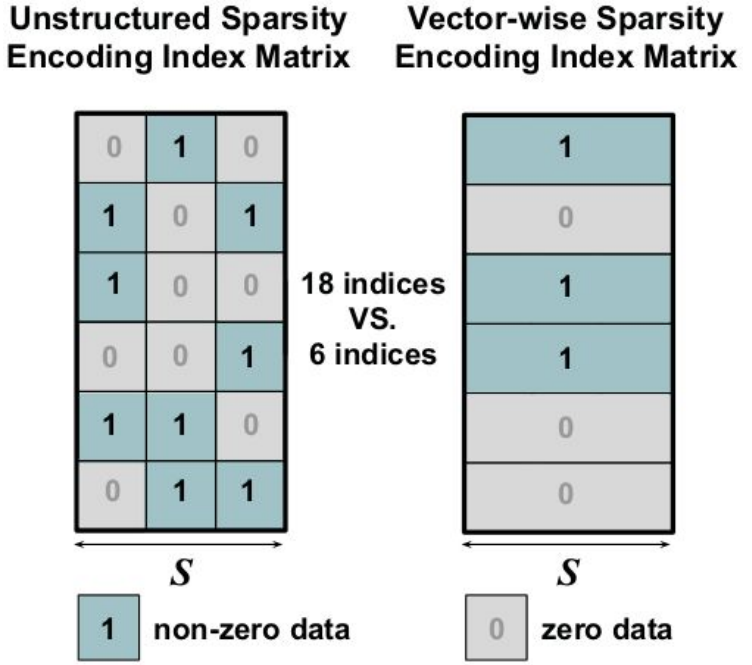


Fig. reduced indexing overhead

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

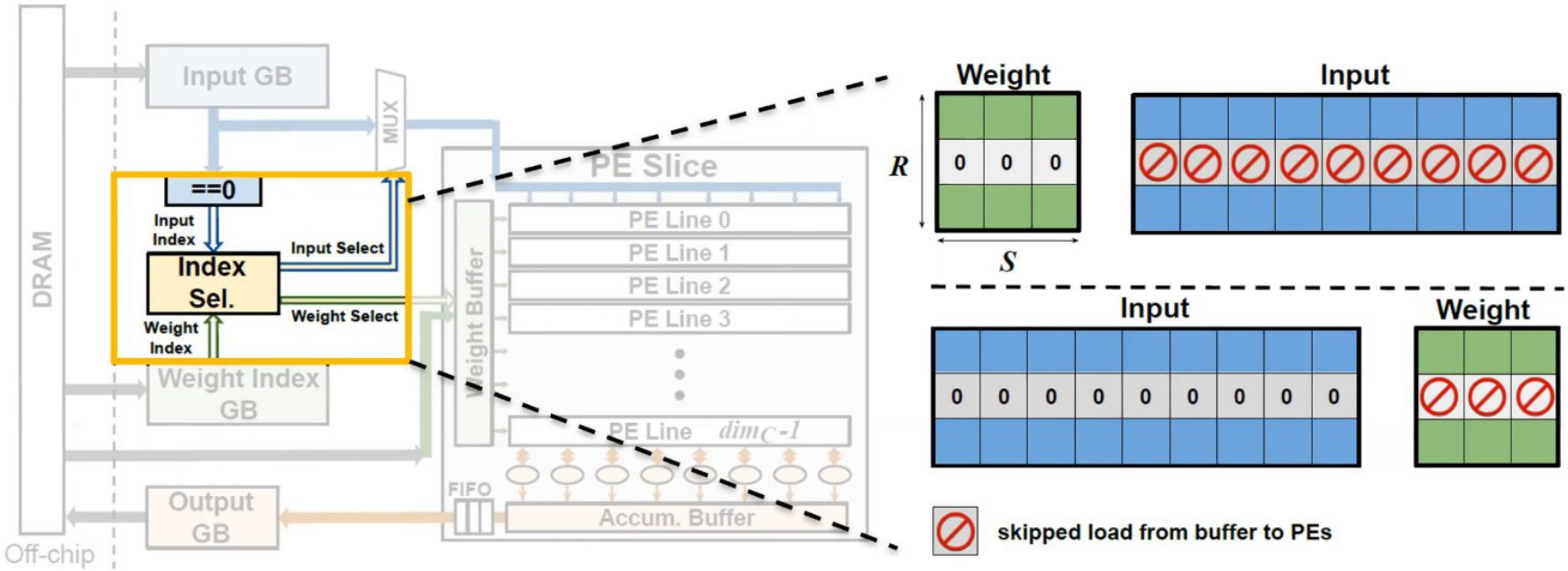


Fig. Index Selector Operation

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Support for Compact Models

- In this course we have learnt about the recently emerging compact models, such as MobileNet and EfficentNet, which often adopt depth-wise convolution and squeeze-and-excite layers, which reduce the data reuse opportunities.
- For example, in a depth-wise convolution layer, the number of convolution channels is only one, which means that the input reuse is much lower than that of a standard convolution layer.
- Similarly, in a squeeze-and-excite layer, there is no weight reuse since it is equivalent to a fully-connected layer. To handle these types of layers efficiently, the following 2 adjustments are made to the accelerator design:
  - **For squeeze-and-excite/fully-connected layers:** Division of each PE line's MAC array of dim_F MACs into multiple clusters with the help of the two REs in one PE line and multiplexing units at the bottom of the MAC array, where each cluster handles computations corresponding to a different output pixel. This allows us to improve the MAC array's utilization and reduce the latency of these layers

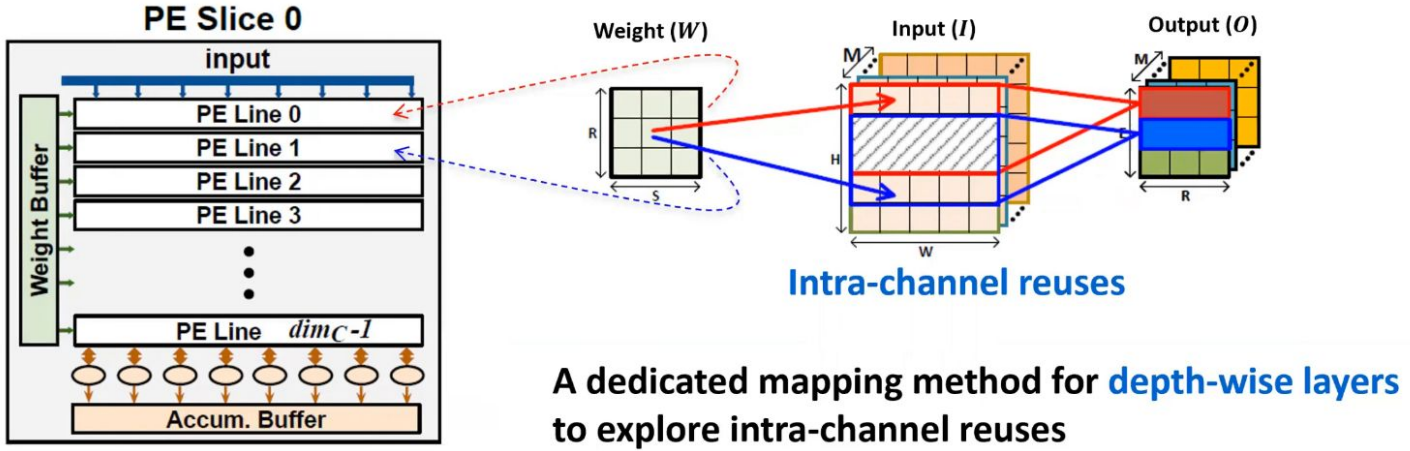


Fig. Adjustment for depth wise convolution layers

For depth-wise convolution layers: The mapping of the R number of 1D convolution operations along the dimension of the weight height to the dim_C PE lines, instead of using them to process different input channels. This way, we can fully utilize the PE slice array and improve the throughput of these layers.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation” ISCA2020 Presentation

Architecture

The accelerator consists of :

1. 3D PE array with dim_M PE slices.

2. Input/ Index/ Output Global Buffers.

3. Buffers associated with index selector for sparsity.

4. DMA for communication with off chip DRAM.

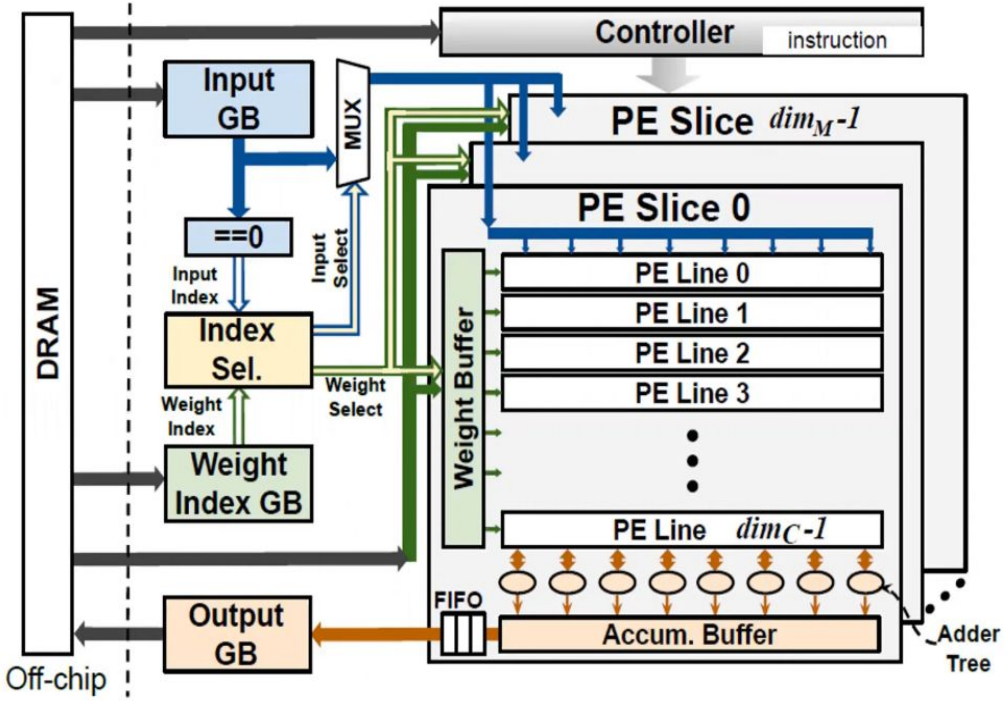


Fig. Architecture of SmartExchange Algorithm

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Architecture aligning with Design Principles

- **Rebuild Engine Design:** It is inserted within the PE lines to reduce the rebuilding overhead.
- **Hybrid Dataflow**: 1D row stationary dataflow is adopted within each PE line for maximising weight and input reuses; while each PE slice uses an output stationary dataflow.
- **Index Selector:** To skip not only computations but data movements associated with the sparse rows of coefficients and activations. It selects the non zero coefficient and activation vector pairs.
- **Data Type Driven Memory Management:** To use matched bandwidths ; so the unit energy cost to access data from the SRAM is reduced.
- **Bit Serial Multiplier Based MAC array:** In each PE line to make use of the activations' bit level sparsity together with a booth encoder.

PE Slices & Dataflow

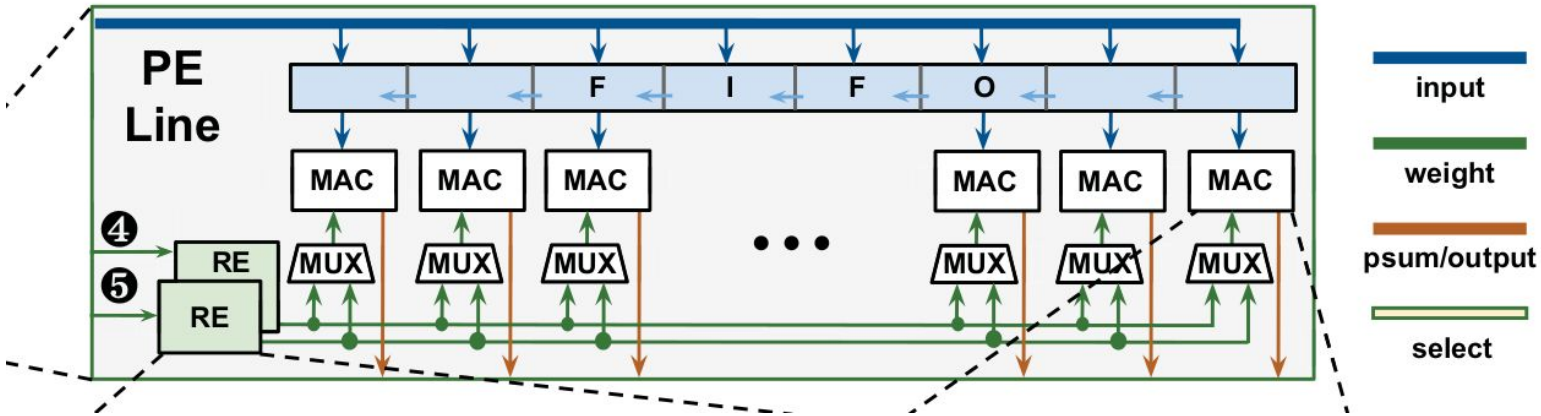


Fig. architecture of a PE line

- **3D PE slice Array:** Enables paralleled processing of computations with the same weight filter using PE slice array of dim_M with the each PE slice having dim_C PE lines.
- **PE line design:** Each PE line has an array of dim_F MACs; one FIFO and two RE units. REs restore the weights in a row wise manner.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Rebuild Engine Design

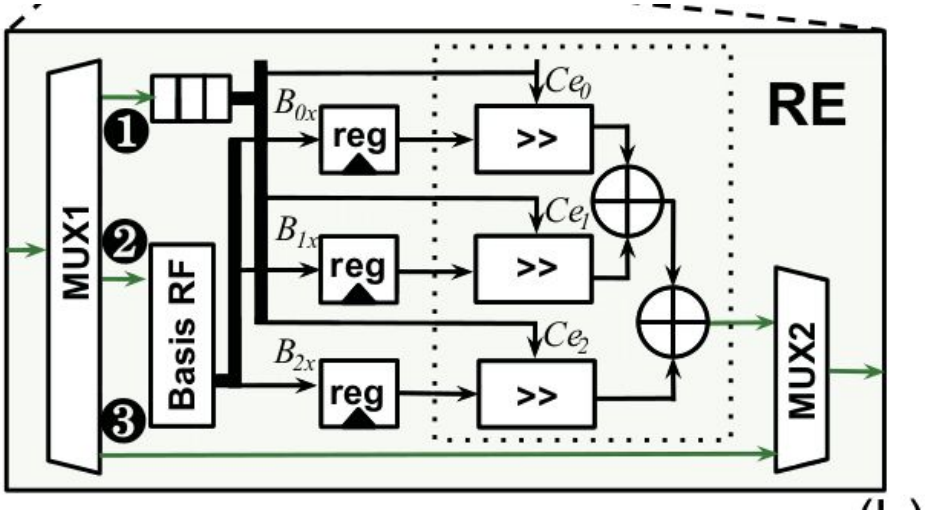


Fig. RE Design

- It has a register file of size S × S to store one basis matrix; it also has a shift and add unit to rebuild weights.
- Time division MUX unit at the left is to fetch: coefficient matrices, basis matrices and original weights. Enables access in a time division manner, reducing bandwidth requirement.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Rebuild Engine Design

- Rebuild Engine accesses the three types of data in a time division manner by taking advantage of the fact these that three types of data don’t need to be fetched simultaneously.
- Specifically the basis matrix is fetched first and stored stationary within the RE until the associated computations are completed.
- The weights are then rebuilt in an RE where each row of the coefficient matrix remains stationary until all the associated computations are completed.
- The third path shown in the figure is for the original weights; it is to handle DNN layers where SmartExchange is not applied.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Handling of Compact Models

- An adjusted dataflow and PE line configuration for improving the utilisation of both the PE slice and the MAC array within each PE line.
- As discussed earlier, for depth wise CONV layers; the no. of CONV channels is only one. So, the dimC PE lines will no longer correspond to input channels.
- So, the R no. of 1D CONV operations are mapped along the weight height of these PE lines.
- For Squeeze and Excite Layers, the MAC array of dimF within each PE line is divided into multiple clusters. With the help of the multiplexing units at the bottom of the MAC array and the two REs to handle computations corresponding to a different output pixel.

Coefficient Matrix Indexing

- We know that the coefficient matrices resulting from the SmartExchange algorithm are highly sparse and readily quantized, which reduces the memory storage and data movement costs but the sparsity patterns of the coefficient matrices are dynamic, which requires encoding the locations of the non-zero elements and their values.
- There are two commonly used methods for encoding the sparse coefficients:
  - A 1-bit direct indexing where the indexes are coded with 1-bit (0 or 1 for zero or non-zero coefficients, respectively)
  - An RLC indexing for the number of zero coefficient rows.
- The SmartExchange algorithm enforces channel-wise sparsity first and then vector-wise sparsity on top of channel-wise sparsity, which leads to clustered zero coefficients in some regions.
- Therefore, a 1-bit direct indexing can be more efficient with those clustered zero coefficients removed, as it reduces the index overhead and the skipping control overhead.
- The index selector in the SmartExchange accelerator uses the 1-bit direct indexing to select the non-zero coefficient and activation vector pairs, and skips the memory accesses and computations of the corresponding zero vectors.

Buffer Design

- We know that DNNs' sparsity helps with skipping corresponding computations/memory accesses but to utilise this sparsity we need a large buffer. The goal here is to form a balance between the skipping convenience and the increased buffer size.
- To enable the processing with sparsity, the row pairs of non zero input activations and coefficients are selected from the InputGB and the IndexGB; using the corresponding coefficient indices.
- These are then sent to the corresponding PE lines with the resulting outputs being collected to the outputGB.

Input GB

- A vanilla design for high utilisation of the PE array would require ($dim_C \times dim_F \times bits_{input}$) × Input activation rows to be fetched to handle the dynamic sparsity patterns.
- SmartExchange accelerator design significantly reduces the required input GB bandwidth.
  - It achieves ≥ 1/S reduction in bandwidth demand.
  - Each PE line handles $dim_C \times dim_F \times bits_{input}$ inputs for every (S + "Booth encoded non-zero activation bits") cycles.
- **Implementation of FIFOs in PE lines using double buffers**: FIFOs (First-In-First-Out) are data storage structures that allow data to be written in and read out in a sequential manner. Implementing FIFOs in PE lines using double buffers means that there are two sets of FIFOs, allowing for simultaneous reading and writing operations.

Input GB

- **Adoption of 1-D row stationary data flow at each PE line:** In a 1-D row stationary data flow, data is streamed through the PE array in a single dimension, typically row-wise, and remains stationary within each row for a defined number of cycles before being replaced by new data. It helps reuse data.
- **Utilization of bit-serial multipliers:** Bit-serial multipliers are hardware components capable of performing multiplication operations on individual bits of data in a serialized manner.

In summary, these features work together to optimize data flow, reduce bandwidth requirements, and enhance the efficiency of input data processing within the PE array, ultimately improving the overall performance of the system.

Weight / Index / Output Buffer

- Exactly like InputGB, weight/index buffers also need to be expanded to handle the sparsity.
- **Handling Basis Matrix Fetching and Computation:** Basis matrices need to be fetched and stored before fetching coefficient matrices and weight reconstruction computation.
  - Computation stalls occur if the next basis matrix is fetched after finishing coefficient fetching and computation for the current basis matrix.
  - To avoid these stalls, leverage two REs in each PE line to operate in a "ping-pong" manner.
- **Output Data Handling:** Utilize a FIFO to buffer outputs from each PE slice.
  - Acts as a cache between the PE array and the output Global Buffer (GB).
  - Aim is to reduce the required output GB bandwidth by capitalizing on the fact that each output is calculated over several clock cycles.

Software-Hardware Interface

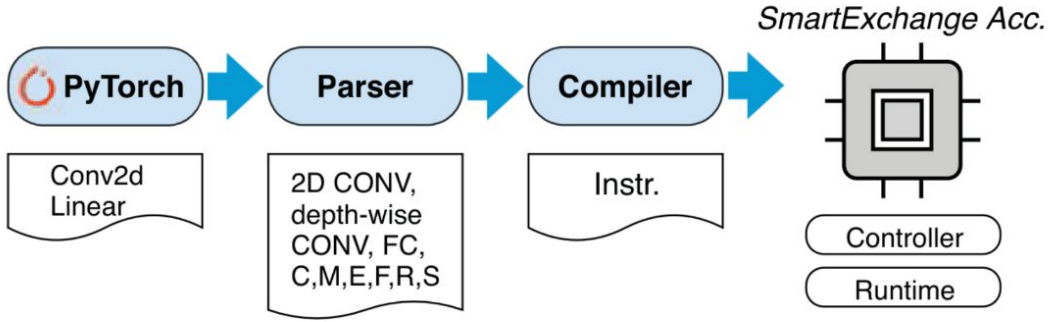


Fig. 7: The software-hardware interface pipeline for the proposed *SmartExchange* accelerator.

- The software-hardware interface plays a crucial role in deploying SmartExchange algorithm-based DNN models onto the SmartExchange accelerator hardware.
- It bridges the gap between deep learning frameworks (e.g., PyTorch) and specialized hardware, facilitating efficient model execution.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Software-Hardware Interface

- Pre-trained DNN models undergo processing through the DNN Parser and Compiler before being loaded into the accelerator hardware.
- The DNN Parser extracts essential **model parameters**, while the Compiler uses the extracted parameters to determine **data flow** and generates sparse index and instructions for hardware configuration.
- Compiled instructions are loaded into the accelerator's controller, controlling processing within the hardware.
- This seamless integration of software and hardware enables efficient execution of DNN models on specialized hardware, maximizing performance and resource utilization.

Experiments and Results

Testing Methodology for the Algorithm

Evaluate the model on:

- Two standard DNNs on ImageNet dataset
  - VGG11
  - ResNet50
- Two standard DNNs on CIFAR-10 dataset
  - VGG19
  - ResNet164
- Two compact DNNs on ImageNet dataset
  - MobileNetV2 (MBV2)
  - EfficientNEt-B0 (Eff-B0)
- DeepLabv3 on CamVid dataset
- Two MLP models on MNIST dataset

Compare different compression techniques such as:

- Two structured pruning techniques
  - Network Slimming
  - Thinet
- Four Quantization techniques
  - Scalable 8-bit (S8)
  - FP8
  - WAGEUBN
  - DoReFa
  - Power of 2 quantization

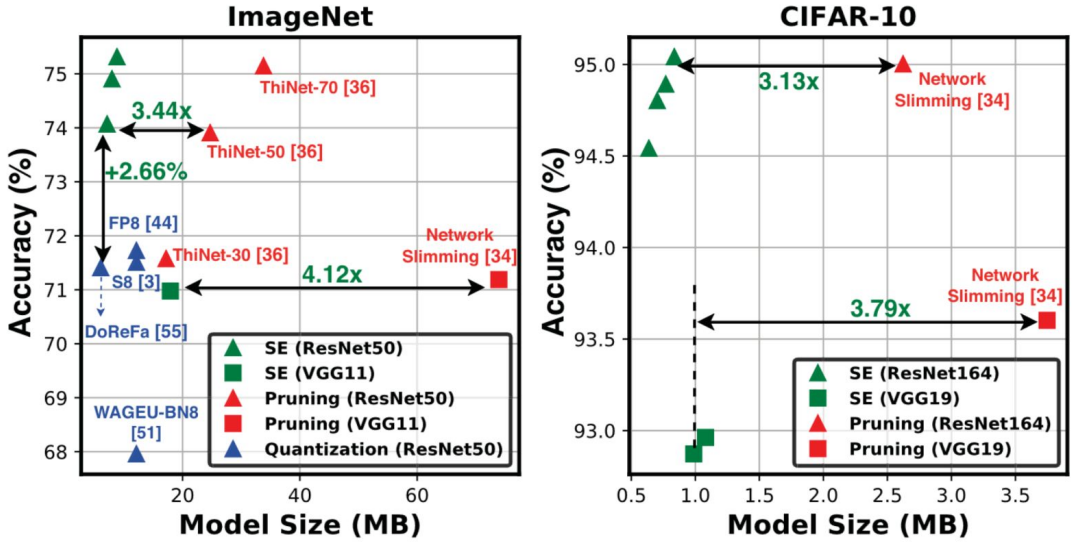


Fig. 8: Accuracy vs. model size comparison of the *SmartExchange* algorithm (SE) and state-of-the-art compression techniques on the (a) ImageNet and (b) CIFAR-10 datasets, where different colors differentiate the SE and baseline techniques.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

- The baseline models use 32-bit floating-point representations for the weights and input/output activations, so as to benchmark with the best achievable accuracy results in the literature.
- The proposed SmartExchange models use 8-bit fixed-point representations for the input/output activations; 4-bit representations for the coefficient and 8-bit representations for basis matrices.

TABLE II: The result summary of the proposed *SmartExchange* with re-training on: 1) VGG11 and ResNet50 using the ImageNet dataset [11]; 2) VGG19 and ResNet164 using the CIFAR-10 dataset [26]; and 3) MLP-1 [40] and MLP-2 [56] using the MNIST dataset.

Model	Top-1 (%)	Top-5 (%)	CR (×)	Param. (MB)	B (MB)	C_e (MB)	Spar. (%)
VGG11	71.18%	90.08%	-	845.75	-	-	-
VGG11_{SE}	70.97%	89.88%	47.04	17.98	1.67	14.77	86.00
ResNet50	76.13%	92.86%	-	102.40	-	-	-
$\text{ResNet50}_{\text{SE}}$	75.31%	92.33%	11.53	8.88	1.40	6.77	45.00
$\text{ResNet50}_{\text{SE}}$	74.06%	91.53%	14.24	7.19	1.40	5.08	58.60
VGG19	93.66%	-	-	80.13	-	-	-
VGG19_{SE}	92.96%	-	74.19	1.08	0.27	0.74	92.80
VGG19_{SE}	92.87%	-	80.94	0.99	0.27	0.65	93.70
ResNet164	94.58%	-	-	6.75	-	-	-
$\text{ResNet164}_{\text{SE}}$	95.04%	-	8.04	0.84	0.25	0.53	37.60
$\text{ResNet164}_{\text{SE}}$	94.54%	-	10.55	0.64	0.25	0.33	61.00
MLP-1	98.47%	-	-	14.125	-	-	-
MLP-1_{SE}	97.32%	-	130	0.11	0.01	0.10	82.34
MLP-2	98.50%	-	-	1.07	-	-	-
MLP-2_{SE}	98.11%	-	45.03	0.024	0.00	0.024	93.33

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

TABLE III: Evaluation of *SmartExchange* with re-training on two compact models with the ImageNet dataset [11].

Model	Top-1 (%)	Top-5 (%)	CR (×)	Param. (MB)	B (MB)	C_e (MB)	Spar. (%)
MBV2	72.19%	90.53%	-	13.92	-	-	-
MBV2_{SE}	70.16%	89.54%	6.57	2.12	0.37	1.74	0.00
Eff-B0	76.30%	93.50%	-	20.40	-	-	-
$\text{Eff-B0}_{\text{SE}}$	73.80%	91.79%	6.67	3.06	0.51	2.55	0.00

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Testing Methodology for the Accelerator

Implemented custom cycle-accurate simulators to test the accelerators

Gate-level netlist and SRAM were generated on a commercial 28nm technology using

- Synopsys Design Compiler
- Arm Artisan Memory Compiler

Energy is calculated using state-of-the-art tool PrimeTime PX

TABLE IV: The design considerations of the baseline and our accelerators.

Accelerator	Design Considerations
DianNao [6]	Dense models
Cambricon-X [54]	Unstructure weight sparsity
SCNN [39]	Unstructure weight sparsity + Activation sparsity
Bit-pragmatic [1]	Bit-level activation sparsity
Ours	Vector-wise weight sparsity + Bit-level and vector-wise activation sparsity

TABLE V: A summary of the computation and storage resources in the *SmartExchange* and baseline accelerators.

***SmartExchange* and Bit-pragmatic [1]**			
dim_M	64	**Input GB**	16KB×32Banks
dim_C	16	**Output GB**	2KB×2Banks
dim_F	8	**Weight Buff./slice**	2KB×2Banks
# of bit-serial mul.	8K	**Precision**	8 bits
DianNao [6], SCNN [39], and Cambricon-X [54]			
The same total on-chip SRAM storage as *SmartExchange*			
# of 8-bit mul.	1K	**Precision**	8 bits

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

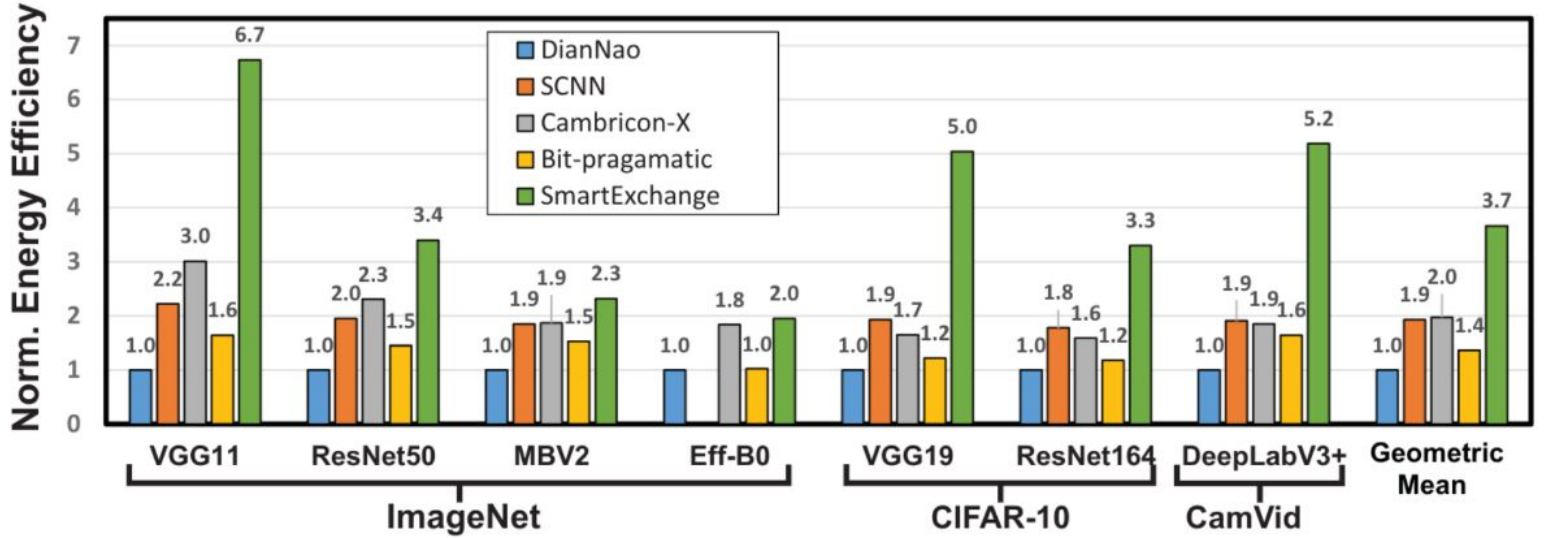


Fig. 10: The normalized energy efficiency (over DianNao) achieved by the *SmartExchange* accelerator over the four state-of-the-art baseline accelerators on seven DNN models and three datasets.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

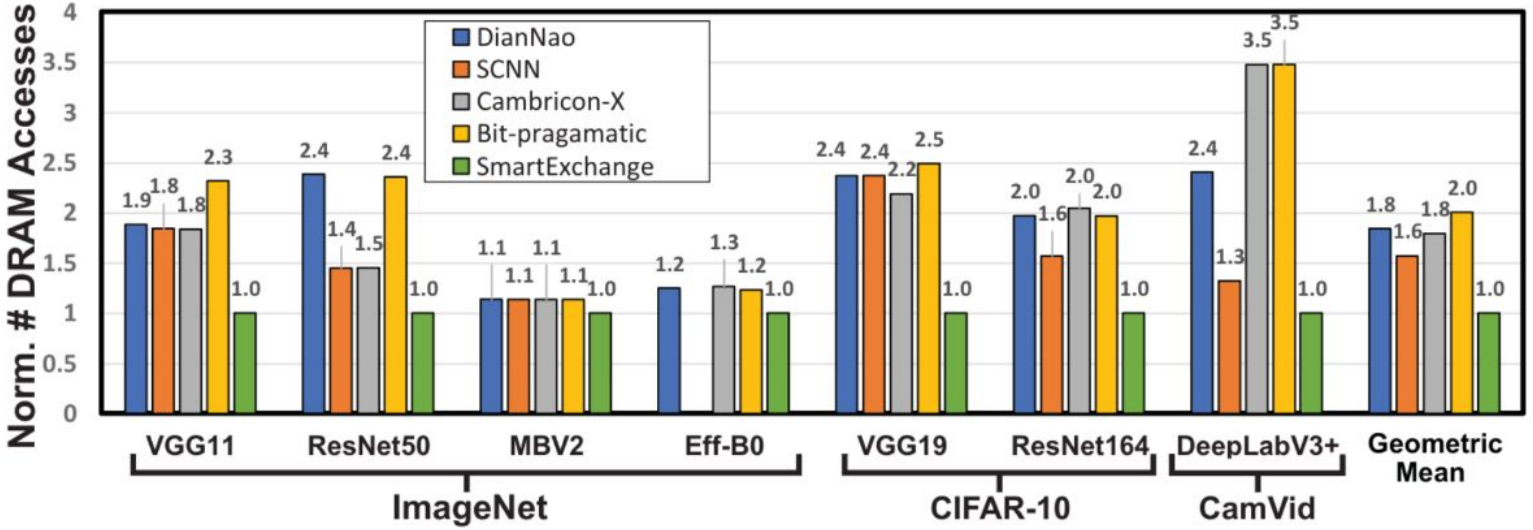


Fig. 11: The normalized number of DRAM accesses (over the *SmartExchange* accelerator) of the *SmartExchange* and four state-of-the-art baseline accelerators on seven DNN models and three datasets.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

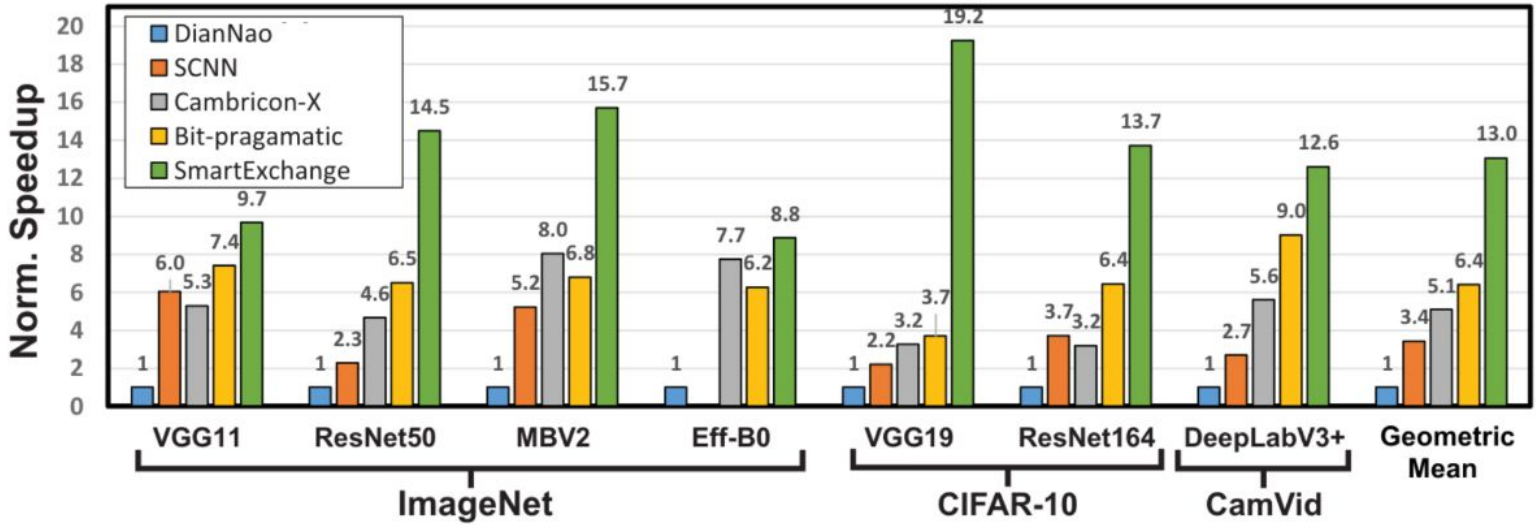


Fig. 12: The normalized speedup (over DianNao) achieved by the *SmartExchange* accelerator over the four state-of-the-art baseline accelerators on seven DNN models and three datasets.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

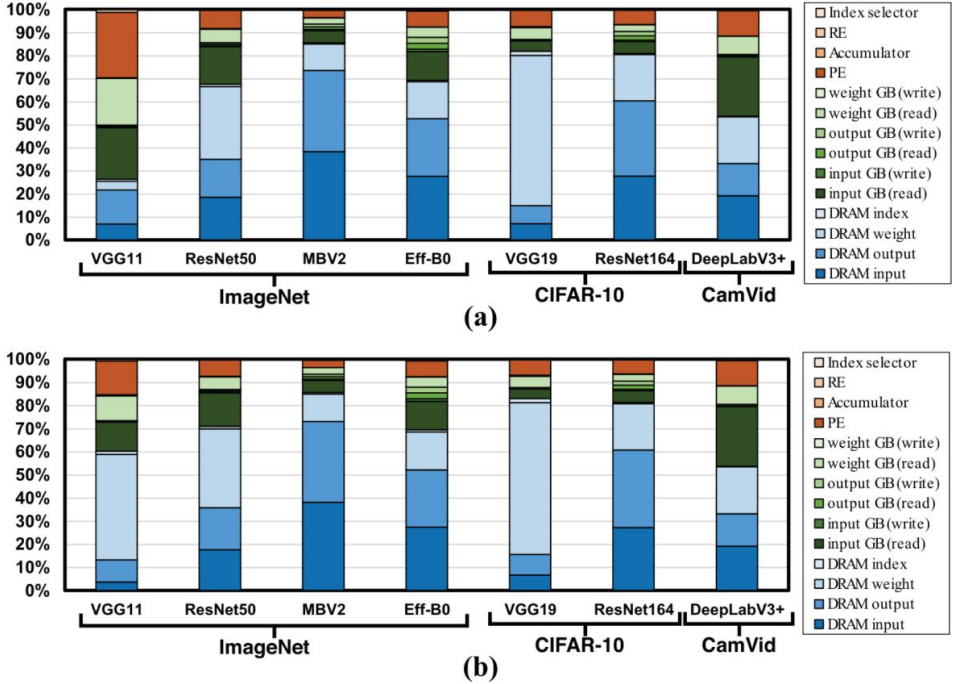
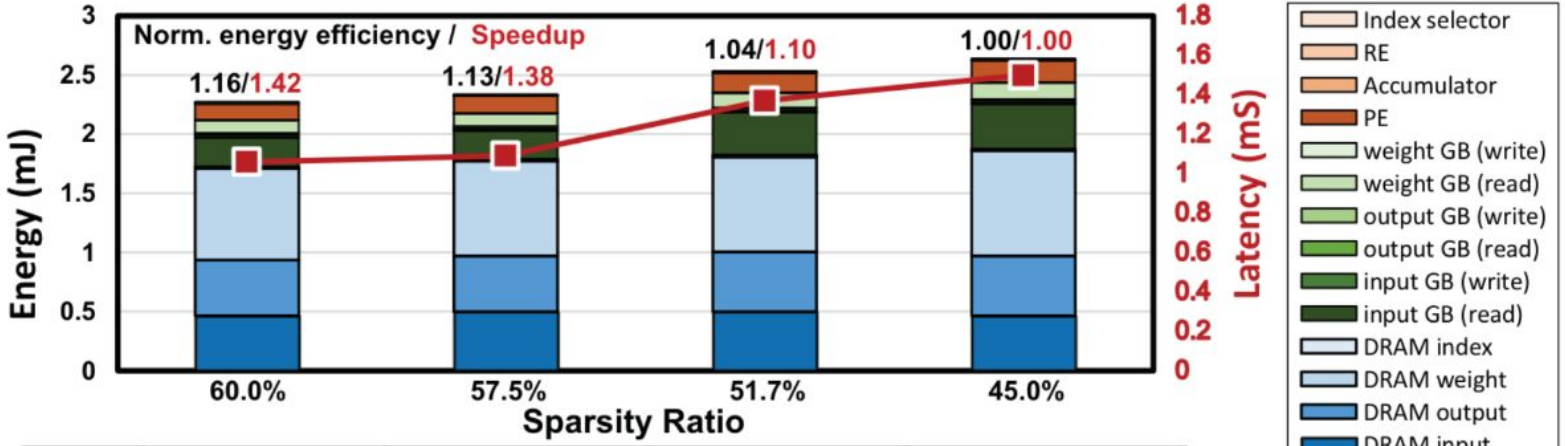


Fig. 13: The energy breakdown of the *SmartExchange* accelerator when running the (a) CONV and squeeze-and-excite layers and (b) CONV, squeeze-and-excite, and FC layers (all types of layers) of seven DNN models on three datasets.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”



Param.	7.68 MB	7.99 MB	8.03 MB	8.88 MB
Top-5	91.83%	91.77%	92.20%	92.33%

Fig. 14: The energy breakdown and latency of the *SmartExchange* accelerator when running ResNet50 with four different sparsity ratios.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

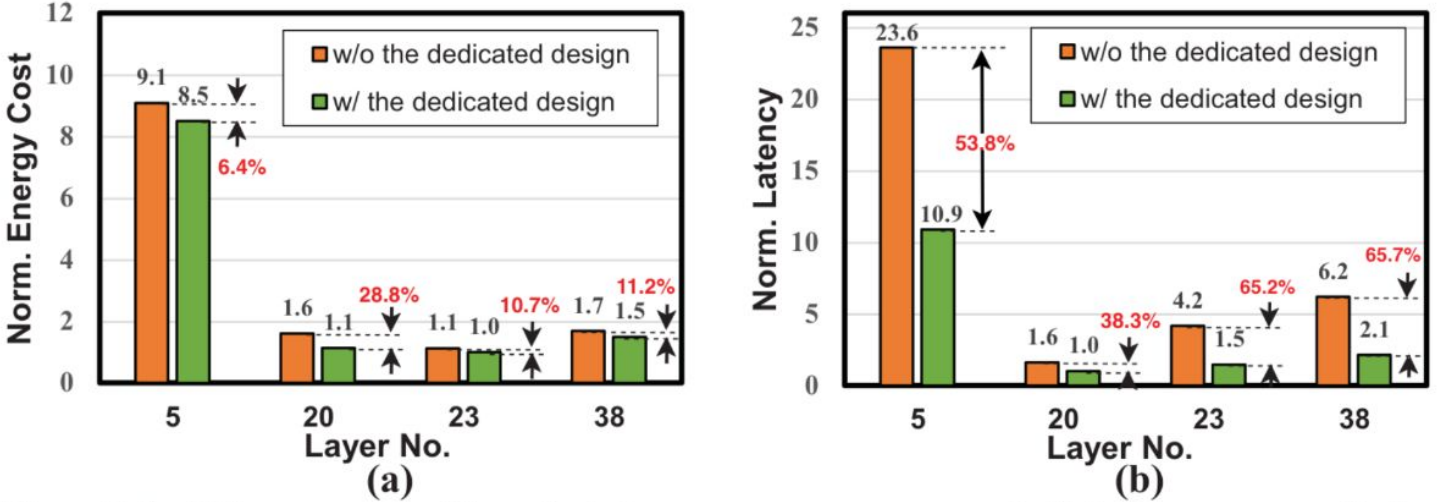


Fig. 15: The normalized (a) energy cost and (b) latency of the depth-wise CONV layers w/ and w/o the proposed dedicated design for compact models, when processing MobileNetV2 on the ImageNet dataset.

Reference: “SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation”

Future Works

- “SmartDeal: Remodeling Deep Network Weights for Efficient Inference and Training”
- Published in IEEE Transactions on Neural Networks and Learning Systems 2023
- Except for one of the authors, all authors of the SmartExchange paper also worked on the SmartDeal paper.
- “Deals” with energy efficient training of a DNN.

Conclusion

- The SmartExchange Algorithm unifies three compression techniques: Pruning, Decomposition and Quantization
- The algorithm aggressively reduces latency and energy costs at inference with a < 2% accuracy loss
- On the hardware level, SmartExchange can boost the energy efficiency by up to 6.7× and reduce the latency by up to 19.2× over four state-of-the-art DNN accelerators.
- Link to the ISCA 2020 Presentation: Link