ComPEND:
Computation Pruning through Early Negative Detection for ReLU in a Deep Neural Network Accelerator

June 13, 2018

Dongwoo Lee, Sungbum Kang, Kiyoung Choi

Neural Processing Research Center (NPRC)

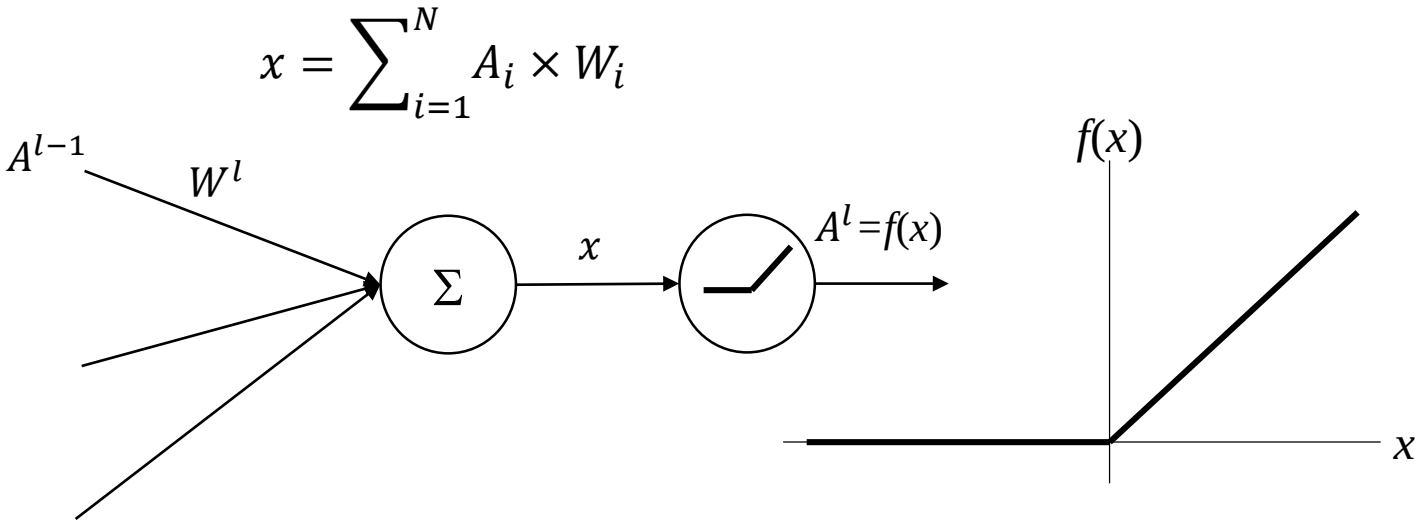
Seoul National University

Outline

- Motivation
- Early Negative Detection (END)
- Computation Pruning thru END (ComPEND)
- Evaluation
- Conclusion

Motivation

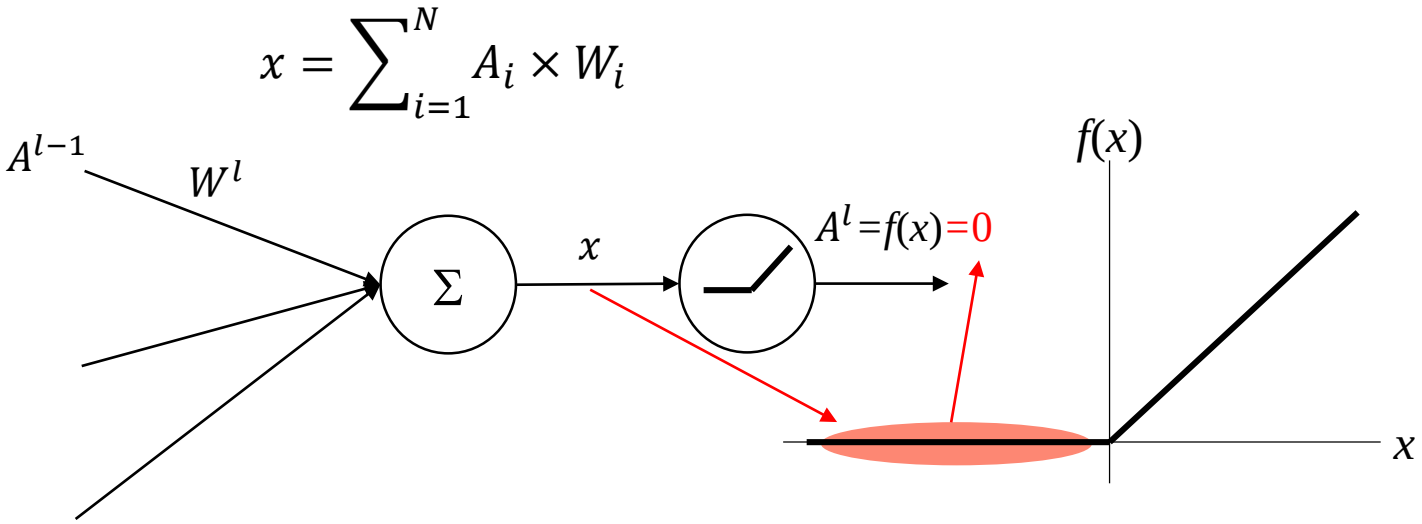
- Perceptron



- Rectified linear unit (ReLU, [$f(x)$ = max(0,x)]) is widely used as an activation function for DNN.

Motivation

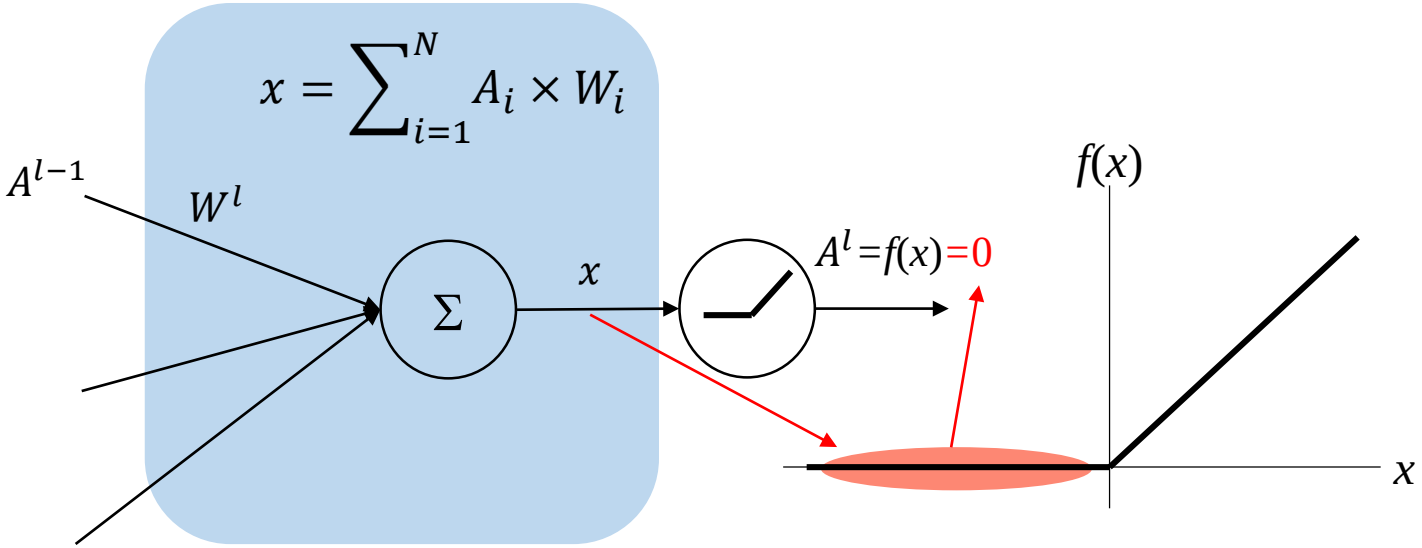
- Perceptron



- Rectified linear unit (ReLU, [$f(x)$ = max(0,x)]) is widely used as an activation function for DNN.

Motivation

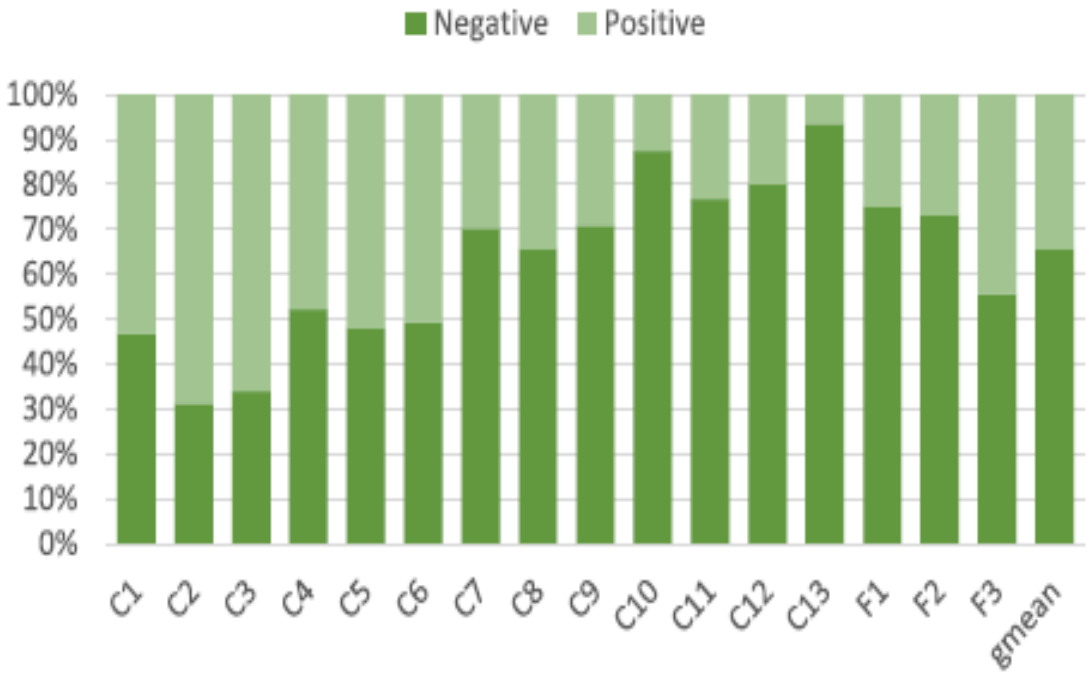
- Perceptron



- Rectified linear unit (ReLU, [$f(x)$ = max(0,x)]) is widely used as an activation function for DNN.
- If we know *a priori* that $x \leq 0$, we can skip unnecessary computations and simply set ReLU output to zero.

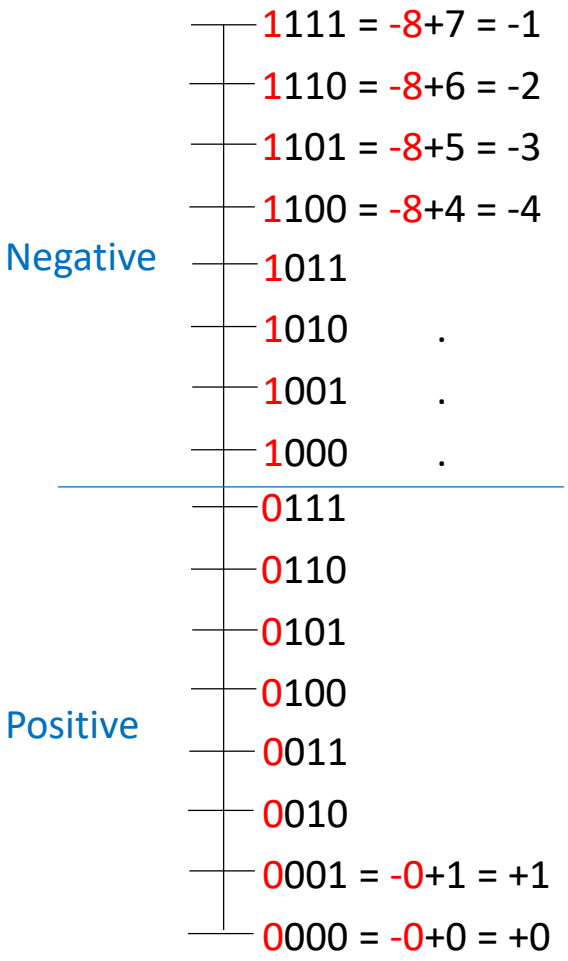
Motivation

- Distribution of negative inputs to ReLU functions in VGG-16
  - More than 60%



Early Negative Detection (END)

- Two's complement number representation (4 bits)

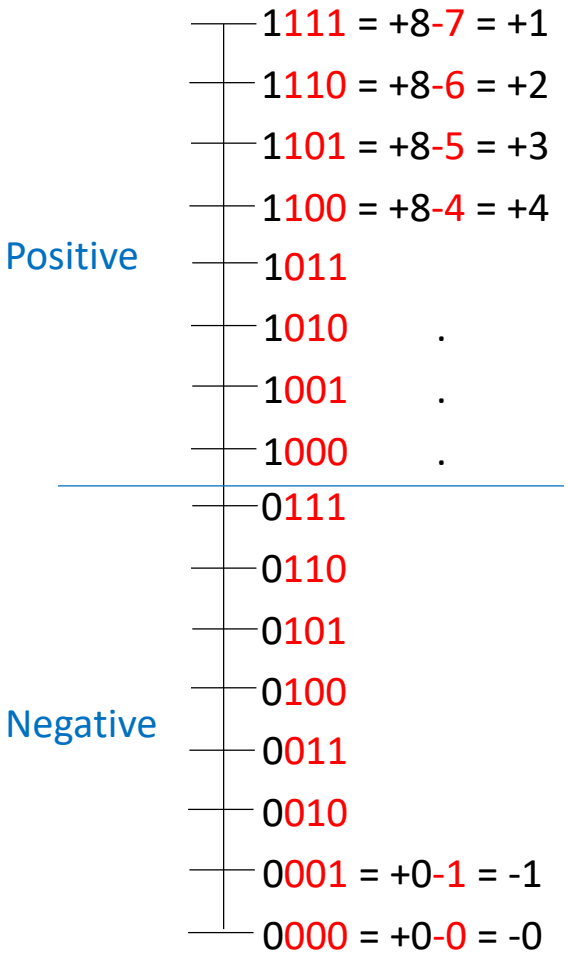


For a B-bit number W : ($w_{B-1}\ w_{B-2}\ w_{B-3} \ldots w_1\ w_0$)

$$W = w_{B-1} \times (-2^{B-1}) + \sum_{k=0}^{B-2} w_k \times (+2^k)$$

Early Negative Detection (END)

- Inverted two's complement number representation (4 bits)

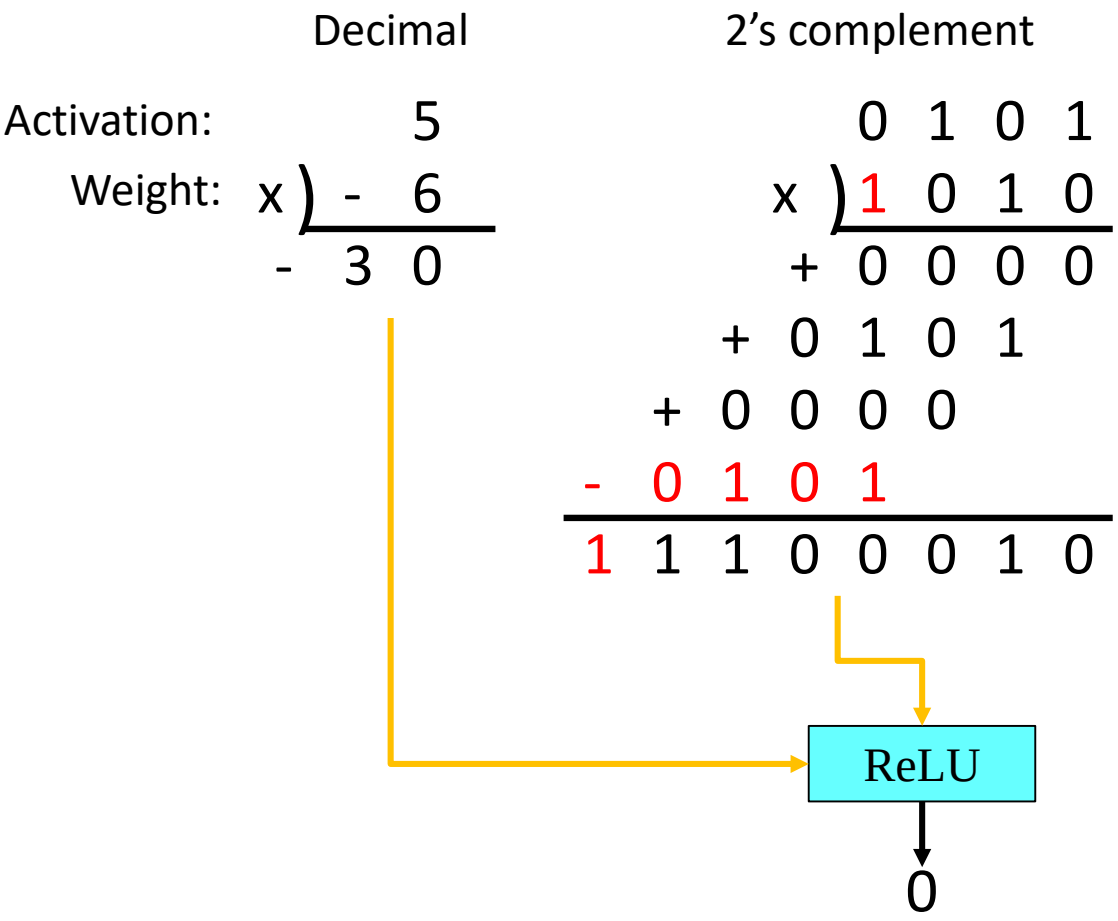


For a B-bit number W : ($w_{B-1}\ w_{B-2}\ w_{B-3} \ldots w_1\ w_0$)

$$W = w_{B-1} \times (+2^{B-1}) + \sum_{k=0}^{B-2} w_k \times (-2^k)$$

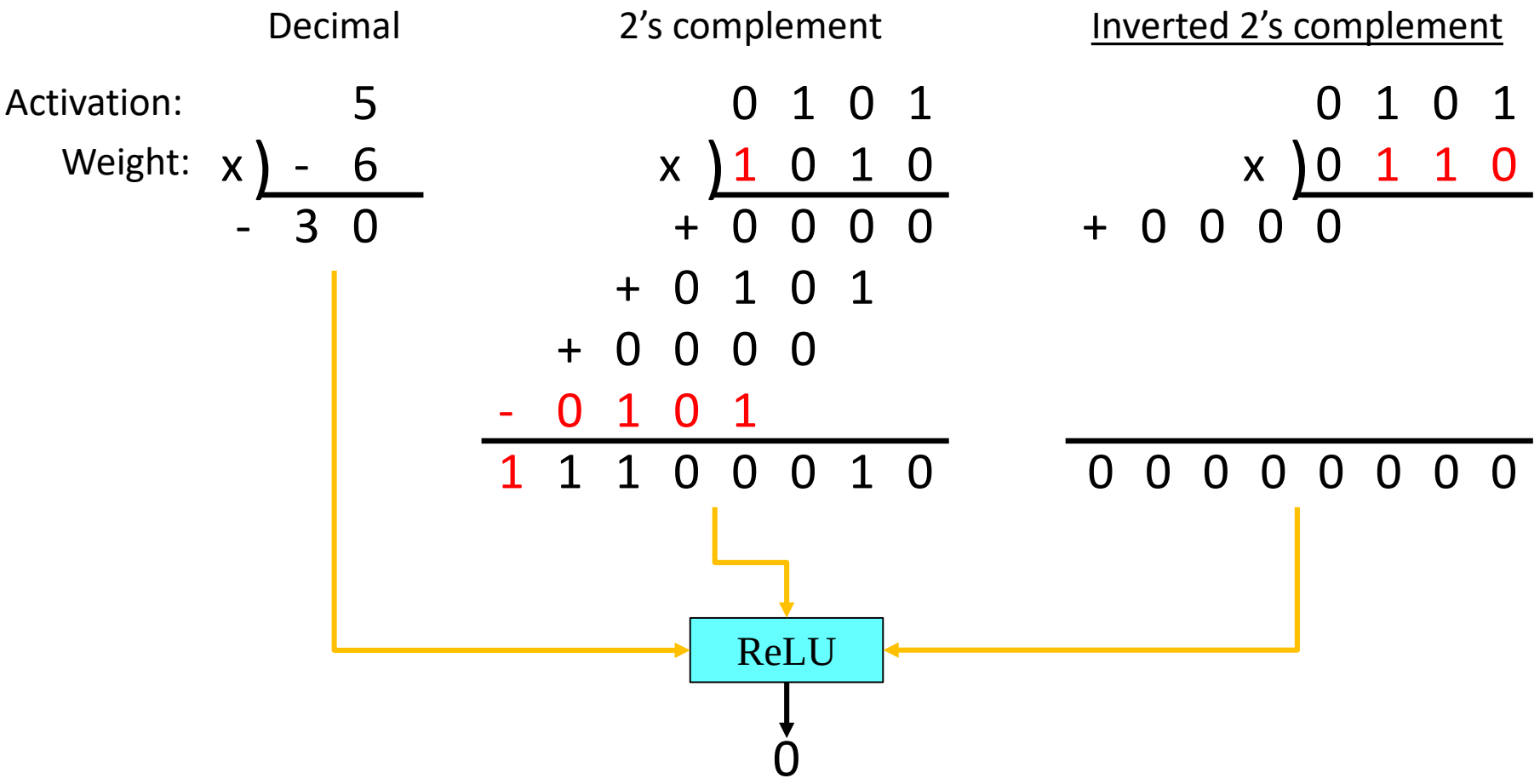
Early Negative Detection (END)

- Inverted two's complement representation for negative detection



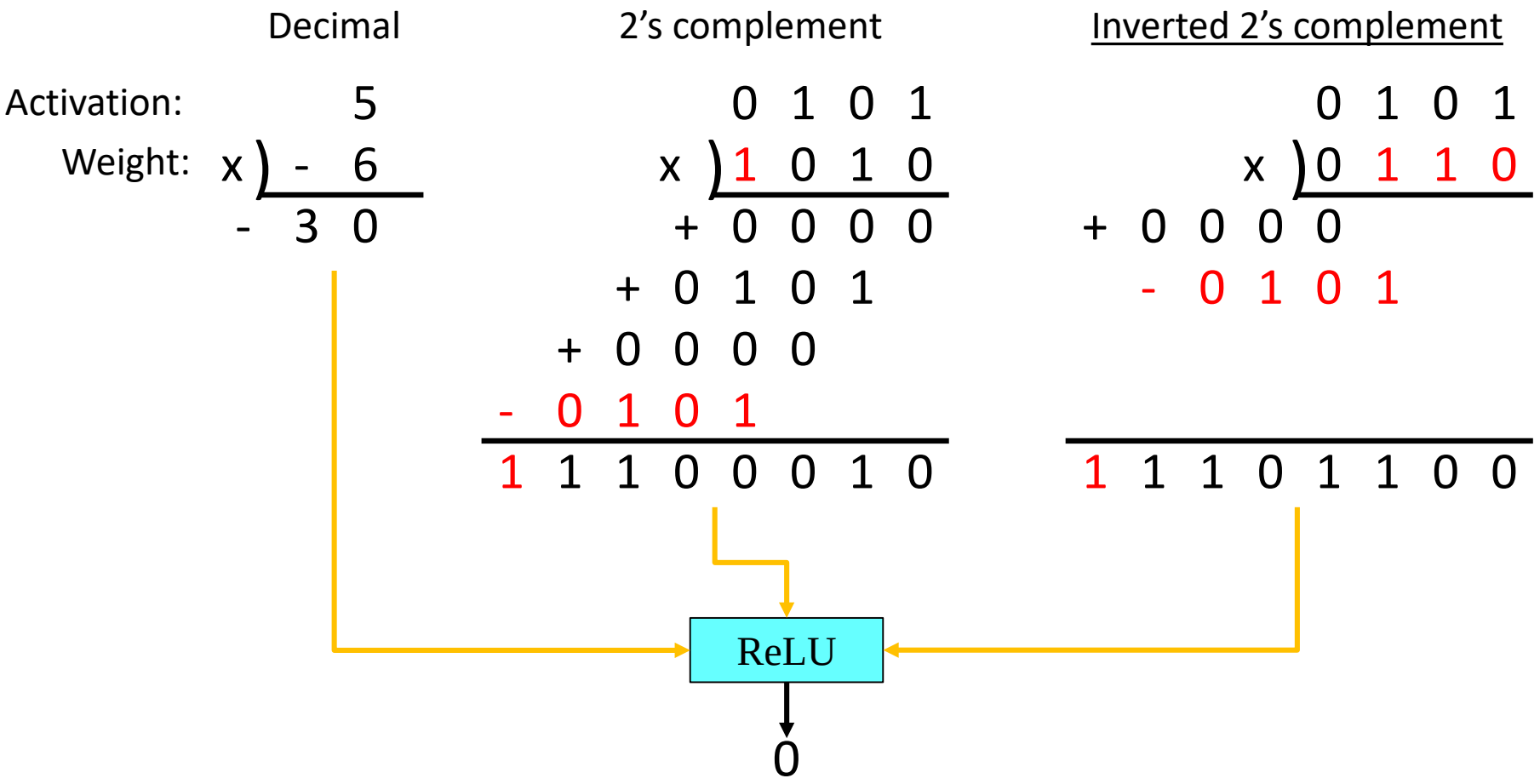
Early Negative Detection (END)

- Inverted two's complement representation for negative detection



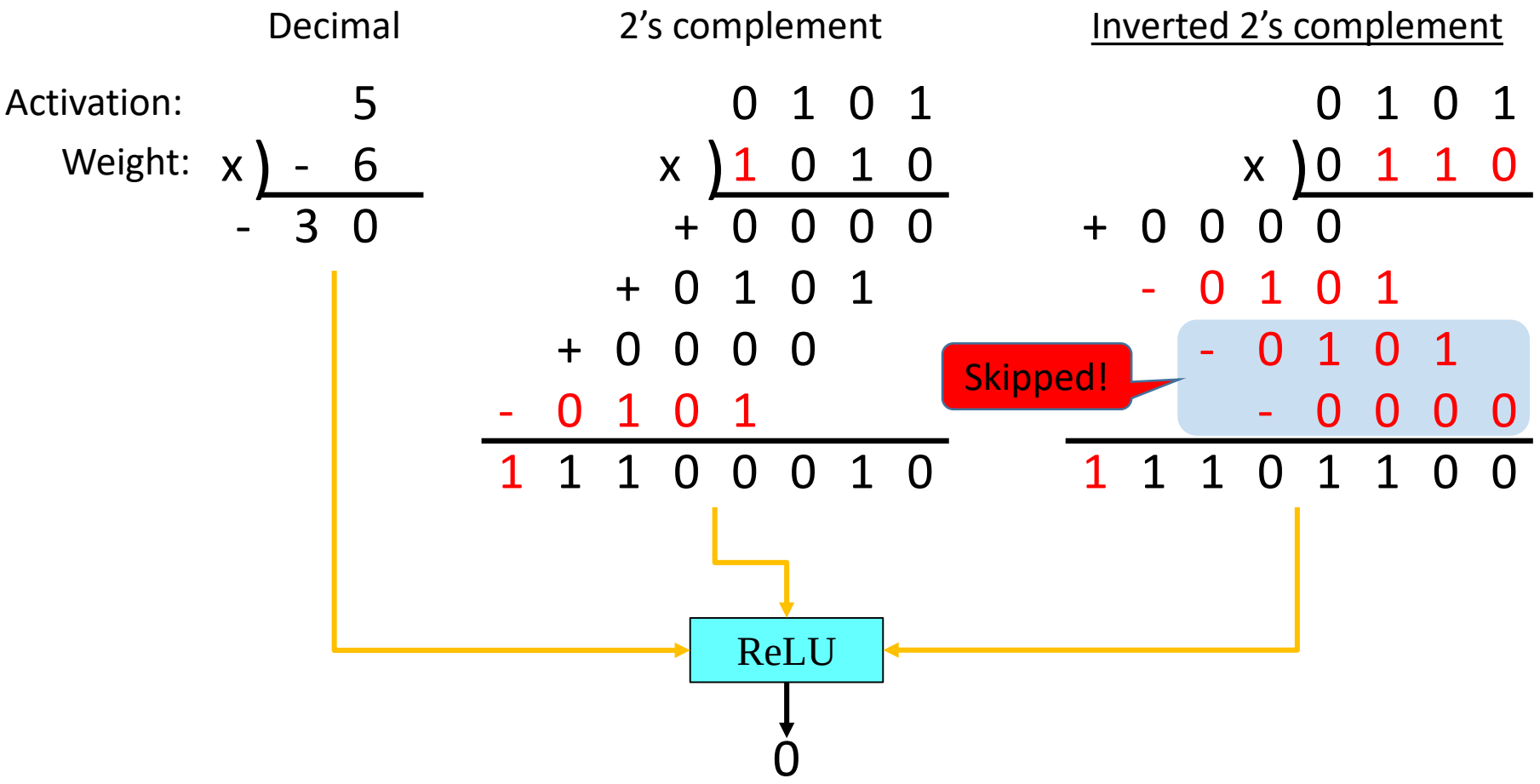
Early Negative Detection (END)

- Inverted two's complement representation for negative detection



Early Negative Detection (END)

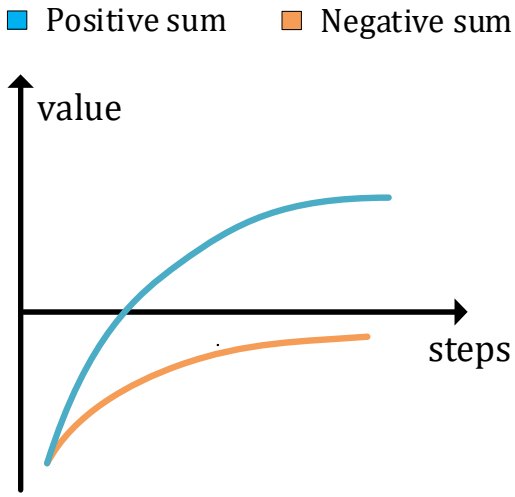
- Inverted two's complement representation for negative detection



Early Negative Detection (END)

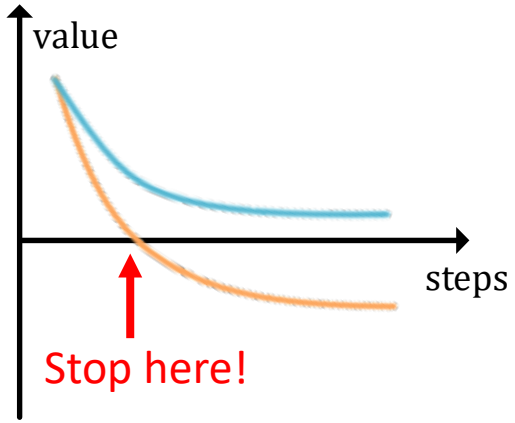
- Two's complement representation

$$W = w_{B-1} \times (-2^{B-1}) + \sum_{k=0}^{B-2} w_k \times (+2^k)$$



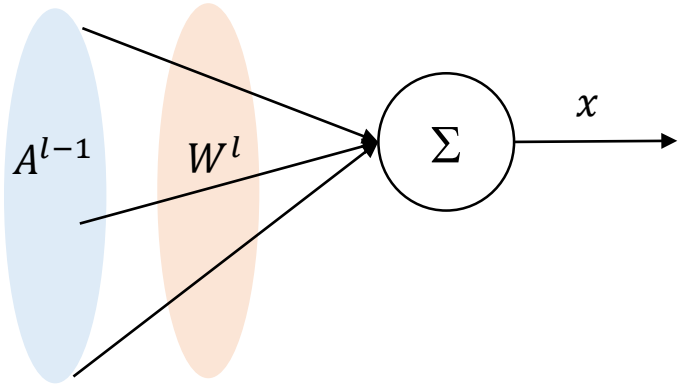
- Inverted two's complement representation

$$W = w_{B-1} \times (+2^{B-1}) + \sum_{k=0}^{B-2} w_k \times (-2^k)$$



Early Negative Detection (END)

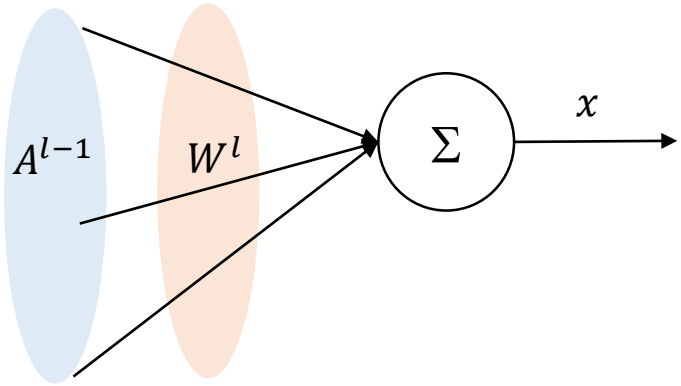
- For multiple inputs



$$x = \sum_{i=1}^{N} A_i \times W_i = A_1 \times [w_{1,B-1} \times 2^{B-1} - w_{1,B-2} \times 2^{B-2} - w_{1,B-3} \times 2^{B-3} - \cdots]$$
$$+A_2 \times [w_{2,B-1} \times 2^{B-1} - w_{2,B-2} \times 2^{B-2} - w_{2,B-3} \times 2^{B-3} - \cdots]$$
$$\cdots$$
$$+A_N \times [w_{N,B-1} \times 2^{B-1} - w_{N,B-2} \times 2^{B-2} - w_{N,B-3} \times 2^{B-3} - \cdots]$$

Early Negative Detection (END)

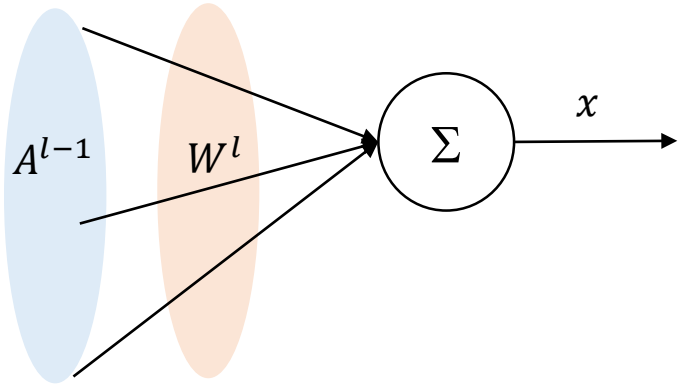
- For multiple inputs



$$x = \sum_{i=1}^{N} A_i \times W_i = A_1 \times [w_{1,B-1} \times 2^{B-1} - w_{1,B-2} \times 2^{B-2} - w_{1,B-3} \times 2^{B-3} - \cdots]$$
$$+A_2 \times [w_{2,B-1} \times 2^{B-1} - w_{2,B-2} \times 2^{B-2} - w_{2,B-3} \times 2^{B-3} - \cdots]$$
$$\cdots$$
$$+A_N \times [w_{N,B-1} \times 2^{B-1} - w_{N,B-2} \times 2^{B-2} - w_{N,B-3} \times 2^{B-3} - \cdots]$$

Early Negative Detection (END)

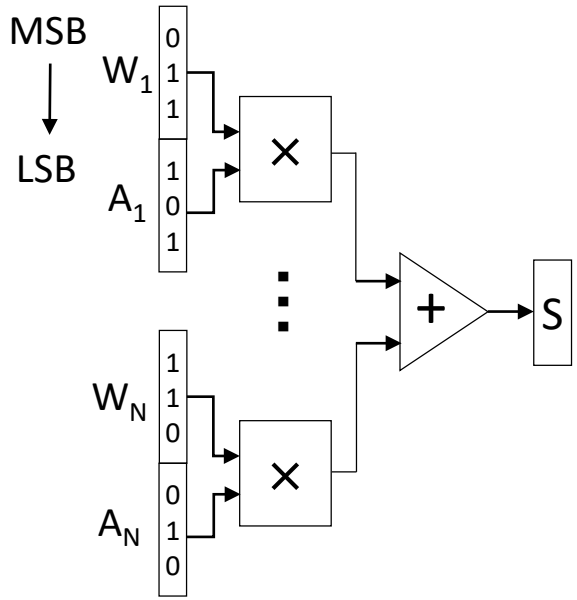
- For multiple inputs



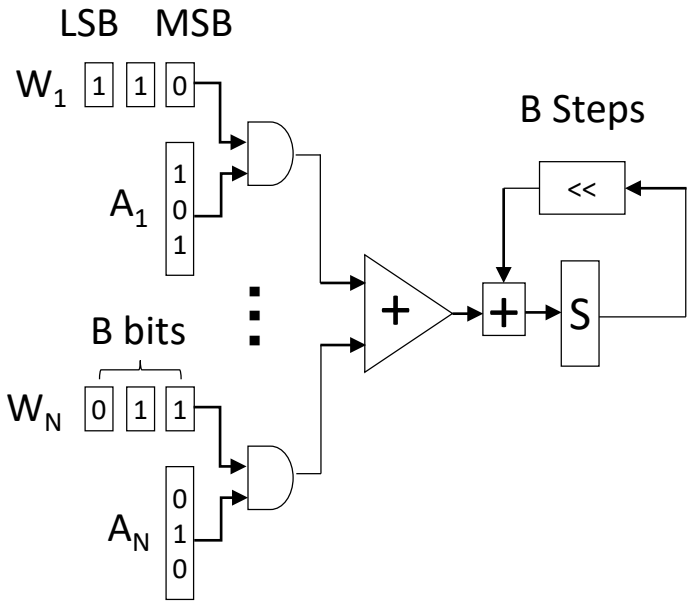
$$x = \sum_{i=1}^{N} A_i \times W_i = A_1 \times [w_{1,B-1} \times 2^{B-1} - w_{1,B-2} \times 2^{B-2} - w_{1,B-3} \times 2^{B-3} - \cdots]$$
$$+A_2 \times [w_{2,B-1} \times 2^{B-1} - w_{2,B-2} \times 2^{B-2} - w_{2,B-3} \times 2^{B-3} - \cdots]$$
$$\cdots$$
$$+A_N \times [w_{N,B-1} \times 2^{B-1} - w_{N,B-2} \times 2^{B-2} - w_{N,B-3} \times 2^{B-3} - \cdots]$$

Computation Pruning thru END (ComPEND)

- Bit-serial sum of products
  - Takes multiple steps, but the area of a bit-serial unit is much smaller
  - Can integrate more units → higher performance
  - Similar to Stripes (P. Judd et al., MICRO 2016)



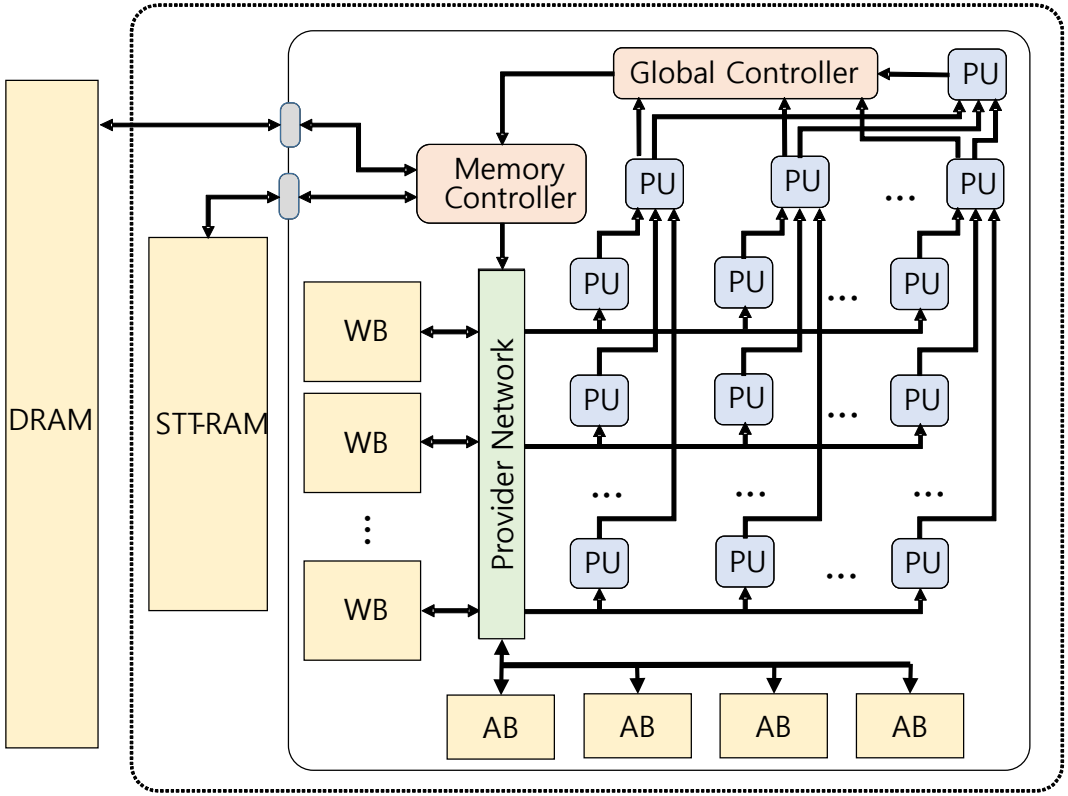
< Conventional sum of products >



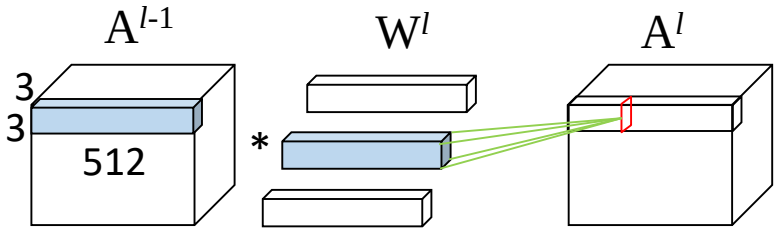
< Bit-seral sum of products >

Computation Pruning thru END (ComPEND)

- Overall architecture of ComPEND

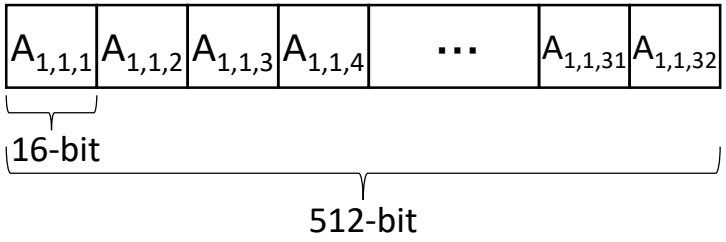


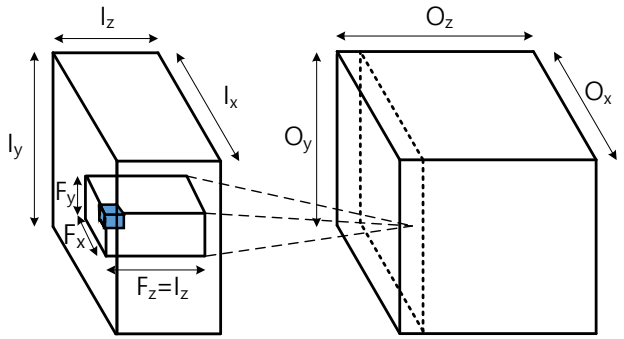
- 9x16 array of PUs
- 32 16-bit inputs per PU
- 9x16x32 inputs at a time (3x3x512 filter)
- 16 + 1 additional PUs



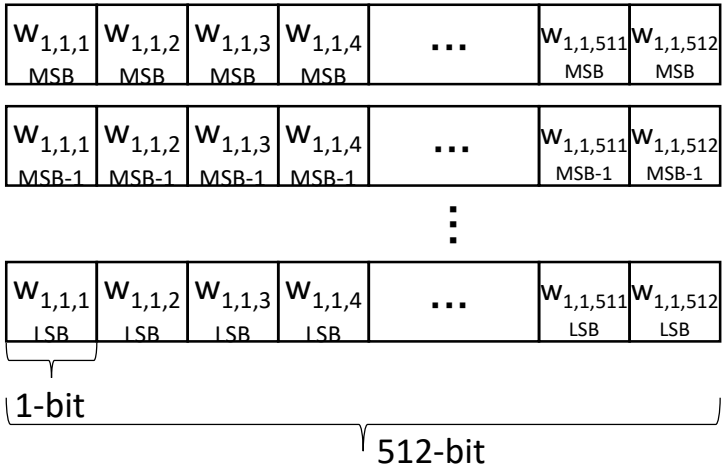
Computation Pruning thru END (ComPEND)

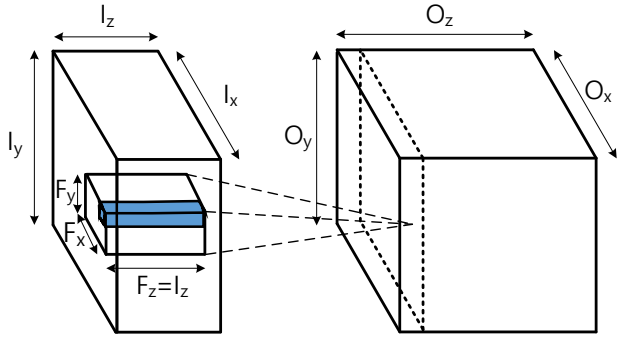
- DATA packing
  - Input activation block
    - 32 activations of same X, Y





  - Weight bits block
    - 512 bits of weights in same bit position

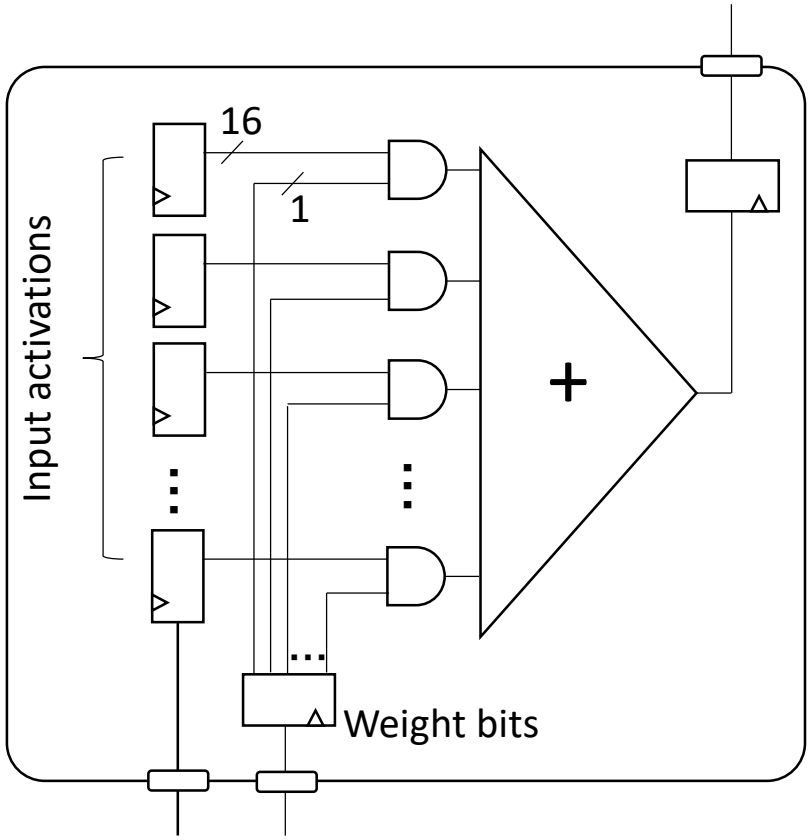




< in the case of F_z = 512 >

Computation Pruning thru END (ComPEND)

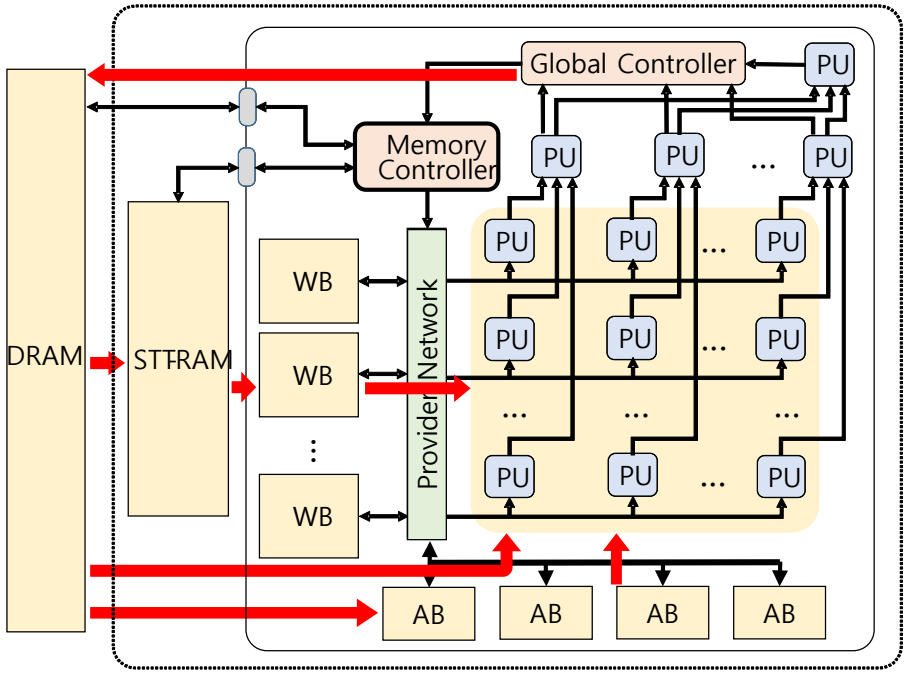
- Processing unit



- 32-input 16-bit adder tree
- 32 16-bit input activation registers
- 32-bit weight bits register

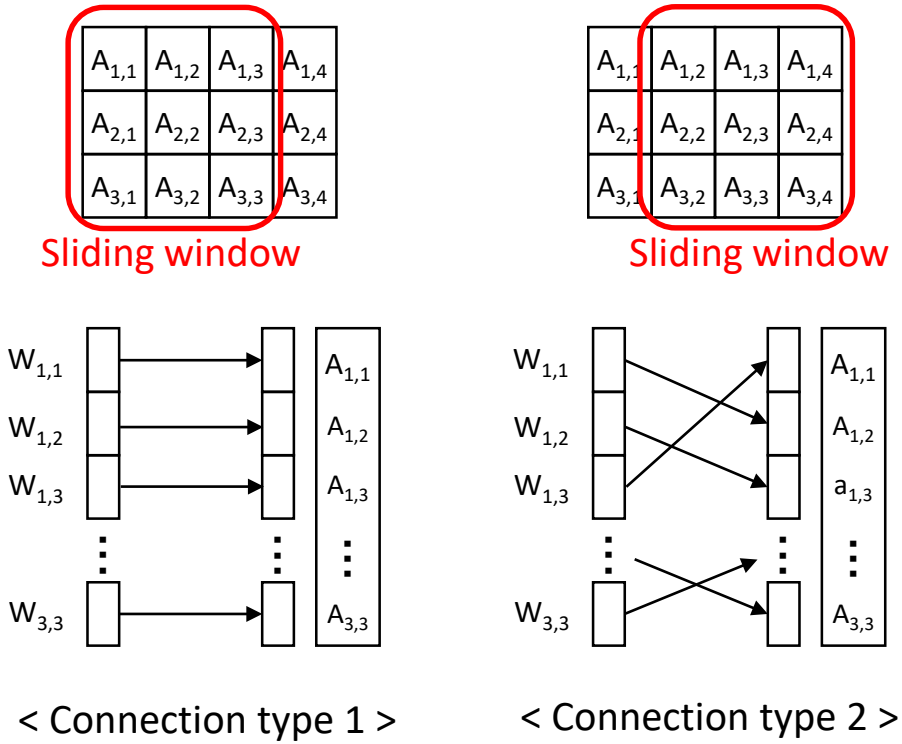
Computation Pruning thru END (ComPEND)

- Memory controller
  - Manages all kinds of memory-involved data transfers
  - Weight blocks
    - Off-chip memory -> STT-RAM
    - STT-RAM -> Weight Buffers (WBs)
    - WBs -> Weight registers in PUs
  - Activation blocks
    - Off-chip memory -> Activation Buffers (ABs)
    - Off-chip memory -> Registers in PUs
      (FC layers: activation blocks are moved directly from off-chip memory to registers)
    - ABs -> Registers in PUs
  - Output activation blocks
    - Global controller -> Off-chip memory



Computation Pruning thru END (ComPEND)

- Provider network

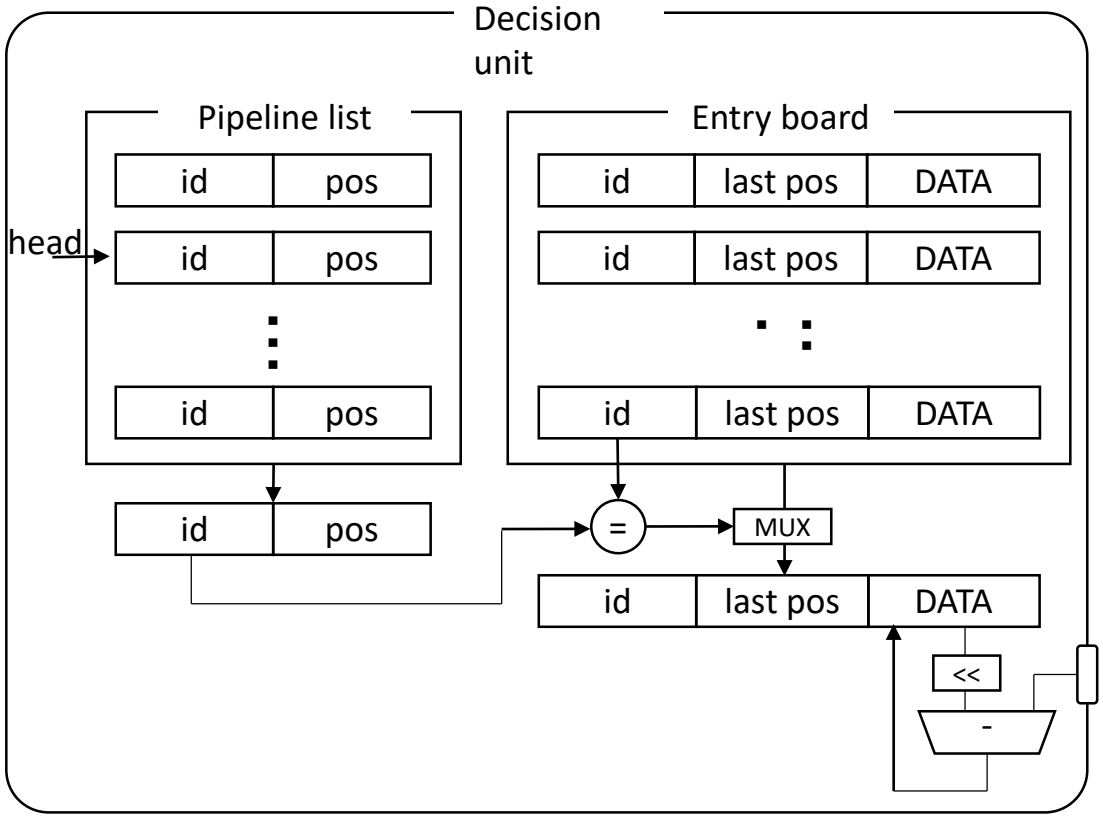


< Connection type 1 > < Connection type 2 >

- Inputs: 32 x 9 x 16 bits
- outputs: 32 x 9 x 16 bits
- Activation reuse in PUs
  - During 2D convolution with 3x3 filters
  - Reconfiguration with 9 types of connections for shuffling weights

Computation Pruning thru END (ComPEND)

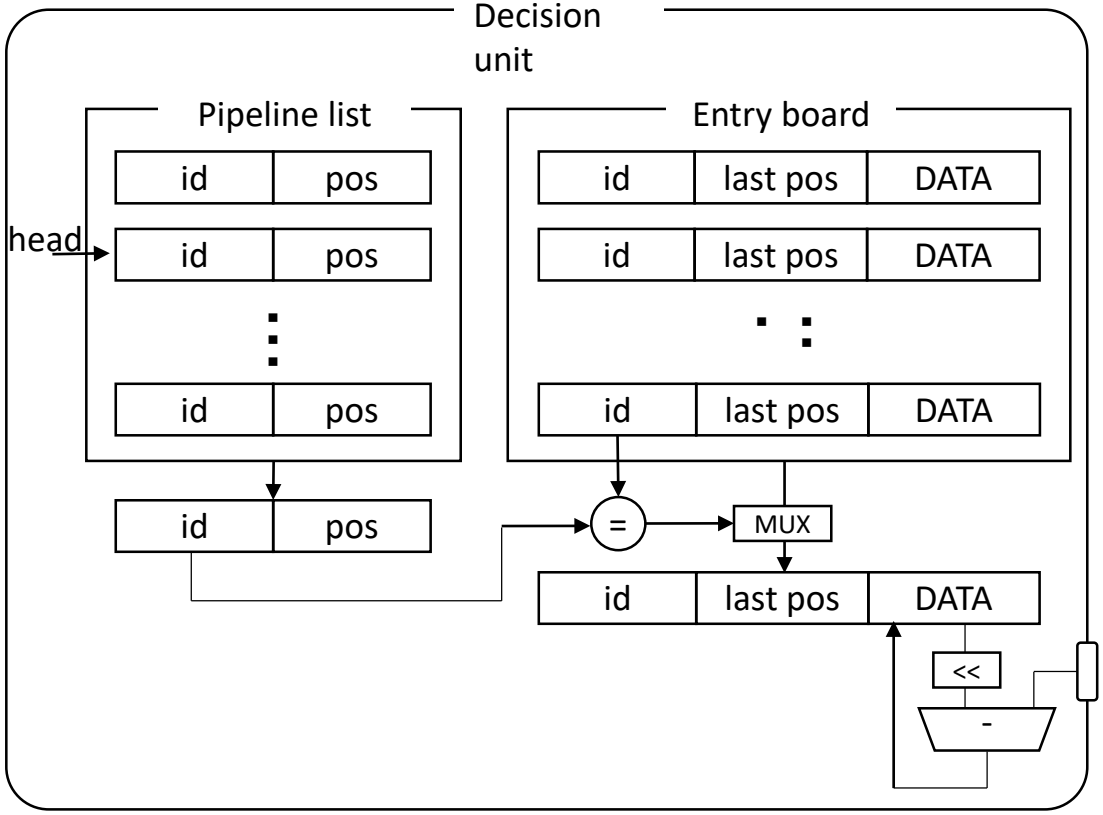
- Global controller



- 16 decision units
- Decision unit
  - Decides final sum of products
    - Zero if DATA is negative
    - DATA if last position is LSB
  - Pipeline list
    - id: filter ID
    - pos: bit position in 16-bit weights
    - head: current output of adder tree
  - Entry board
    - id: filter ID
    - last pos: last position in the pipeline
    - DATA: partial sum

Computation Pruning thru END (ComPEND)

- Global controller



- Filling up the pipeline

P1: The next bit in the bit-serial computation

P2: A new sum of products that has not yet been entered into the pipeline

P3: The next step of a sum of products whose prior step is still in the pipeline

completed P1 P3

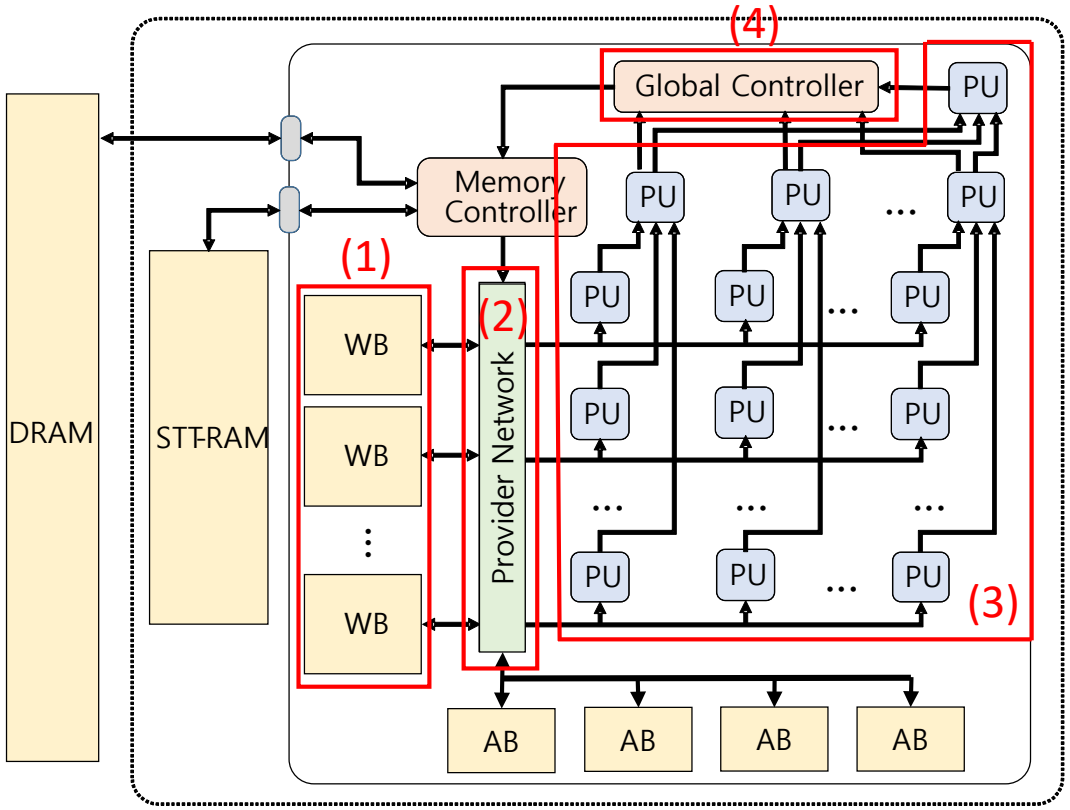
F_p: ($w_{i,B-1}$ $w_{i,B-2}$ $w_{i,B-3}$ … $w_{i,1}$ $w_{i,0}$)

F_q: ($w_{j,B-1}$ $w_{j,B-2}$ $w_{j,B-3}$ … $w_{j,1}$ $w_{j,0}$)

P2

Computation Pruning thru END (ComPEND)

- Operation pipeline



(1) Weight buffers ->

(2) Provider network (2) ->

(3) Processing unit array (3) ->

(4) Global controller

Evaluation

- Pre-trained weights of VGG-16 network and 1000 images from ImageNet ILSVRC-2012
- In-house cycle-accurate timing simulator by using C++ with DRAMSim2 for off-chip memory
- CACTI 6.5 to model SRAM
- NVSim for on-chip STT-RAM
- Synopsys Design Compiler with TSMC 45nm technology library with 0.9V to get parameters of timing/power/area for PUs and Provider Network

Evaluation

- VGG-16 network

Layer	Imap	filter	# of filter	Omap
C1	224x224x3	3x3x3	64	224x224x64
C2	224x224x64	3x3x64	64	224x224x64
C3	112x112x64	3x3x64	128	112x112x128
C4	112x112x128	3x3x128	128	112x112x128
C5	56x56x128	3x3x128	256	56x56x256
C6	56x56x256	3x3x256	256	56x56x256
C7	56x56x256	3x3x256	256	56x56x256
C8	28x28x256	3x3x256	512	28x28x512
C9	28x28x512	3x3x512	512	28x28x512
C10	28x28x512	3x3x512	512	28x28x512
C11	14x14x512	3x3x512	512	14x14x512
C12	14x14x512	3x3x512	512	14x14x512
C13	14x14x512	3x3x512	512	14x14x512
F1	7x7x512	7x7x512	4096	1x1x4096
F2	1x1x4096	1x1x4096	4096	1x1x4096
F3	1x1x4096	1x1x4096	1000	1x1x1000

- We use 15 layers in the VGG-16 network as workloads, excluding layer F1.
- F1 is excluded since the total size of input activations is too big.
- Inputs to C1 are raw data that can be negative.
  → The pruning scheme cannot be applied.
  → C1 is implemented without ComPEND.

Evaluation

- Configuration

Frequency	1 GHz
Processing unit	Total 161 PUs (9 x 16 + 16 + 1), 2 bytes 32-input PU, 1 cycle for a PU, 3 pipeline stages for a PU array
Peak throughput	Total 4.608 TOPS or 288 GMACS
Weight buffer	Total 288 Kbytes, 9 x 32 Kbytes SRAM, 64 bytes, 1 cycle for a 32 Kbytes WB
Activation buffer	Total 128 Kbytes, 4 x 32 Kbytes SRAM, 64 bytes, 1 cycle for a 32 Kbytes AB
Provider network	9 x 64 bytes input, 9 x 64 bytes output, 1 cycle for a stage, 2 pipeline stages
Global controller	16 decision units, 1 cycle for each unit
STT-RAM	Total 4.5 Mbytes, 9 x 512 Kbytes STT-RAM, 64 bytes, 2 cycles for read, 11 cycles for write of a 512 Kbytes STT-RAM
DRAM	16 Gbytes, 8 Banks, 1333 MHz (DDR3_micron_32M_8B_x8_sg15)

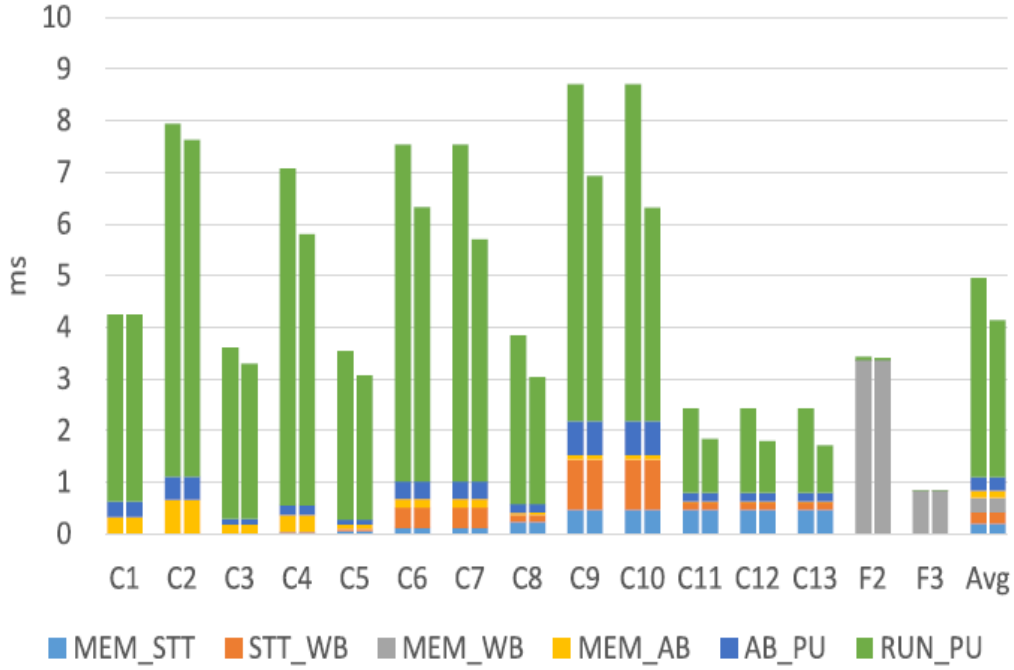
※ Peak throughput (32-input PU × 16 PUs in a row × 9 rows × 1GHz = 4.6 TOPS)

- Area

	Area (mm^2)	%
PUs	0.75	13%
Provider	0.09	2%
WBs	1.27	23%
ABs	0.56	10%
STT-RAM	2.93	52%
Total	5.62	100%

Evaluation

- Runtime
  - Reduced by 16.62% on average compared to that without ComPEND for 15 layers



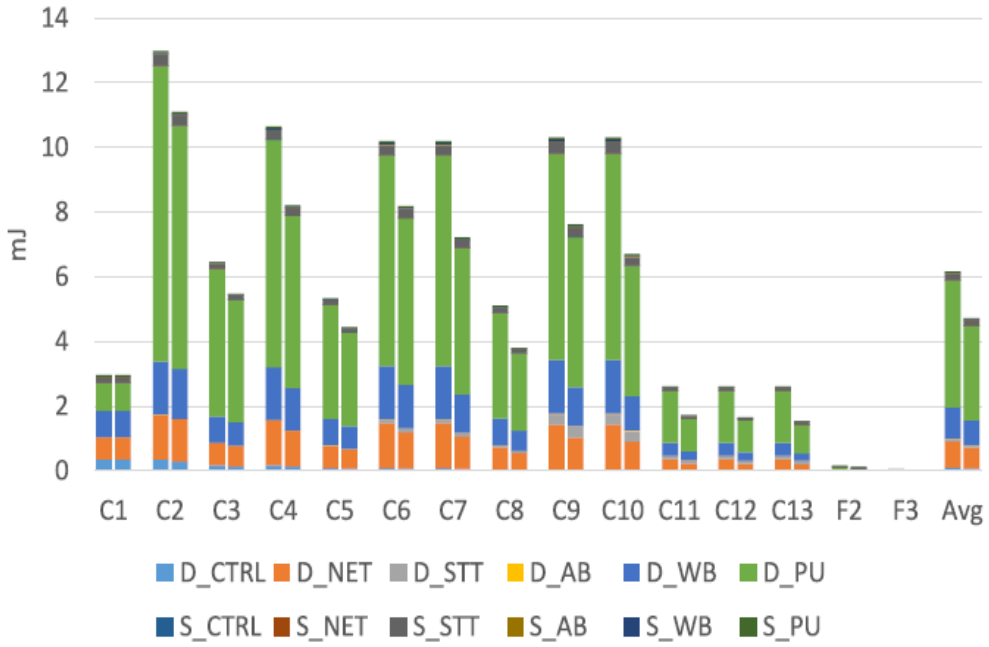
Left bars: without ComPEND
Right bars: with ComPEND

< for VGG-16 layers >

- MEM_STT: reads/writes between off-chip memory and STT-RAM
- STT_WB: runtime of reads/writes between STT-RAM and WB
- MEM_WB: reads/writes between off-chip memory and WB
- MEM_AB: reads/writes between off-chip memory and AB
- AB_PU: reads/writes between AB and registers in PUs
- RUN_PU: computation in PUs

Evaluation

- Energy (dynamic & static) consumption
  - Reduced by 23.50% on average for 15 layers



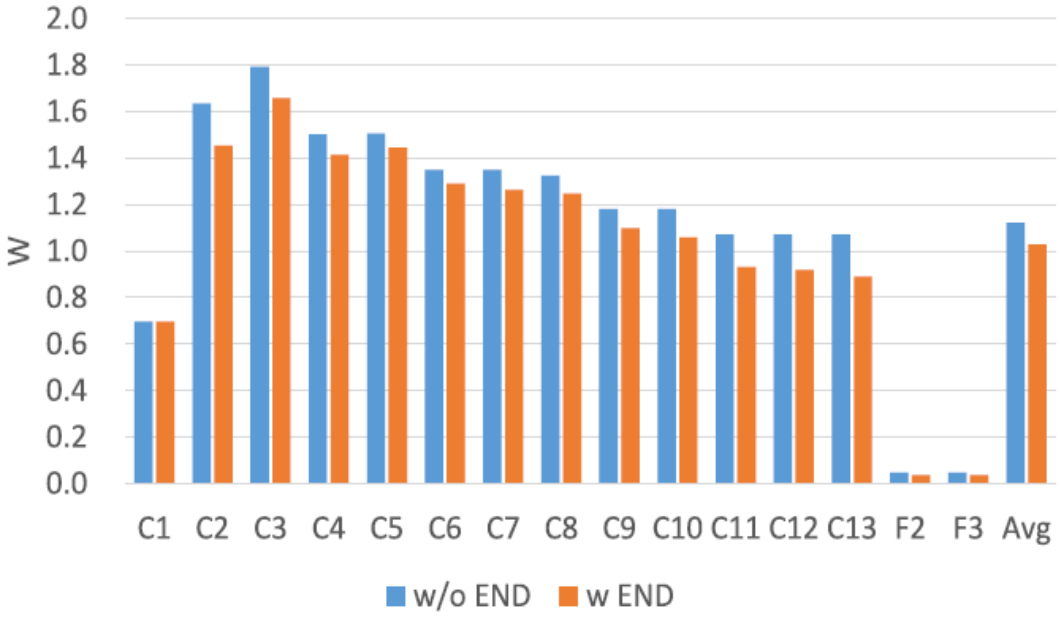
Left bars: without ComPEND
Right bars: with ComPEND

< for VGG-16 layers >

- D/S_CTRL: global controller
- D/S_NET: provider network
- D/S_STT: STT-RAM.
- D/S_AB: activation buffers
- D/S_WB: weight buffer
- D/S_PU: processing units

Evaluation

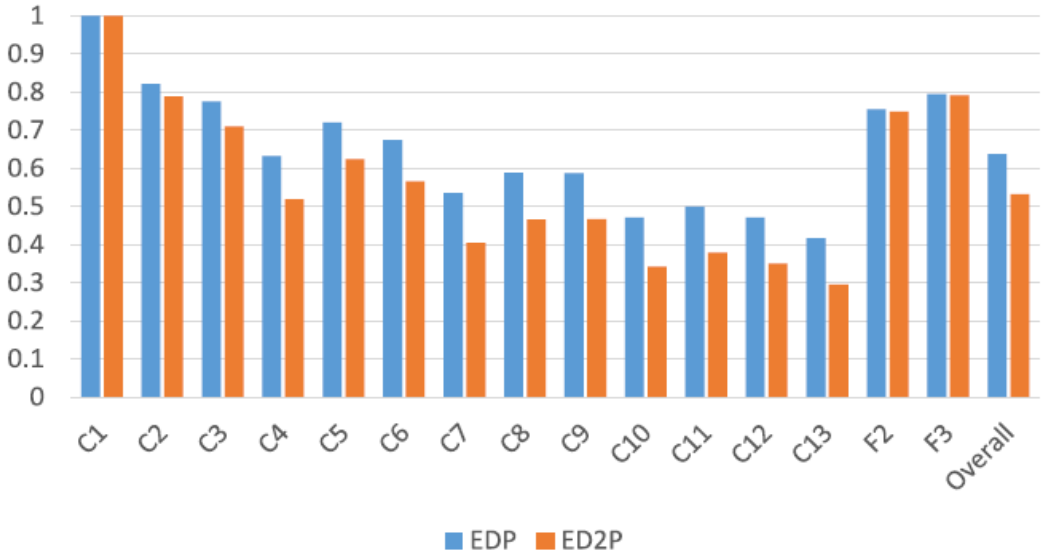
- Power consumption
  - Average over 15 layers
    - Without ComPEND: 1.12 Watt
    - With ComPEND: 1.03 Watt



< for VGG-16 layers >

Evaluation

- Energy-delay product
  - ComPEND reduces EDP and ED^{2}P by 36.21% and 46.81% for the execution of the 15 layers



< for VGG-16 layers >

Conclusion

- Proposed the concept of END (early negative detection) based on inverted two's complement
- Proposed an architecture that implements ComPEND
- Achieved 16.62% higher speed and 23.50% less energy consumption for inference
- Future work
  - Combining with other zero-skipping approaches
  - Handling layers (say, F1 in VGG-16) exceeding the capacity of the architecture

THANK YOU