Gemmini: An Agile Systolic Array Generator Enabling Systematic Evaluations of Deep-Learning Architectures

2020년 03월 15일

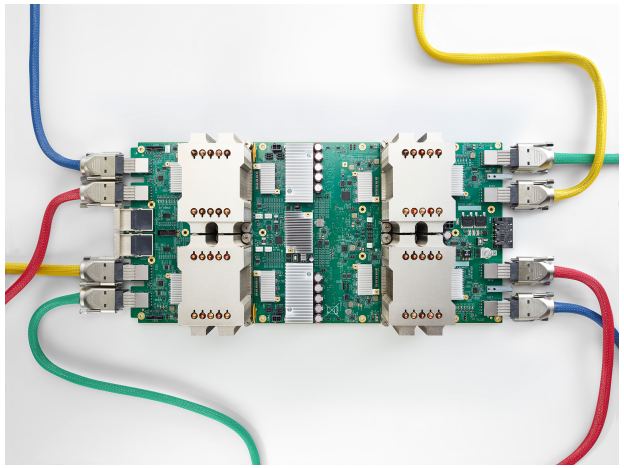
Constant Park

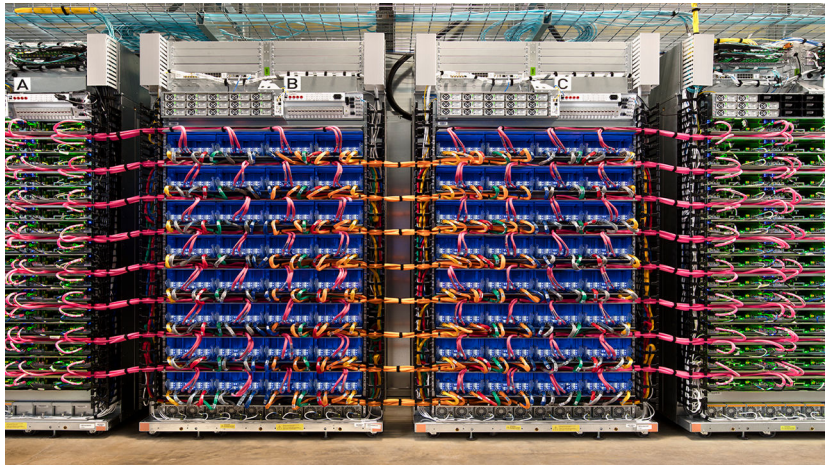
(http://esoc.hanyang.ac.kr/people/sangsoo_park/index.html)

Neural Acceleration Study

Motivation

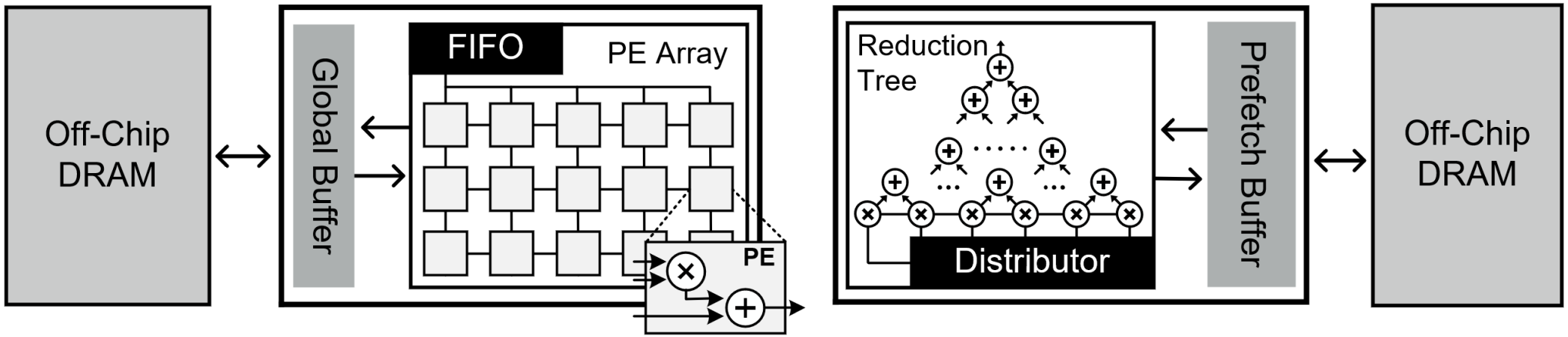
- DNN accelerators range from edge-device to cloud
  - **Rapid development** of deep learning and neural accelerator
  - Requirement for agile HW design and generator methodology
  - Different constraints on the accelerator (Latency, power, etc.)





DNN Accelerators

- Mainstream of DNN accelerators
  - Processing element (PE) array, Reduction-tree structure
  - Scratch pad: Global, Prefetch buffer
  - Multiplication and Accumulate (MAC): PE array, Reduction tree

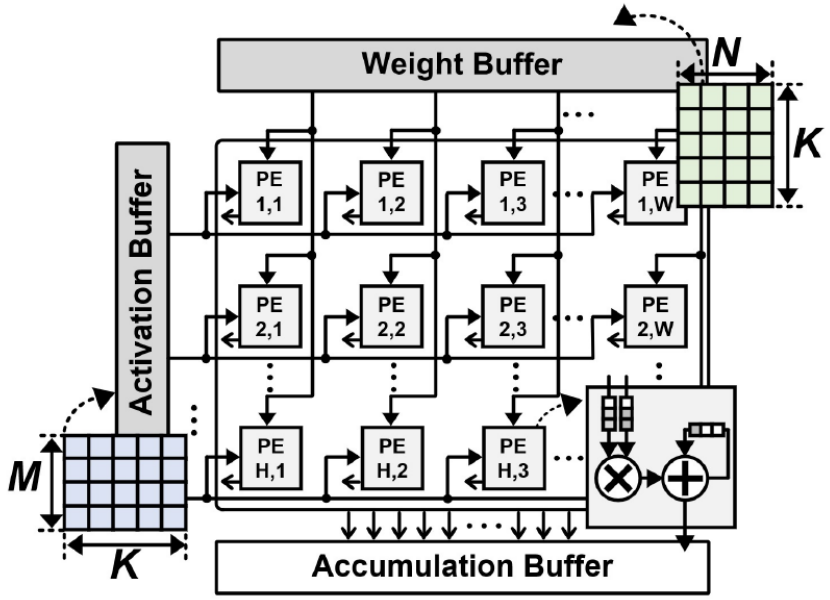


PE array structure

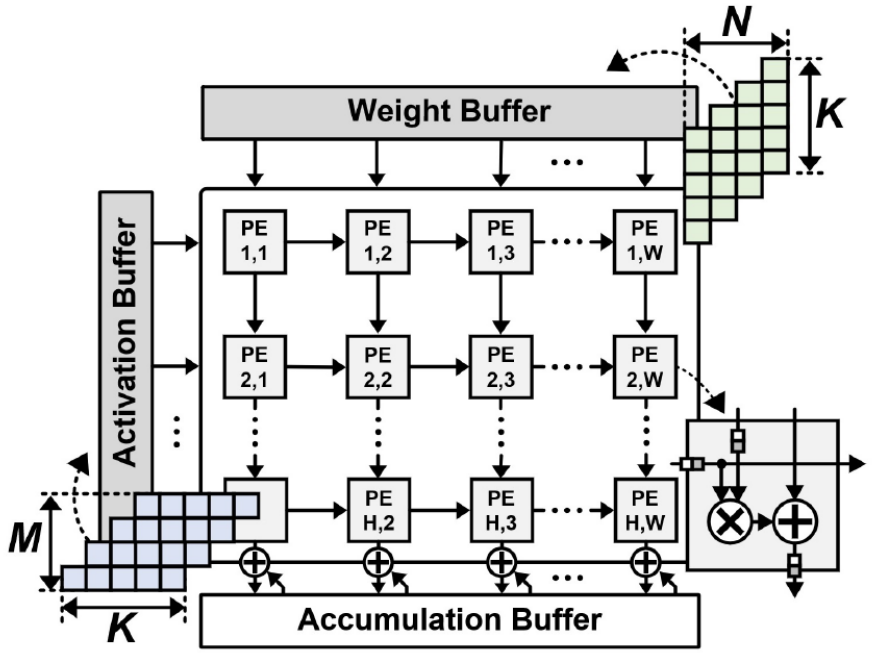
Reduction-tree structure

PE array: 2D-SIMD

- Two-dimensional single instruction multiple data (SIMD)
  - Load multiple data into vector register to compute together
  - Separated data distribution fabric (i.e. bus, tree) for separate control



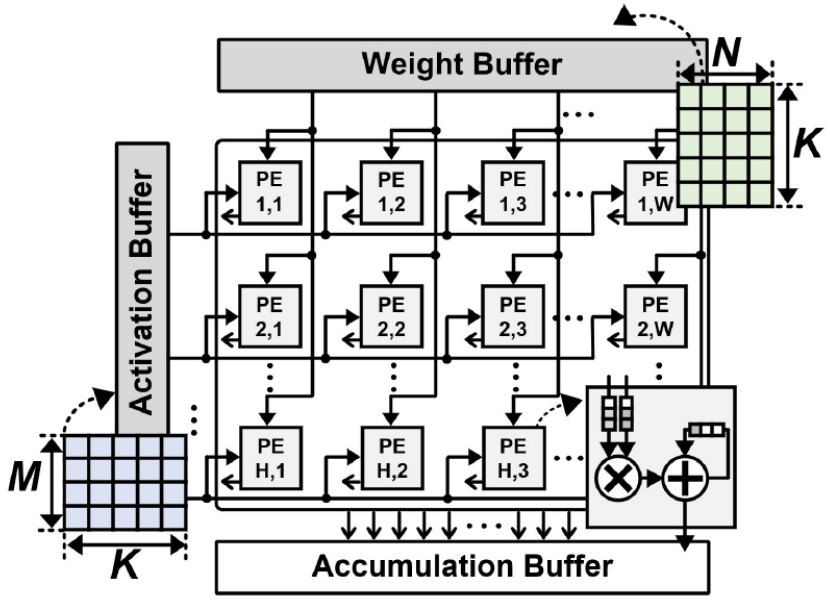
2D-SIMD structure



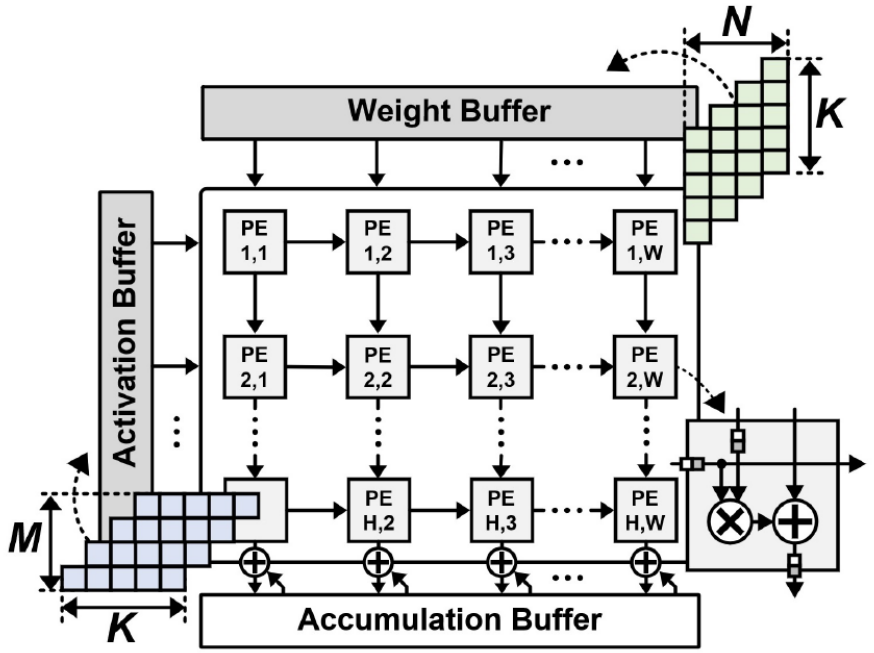
Systolic array structure

PE array: Systolic array

- Homogenous network of tightly coupled data units (DPUs)
  - Receive data from their neighbor PEs
  - Perform MAC operation simultaneously with 2*N channel



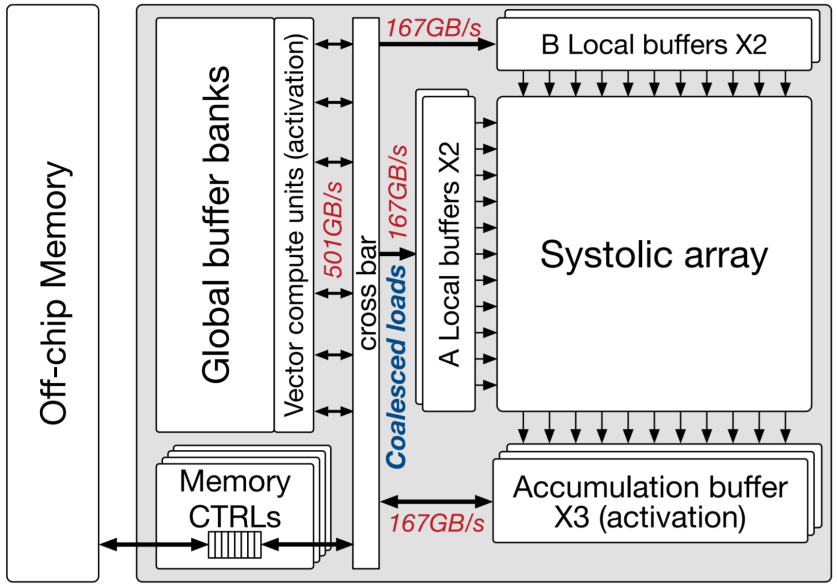
2D-SIMD structure (N*N)



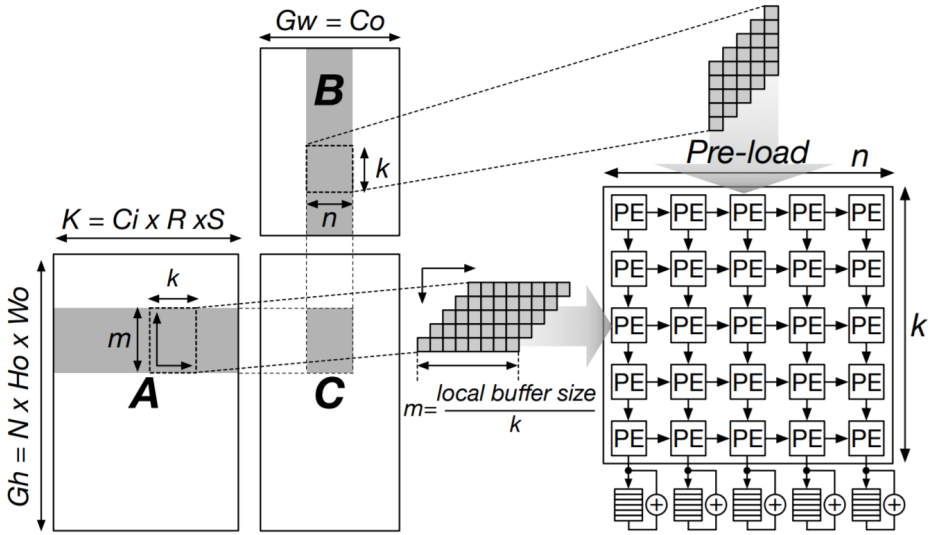
Systolic array structure (2*N)

PE array: Systolic array

- Set of interconnected PEs
  - PE performing simple operation (MAC operation)
  - 2D data flow can be at multiple speeds in different directions



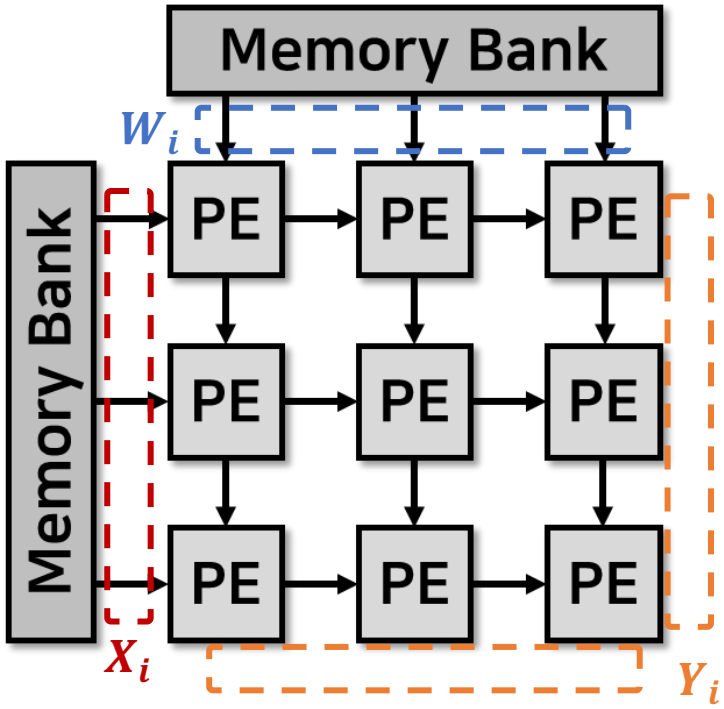
Systolic accelerator



Computation in systolic architecture

PE array: Systolic array example #1

- Multiplication and addition in each PE
  - Each PE stores a weight, multiplies its input and add partial sums
  - Correlation operation can be expressed as



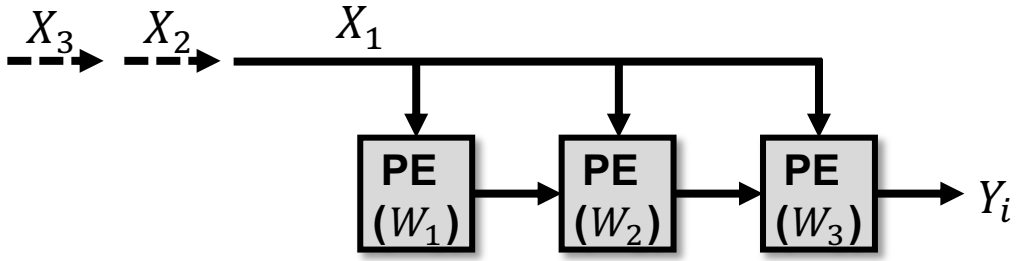
$$Y_i = W_1 * X_i + W_2 * X_{i+1} + W_3 * X_{i+2}$$

$$Y_1 = W_1 * X_1 + W_2 * X_2 + W_3 * X_3$$

$$Y_2 = W_1 * X_2 + W_2 * X_3 + W_3 * X_4$$

PE array: Systolic array example #2

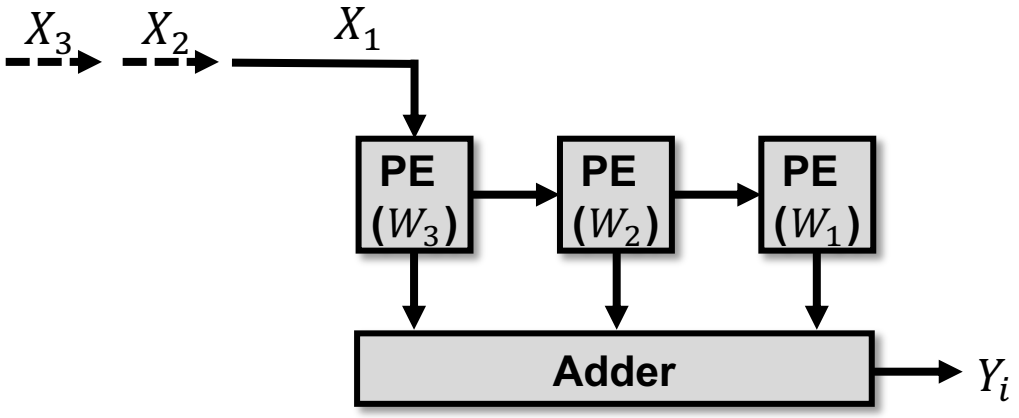
- Weight Stationary (WS)
  - PE holds weights, multiplies it with its input and add partial sums
  - **Weight stationary** as weight stay stationary and the input are streamed in



$t = 0$	$W_1 * X_1$	$W_2 * X_1$	$W_3 * X_1$	$W_3 * X_1$
$t = 1$	$W_1 * X_2$	$W_2 * X_2 + W_1 * X_1$	$W_3 * X_2 + W_2 * X_1$	$W_3 * X_2 + W_2 * X_1$
$t = 2$	$W_1 * X_3$	$W_2 * X_3 + W_1 * X_2$	$W_3 * X_2 + W_2 * X_1 + W_1 * X_1$	$W_3 * X_2 + W_2 * X_1 + W_1 * X_1$

PE array: Systolic array example #3

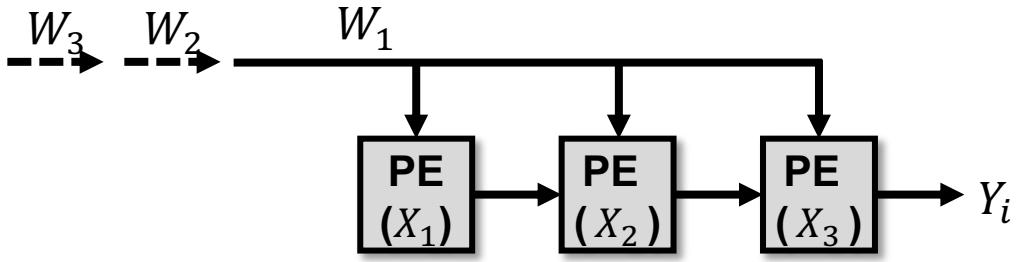
- Output Stationary (OS)
  - Each PE stores and accumulate the partial sums
  - Results stay and the inputs/weights move in opposite directions



$t = 0$	$W_3 * X_1$
$t = 1$	$W_3 * X_2 + W_2 * X_1$
$t = 2$	$W_3 * X_3 + W_2 * X_2 + W_1 * X_1 = Y_1$
$t = 3$	$W_3 * X_4 + W_2 * X_3 + W_1 * X_2 = Y_2$

PE array: Systolic array example #4

- Input Stationary (IS)
  - PE holds inputs, multiplies it with its weight and add partial sums
  - **Input stationary** as input stay stationery and weights are streamed in



$t = 0$	$X_1 * W_1$	$X_2 * W_1$	$X_3 * W_1$	$X * W_1$
$t = 1$	$X_1 * Y_2$	$X_2 * W_2 + X_1 * W_1$	$X_3 * W_2 + X_2 * W_1$	$X_3 * W_2 + X_2 * W_1$
$t = 2$	$X_1 * Y_3$	$X_2 * W_3 + X_1 * W_2$	$X_3 * W_2 + X_2 * W_1 + X_1 * W_1$	$X_3 * W_2 + X_2 * W_1 + X_1 * W_1$

PE array: Systolic array conclusion

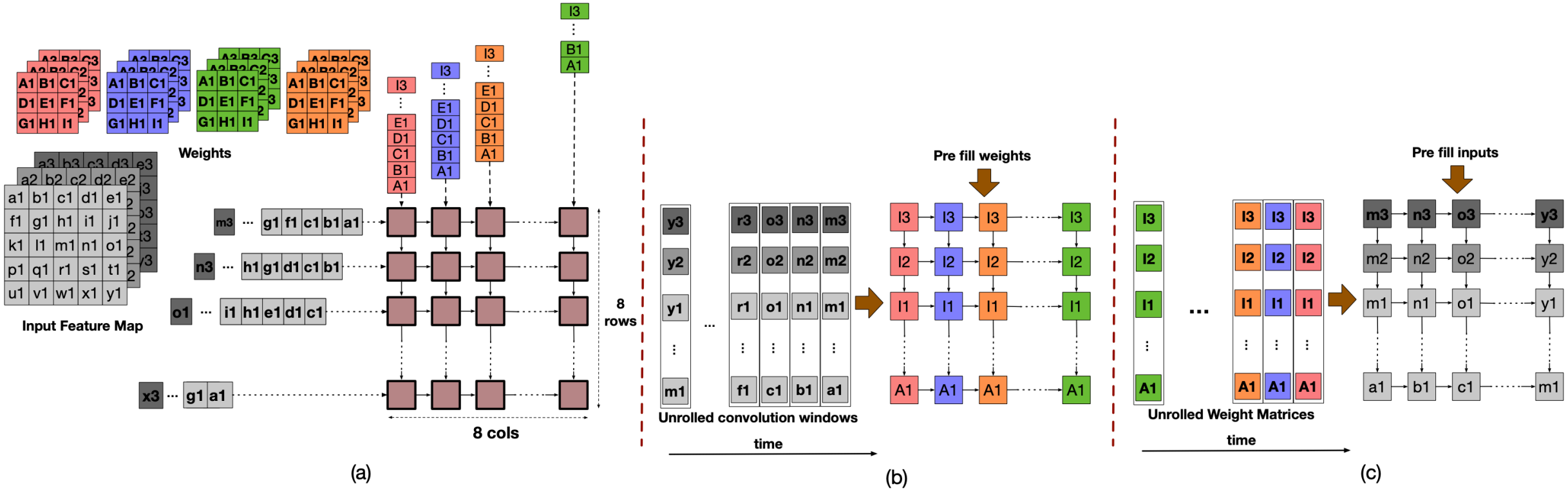
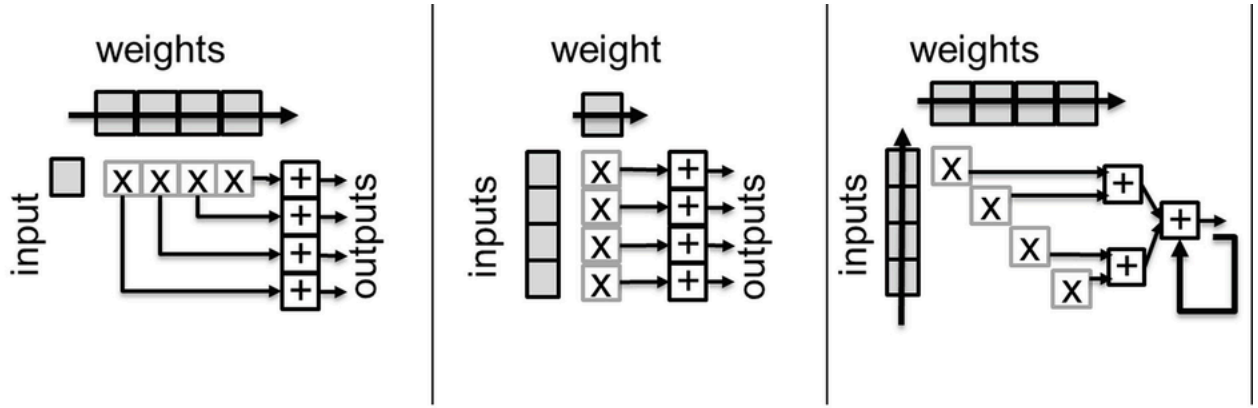


Fig. 2: Schematic showing the mapping in various dataflows (a) Output stationary; (b) Weight stationary; (c) Input stationary

PE array: Systolic array conclusion

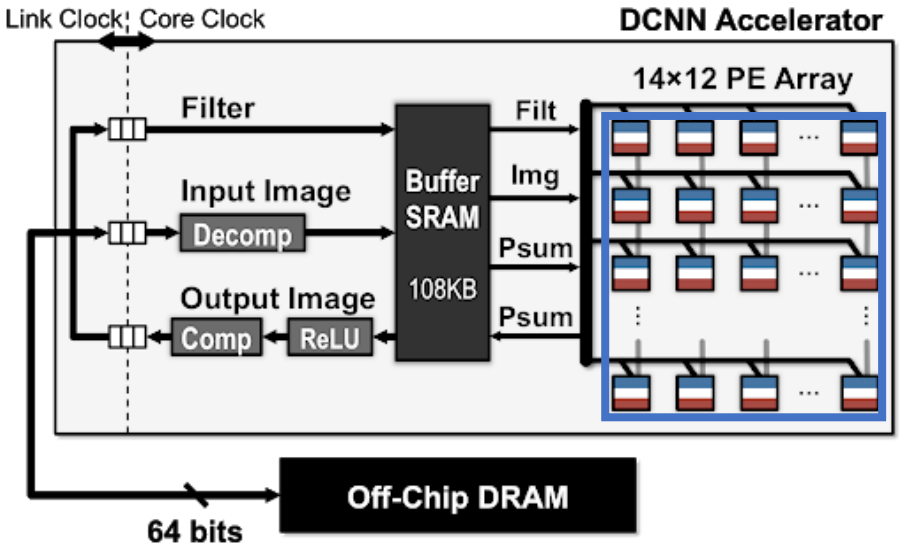
- Stationary Series
  - Each scheme has different memory bandwidth, optimal computation



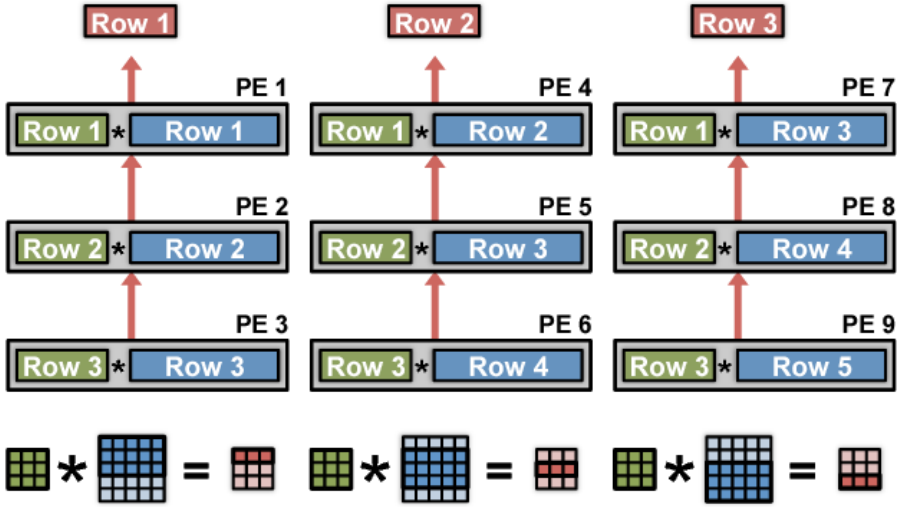
	Input-stationary *(weight parallel)*	Weight-stationary *(input parallel)*	Output-stationary
Input BW	**low**	high	high
Weight BW	high	**low**	high
Output BW	high	high	**low**

PE array arch: Eyeriss v1.0

- 1st neural network accelerator project in NVIDIA and MIT
  - Row stationary scheme in PE communication
  - Reducing data transaction between accelerator and off-chip DRAM



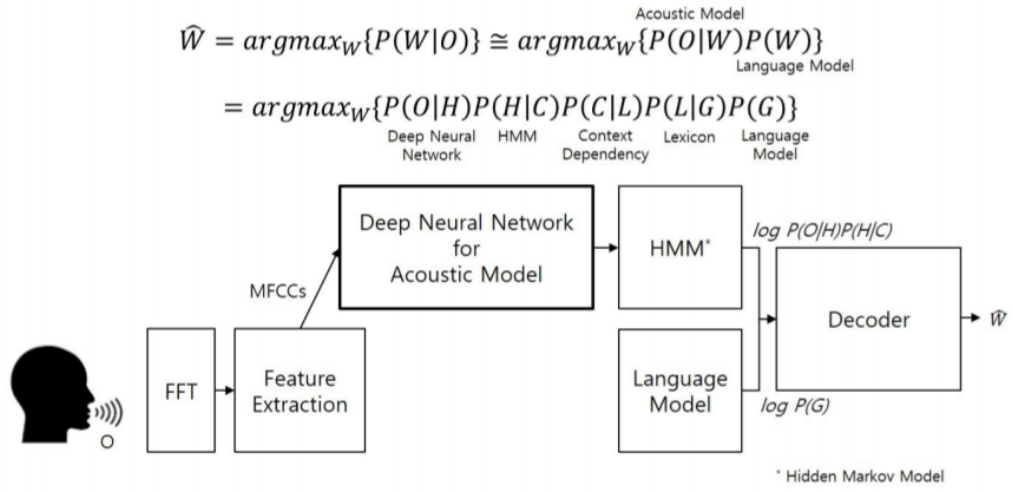
Eyeriss v1.0 architecture



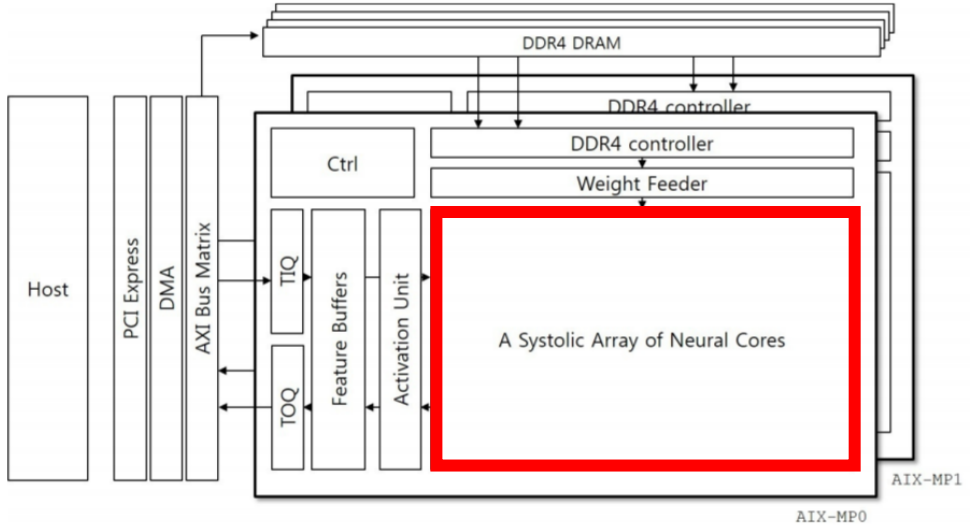
Row stationary scheme

PE array arch: AI FPGA accelerator (AIX)

- 1st commercial Korean virtual personal assistant (VPA)
  - Systolic array of neural cores using Xilinx DSP
  - Acoustic speech recognition accelerator (MFCC, HMM, LM)



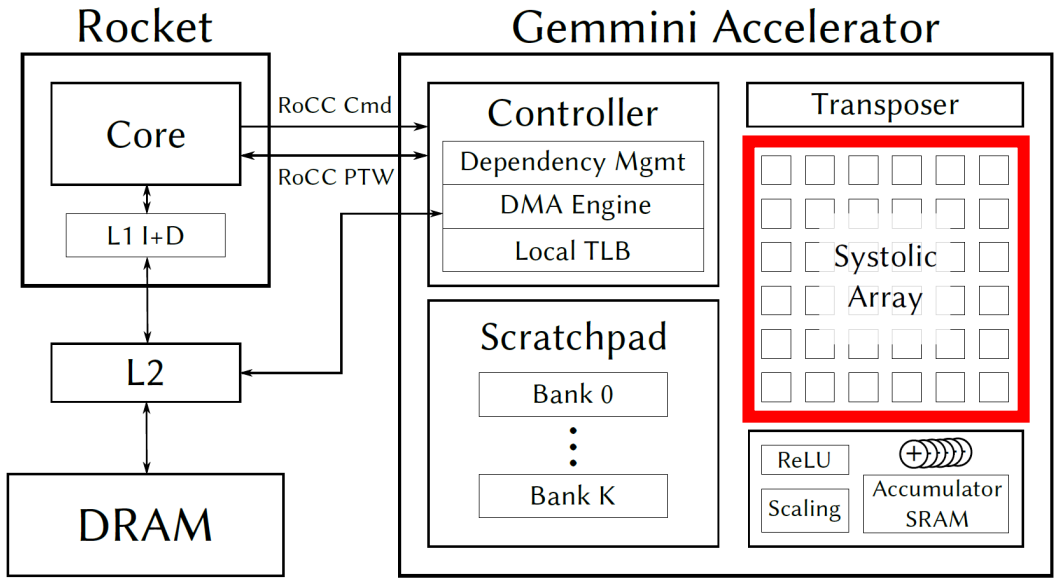
Block diagram of ASR



AIX architecture

Arch: System overview

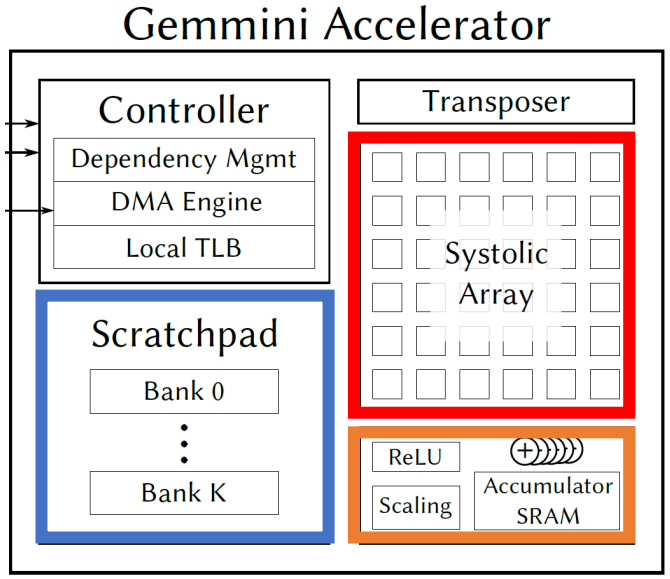
- SoC architecture with Rocket generator and Boom OOO arch
  - HW: RISC-V core and ISA extension-based accelerator (ROCC extension)
  - SW: C libraries for Systolic array (Inline assembly)

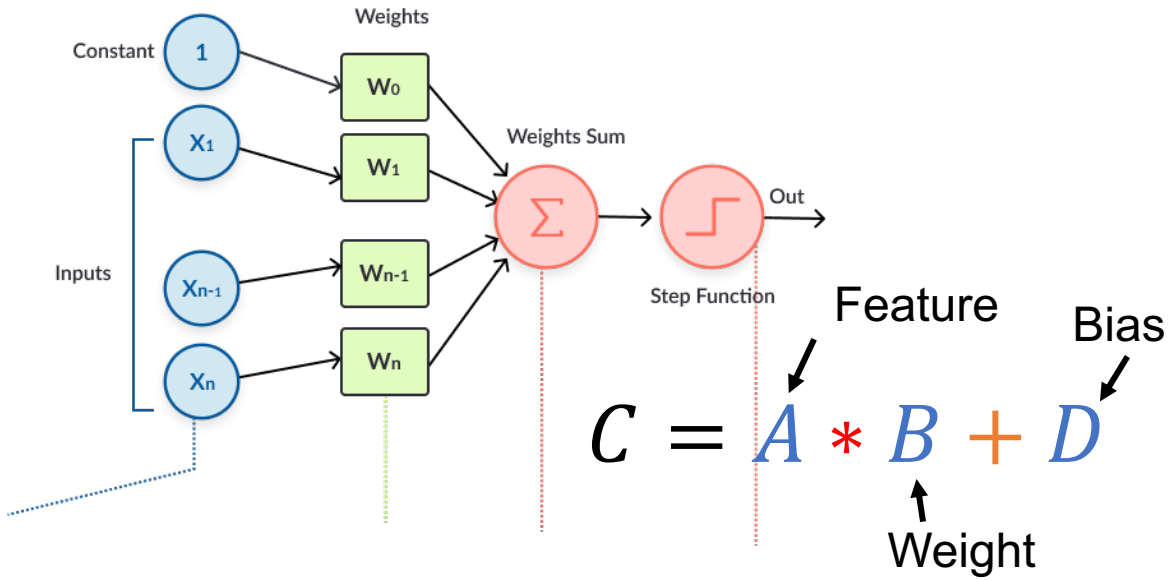


System overview of the Gemmini based systolic array generator

Arch: Matrix multiplication #1

- Multiple components of accelerator
  - Scratchpad: Feature map, weight, bias
  - Peripheral: ReLU/ReLU6, bit shift/scaling, accumulation

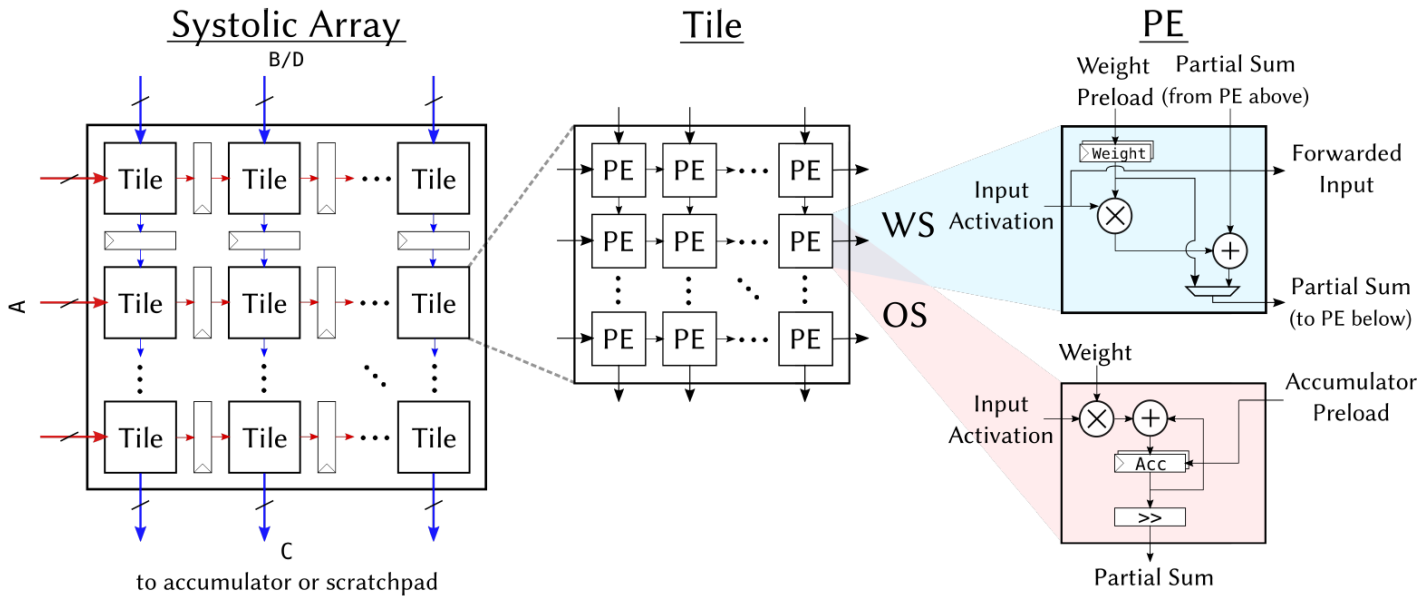




Function mapping to HW design

Arch: Matrix multiplication #2

- Multiple components of accelerator
  - Special dataflow scheme
  - Weight stationary (WS), Output stationary (OS)



Systolic architecture in Gemmini

Arch: Parameters #1

- Dataflow
  - Support WS, OS (Fixed at elaboration time or configurated at runtime)
  - Configurated implementation in SqueezeNext accelerator

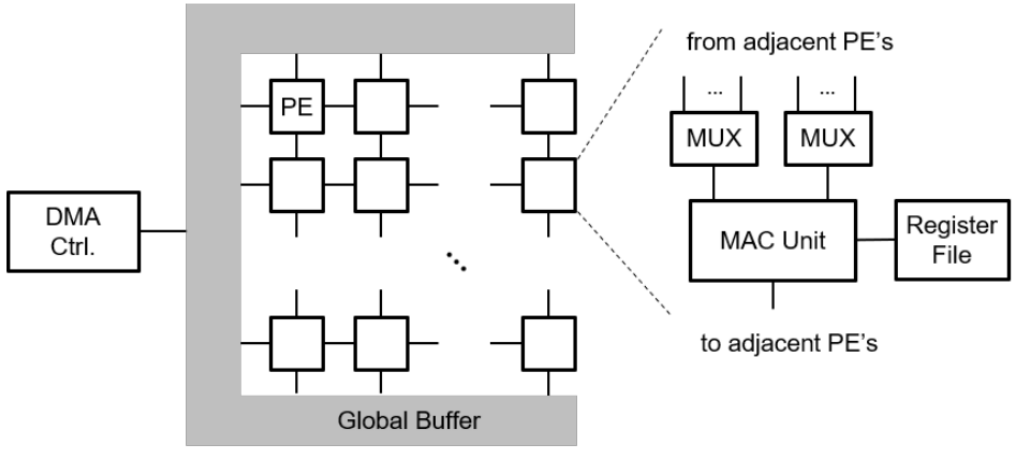


Figure 5: *Block diagram of the neural network accelerator used as the reference hardware for inference speed and energy estimation of various neural networks.*

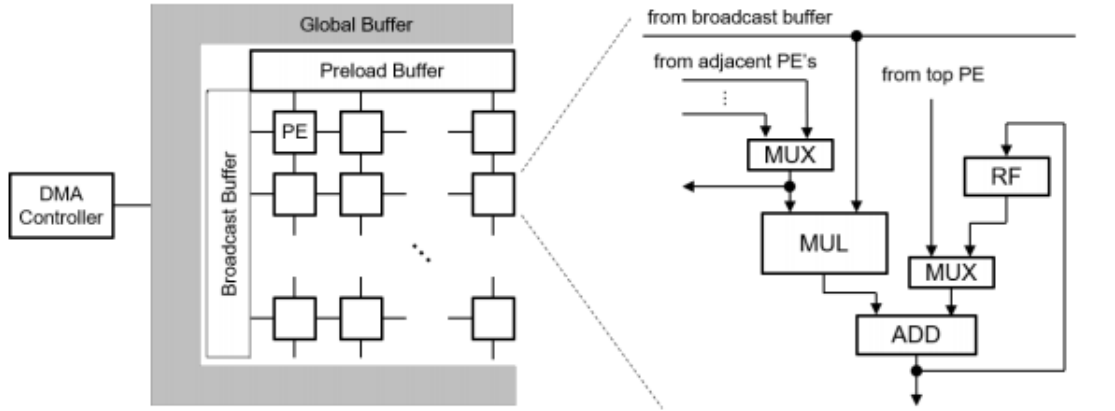
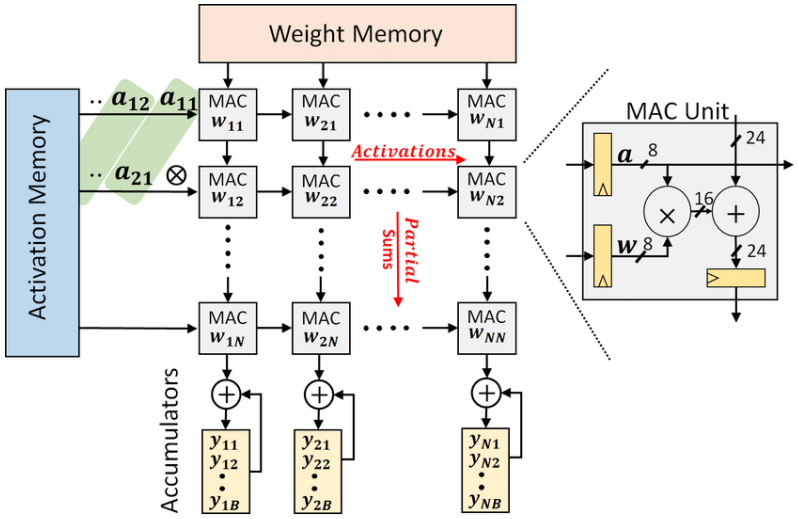


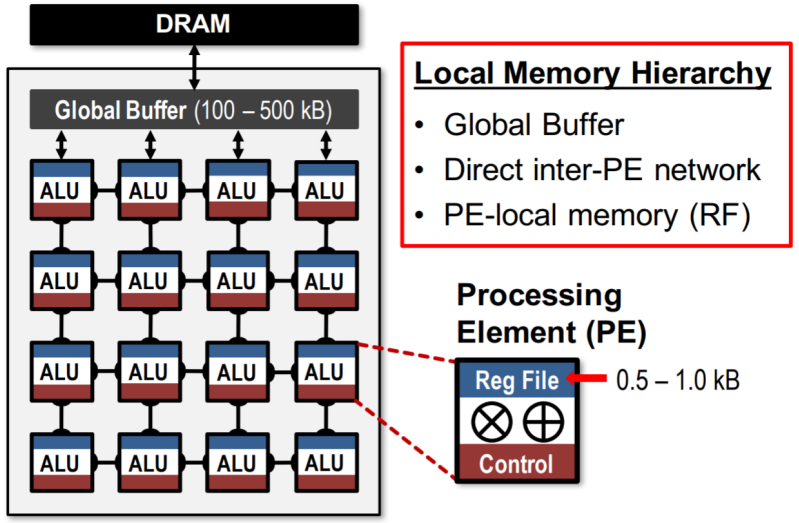
Figure 2: The block diagram of Squeezelerator (left) and PE (right)

Architecture of SqueezeNext accelerator

Arch: Parameters #2

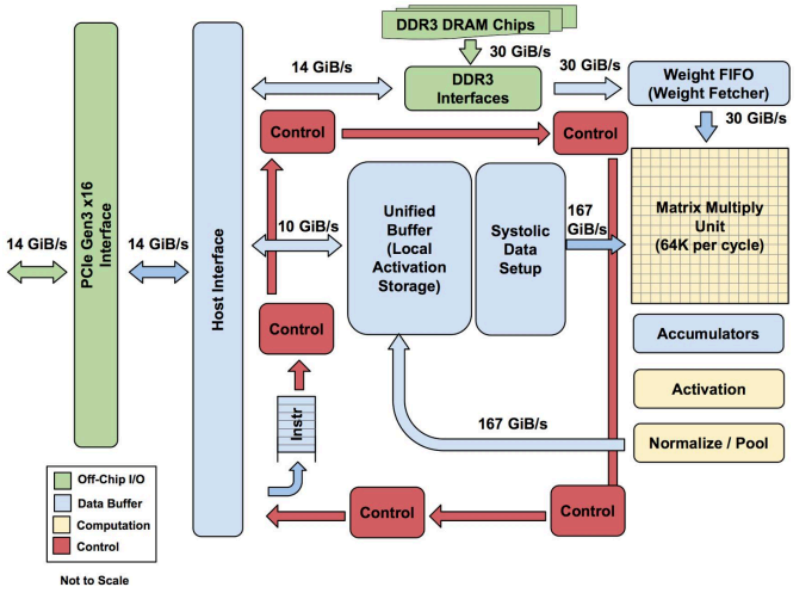
- Dimension/Bit-width/Pipeline Depth
  - Dimension, shape of PE array
  - Bit-width of input and output, partial sum (Related to quantization)
  - Computation and Power performance (Related to # # of register)

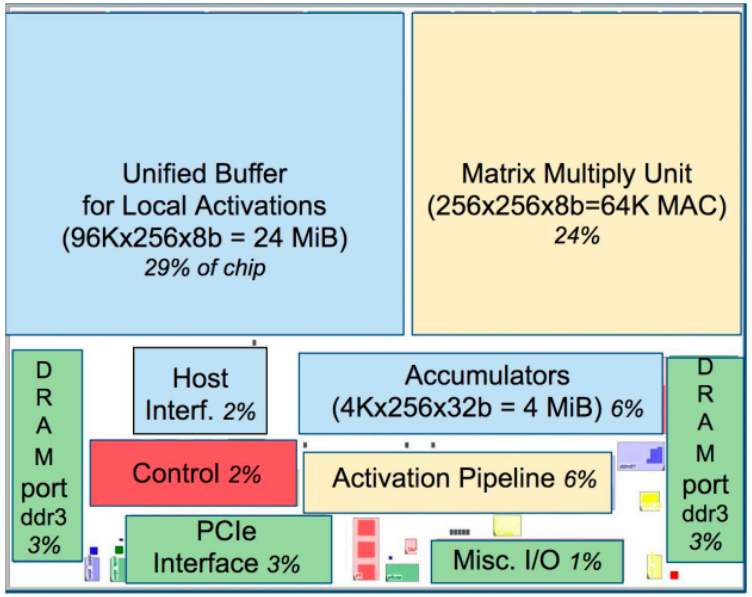




Arch: Parameters #3

- Memory Capacity/Bank and System
  - Both scratchpad and accumulator memory can be configured
  - # of memory bank maximizes/minimizes read/write throughput
  - Configure In-order or Out-of-order BOOM core with chisel





Arch: Programming model

- Decoupled-access-execute architecture
  - Three independent, parallel command queues
  - LOAD queue (mvin), STORE queue (mvout), EXECUTE queue (compute)
  - GEMM API routine and Applied Im2col API routine

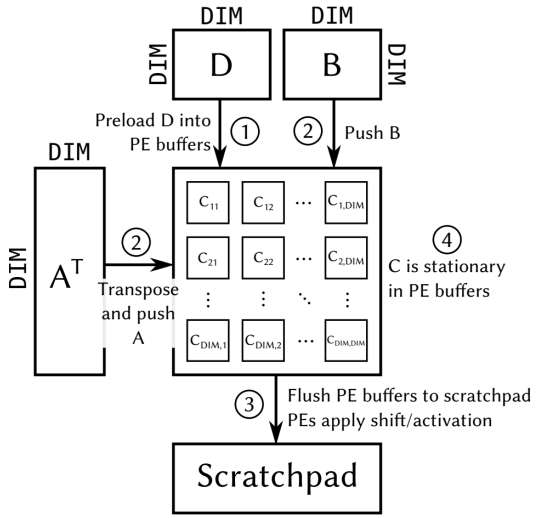


Figure 4: Execution of an output-stationary `compute` instruction.

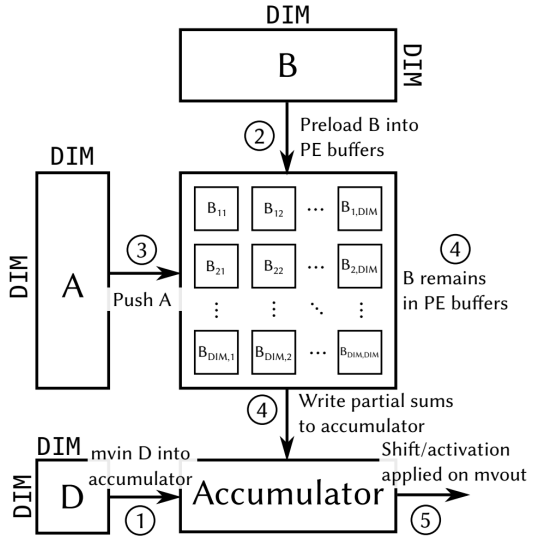


Figure 5: Execution of a weight-stationary `compute` instruction.

Design Space Exploration: Env

- Environment and Evaluation
  - FPGA-accelerated cycle-exact simulation platform (FireSim)
  - TSMC 16nm, Frequency (0.5/1GHz), 256KiB L2 Cache, 4MiB LLC
  - CNN (MobileNet, ResNet50, ResNet152), 4 MLP models in TPU paper
  - Im2col based Primitive acceleration

Design Space Exploration: Design Point

No.	Dataflow	Bitwidth	Dimensions	Pipeline Depth	Memory	Banks	Bus width	Host CPU
①	OS	8 bit input 32 bit result	16×16	fully pipelined	64 KiB	5	128 bits	rocket
②	**WS**	8 bit input 32 bit result	16×16	fully pipelined	64 KiB	5	128 bits	rocket
③	**OS + WS**	8 bit input 32 bit result	16×16	fully pipelined	64 KiB	5	128 bits	rocket
④	OS	**32 bit input** **32 bit result**	16×16	fully pipelined	64 KiB	5	128 bits	rocket
⑤	OS	8 bit input 32 bit result	$\mathbf{32 \times 32}$	fully pipelined	64 KiB	5	128 bits	rocket
⑥	OS	8 bit input 32 bit result	16×16	**fully combin.**	64 KiB	5	128 bits	rocket
⑦	OS	8 bit input 32 bit result	16×16	fully pipelined	**256 KiB**	5	128 bits	rocket
⑧	OS	8 bit input 32 bit result	16×16	fully pipelined	64 KiB	**33**	128 bits	rocket
⑨	OS	8 bit input 32 bit result	16×16	fully pipelined	64 KiB	5	**64 bits**	rocket
⑩	OS	8 bit input 32 bit result	16×16	fully pipelined	64 KiB	5	128 bits	**BOOM**

Design Space Exploration: Performance

- Implementation cost (Area and Power)
  - Baseline: 500MHz, 0.467mm^2, 611mW including RISC-V and SA (OS)
  - Bit-width, # of PE array, and processor type are major impact

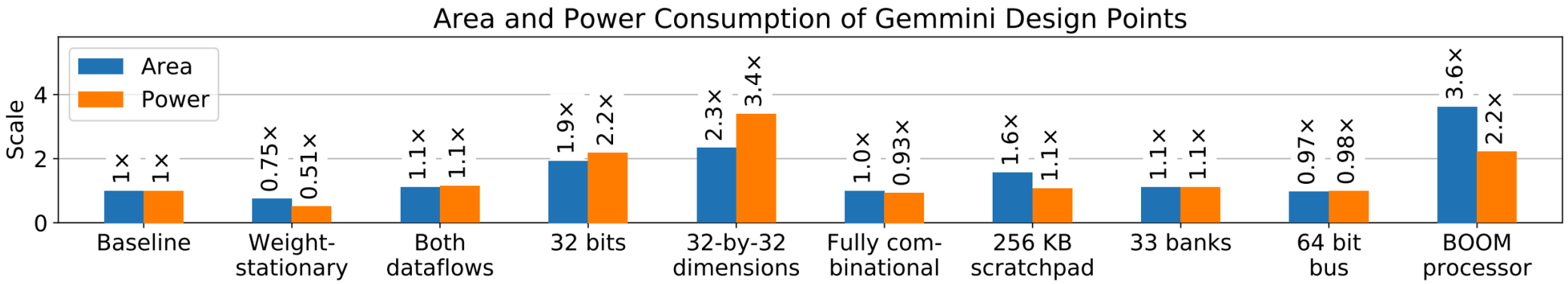
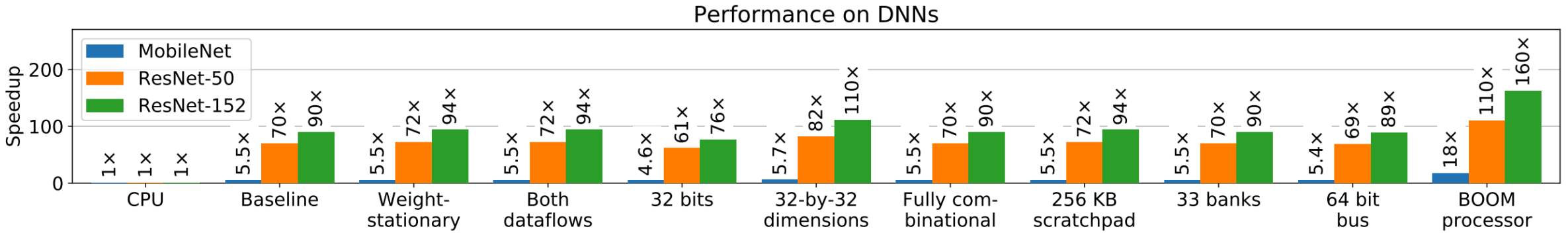


Figure 6: The area and power consumption of synthesized Gemmini designs, normalized to the area and power consumed by the baseline design.

Design Space Exploration: Performance

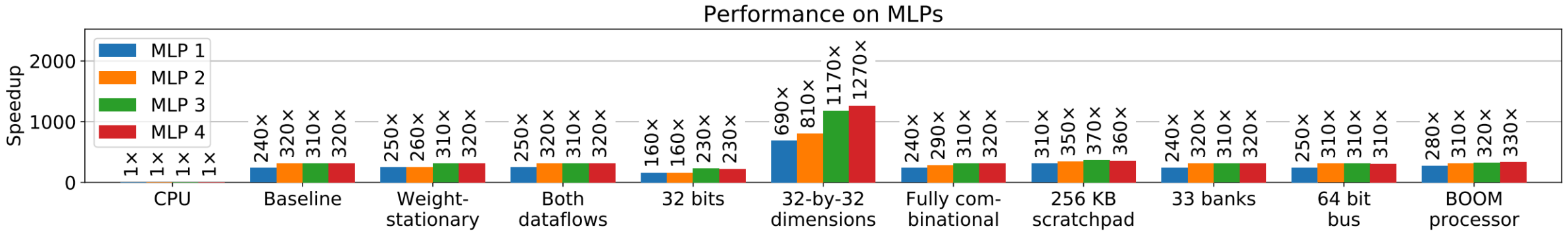
- DNN inference performance (Latency)
  - Processor type boosts performance, while scratchpad had little impact
  - # of PE array is major impact
  - Memory bandwidth and Fully combination are minor impact



(a) The performance of Gemmini designs on various deep neural networks, normalized to the performance of a cache-blocking algorithm on a CPU.

Design Space Exploration: Performance

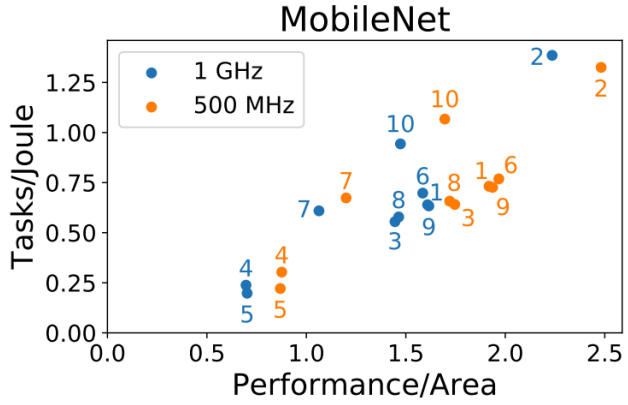
- MLP inference performance (Latency)
  - # of PE array is major impact
  - Increasing # of PE array results a large blocks of memory requested
  - Quadruples the compute throughput
  - Shapes of the layers affect input/weight reuse, tiled GEMM in scratchpad



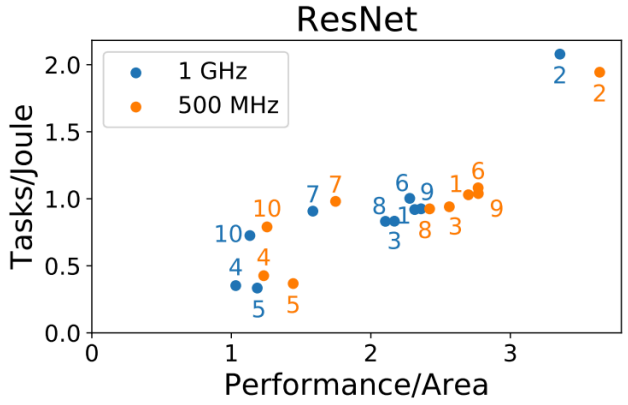
(b) The performance of Gemmini designs on various multi-layer perceptrons, normalized to the performance of a cache-blocking algorithm on a CPU.

Design Space Exploration: Performance

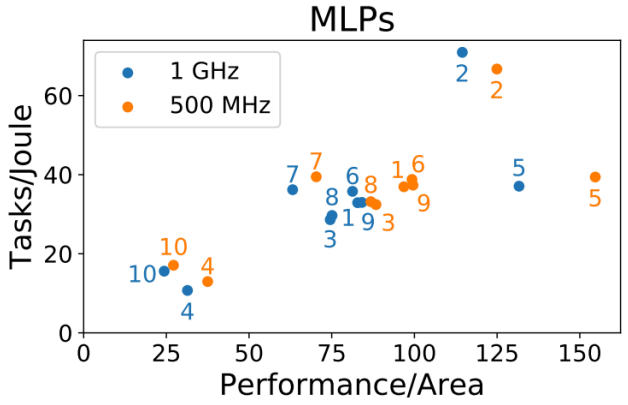
- Efficiency performance (Latency)
  - WS did not noticeably improve performance, but energy and area efficiency
  - 32x32 PE array suffered from very low efficiency (Power and Area)
  - 500MHz designs were more energy- and area-efficient than 1GHz designs



(a) Performance-energy-area trade-offs for MobileNet inference.



(b) Performance-energy-area trade-offs for inference on ResNet50 and ResNet152.



(c) Performance-energy-area trade-offs for inference on various MLPs.

Thank you