2-hour Lecture on May 29th

- 1st hour
  - 9am-9:50am
  - eTL zoom
  - **Online** invited talk by Dr. Donghyuk Woo, Google Edge TPU team
  - Attendance check on eTL zoom

- 2nd hour Prof Yoo's Lecture
  - 10:00am-10:50am
  - eTL zoom
  - Prof Yoo's **online** lecture
  - Attendance check on eTL zoom

Advanced Hardware Accelerator: Zero Skipping and Low Precision

May 22, 2023

Sungjoo Yoo

Computing Memory Architecture Lab.

CSE, SNU

http://cmalab.snu.ac.kr

Weekly Lecture / Lab Schedule

- W1 (March 6) Class introduction / (March 8) Orientation & team formation
- W2 13 Verilog 1 Combinational circuits / 15 Verilog 1 (tool installation, adder & multiplier combinational logic)
- W3 20 Verilog 2 Sequential circuits / 22 Verilog 2 (memory i/o, FSM sequential logic)
- W4 27 AI application introduction 1, Amaranth introduction 1 / 29 Amaranth (tool installation, MAC, adder tree)
- W5 4/3 AI application introduction 2, Amaranth introduction 2 (memory i/o, FSM sequential logic) / 5 Amaranth (PE)
- W6 10 AI application introduction 3, Neural network accelerator 1 / 12 Amaranth (PE controller)
- W7 17 Neural network accelerator 2 / 19 Q&A
- W8 24 **Mid-term exam** / 26 Amaranth (outer product accelerator)
- W9 5/1 Reading data from memory 1 (VA2PA, interconnect) / 3 PyTorch (simple CNN on MNIST)
- W10 8 Reading data from memory 2 (DRAM main memory) / 10 Quantization aware training (QAT)
- W11 15 Compressing networks (pruning, low precision) / 17 Convolution lowering & tiling
- W12 22 Compressing networks (pruning, low precision) / 24 PyTorch – Amaranth communication
- W13 29 Invited talk (1h online Google Edge TPU), Zero-skipping & low-precision hardware accelerator / 31 Zero skipping
- W14 6/5 **Final exam** / 7 Project Q&A
- W15 12 (online) Claim & Project Q&A / 14 (online) Project Q&A, submission

[Kim, 2017]

Observation: Abundant Zero Input Values in CNNs

Layer	Zero Weight [%] [1]	Zero Activation [%] [2]
conv1	15.7	0
conv2	62.1	50.9
conv3	65.4	76.3
conv4	62.8	61.8
conv5	63.1	59.0

Pruning

ReLU

AlexNet

- **Significant portion of input values in a convolutional layer is zero**

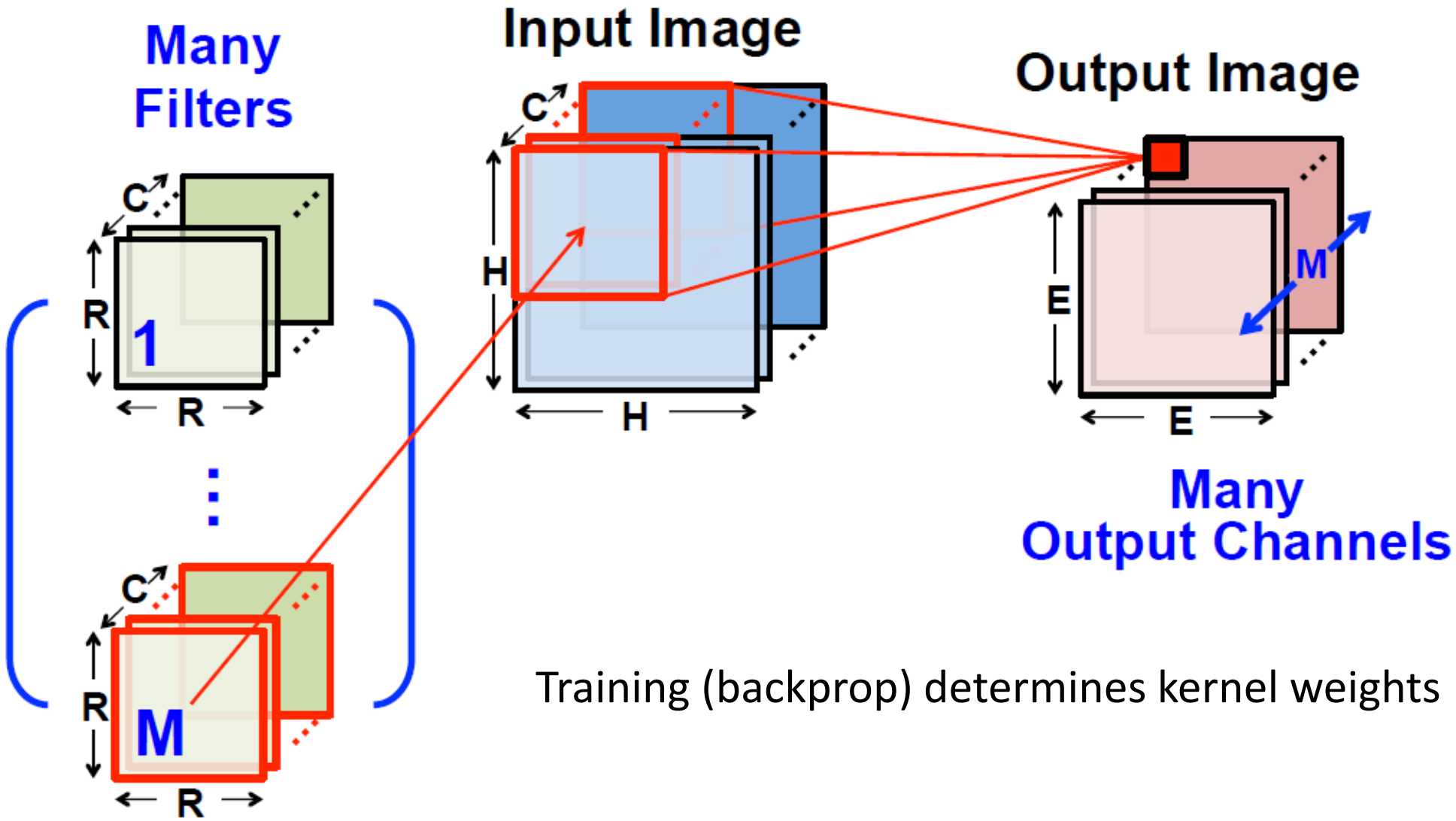
1. S. Han et al., "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding,". *ICLR*, 2016.
2. We obtained the ratio of zero activations by applying 100 randomly selected images from ImageNet validation set to the pruned AlexNet.

Agenda

- Zero-weight skipping accelerator
  - NVIDIA Sparse TensorCore, Samsung NPU v1
- Zero-activation skipping accelerator
  - Samsung NPU v2
- Low precision accelerator
  - NVIDIA TensorCore

[Chen, 2016]

Convolution: 3D Input / 3D Output

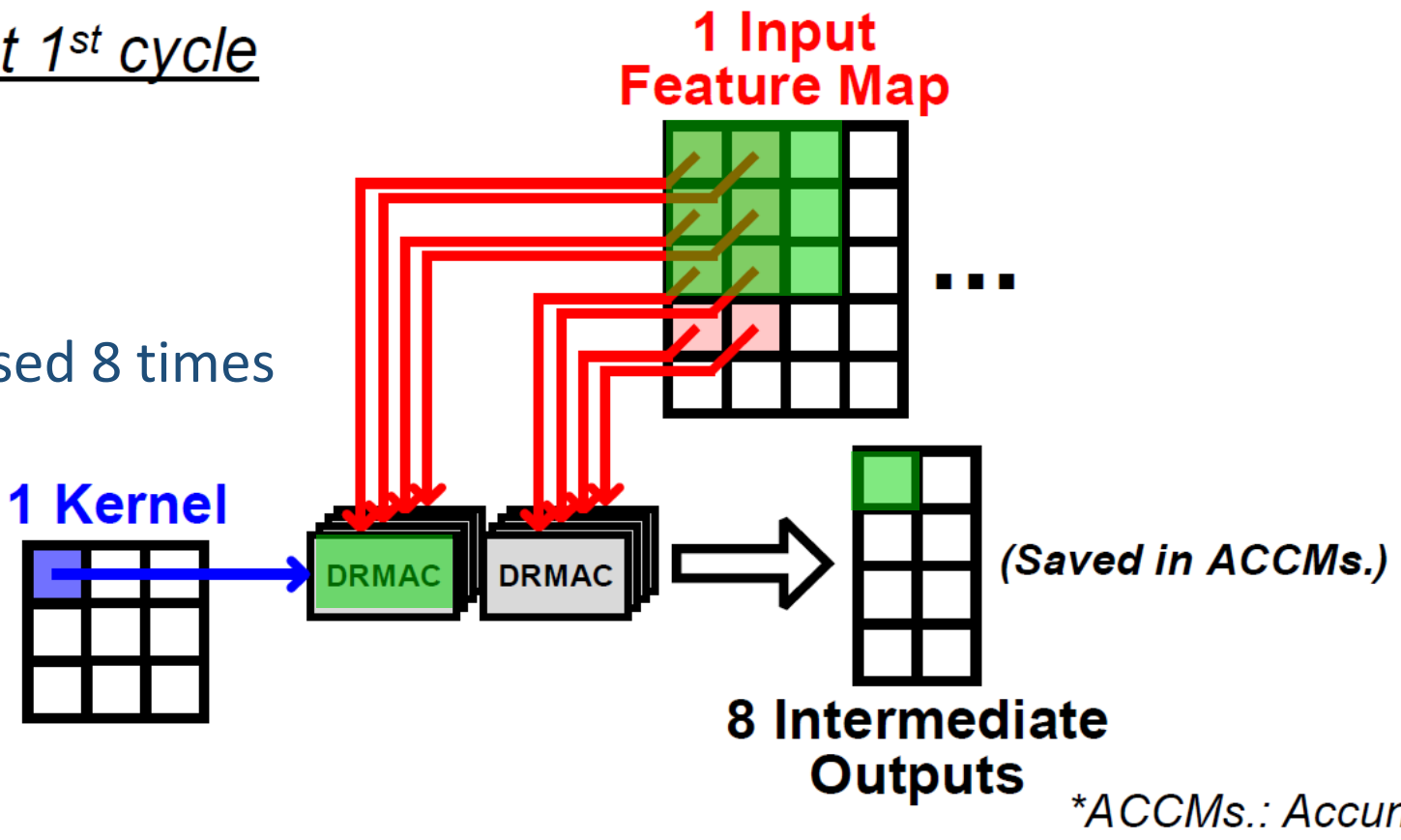


Training (backprop) determines kernel weights

Neuron Processing Engine

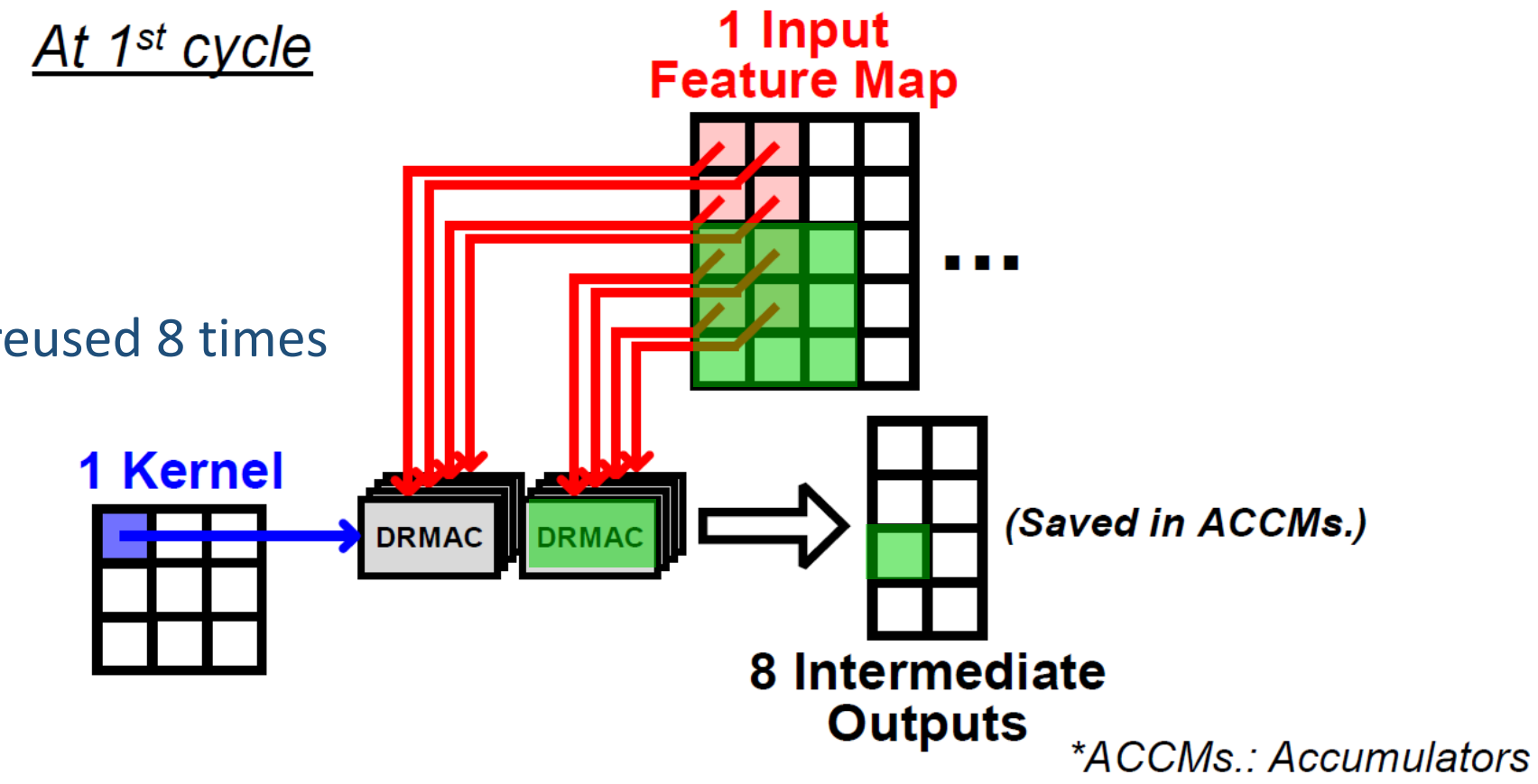
- **Convolution Operation**
  - 8 intermediate words are generated by performing MACs on 8 image words with 1 kernel word

Kernel weight is reused 8 times



Neuron Processing Engine

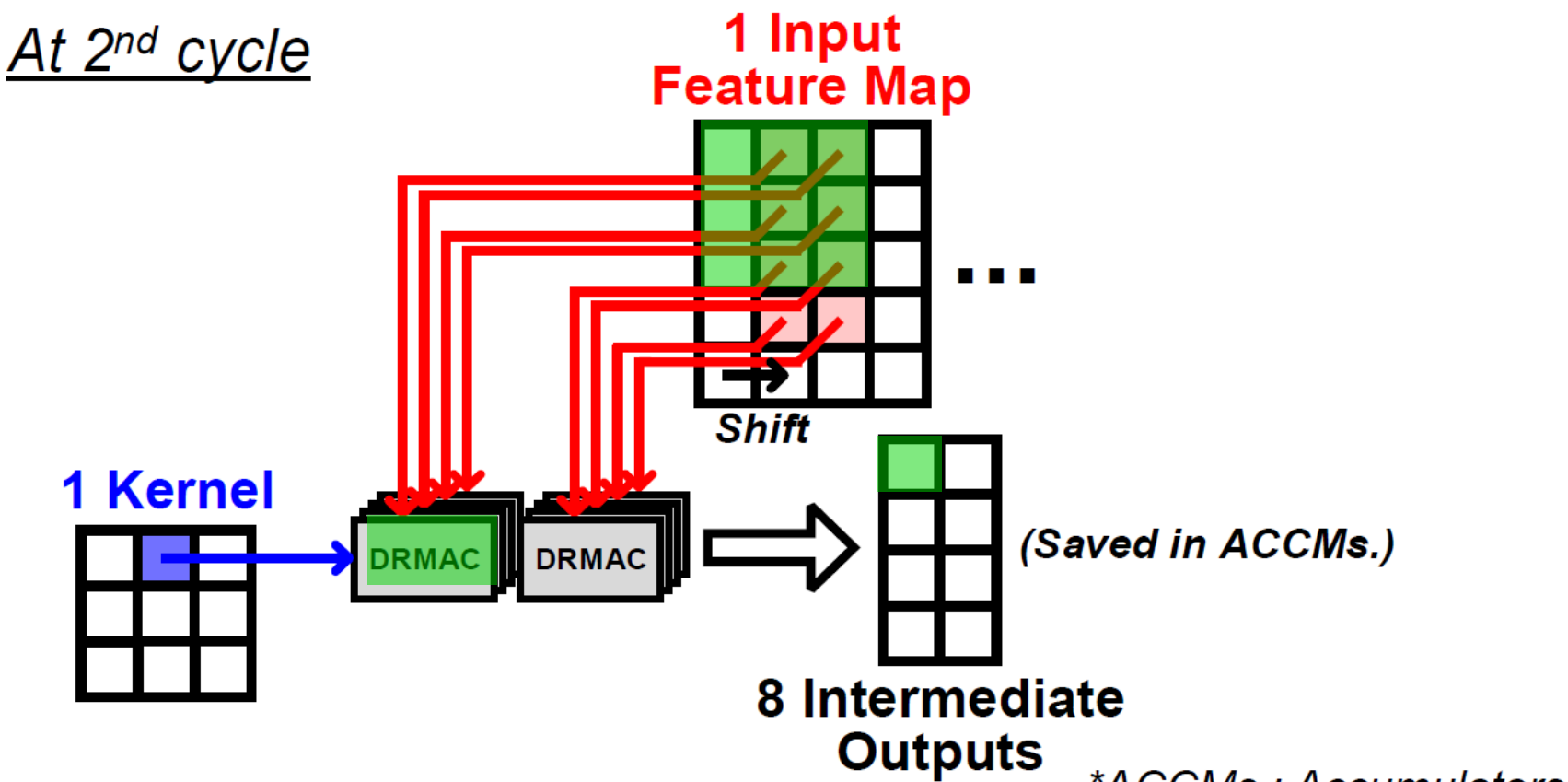
- **Convolution Operation**
  - 8 intermediate words are generated by performing MACs on 8 image words with 1 kernel word



Kernel weight is reused 8 times

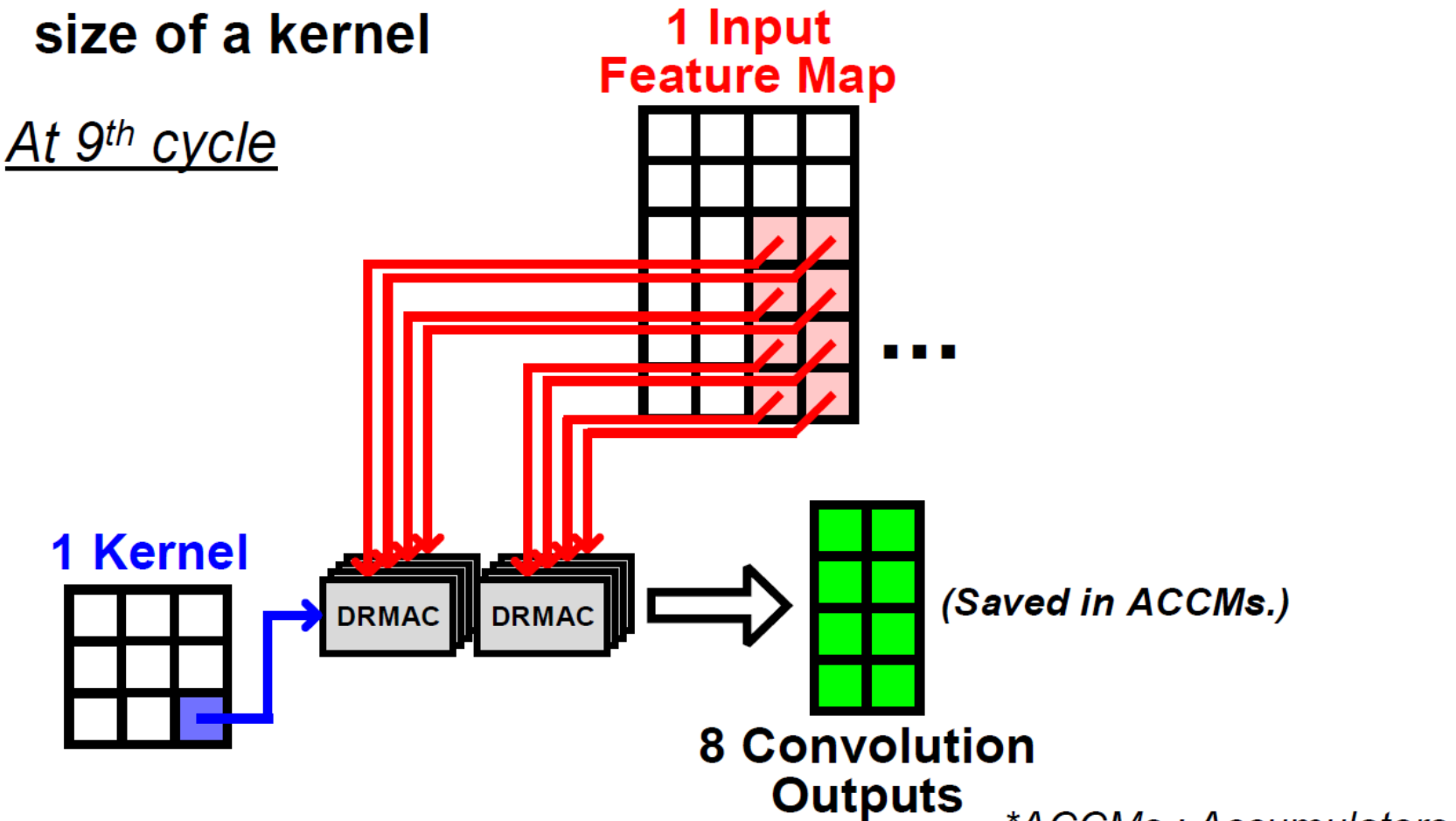
Neuron Processing Engine

- **Convolution Operation**
  - **Next words from the input feature map are reused by shift registers.**



Neuron Processing Engine

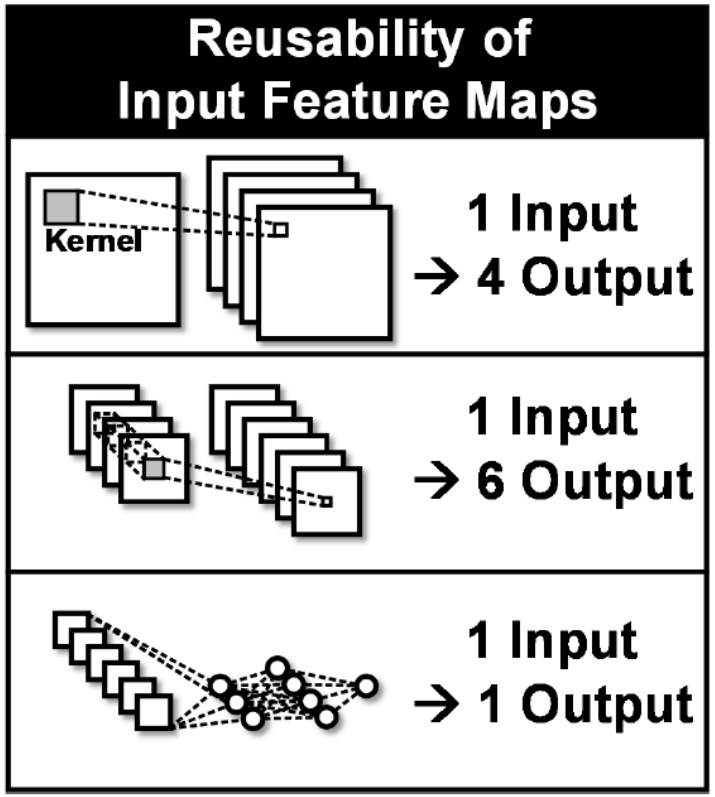
- **Convolution Operation**
  - **A single convolution takes cycles as many as the size of a kernel**

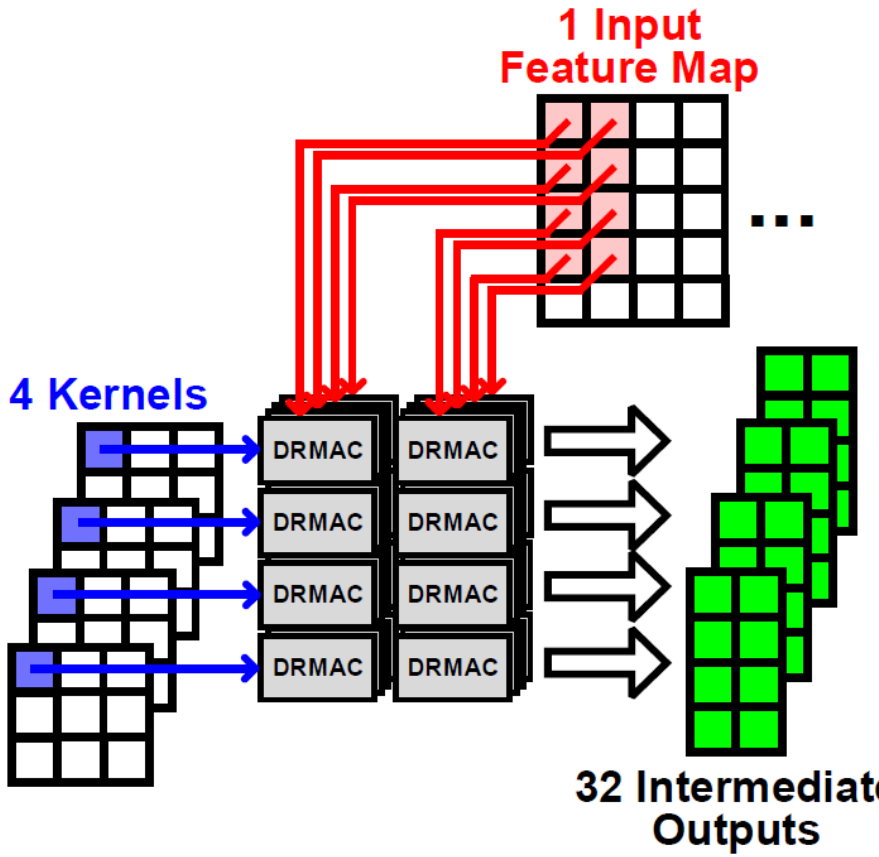


Neuron Processing Engine

- **Convolution Operation**
  - 4 groups of 8 MACs share a input feature map

→ **Maximum utilization of data with limited internal BW**





Kernel weight is reused 8 times

Data item is reused 4 times

Zero Weight Skipping Accelerator

An 11.5 TOPS/W 1024-MAC Butterfly-Structure Dual-Core Sparsity-Aware Neural Processing Unit in 8nm Flagship Mobile SoC

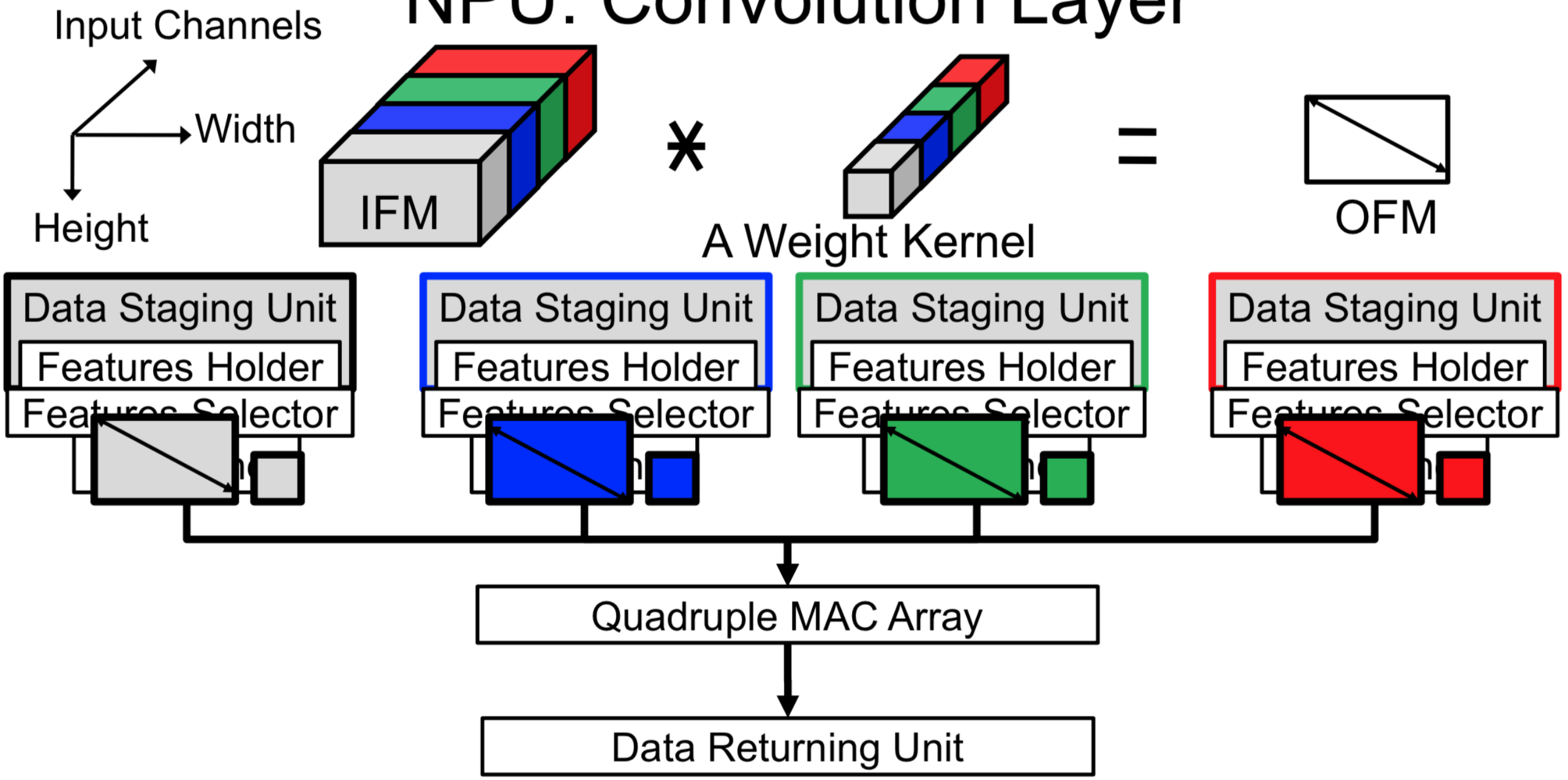
Jinook Song[1], Yunkyo Cho[1], Jun-Seok Park[1], Jun-Woo Jang[2], Sehwan Lee[2], Joon-Ho Song[2], Jae-Gon Lee[1], Inyup Kang[1]

SAMSUNG Exynos

[1]Samsung Electronics, Hwaseong, Korea

[2]Samsung Advanced Institute of Technology, Suwon, Korea

NPU: Convolution Layer



Convolution: Moving Output Feature Maps

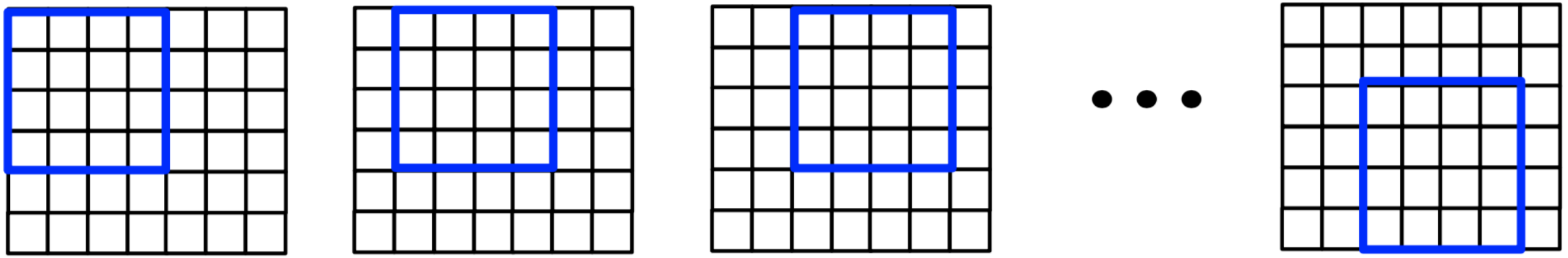
Weight Kernel

0	1	2
3	4	5
6	7	8

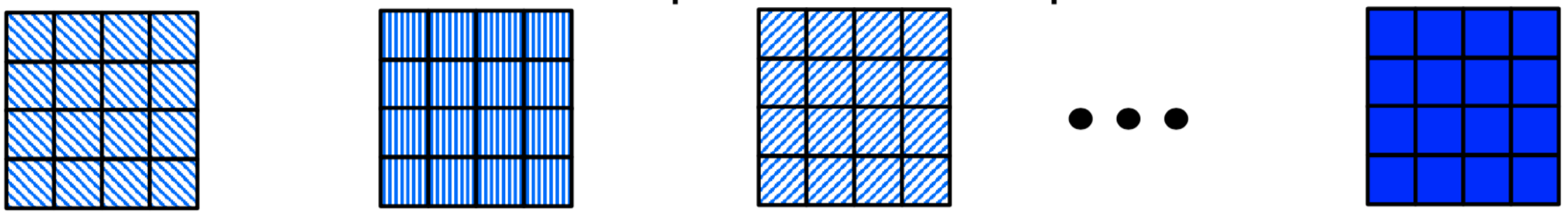
Non-zero Weights

0 1 2 8

Input Feature Map



Output Feature Map



Cycle #1 Cycle #2 Cycle #3 Cycle #9

Skipping Convolution: Moving OFM

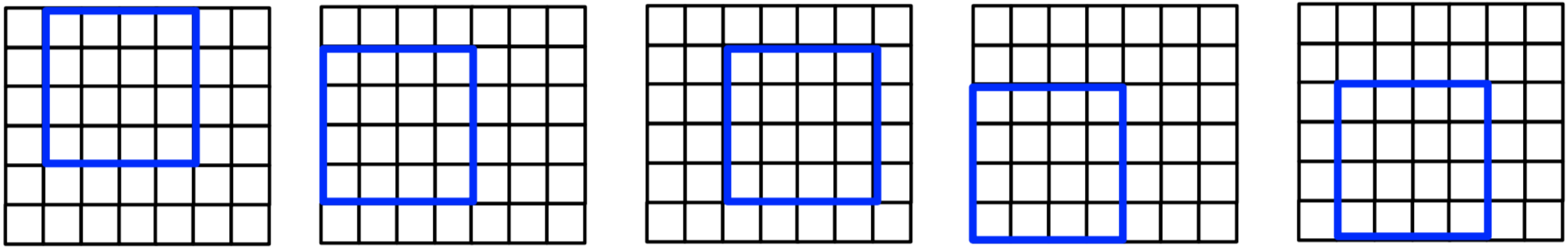
Non-zero Weights

1 3 5 6 7

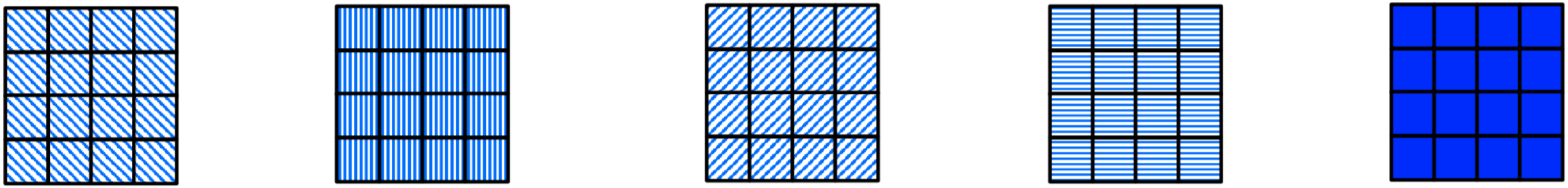
Weight Kernel

	1	
3		5
6	7	

Input Feature Map

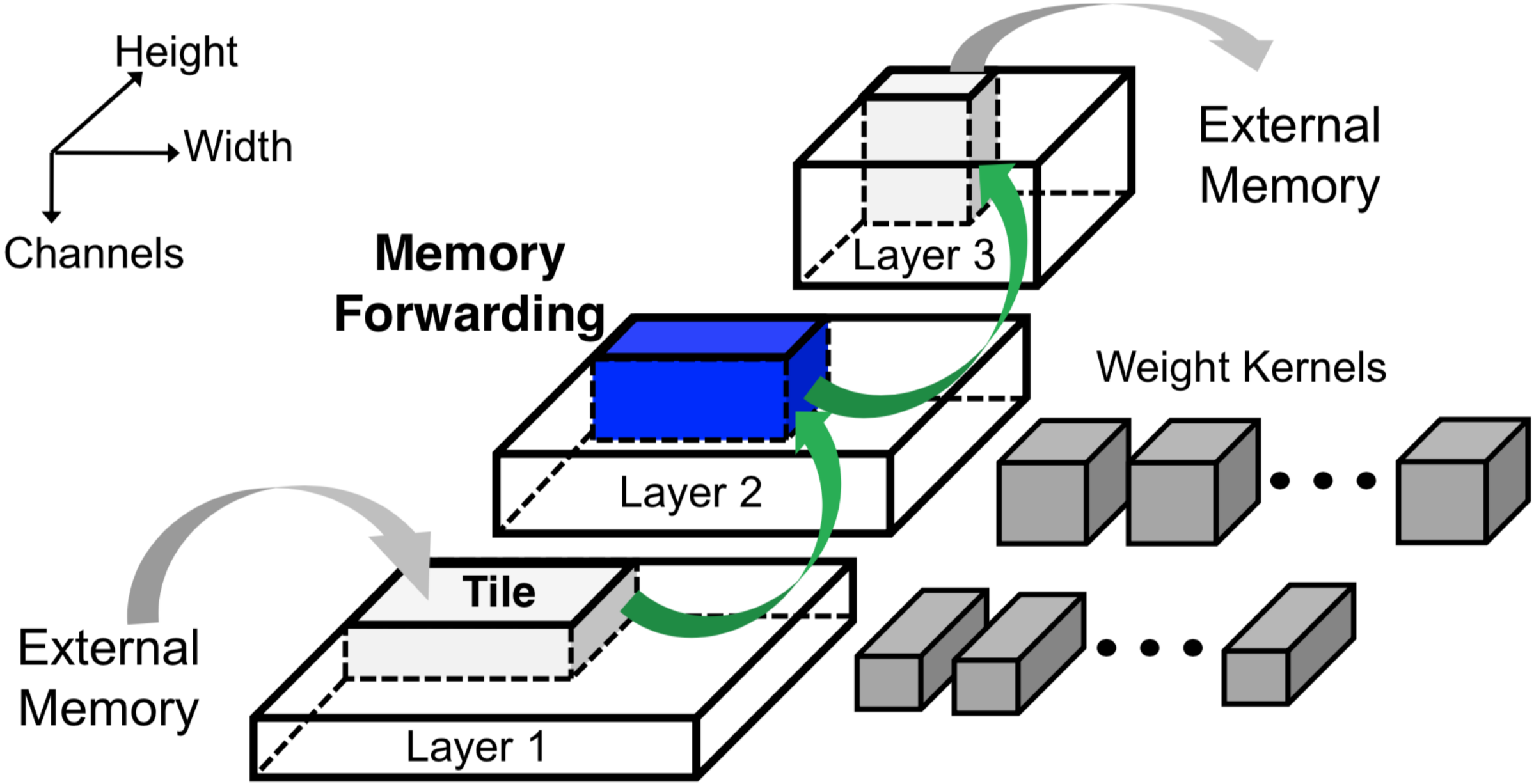


Output Feature Map



Cycle #1 Cycle #2 Cycle #3 Cycle #4 Cycle #5

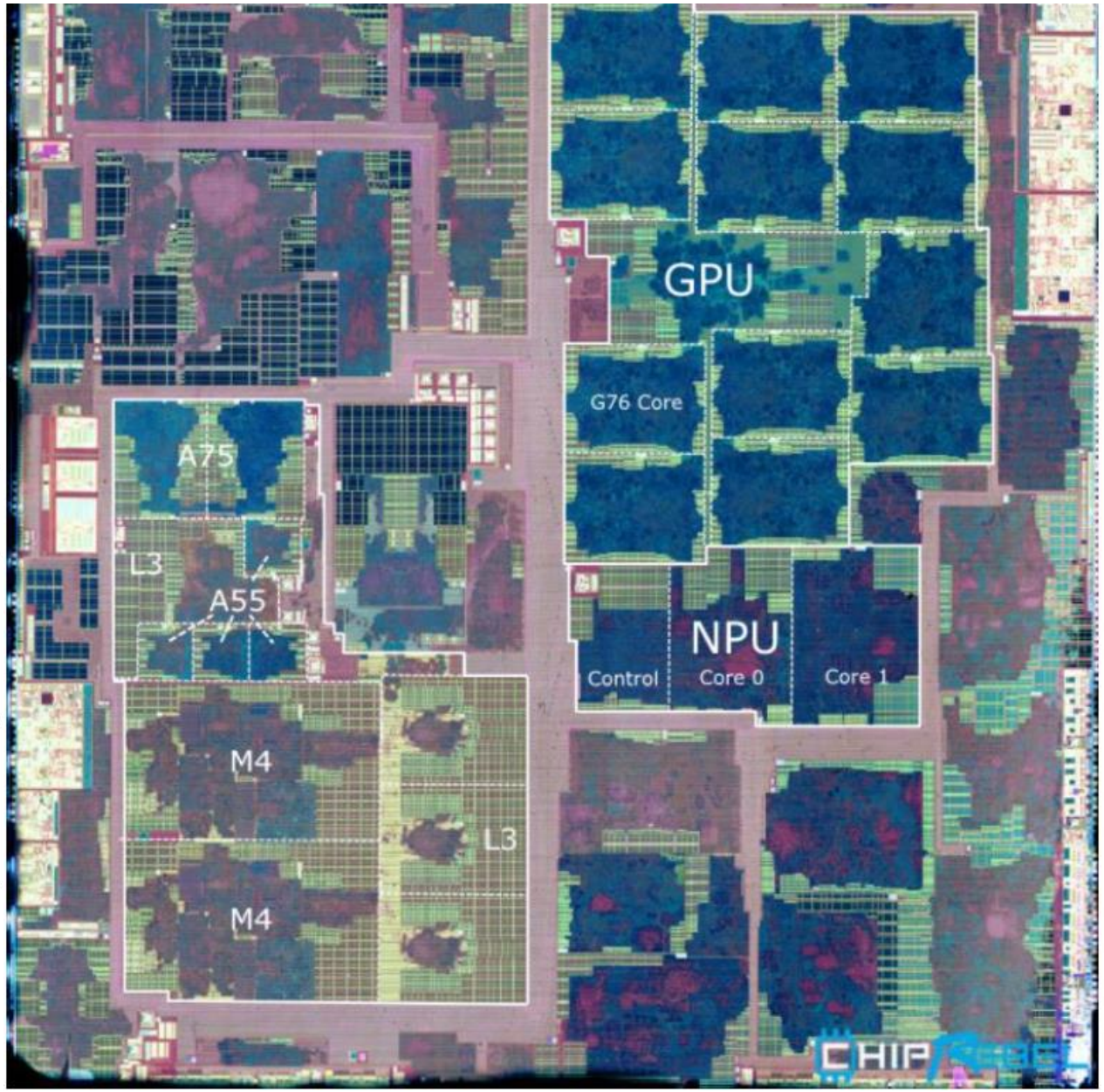
Network Traversal: Feature-Map Forwarding



Die Photo

Process	Samsung 8nm
Area	5.5 mm^2
Voltage	0.5 V ~ 0.8V
Freq.	67 MHz ~ 933 MHz
Best Peak Performance	6.93* TOPS (8b) @ 0.8 V
Best Energy Efficiency	11.5 * TOPS/W (8b) @ 0.5 V

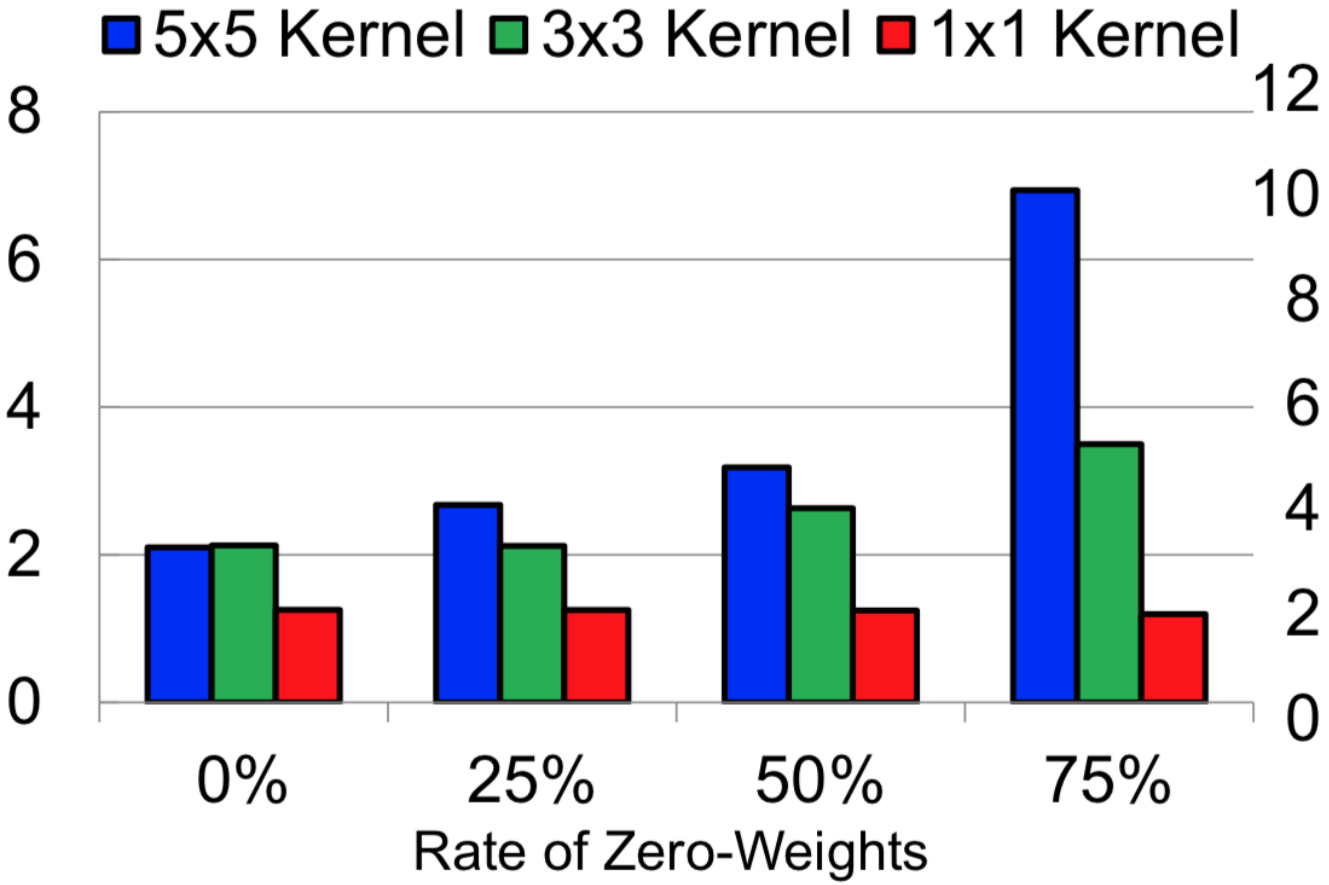
* 75% zero-weights



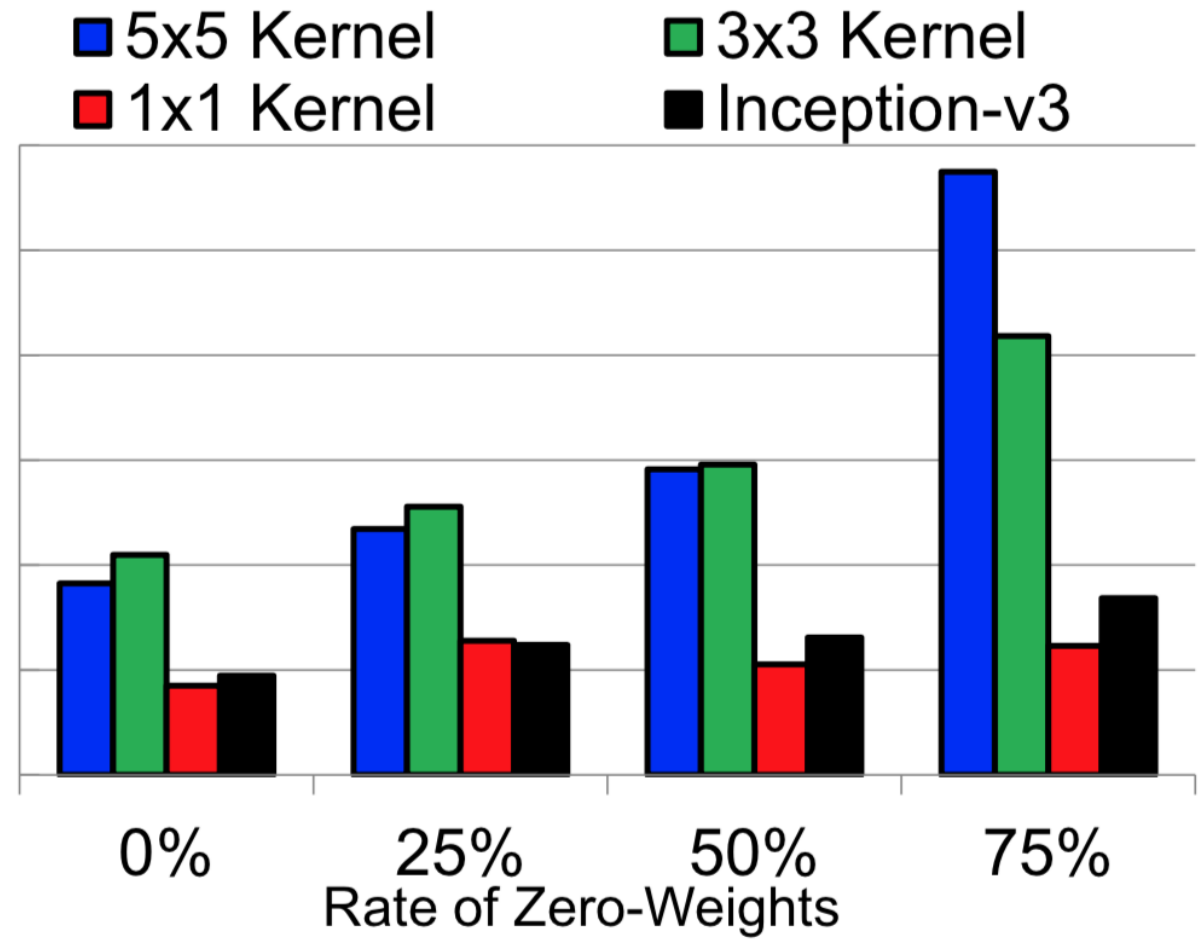
Exynos9820

Measurement Results

Peak Performance (TOPS @ 933 MHz)



Energy Efficiency (TOPS/W @ 0.5V)



Agenda

- Zero-weight skipping accelerator
  - NVIDIA Sparse TensorCore, Samsung NPU v1
- Zero-activation skipping accelerator
  - Samsung NPU v2
- Low precision accelerator
  - NVIDIA TensorCore

A 6K-MAC Feature-Map-Sparsity-Aware Neural Processing Unit in 5nm Flagship Mobile SoC

Jun-Seok Park[1], Jun-Woo Jang[2], Heonsoo Lee[1], Dongwoo Lee[1], Sehwan Lee[2], Hanwoong Jung[2], Seungwon Lee[2], Suknam Kwon[1], Kyungah Jeong[1], Joon-Ho Song[2], SukHwan Lim[1], Inyup Kang[1]

SAMSUNG
Exynos

[1] Samsung Electronics
[2] Samsung Advanced Institute of Technology

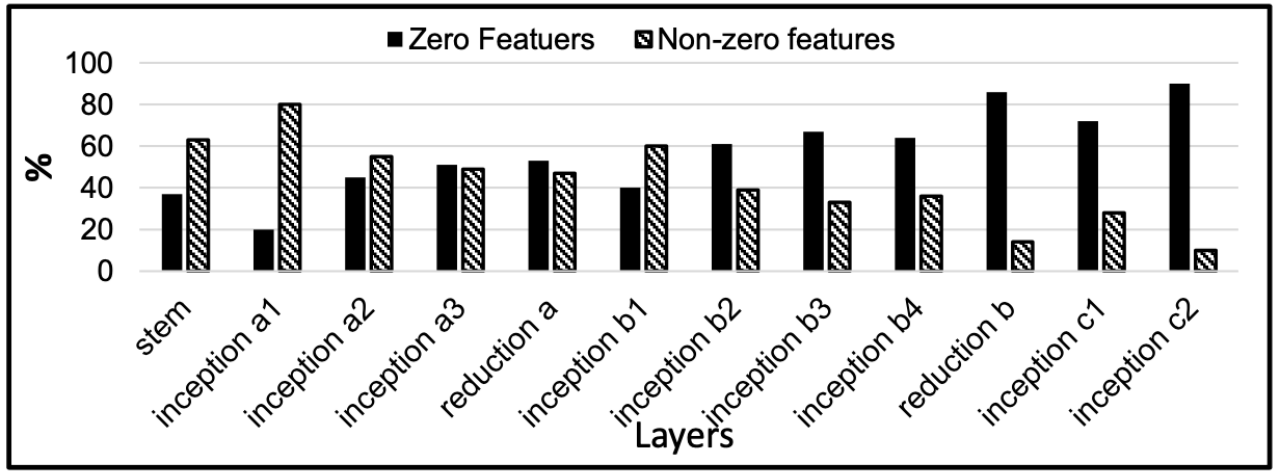
Key points

Challenges

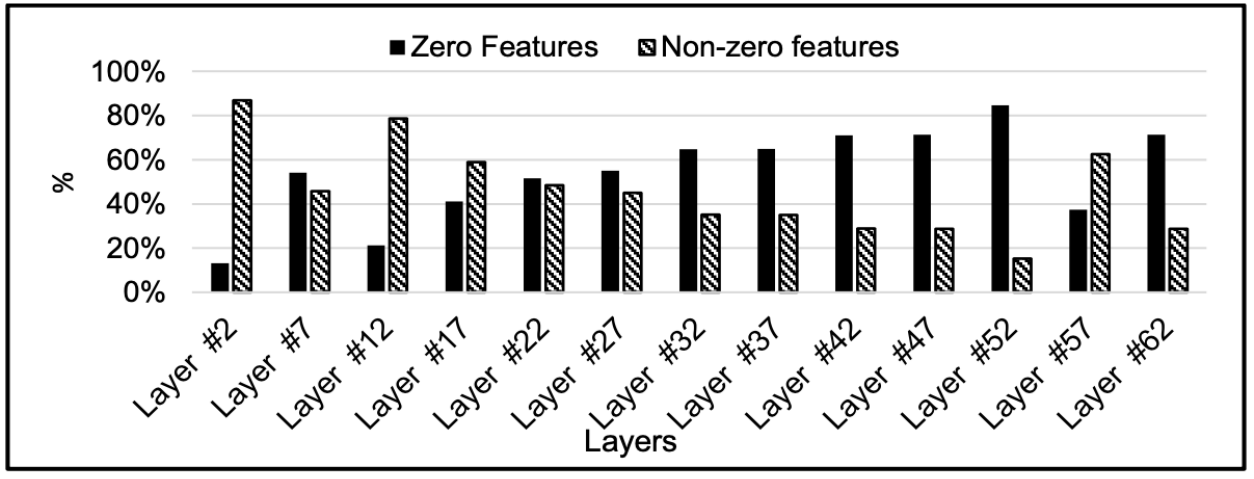
- To maintain a high utilization factor for those diverse convolutions
- To achieve high energy efficiency

Solutions

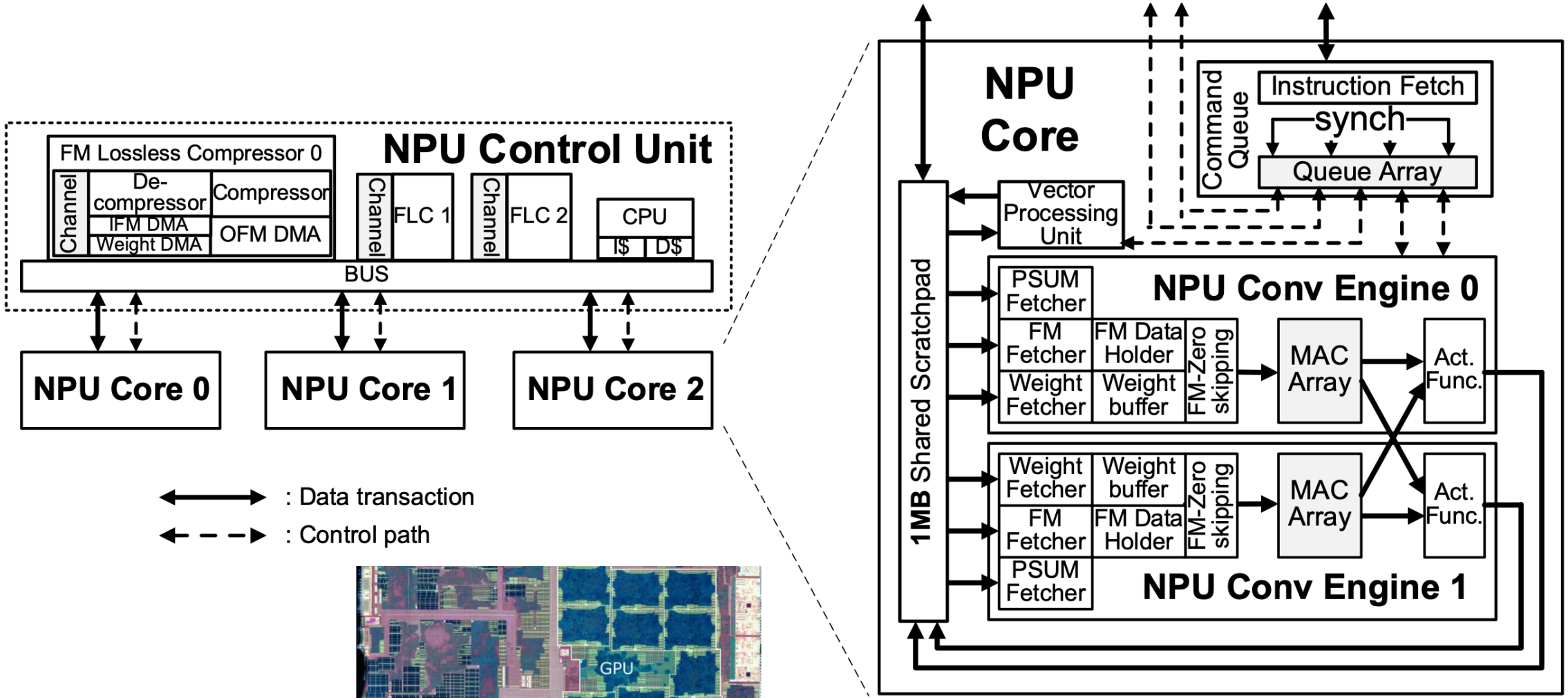
- Serialize the convolutional operations along spatial direction
- Skip redundant operations exploiting sparseness of feature-map.

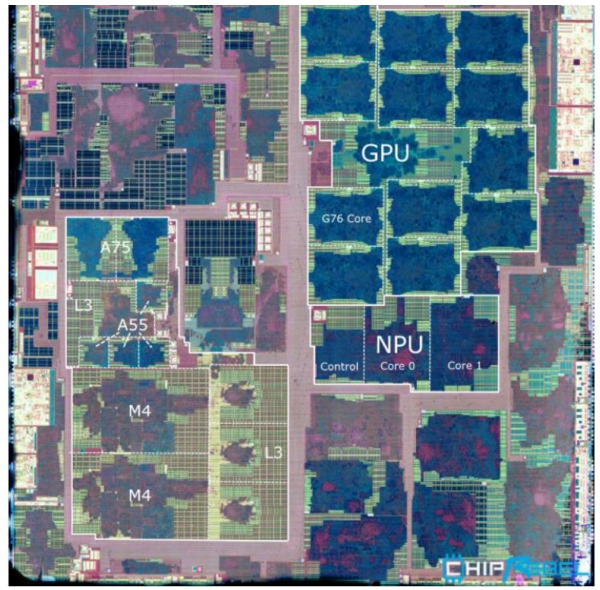


Feature-map Distribution for Neural Layers on inceptionV3



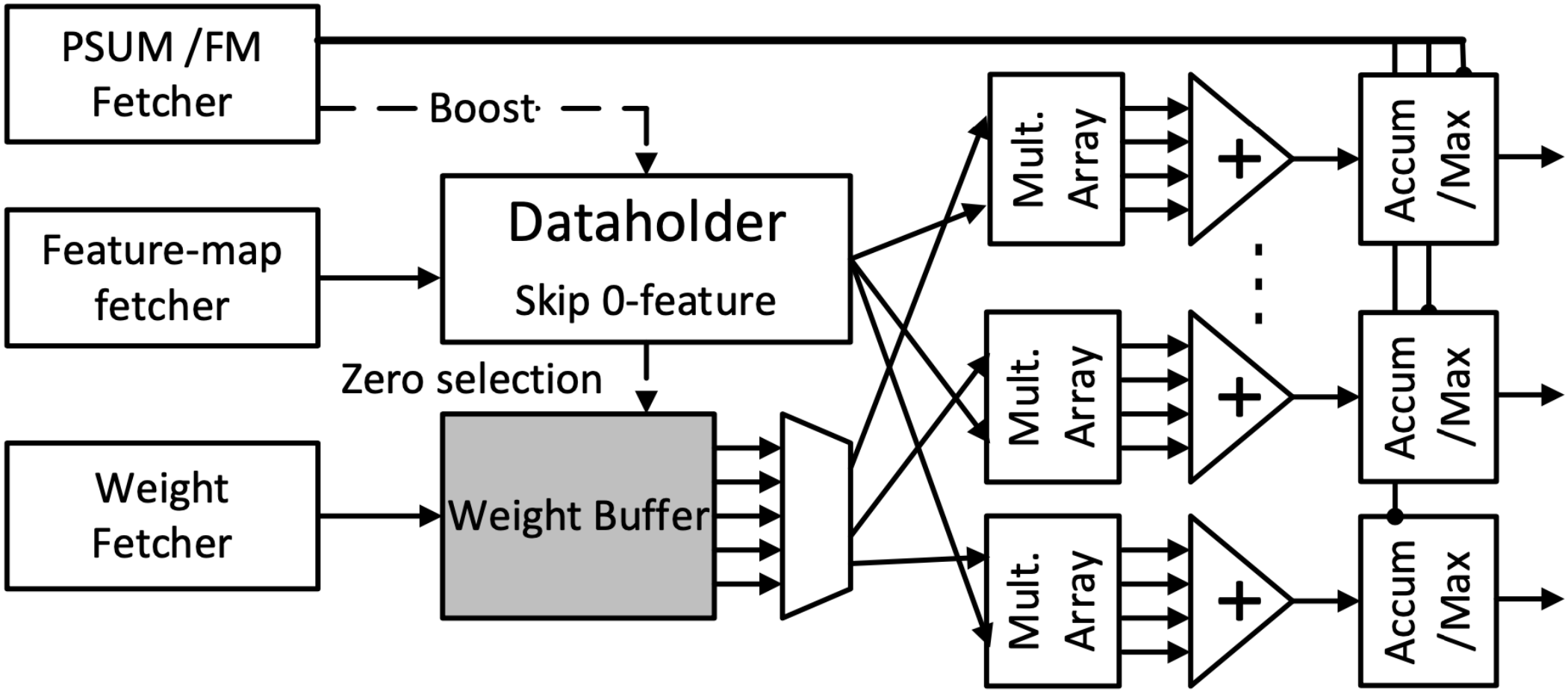
Feature-map Distribution for Neural Layers on DeepLabV3


NPU Control Unit
FM Lossless Compressor 0
Channel
De-compressor
Compressor
IFM DMA
OFM DMA
Weight DMA
Channel
FLC 1
Channel
FLC 2
CPU
I$
D$
BUS
NPU Core 0
NPU Core 1
NPU Core 2
: Data transaction
: Control path
NPU Core
Command Queue
Instruction Fetch
synch
Queue Array
Vector Processing Unit
1MB Shared Scratchpad
PSUM Fetcher
NPU Conv Engine 0
FM Fetcher
FM Data Holder
FM-Zero skipping
Weight Fetcher
Weight buffer
MAC Array
Act. Func.
Weight Fetcher
Weight buffer
FM Fetcher
FM Data Holder
FM-Zero skipping
MAC Array
Act. Func.
PSUM Fetcher
NPU Conv Engine 1


GPU
G76 Core
A75
L3
A55
NPU
Control
Core 0
Core 1
M4
L3
M4

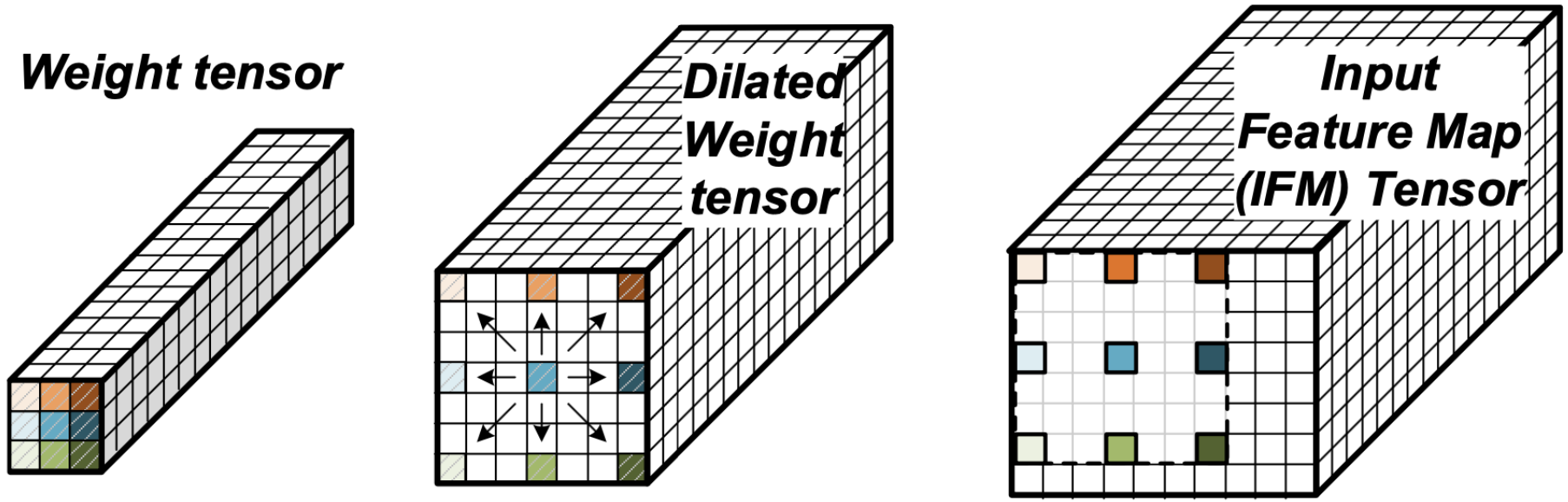
Exynos9820

Convolutional engines (CE)

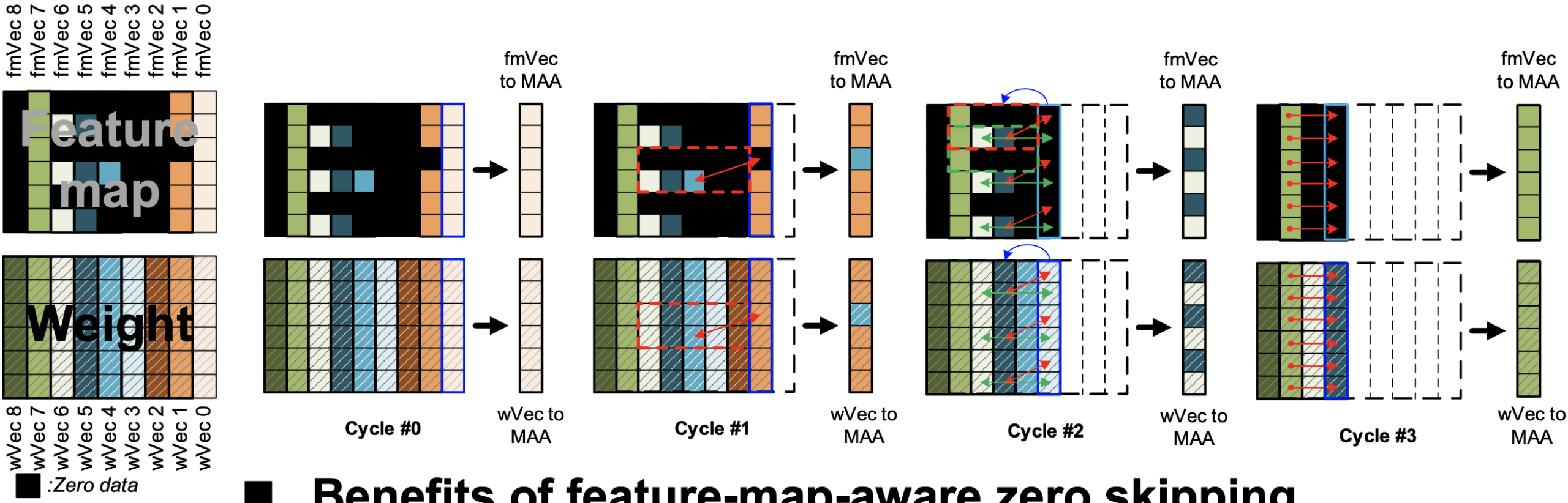


- CE executes 16 dim. data in parallel along the channel
- If the smallest unit of compute is 1x1x16, convolutions with arbitrary kernels is straightforward

Dilated convolution



Feature-map-aware zero skipping

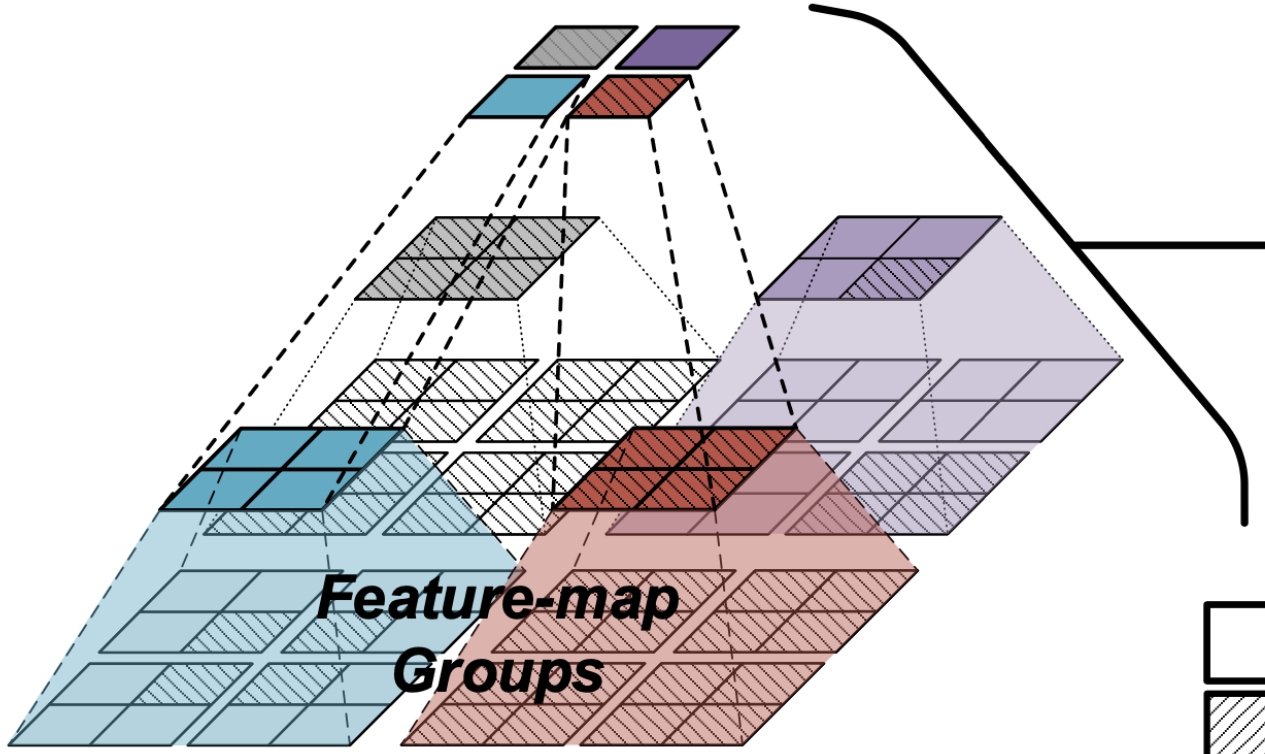


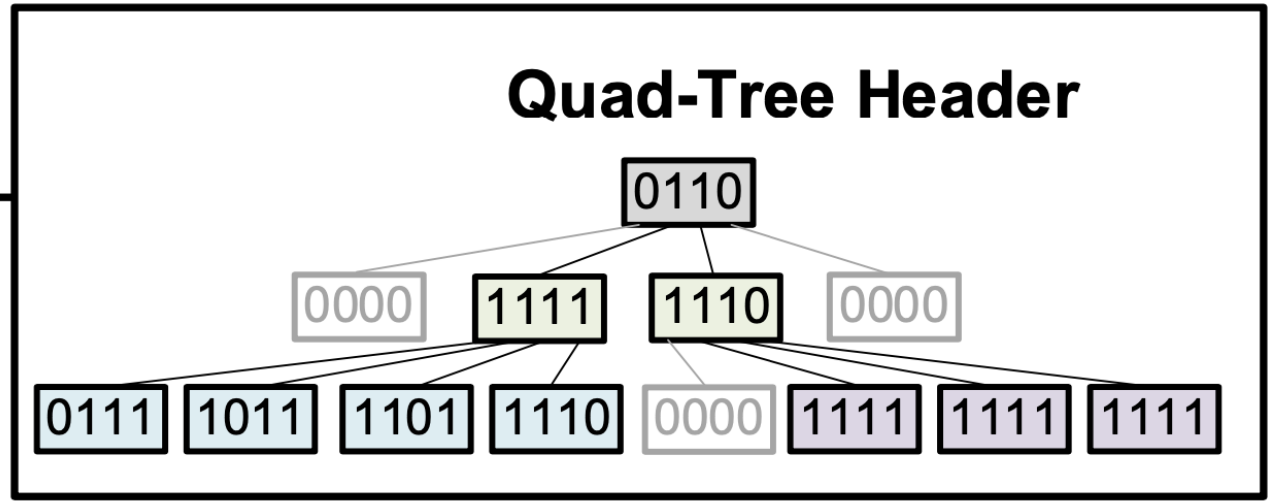
- **Benefits of feature-map-aware zero skipping**
  - Effective performance
  - Energy efficiency
  - HW Utilization

*In a MAC array, 1k MACs perform 64 dot-products of 16-dim vectors

*Zero skipping improves utilization by 36%

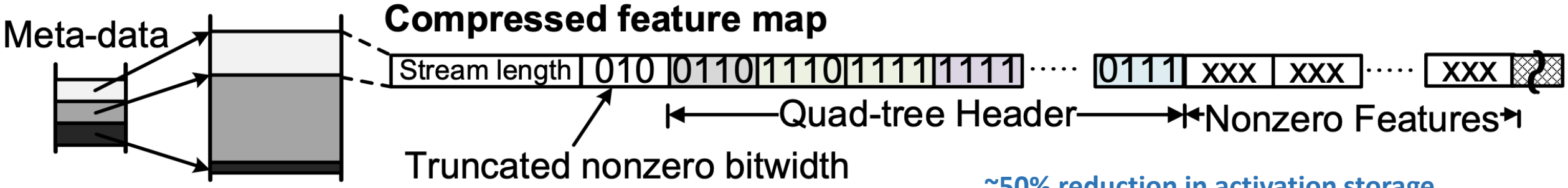
Feature Map Lossless Compressor





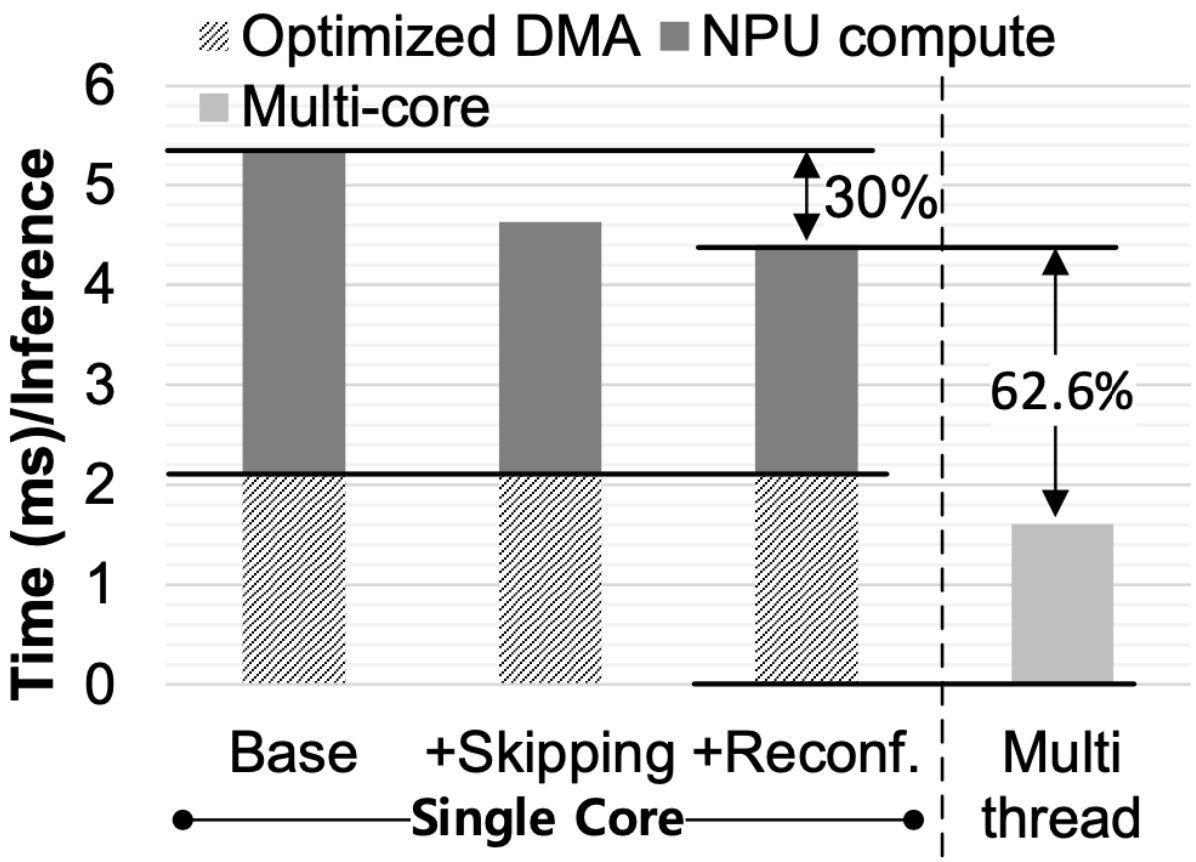
: Non-zero Features

: Zero Features



~50% reduction in activation storage

Key Results



	This work
Process (nm)	5
Area (mm^2)	5.46
Supply Voltage(V)	0.55 – 0.9
Frequency (MHz)	332 - 1196
On-Chip memory (kB)	3,072
Bit Precision	8, 16
The number of MACs	**6,144**
Peak Performance (TOPS)	14.7(8b) @noskip, 29.4(8b) @maxskip
Power (mW)	327 @0.6V, 794 @0.9V
Measured network	Inception V3 (8bit)
Energy efficiency (TOPS/W)	**13.6** (8b) @0.6V
Energy efficiency (mJ/Inference)	**0.840**
Peak TOPS/mm^2	**2.69**

13.6TOPS/W for Inception-V3, where MAC utilization of the NPU reaches 83.8% for the time period when CE is active

Agenda

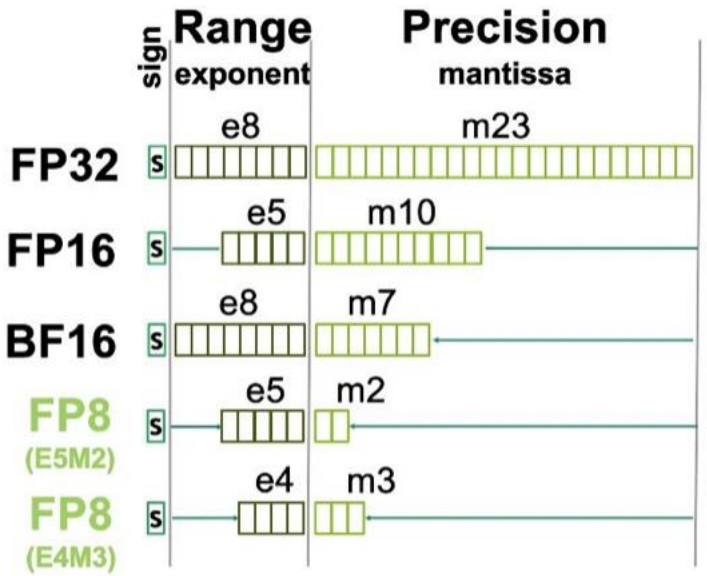
- Zero-weight skipping accelerator
  - NVIDIA Sparse TensorCore, Samsung NPU v1
- Zero-activation skipping accelerator
  - Samsung NPU v2
- Low precision accelerator
  - NVIDIA TensorCore

[https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf]

[https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/]

Precision in Nvidia A100 and H100

- 8-bit floating point (FP8) is new in H100

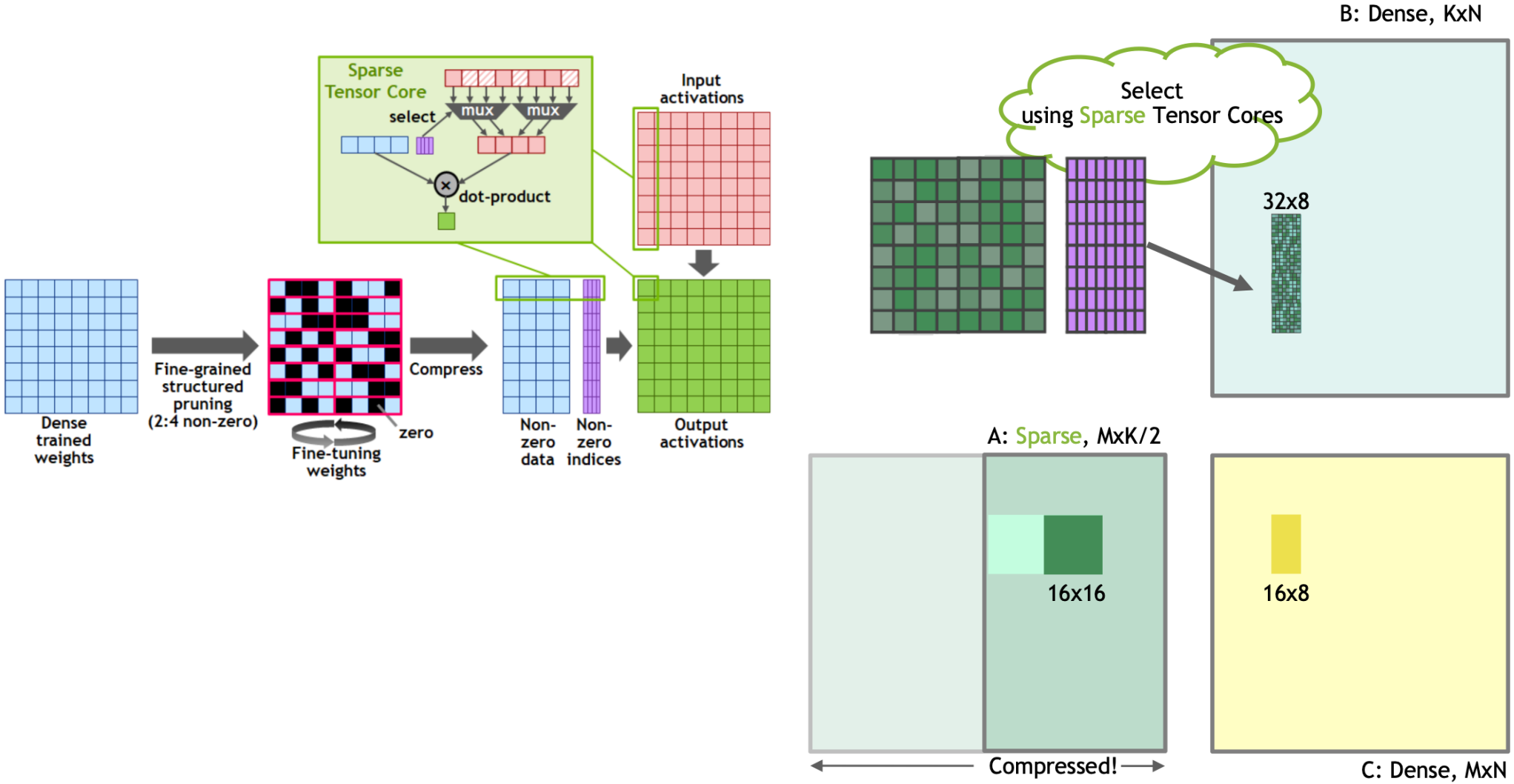


	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32	FP32	15.7	1x	-	-
	FP16	FP32	125	8x	-	-
A100	FP32	FP32	19.5	1x	-	-
	TF32	FP32	156	8x	312	16x
	FP16	FP32	312	16x	624	32x
	BF16	FP32	312	16x	624	32x
	FP16	FP16	312	16x	624	32x
	INT8	INT32	624	32x	1248	64x
	INT4	INT32	1248	64x	2496	128x
	BINARY	INT32	4992	256x	-	-
	IEEE FP64		19.5	1x	-	-

[https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf]

TENSOR CORE OPERATION

Sparse Tensor Cores – Hardware Magic

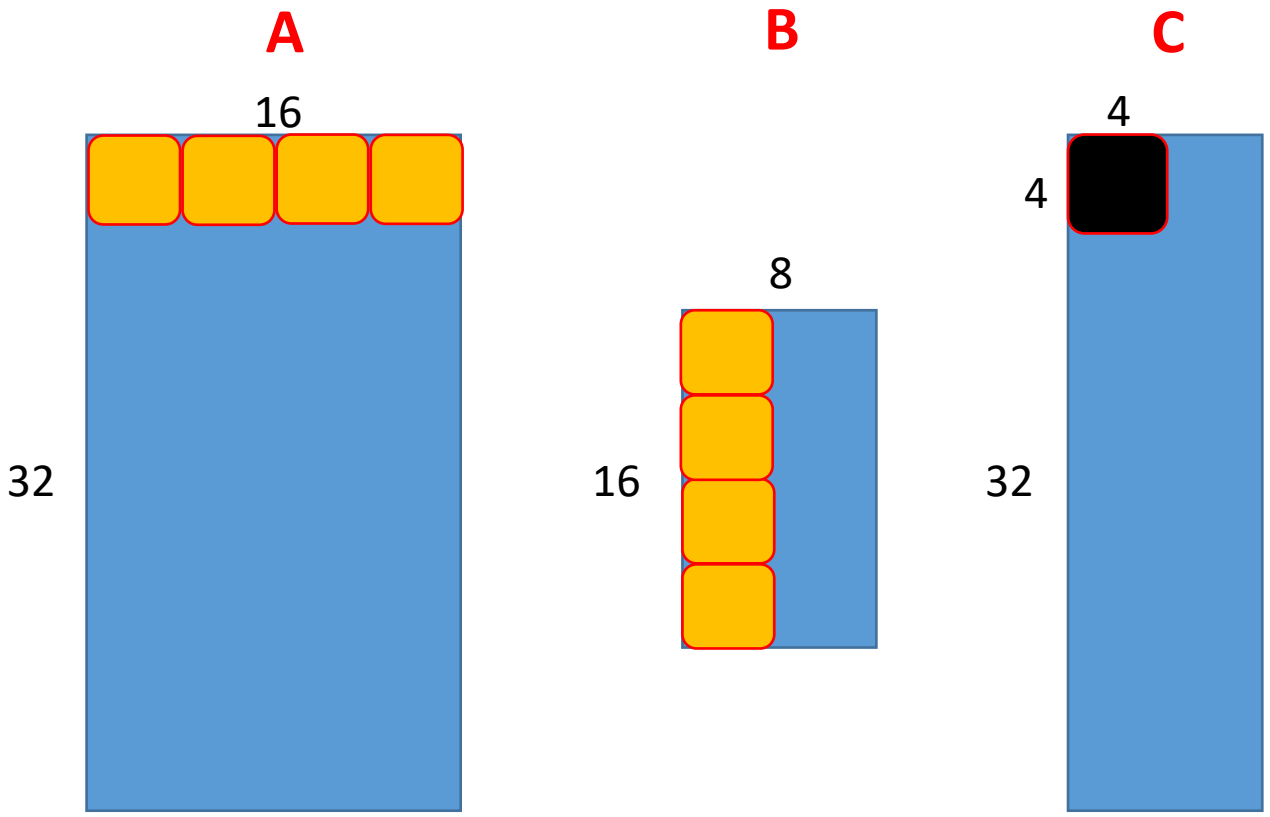


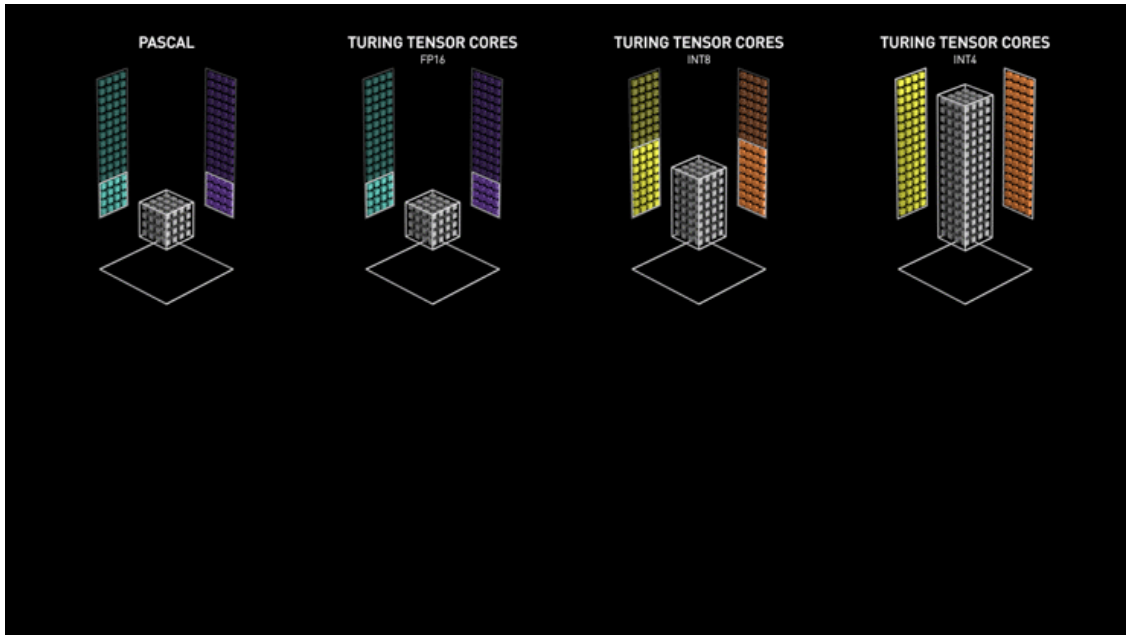
NVIDIA

[https://www.nvidia.com/en-us/data-center/tensor-cores/]

Lower Precision can Further Boost Performance

- Assume fp16 data and Tensor Core takes as input 4 pairs of input tiles and produces a 4x4 output tile in 4 cycles

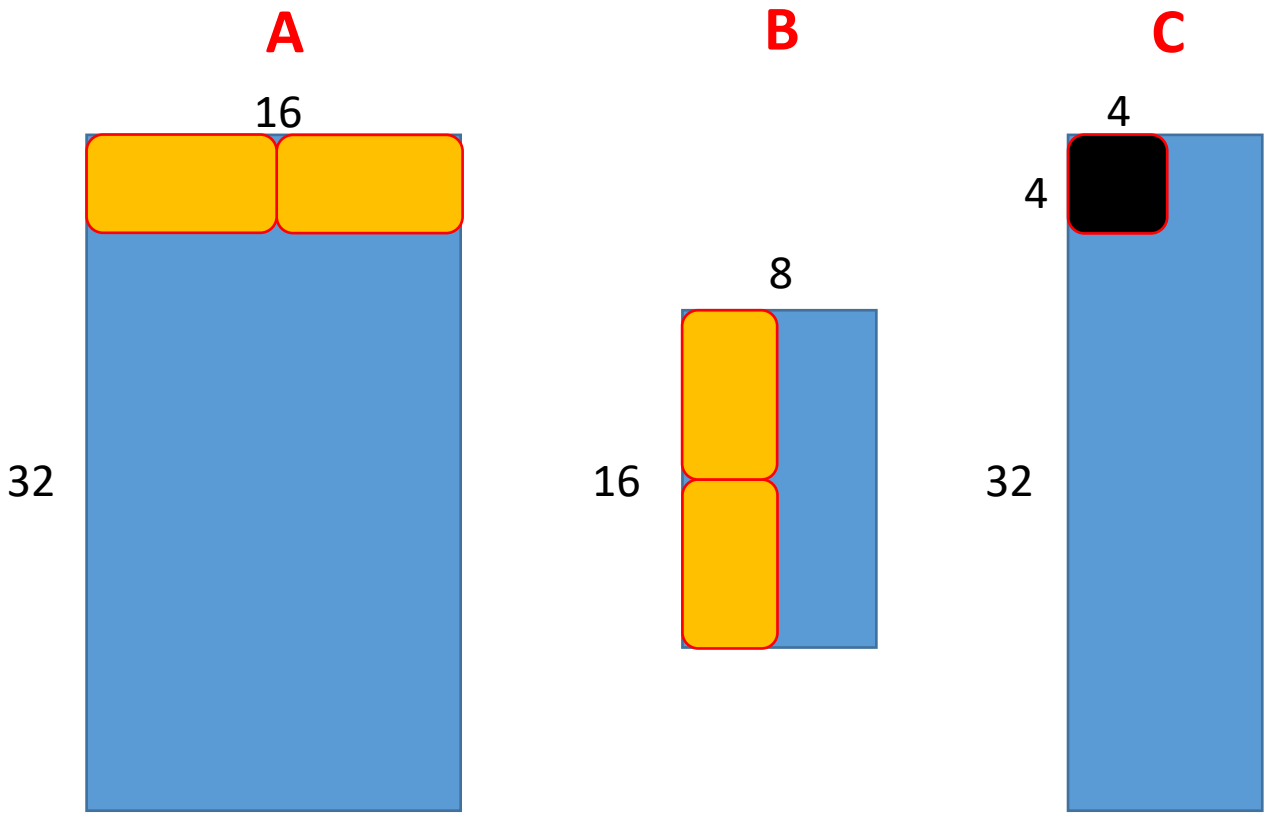


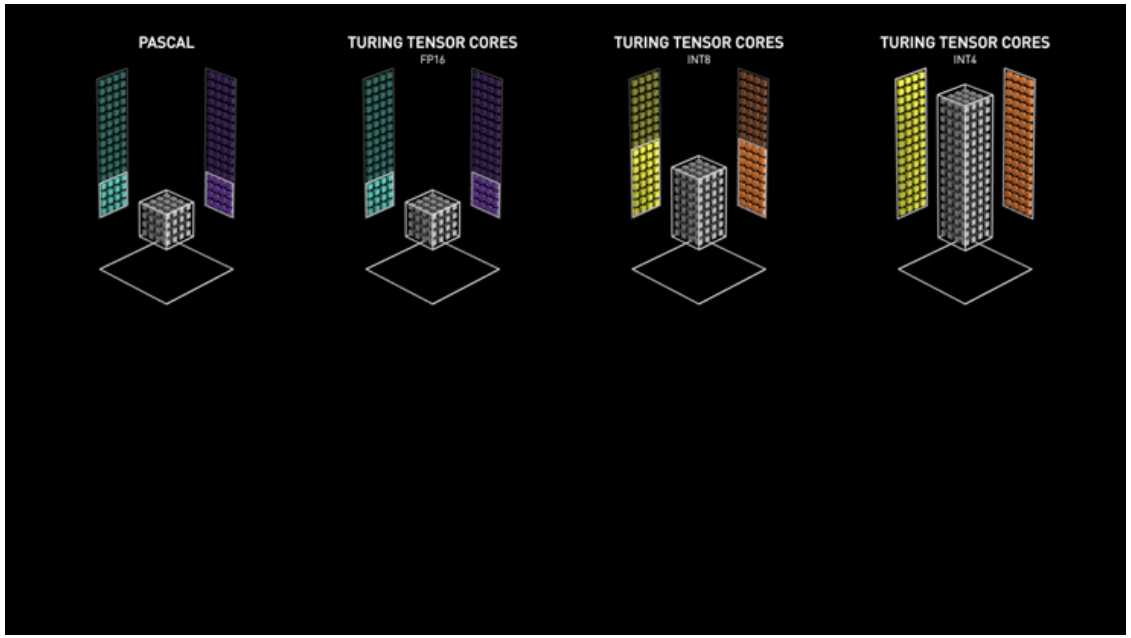


[https://www.nvidia.com/en-us/data-center/tensor-cores/]

Int8 Offers 2X Performance w.r.t. fp16

- Assume int8 data and Tensor Core takes as input 2 pairs of input tiles and produces a 4x4 output tile in 2 cycles

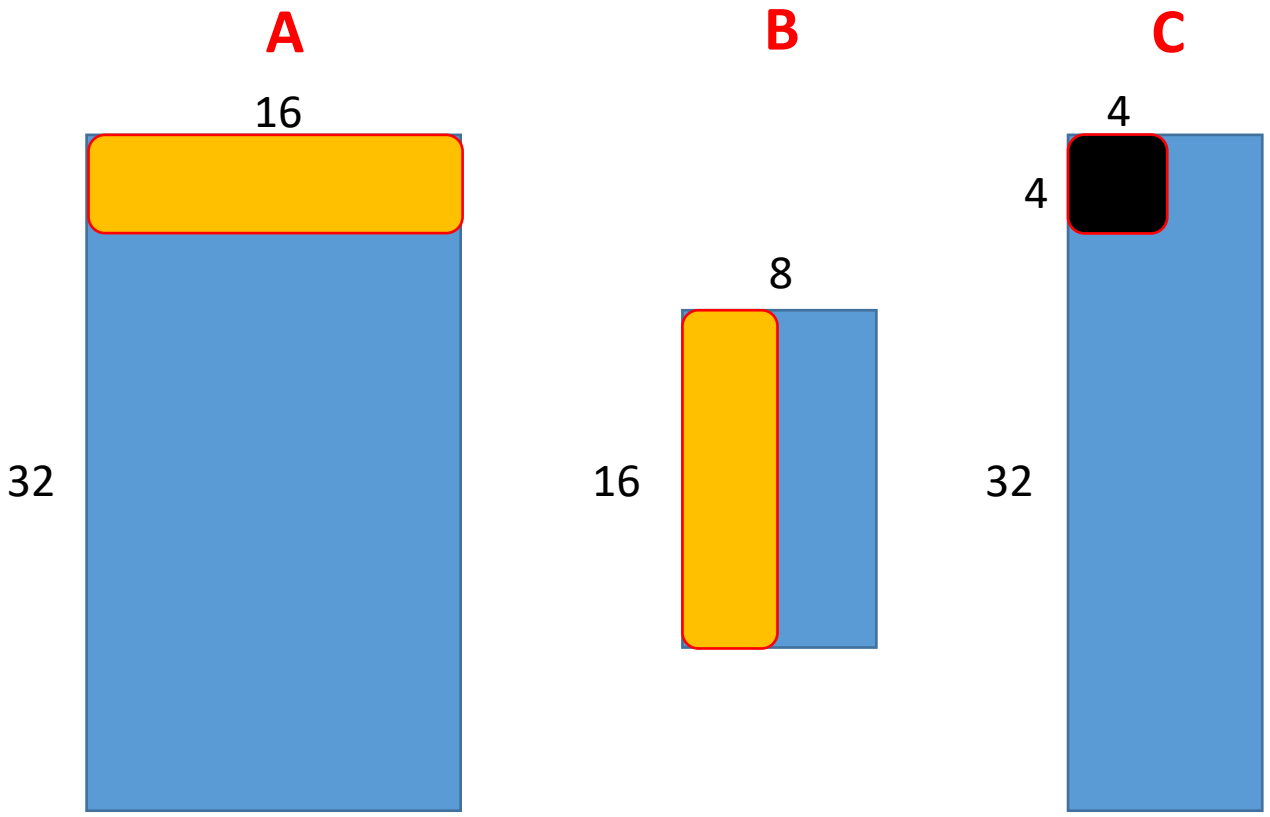


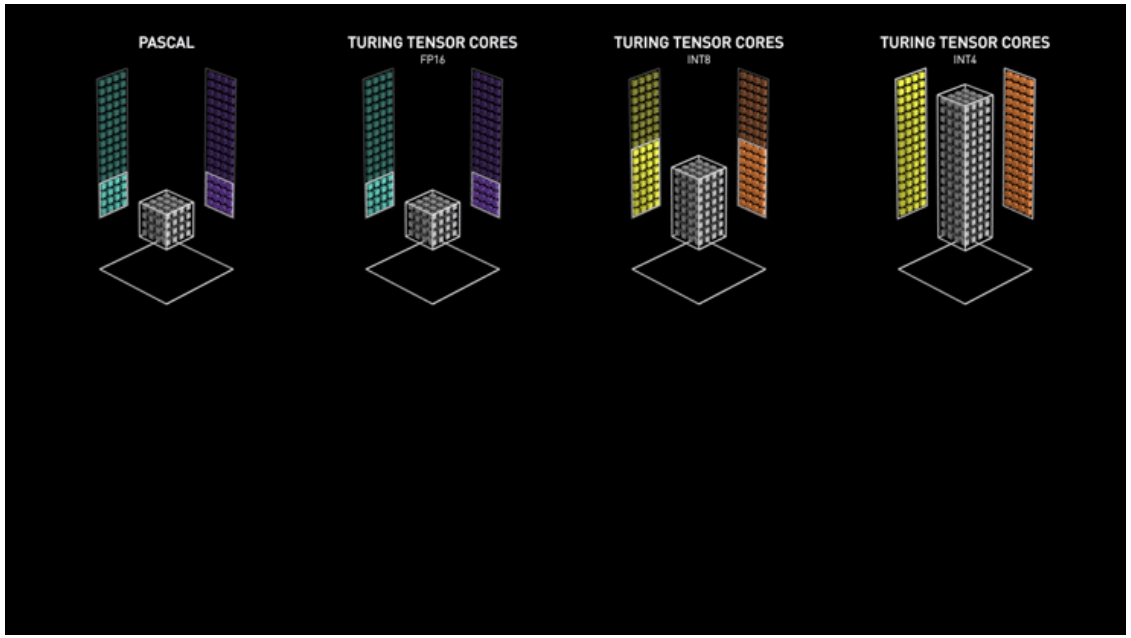


[https://www.nvidia.com/en-us/data-center/tensor-cores/]

Int4 Offers 4X Performance w.r.t. fp16

- Assume int4 data and Tensor Core takes as input 1 pair of input tiles and produces a 4x4 output tile in 1 cycle





[https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf]

IMAGE CLASSIFICATION

ImageNet

Network	Accuracy				
	Dense FP16	Sparse FP16		Sparse INT8	
ResNet-34	73.7	73.9	0.2	73.7	-
ResNet-50	76.6	76.8	0.2	76.8	0.2
ResNet-101	77.7	78.0	0.3	77.9	-
ResNeXt-50-32x4d	77.6	77.7	0.1	77.7	-
ResNeXt-101-32x16d	79.7	79.9	0.2	79.9	0.2
DenseNet-121	75.5	75.3	-0.2	75.3	-0.2
DenseNet-161	78.8	78.8	-	78.9	0.1
Wide ResNet-50	78.5	78.6	0.1	78.5	-
Wide ResNet-101	78.9	79.2	0.3	79.1	0.2
Inception v3	77.1	77.1	-	77.1	-
Xception	79.2	79.2	-	79.2	-
VGG-16	74.0	74.1	0.1	74.1	0.1
VGG-19	75.0	75.0	-	75.0	-

NVIDIA.

Weekly Lecture / Lab Schedule

- W1 (March 6) Class introduction / (March 8) Orientation & team formation
- W2 13 Verilog 1 Combinational circuits / 15 Verilog 1 (tool installation, adder & multiplier combinational logic)
- W3 20 Verilog 2 Sequential circuits / 22 Verilog 2 (memory i/o, FSM sequential logic)
- W4 27 AI application introduction 1, Amaranth introduction 1 / 29 Amaranth (tool installation, MAC, adder tree)
- W5 4/3 AI application introduction 2, Amaranth introduction 2 (memory i/o, FSM sequential logic) / 5 Amaranth (PE)
- W6 10 AI application introduction 3, Neural network accelerator 1 / 12 Amaranth (PE controller)
- W7 17 Neural network accelerator 2 / 19 Q&A
- W8 24 **Mid-term exam** / 26 Amaranth (outer product accelerator)
- W9 5/1 Reading data from memory 1 (VA2PA, interconnect) / 3 PyTorch (simple CNN on MNIST)
- W10 8 Reading data from memory 2 (DRAM main memory) / 10 Quantization aware training (QAT)
- W11 15 Compressing networks (pruning, low precision) / 17 Convolution lowering & tiling
- W12 22 Compressing networks (pruning, low precision) / 24 PyTorch – Amaranth communication
- W13 29 Invited talk (1h online Google Edge TPU), Zero-skipping & low-precision hardware accelerator / 31 Zero skipping
- W14 6/5 **Final exam** / 7 Project Q&A
- W15 12 (online) Claim & Project Q&A / 14 (online) Project Q&A, submission

Final Exam

- 9am – 10:50am, June 5th
- Closed book
- All the slides (week 1 ~ 13) explained in the lectures
  - Excluding Verilog language parts in lectures
- All the contents covered in the practices

Possible Problems for Final Exam (NOT Limited to These) Will be Useful for Key Ideas Review: Page 1

- Explain why a layer in MLP performs matrix vector multiplication
- Explain how convolution is converted to matrix multiplication
  - Matrix size, …
- Explain how tiling can improve the speed of matrix multiplication when main memory access is slow
- Complete the given Verilog code of systolic array PE
- Complete the given Verilog code of systolic array structure
- Explain how output stationary mode (A&B input) in systolic array
- Explain how weight stationary mode (A input) in systolic array
- Calculate how many cycles it takes to obtain an output tile of matrix multiplication via tile pipelining
- TPUv1: Explain what a hardware design of im2col looks like
- TPUv4i: Explain systolic array design difference w.r.t. TPUv1
- TPUv4i: What is the utility of large on-chip memory?
- Explain an outer product based MM accelerator with 4x4 PEs
- Explain how the 4x4 PEs can be used to realize TensorCore
- Calculate model size & # multiplications in Transformer model
- Compare the amount of data transfer btw 2 GPUs on data and model parallelism
- Explain how MicroSoft ZeRO works
- Explain how a layer can be executed in parallel on Tesla supercomputer
- Explain how zero-skipping in matrix multiplication is performed on Cerebras CS-2
- A few Amaranth coding problems
  - Functionality and waveform problems
  - Code writing problems, e.g.,
    - A functionality description is given, write an Amaranth code
    - An example Verilog code is given, write an equivalent code in Amaranth

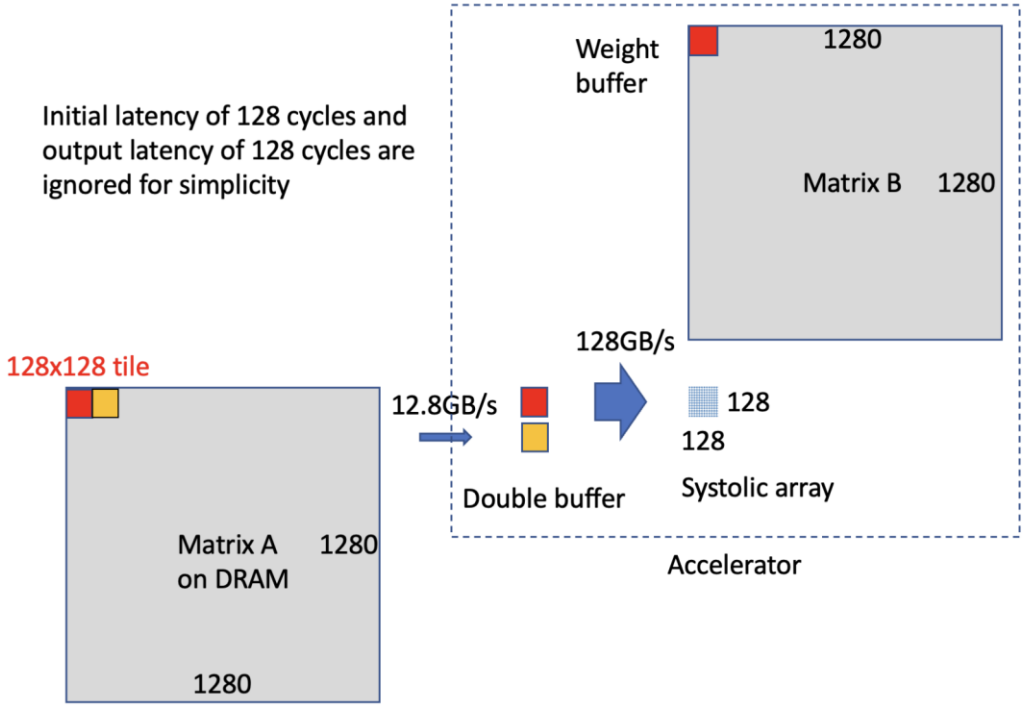
Possible Problems for Final Exam (NOT Limited to These) Will be Useful for Key Ideas Review: Page 2

- Explain virtual to physical address translation
- What is IOMMU?
- Communication between hardware components on request and data channels
- Multiple outstanding requests
- Basic DRAM operations
- Memory access scheduling
- Address mapping for bank parallelism
- DRAM refresh overhead, per-bank refresh
- Row hammer problem and solutions
- Error correction code, how to detect and locate error in [7,4] code?
- Partial write on in-DRAM ECC
- Compressed sparse row (CSR) to store non-zero weights in pruned models
- Clustering non-zero weight values
- What is column-wise pruning?
- Explain NVIDIA's 2:4 pruning and optimization method
- Explain how NVIDIA's Sparse TensorCore reduces computation cost (i.e., # multiplications)
- What is FP8 number?
- What is the FP8 (E4M3) representation of a real value, 447?
- What is fake quantization?
- What is straight-through estimator (STE)?
- Show the clipping threshold of PACT is trainable, i.e., the partial derivative of training loss (L) with respect to the threshold α can be calculated
- Show # bits (B) of DiffQ is trainable
- Show that the binarization of weight gives binary weight matrix (of the sign of the original values) and a scale of average weight magnitude
- Explain Samsung's zero-weight-skipping accelerator
- Explain Samsung's zero-activation-skipping accelerator

’22 Final Exam

Q1. Accelerator (15 points)

The following figure illustrates the systolic array (SA) accelerator (corresponding to the dashed box), explained in our lecture, which is used to multiply matrix A (stored in DRAM) and matrix B (stored in the weight buffer inside of the accelerator). The matrix size is 1280x1280 and each element has 1 byte. We have the same baseline architecture where DRAM bandwidth is 12.8GB/s, SA of 128x128 runs at 1GHz in a pipelined manner, the double buffer size is each 128x128, and the initial and output latency of SA is ignored for simplicity. Each input tile is reused 10 times inside of accelerator and we ignore the latency for the transfer of partial sums, i.e., we assume the partial sum buffer (not shown in the accelerator) can be accessed with no latency.

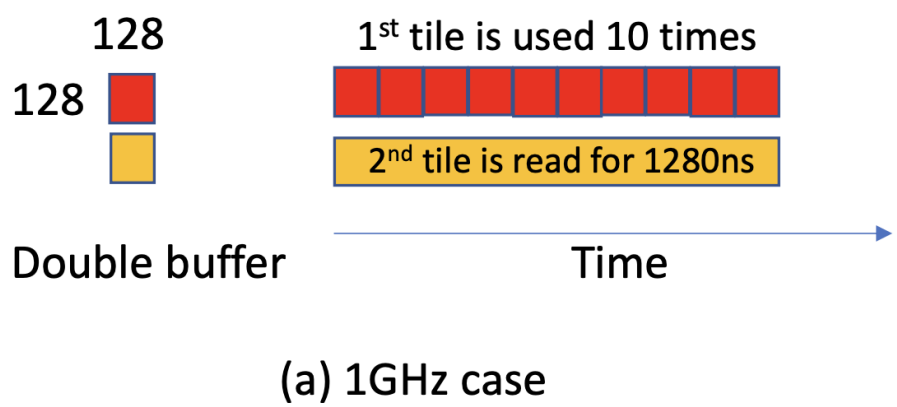


Q1-1. Calculate the total latency (in nano-seconds) of matrix multiplication to obtain the entire output matrix. (10pts)

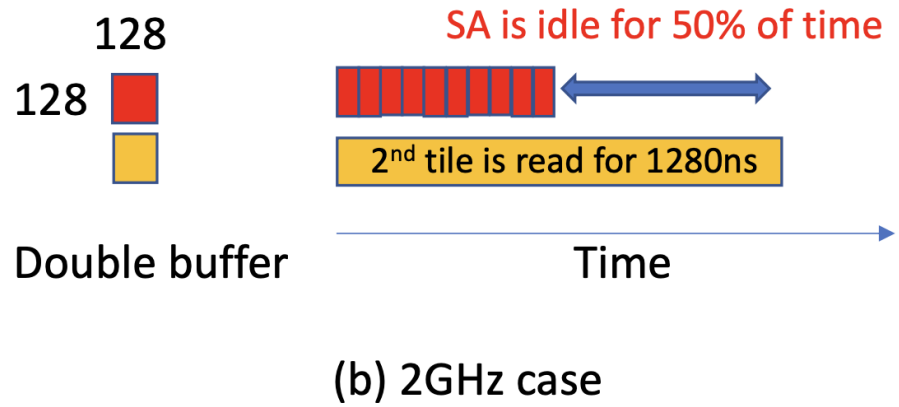
[Answer]
cycles = total # multiplications / # SA multiplications = 1280x1280x1280 / 128x128 = 128,000 nano-seconds (1 cycle @ 1GHz = 1 nano-second)

Q1-2. Assume the SA runs at 2GHz and calculate the total latency of matrix multiplication (2pt). Explain the difference between 1GHz and 2GHz operations (3pts)

[Answer]
The same as the runtime of problem 1-1. (2pt)
As the following figures show, the runtime of 2GHz is determined by memory access time, i.e., memory bandwidth, which is the same as in problem 1.1. (3pt)



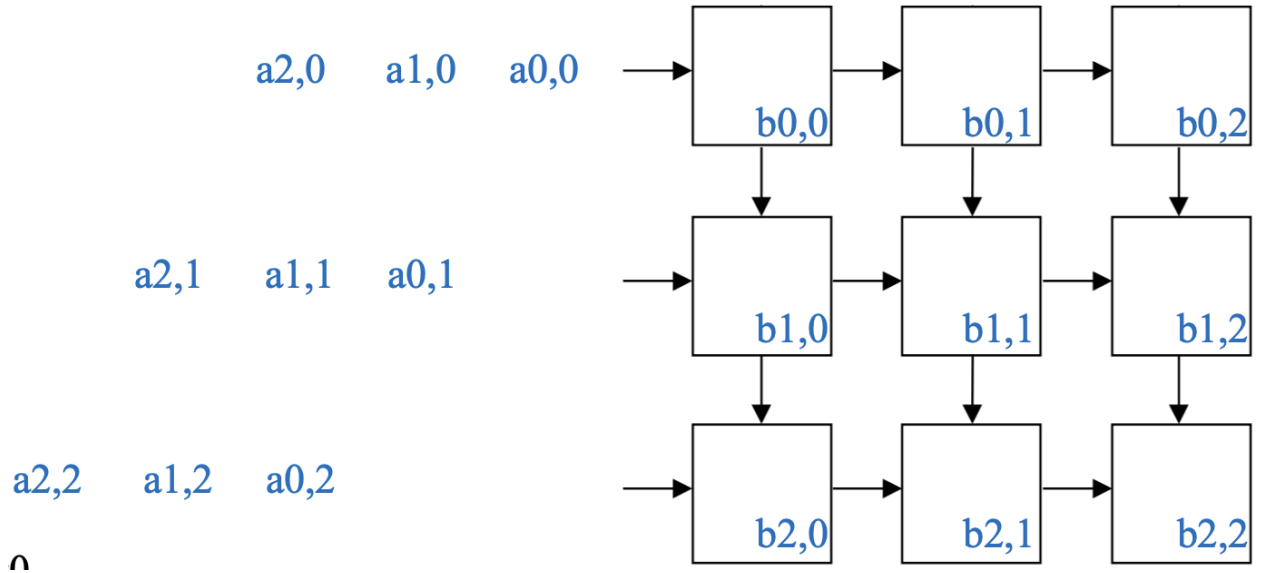
(a) 1GHz case

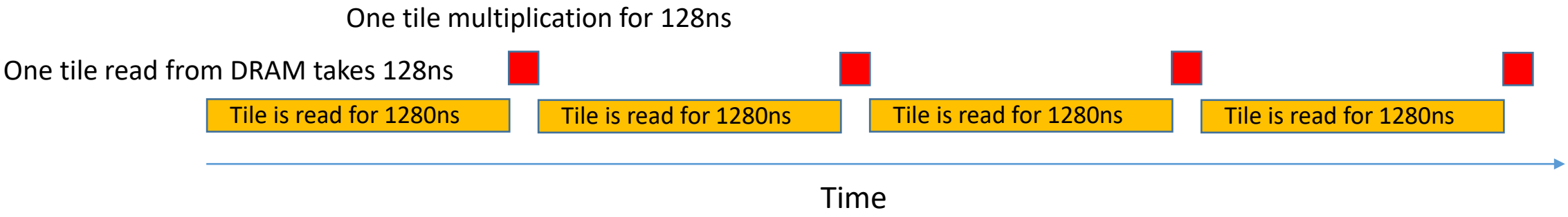


(b) 2GHz case

Bandwidth Requirement of Systolic Array

- Every cycle (1GHz), 128 byte input is needed
  - 128 bytes x 1GHz = 128GB/s
- DRAM bandwidth = 12.8GB/s
- If the DRAM data is not reused, the SA utilization will be 10%

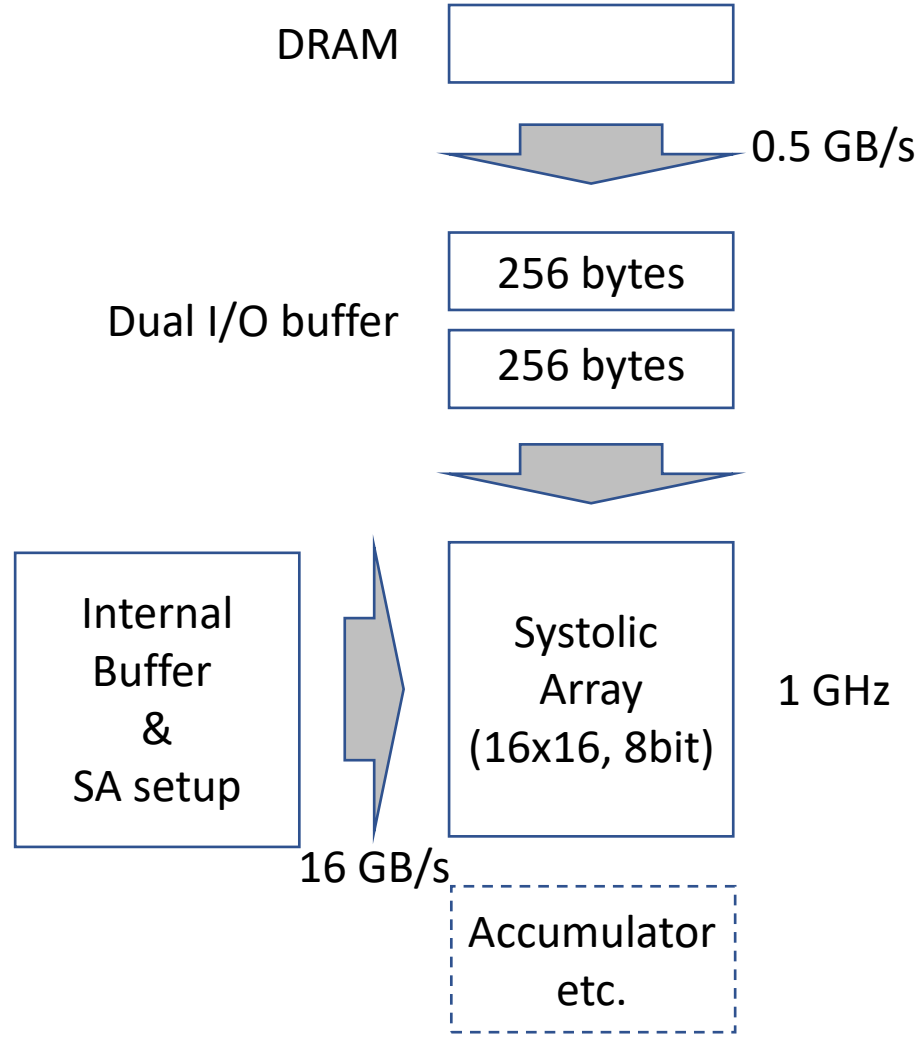




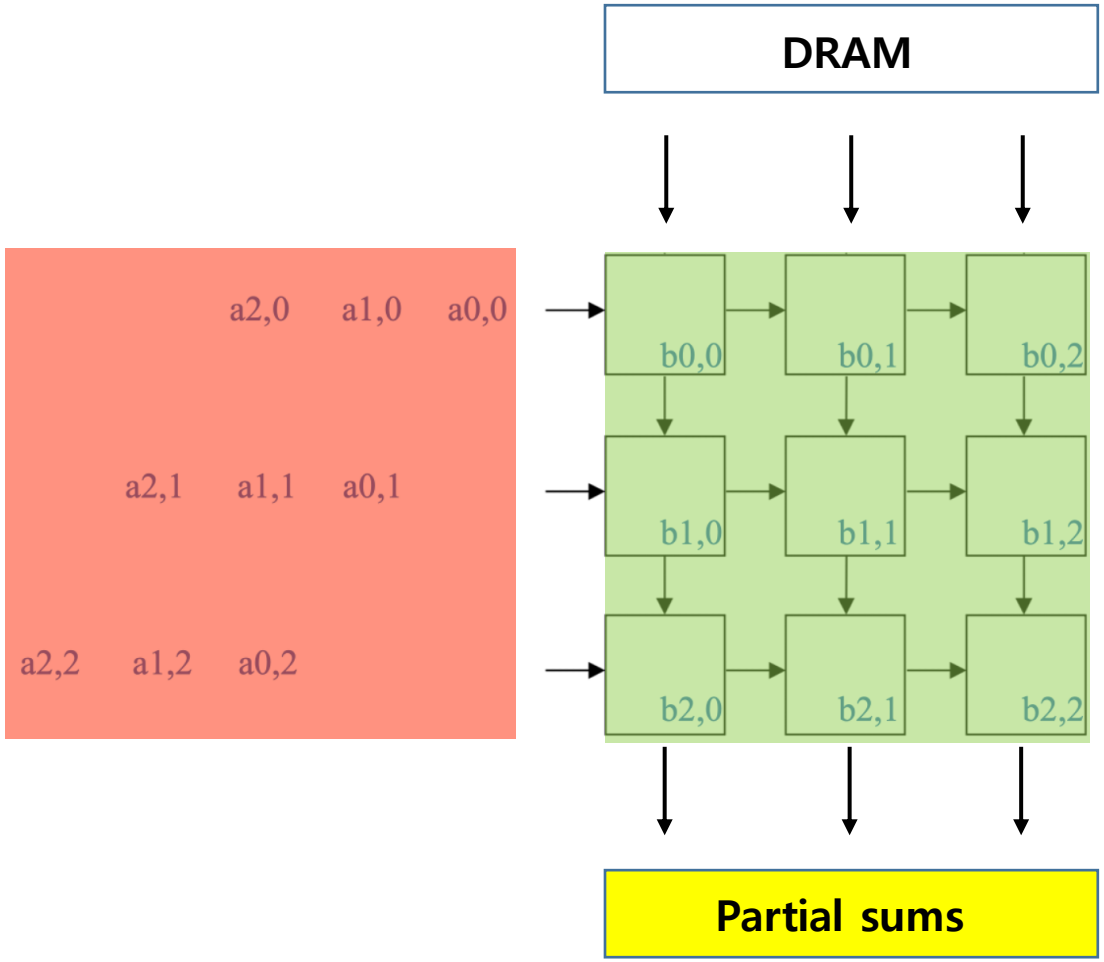
OPS/Byte (# Operations/Byte)

- Goal
  - Hiding memory access latency with computation
- Double buffer alone is not enough

- OPS/byte matters
  - Computation (# multiplications) / amount of data from slow memory needs to be increased!
  - OPS/byte can be increased by reusing data read from slow memory in computation

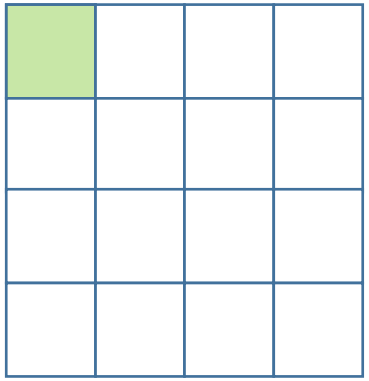


Tiling in Weight Stationary Mode

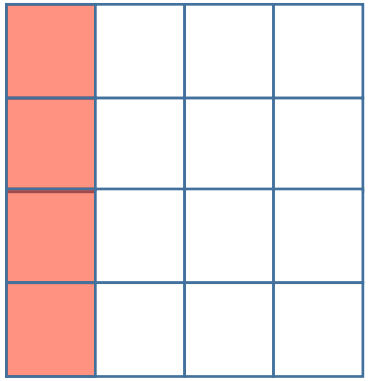


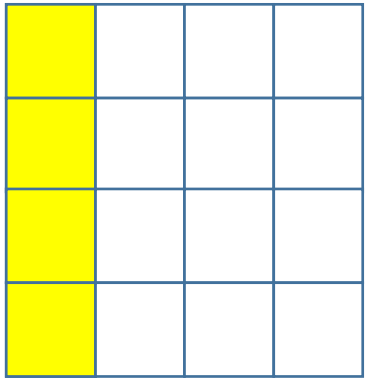
Read a tile from DRAM

Matrix B
(DRAM)



Matrix A
(Internal buffer)



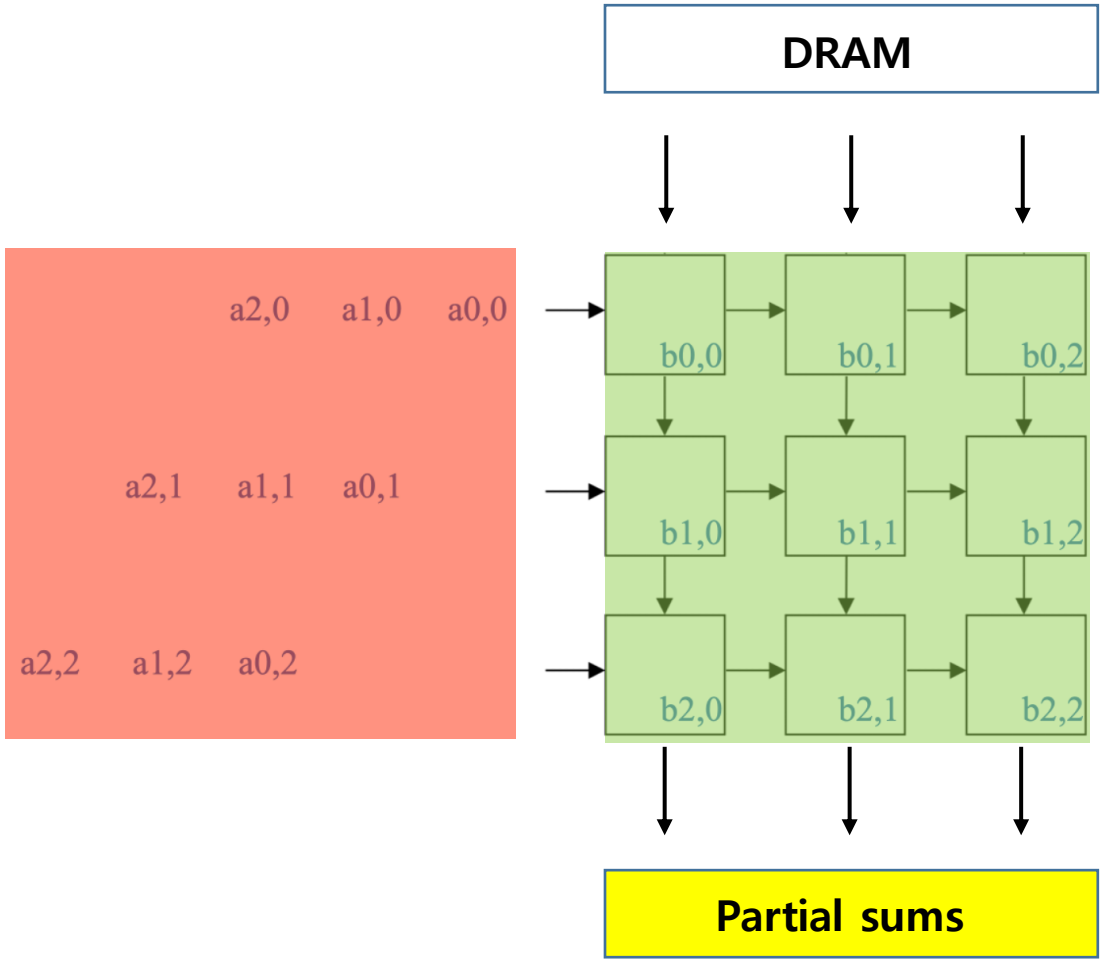


Matrix C

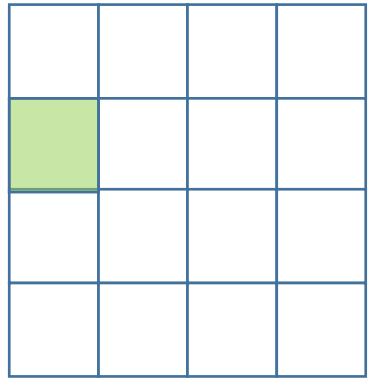
Partial sums for multiple output tiles are assumed to be managed by large accumulators and buffers

Tiling in Weight Stationary Mode

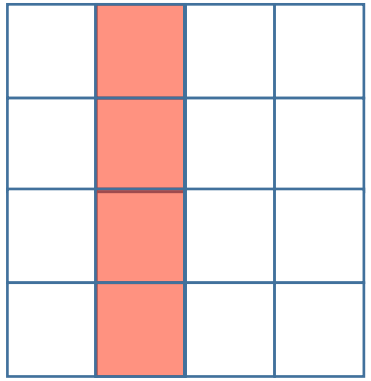
Each tile from DRAM is reused 4X
i.e., OPS/byte gets increased 4X

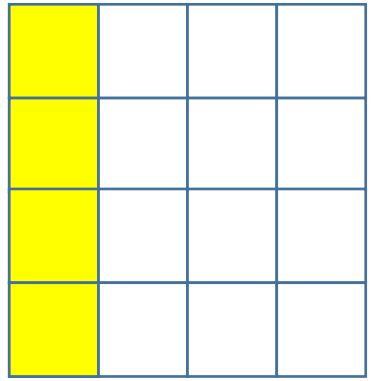


Matrix B
(DRAM)



Matrix A
(Internal buffer)



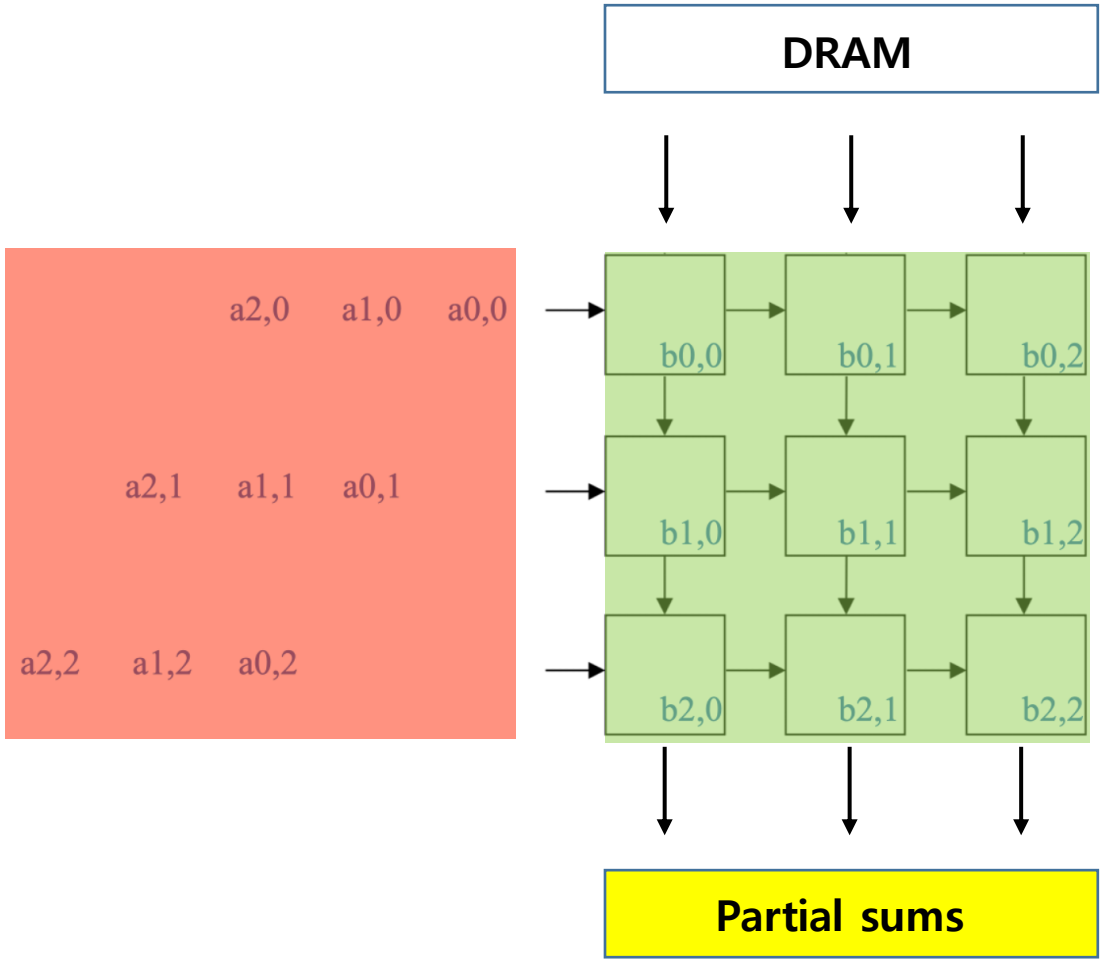


Matrix C

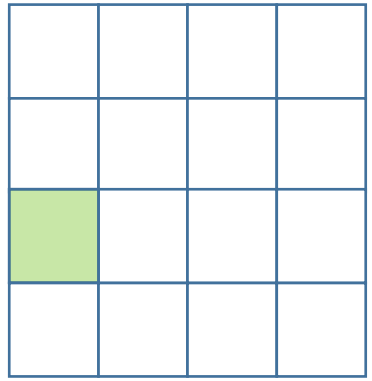
Associated tiles are read from the fast internal buffer

Tiling in Weight Stationary Mode

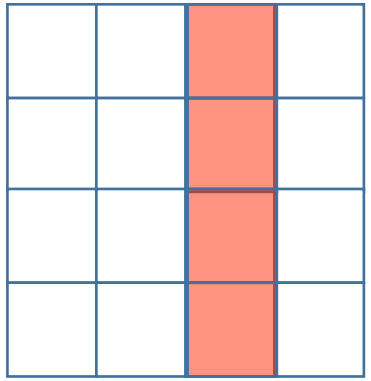
Each tile from DRAM is reused 4X
i.e., OPS/byte gets increased 4X

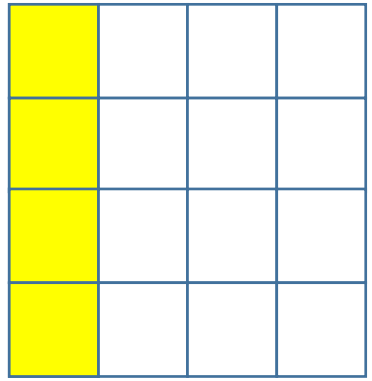


Matrix B
(DRAM)



Matrix A
(Internal buffer)



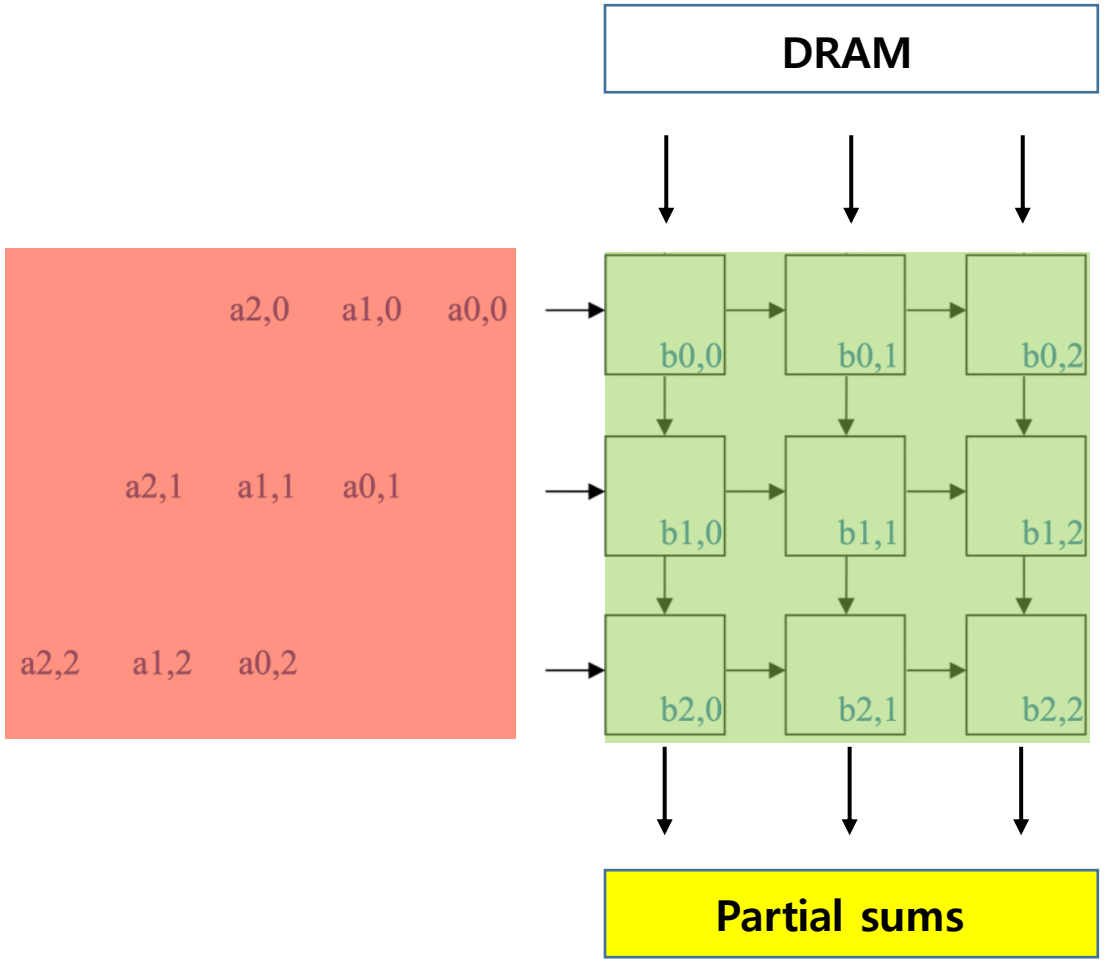


Matrix C

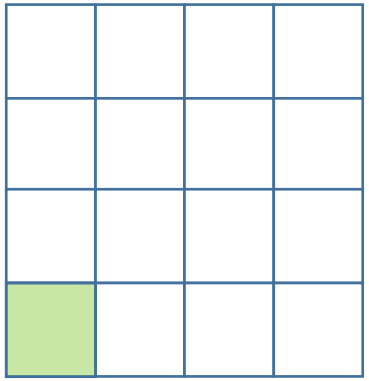
Associated tiles are read from the fast internal buffer

Tiling in Weight Stationary Mode

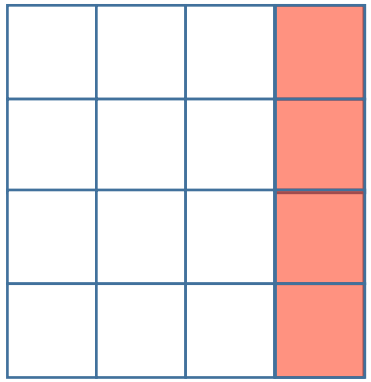
Each tile from DRAM is reused 4X
i.e., OPS/byte gets increased 4X



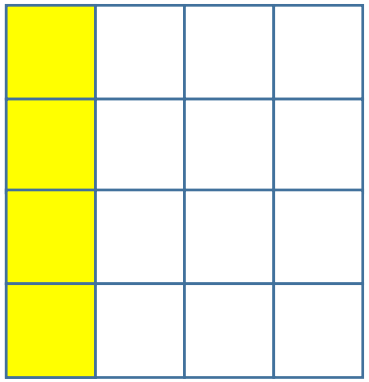
Matrix B
(DRAM)



Matrix A
(Internal buffer)



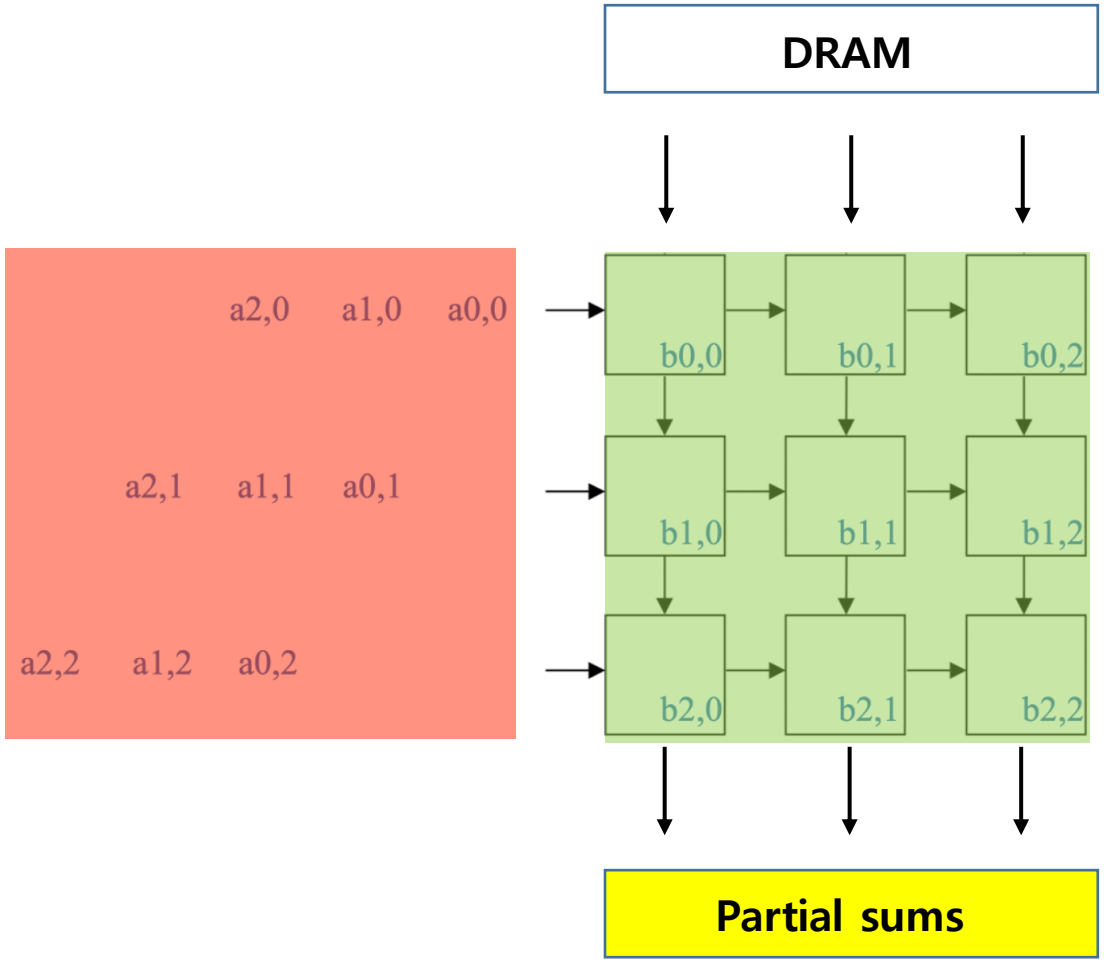
Associated tiles are read from the fast internal buffer



Matrix C

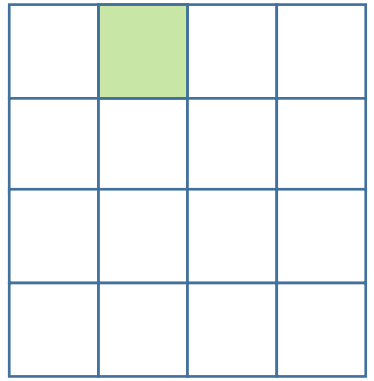
Finally, output tile computation ends on the first column

Tiling in Weight Stationary Mode

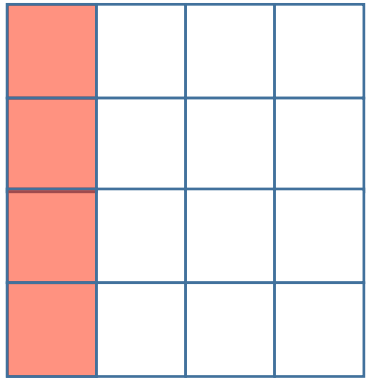


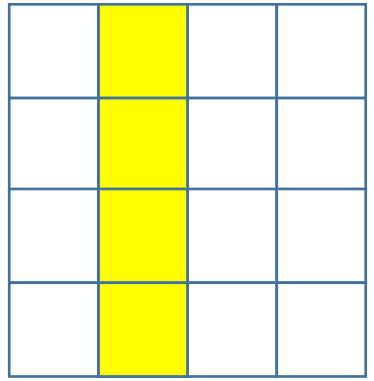
A tile is read from DRAM only once

Matrix B (DRAM)



Matrix A (Internal buffer)

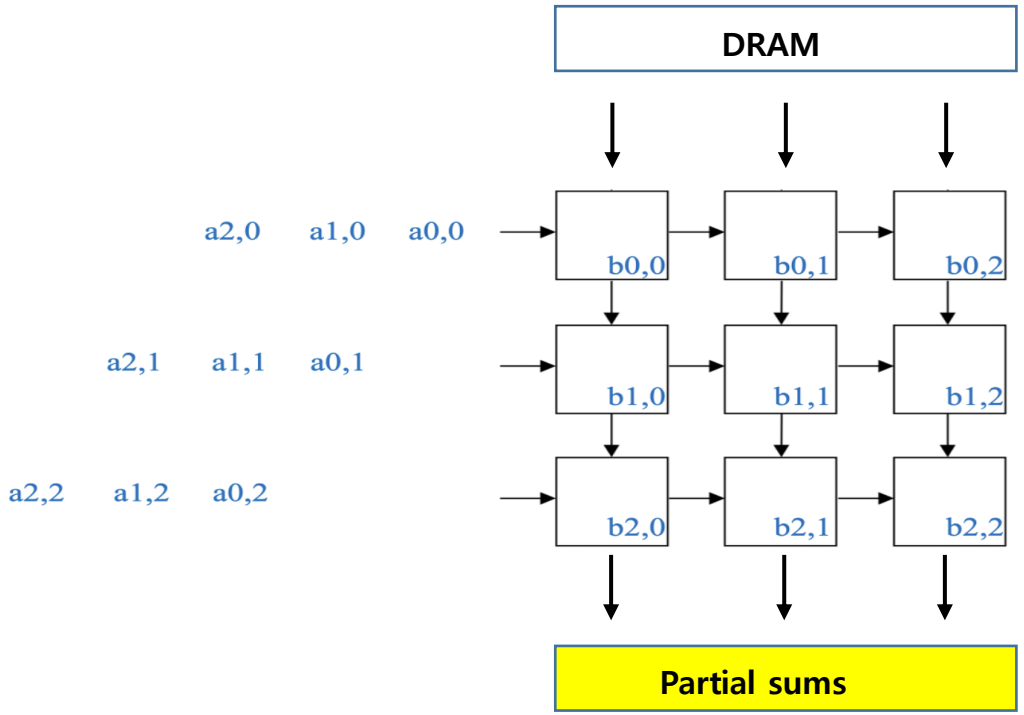


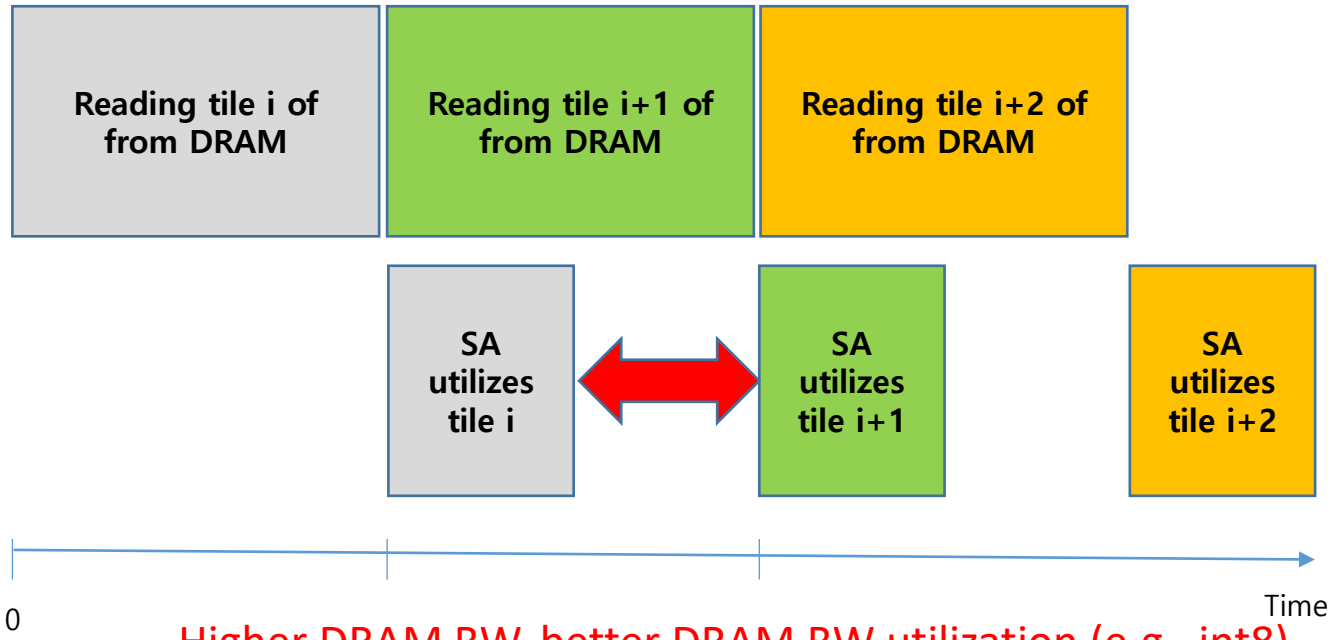


Matrix C

Hiding Latency of Reading Weights from DRAM with Computation

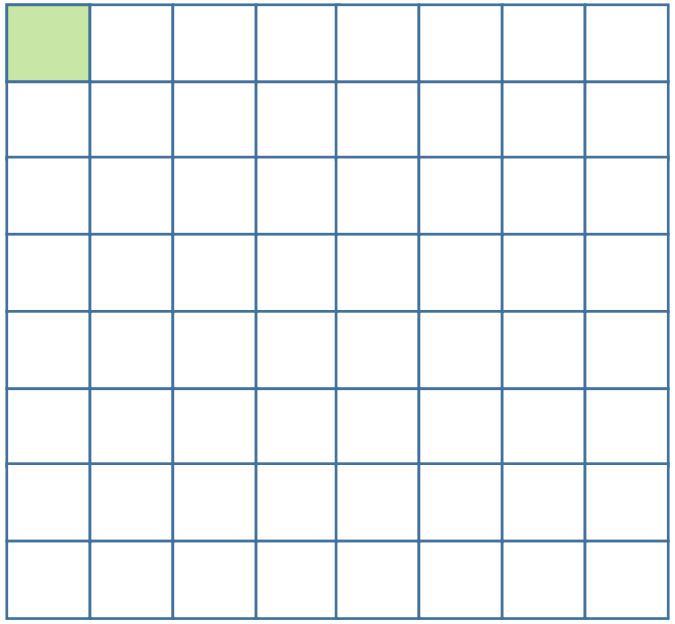
- Due to low bandwidth (BW) of DRAM, DRAM accesses can still dominate total runtime although some of the DRAM latency is hidden by computation on systolic array





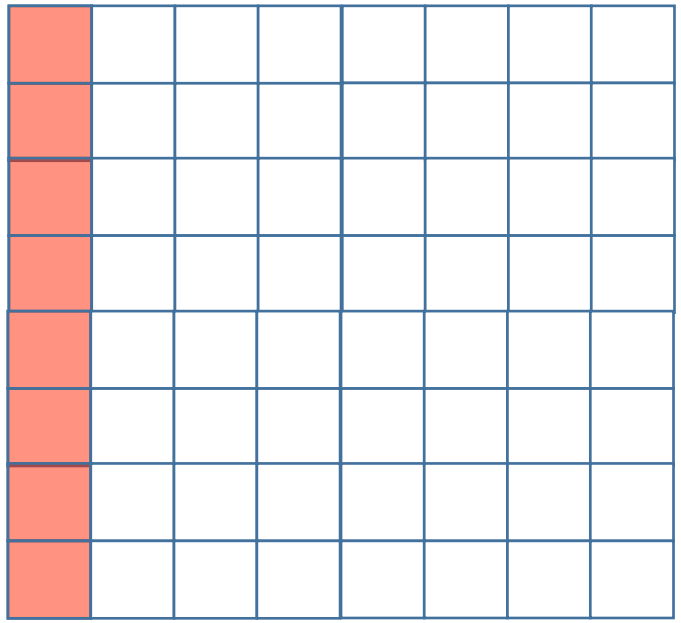
Higher DRAM BW, better DRAM BW utilization (e.g., int8), larger matrix, … are needed to remove this

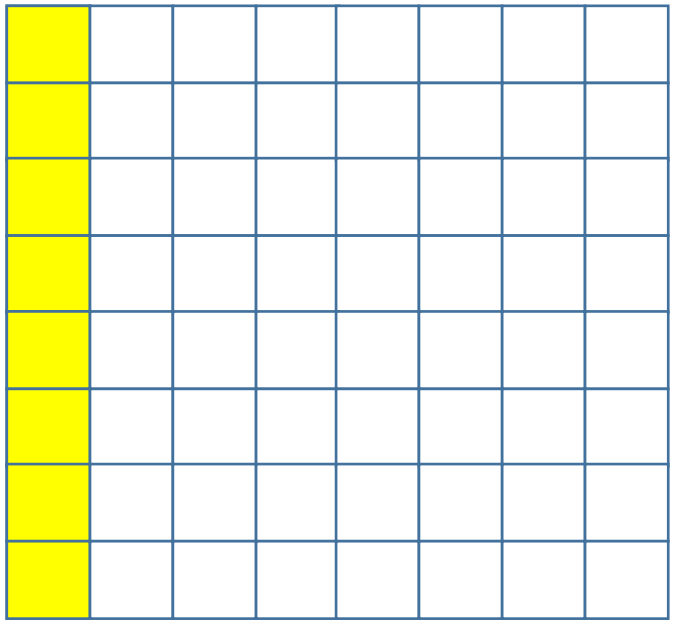
Larger Matrix Can Help Overlap DRAM Access with Longer Computation by More Data Reuse (8X) or OPS/Byte



Matrix B
(DRAM)

Matrix A
(Internal buffer)

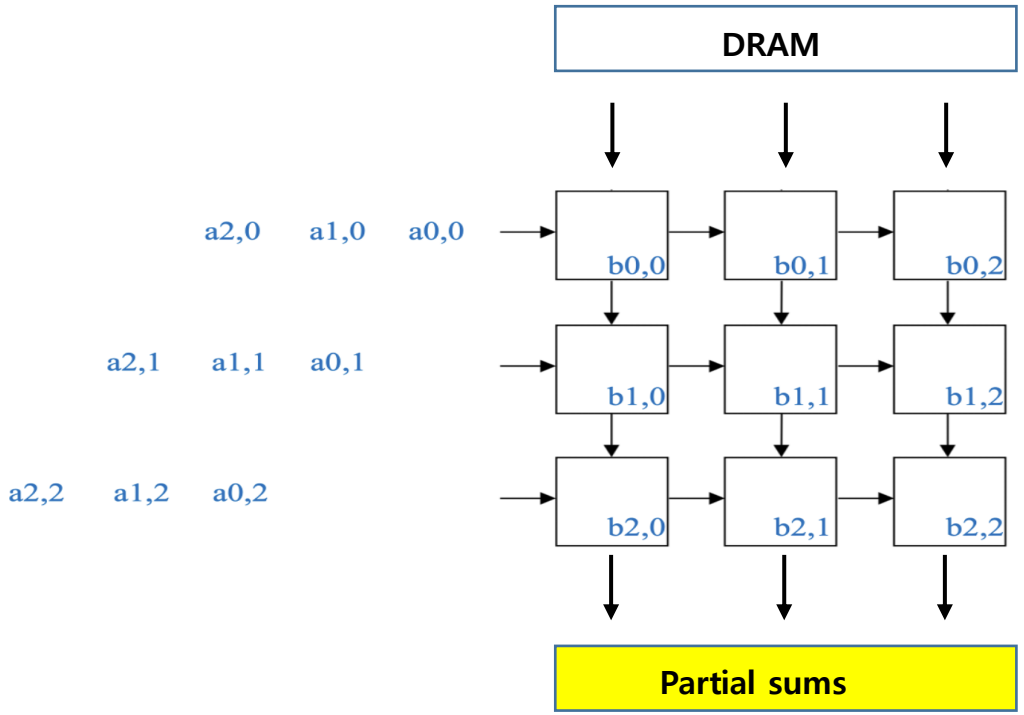


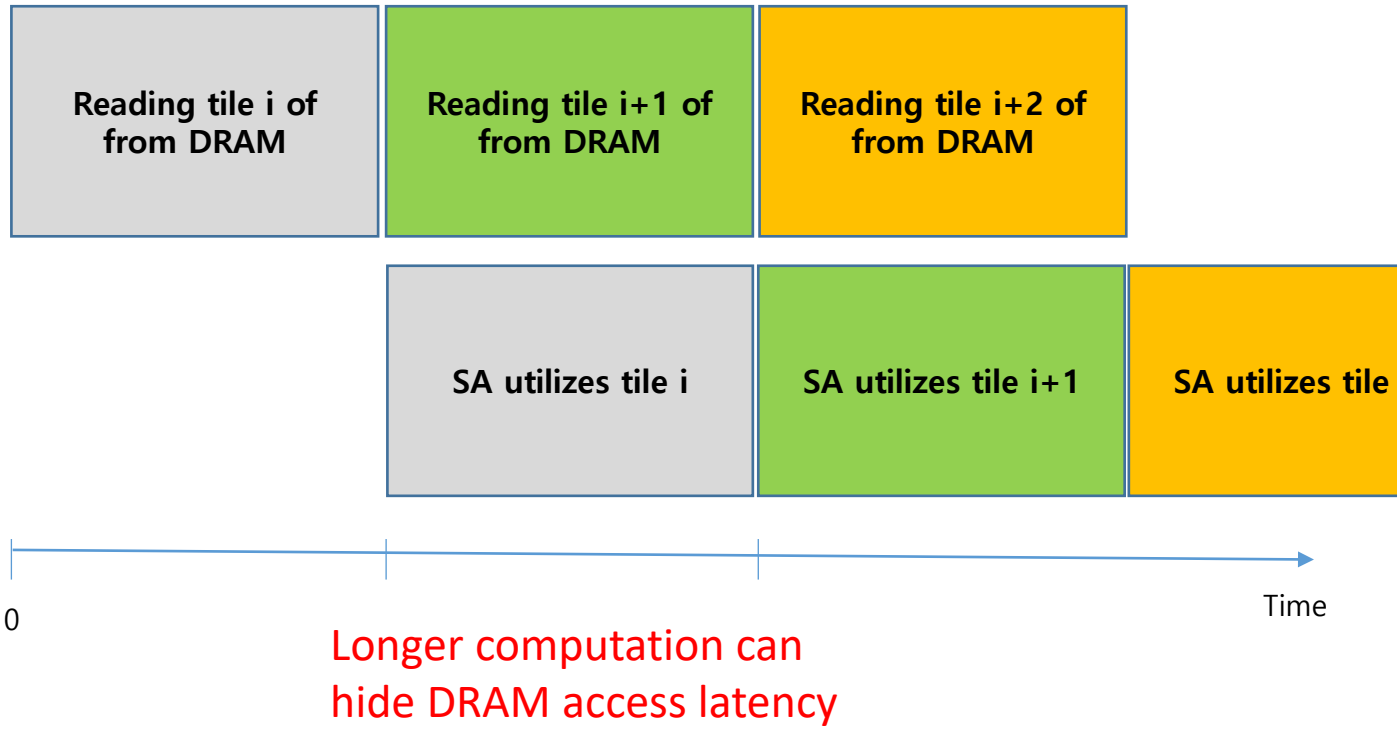


Matrix C

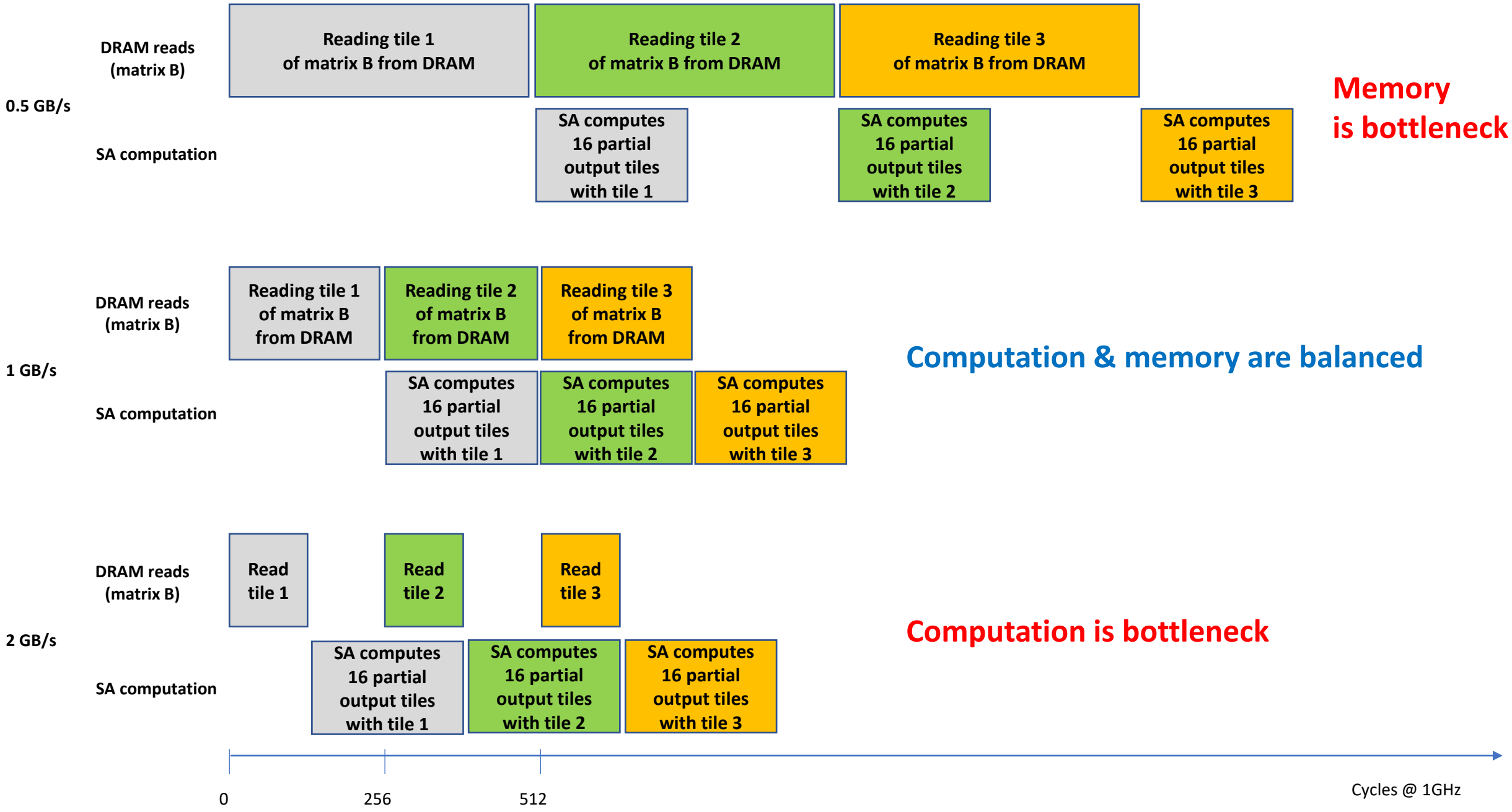
In case of Large Matrices, Longer Computation Can Hide DRAM Access Latency

- However, what if matrix size is not large enough?
- Improvements in memory bandwidth and efficiency (via low precision)!





Longer computation can hide DRAM access latency

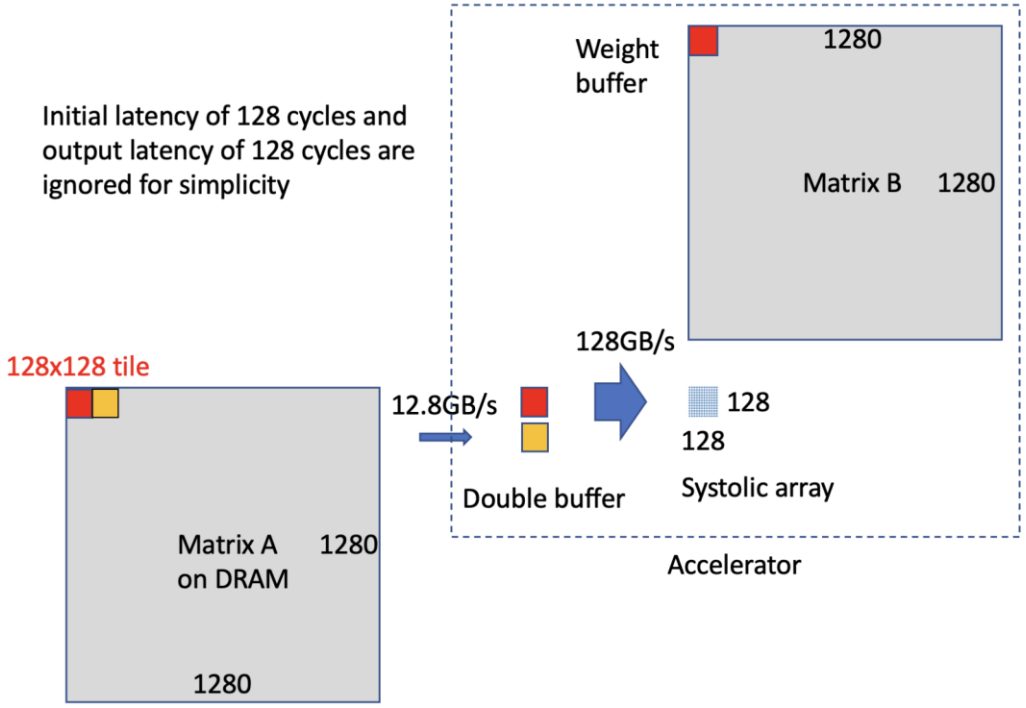

0.5 GB/s
DRAM reads (matrix B)
SA computation
Reading tile 1 of matrix B from DRAM
Reading tile 2 of matrix B from DRAM
Reading tile 3 of matrix B from DRAM
SA computes 16 partial output tiles with tile 1
SA computes 16 partial output tiles with tile 2
SA computes 16 partial output tiles with tile 3
Memory is bottleneck
1 GB/s
DRAM reads (matrix B)
SA computation
Reading tile 1 of matrix B from DRAM
Reading tile 2 of matrix B from DRAM
Reading tile 3 of matrix B from DRAM
SA computes 16 partial output tiles with tile 1
SA computes 16 partial output tiles with tile 2
SA computes 16 partial output tiles with tile 3
Computation & memory are balanced
2 GB/s
DRAM reads (matrix B)
SA computation
Read tile 1
Read tile 2
Read tile 3
SA computes 16 partial output tiles with tile 1
SA computes 16 partial output tiles with tile 2
SA computes 16 partial output tiles with tile 3
Computation is bottleneck
0
256
512
Cycles @ 1GHz

’22 Final Exam

Q1. Accelerator (15 points)

The following figure illustrates the systolic array (SA) accelerator (corresponding to the dashed box), explained in our lecture, which is used to multiply matrix A (stored in DRAM) and matrix B (stored in the weight buffer inside of the accelerator). The matrix size is 1280x1280 and each element has 1 byte. We have the same baseline architecture where DRAM bandwidth is 12.8GB/s, SA of 128x128 runs at 1GHz in a pipelined manner, the double buffer size is each 128x128, and the initial and output latency of SA is ignored for simplicity. Each input tile is reused 10 times inside of accelerator and we ignore the latency for the transfer of partial sums, i.e., we assume the partial sum buffer (not shown in the accelerator) can be accessed with no latency.

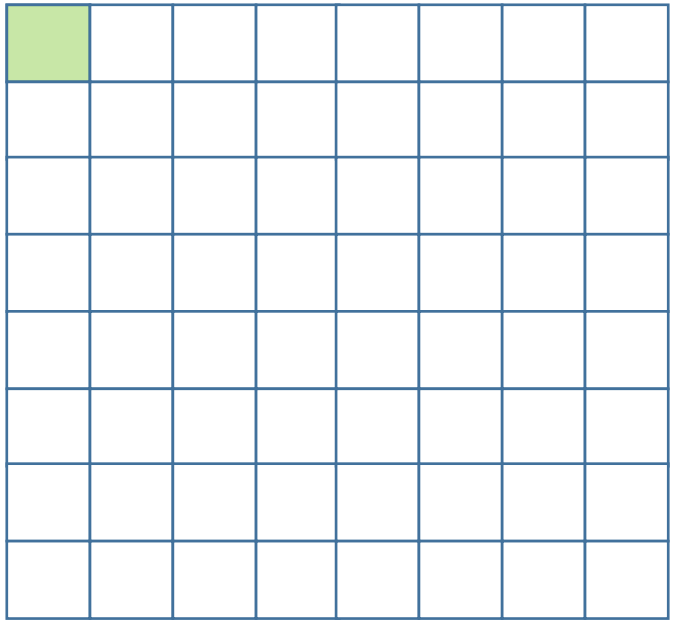
10X reuse



Q1-1. Calculate the total latency (in nano-seconds) of matrix multiplication to obtain the entire output matrix. (10pts)

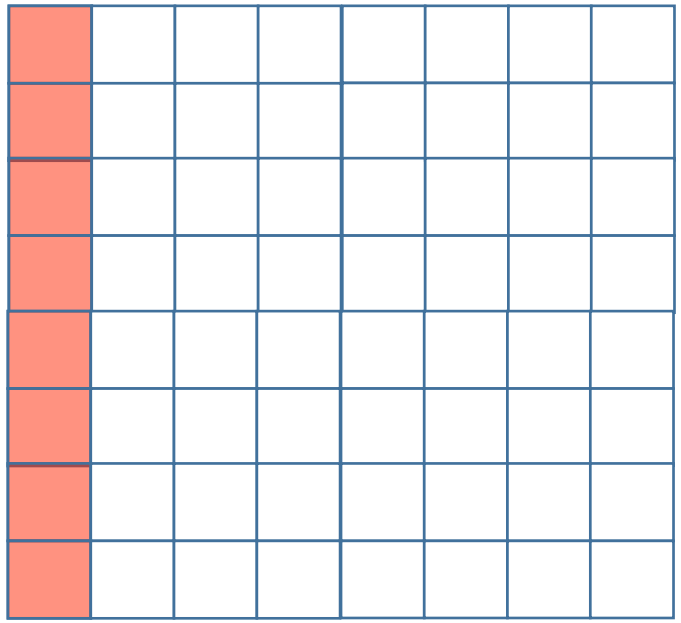
[Answer]
cycles = total # multiplications / # SA multiplications = 1280x1280x1280 / 128x128 = 128,000 nano-seconds (1 cycle @ 1GHz = 1 nano-second)

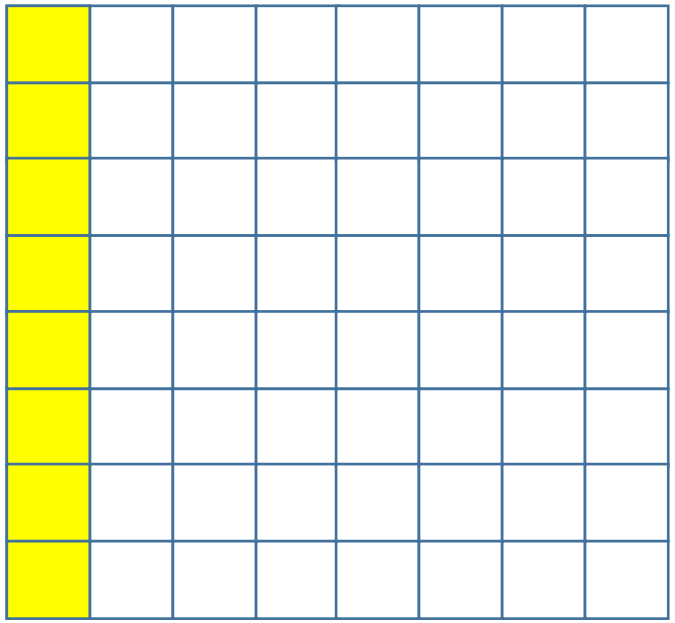
8X Reuse (or OPS/Byte) Case



Matrix B
(DRAM)

Matrix A
(Internal buffer)

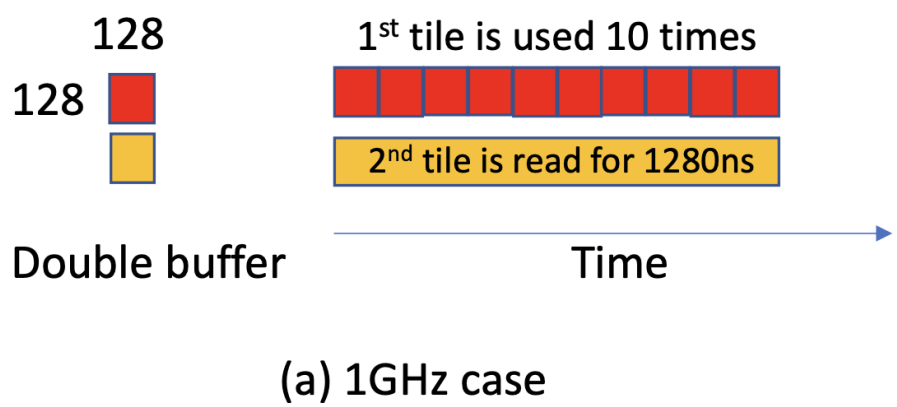




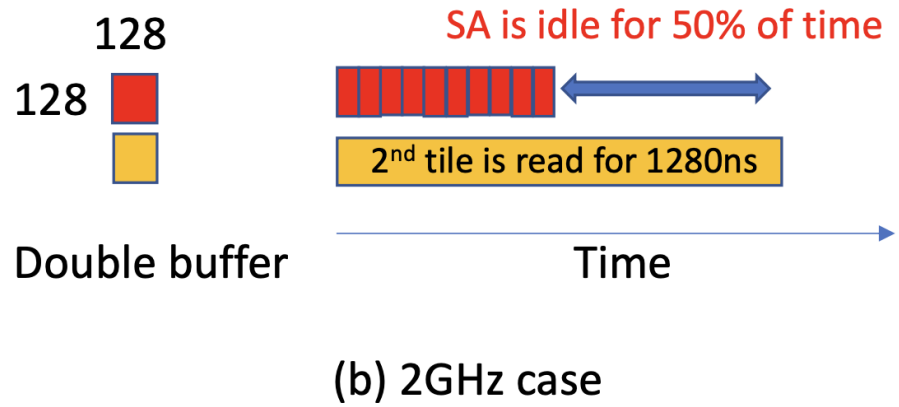
Matrix C

Q1-2. Assume the SA runs at 2GHz and calculate the total latency of matrix multiplication (2pt). Explain the difference between 1GHz and 2GHz operations (3pts)

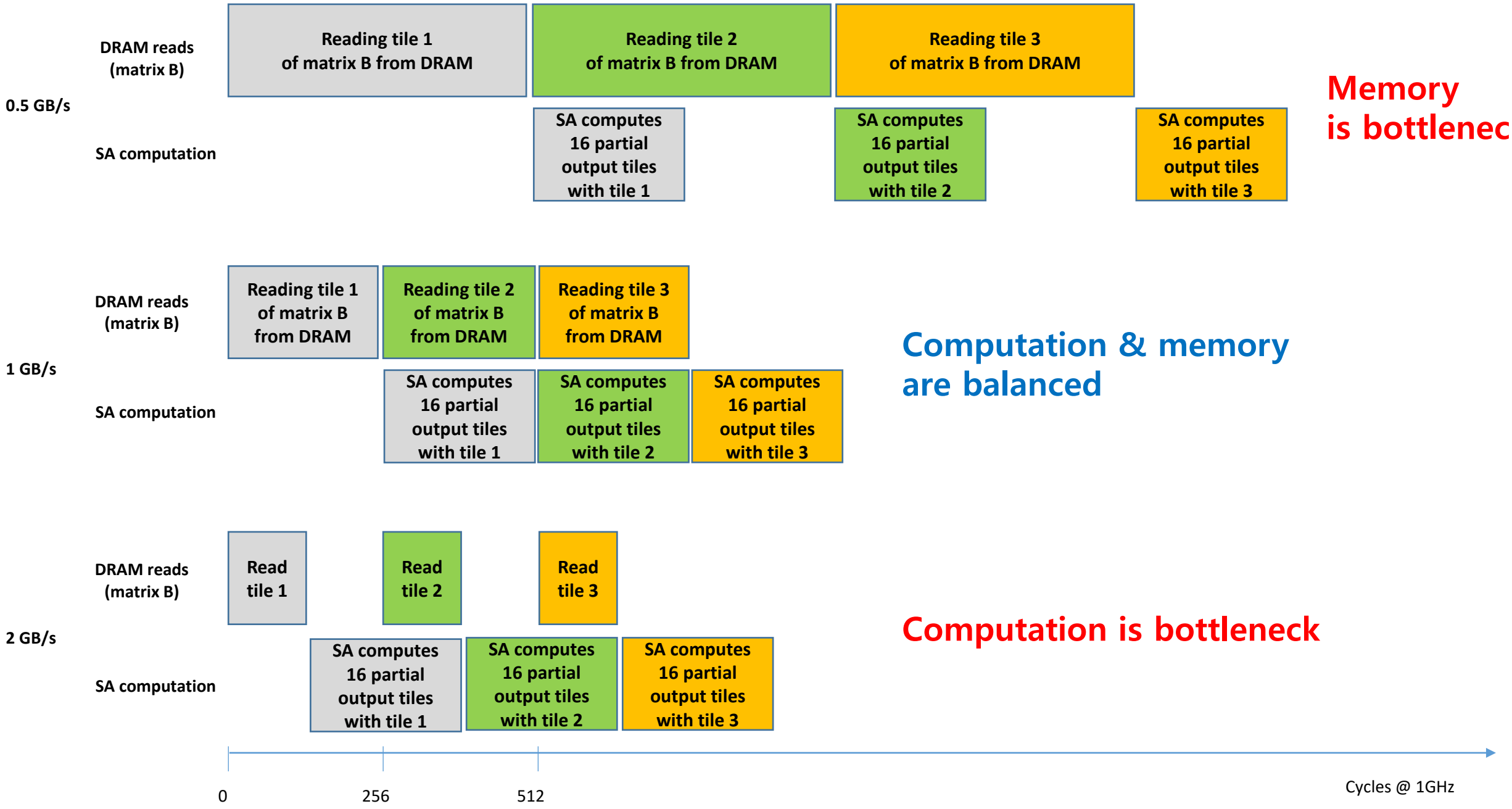
[Answer]
The same as the runtime of problem 1-1. (2pt)
As the following figures show, the runtime of 2GHz is determined by memory access time, i.e., memory bandwidth, which is the same as in problem 1.1. (3pt)



(a) 1GHz case



(b) 2GHz case


DRAM reads (matrix B)
0.5 GB/s
SA computation
Reading tile 1 of matrix B from DRAM
Reading tile 2 of matrix B from DRAM
Reading tile 3 of matrix B from DRAM
SA computes 16 partial output tiles with tile 1
SA computes 16 partial output tiles with tile 2
SA computes 16 partial output tiles with tile 3
Memory is bottleneck
DRAM reads (matrix B)
1 GB/s
SA computation
Reading tile 1 of matrix B from DRAM
Reading tile 2 of matrix B from DRAM
Reading tile 3 of matrix B from DRAM
SA computes 16 partial output tiles with tile 1
SA computes 16 partial output tiles with tile 2
SA computes 16 partial output tiles with tile 3
Computation & memory are balanced
DRAM reads (matrix B)
2 GB/s
SA computation
Read tile 1
Read tile 2
Read tile 3
SA computes 16 partial output tiles with tile 1
SA computes 16 partial output tiles with tile 2
SA computes 16 partial output tiles with tile 3
Computation is bottleneck
0
256
512
Cycles @ 1GHz