深度神经网络的VLSI优化实现

王中风， 国家特聘专家， IEEE Fellow
南京大学 集成电路与智能系统实验室

2019-08-18， 厦门

Outlines

集成电路与系统（ICAIS）实验室

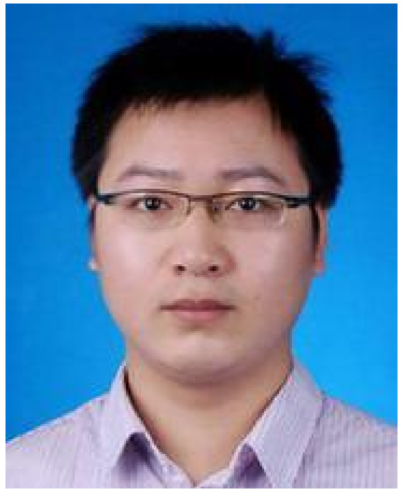
- **集成电路与系统**（ICAIS）实验室成立于2016年5月，已累计发表国际论文约80篇（其中IEEE 期刊论文约25篇)，获竞争性科研经费超千万元。目前拥有4位美国博士和40余位研究生的科研团队。
- 团队成员在过去3年内在**深度学习的硬件实现**方面，**在IEEE 相关期刊上已经发表论文10 余篇**, (同期成果) 在国际同行中处于前列
- 承担了军委科技委创新特区、国家自然科学基金委等资助的关于深度学习VLSI实现方面的国家级项目。 与大企业在通信系统与深度学习等方面有广泛的科研合作
- 过去两年内**共有四篇论文进入IEEE 行业旗舰会议最佳论文奖的终选名单，已经获最佳论文奖一次。**实验室学生团队获得了2018年**全国大学生集成电路创新创业大赛总决赛的一等奖两项**
- 2018年在IEEE Trans. on Circuits and Systems-I （TCAS-I）上发表的关于CNN架构设计的论文自发表以来已经**连续11个月位居该期刊按月下载排行榜 前五名**。
- IEEE TCAS-II **2019年第三期同期发表了ICAIS 实验室的三篇论文**

集成电路与系统（ICAIS）实验室

- 实验室主任：王中风博士

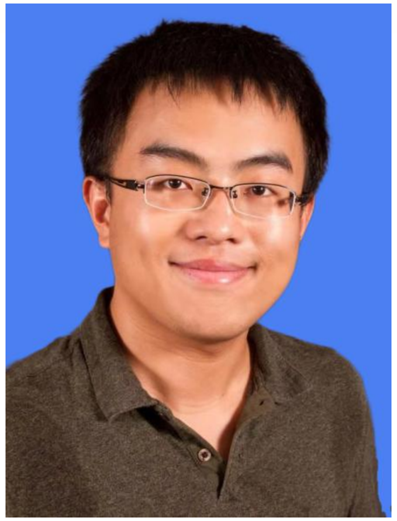
  - **文革后以中专学历考进清华大学的第一人, 本科提前毕业**，在自动化系相继获得学士和硕士学位。2000年获美国明尼苏达大学电机系博士学位
  - **国家“千人计划”专家，IEEE Fellow**，南京大学特聘教授、博导，微电子学院副院长
  - 国际知名的**低功耗超大规模集成电路设计专家**
  - 拥有**二十余年数字信号处理与IC设计领域丰富的研发经验**，在学术界和工业界都有广泛的国际影响，在通信系统的VLSI实现方面有许多技术性突破， 是纠错码设计与 **IC**实现领域里的国际著名专家
  - 曾在美国Oregon State University 任教和在多家半导体公司从事通信系统的设计工作，包括著名的国家半导体公司（现德州仪器）和博通公司。曾任博通公司技术副总监。**先后参与十余款商用芯片的设计**
  - 在高速网络通信方面掌握行业的许多关键技术并发明了多种先进的信道编码方法，其**设计的FEC方案已经被 IEEE等十五种以上网络通信国际标准所采纳**
  - 在国际会议和期刊上**发表200余篇技术论文**，**拥有60多项美国和中国的专利与发明**，编著“VLSI“专辑一部，**先后三次获IEEE行业学会颁发的最佳论文奖**，先后八次担任IEEE四种会刊的编委或客座编辑

实验室教师成员



- 林军博士
  - ◆ 副教授
  - ◆ 美国 Lehigh 大学博士
  - ◆ 在IEEE TVLSI、IEEE TCAS-I/II等顶级杂志和会议上发表 80余篇论文， Google学术引用700余次。获IEEE ISVLSI年会最佳论文奖一次
  - ◆ 近年来主要从事新一代人工智能芯片架构、5G智能基带算法与实现及安全芯片设计等方面的研究

- 杜力博士
  - ◆ 副教授
  - ◆ 美国UCLA 电机系博士
  - ◆ 在IEEE芯片设计类的期刊和会议（包括ISSCC，TCAS-I,TCAD，和JSSC等）上发表论文20余篇
  - ◆ 具有8年的电路领域工业设计经验，申报美国专利12项
  - ◆ 主要从事模数混合，高速光通信和数字电路设计的研究



- 马宇飞博士
  - ◆副研究员
  - ◆美国ASU 电机系博士
  - ◆主要从事深度学习系统硬件加速以及AI系 统的设计自动化方面的研究
  - ◆曾在飞步科技美国公司负责开发针对无人驾驶的SoC智能感知处理器核心模块设计
  - ◆在IEEE 电路设计领域主流期刊和会议上发表高质量论文10余篇，已被引用500多次

Part I: Introduction to VLSI for Signal Processing

Introduction

- More advanced/complicated algorithms are being introduced to improve the system performance, e.g., DNN, LDPC, MIMO, etc, which usually leads to larger power consumption in a real system

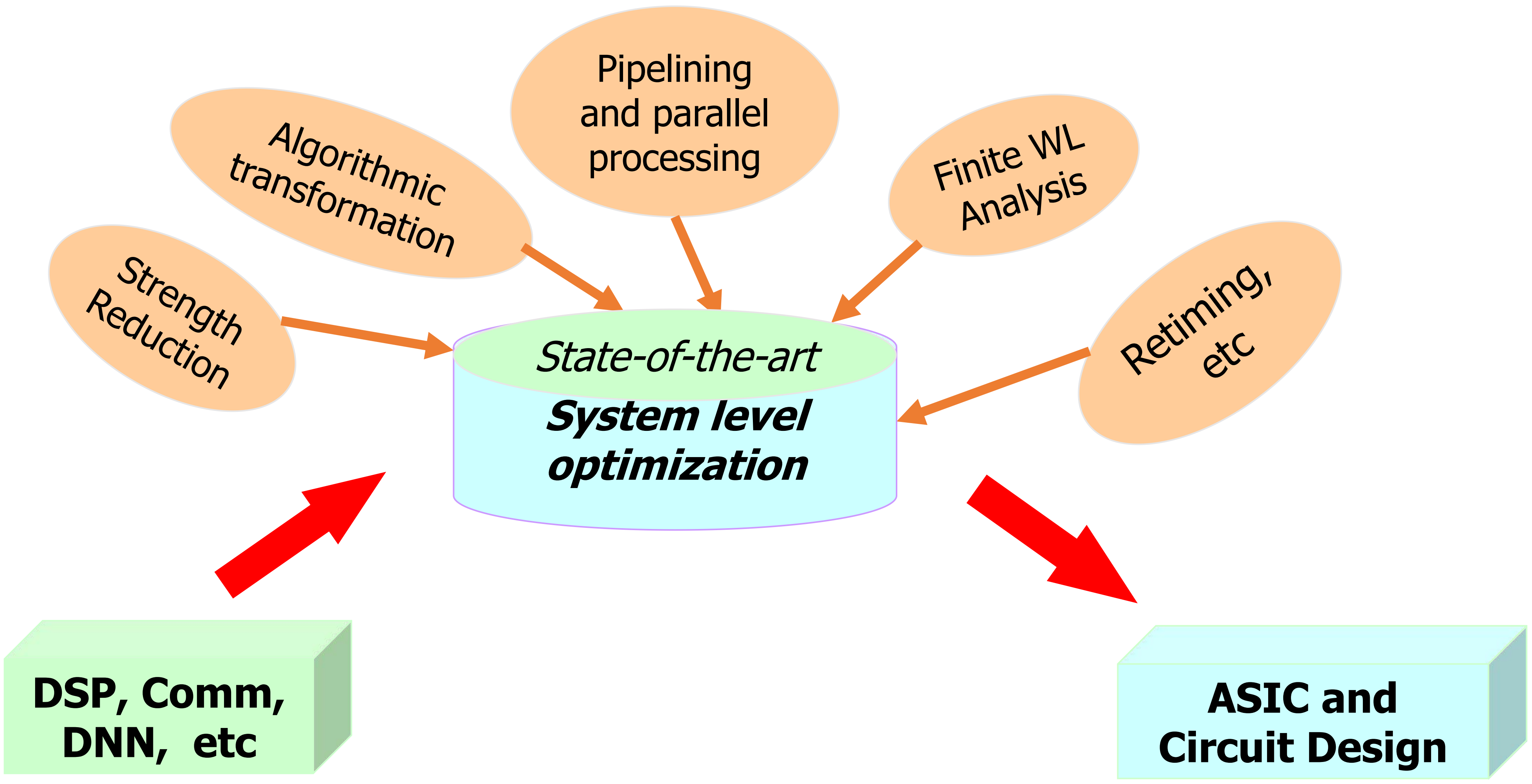
- Real applications are usually speed hungry, e.g., Ethernet speed increases by approximately 10 times every 8 years. Increased data rate directly leads to increased power consumption

- With the popularity of portable computing devices and the increasing need to reduce packaging cost and size, low power dissipation is highly desired

- VLSI optimizations on power or speed are generally performed at multiple levels

- VLSI Signal Processing plays a major role at algorithm level, architecture level and even circuit level optimizations.

The Way of VLSI for Signal Processing

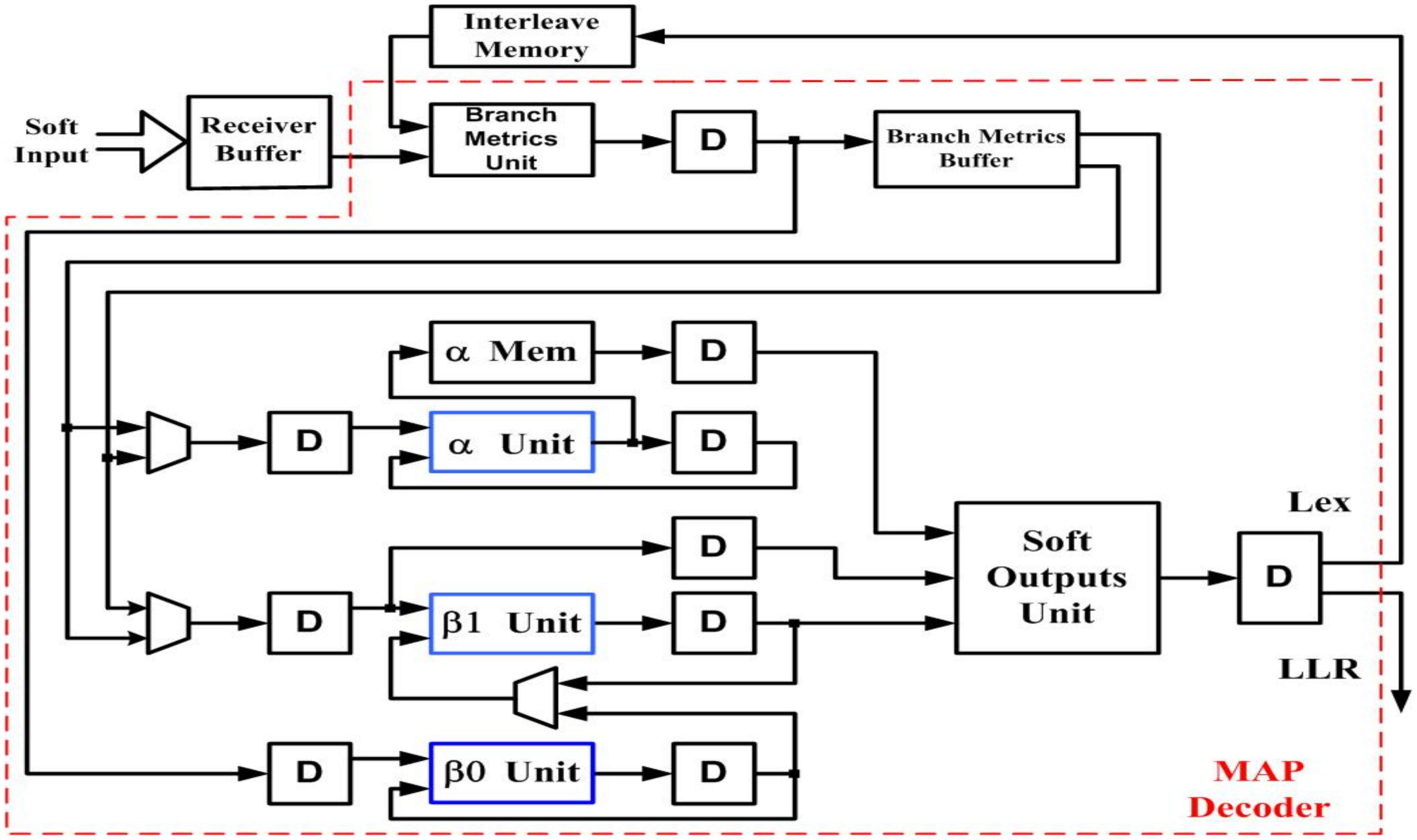


High Speed/Low Power Design Methods

- Generic Design Techniques
  - Power Island,
  - Clock Gating,
  - Transistor Resizing,
  - Operating in subthreshold regime,
  - MTCMOS, VTCMOS, Dual-Vdd, etc,
  - Sleep Transistor (cluster based, DSTN, etc),
  - Forward/Reverse Body Biasing (FBB/RBB),
- Application Specific Techniques
  - Pipelining/parallel processing/retiming w/ reduced Vdd,
  - Numerical/algorithmic strength reduction (e.g., FFT/DCT),
  - Joint algorithmic and architectural level optimization,
  - Adaptive computing, etc.

Hi-Speed MAP Decoder Architectures

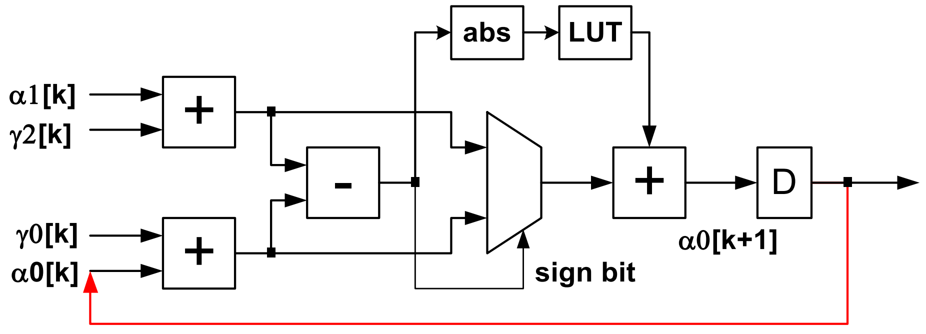
- Max-a-posterior (MAP) algorithm: optimal for turbo decoding
- Recursive computations form the bottleneck in hi-speed design



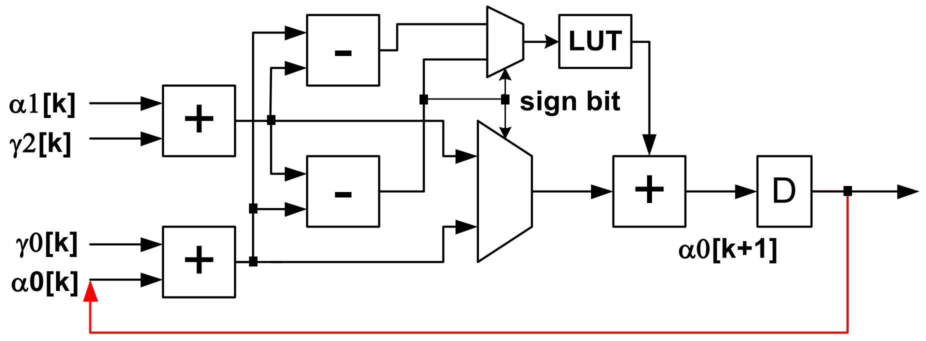
Recursive State Metrics Computation

$$\alpha_0(k+1) = \max\{\alpha_0 + \gamma_0(k),\ \alpha_1 + \gamma_2(k)]\} + \log(1 + e^{-|\Delta|})$$
$$\equiv \max{}^*\{\alpha_0 + \gamma_0(k), \alpha_1 + \gamma_2(k)\}$$

- Recursive computation forms the high speed bottleneck
- The delay of computing the absolute value can be saved by introducing an extra subtraction unit.



(a) traditional recursion arch A

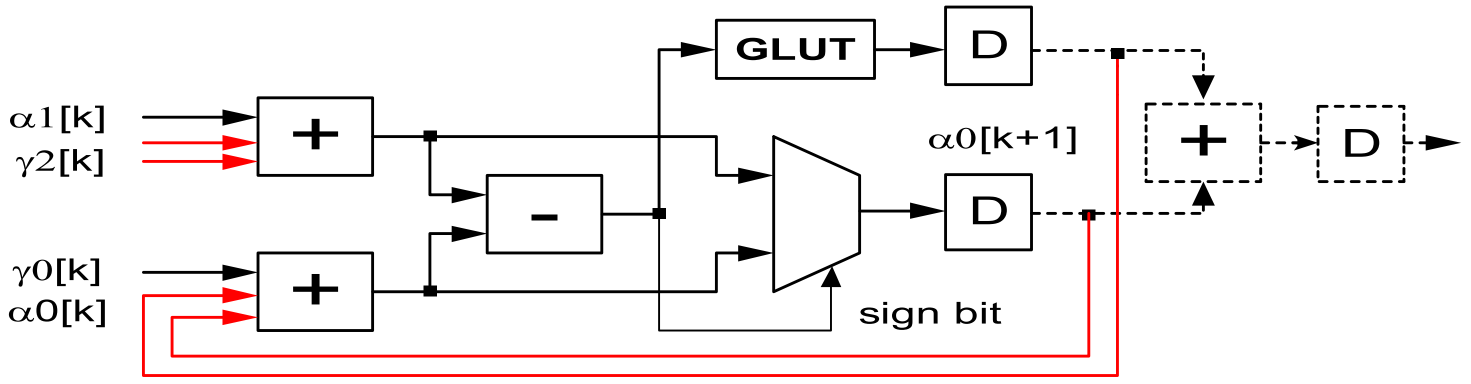


(b) traditional recursion arch B

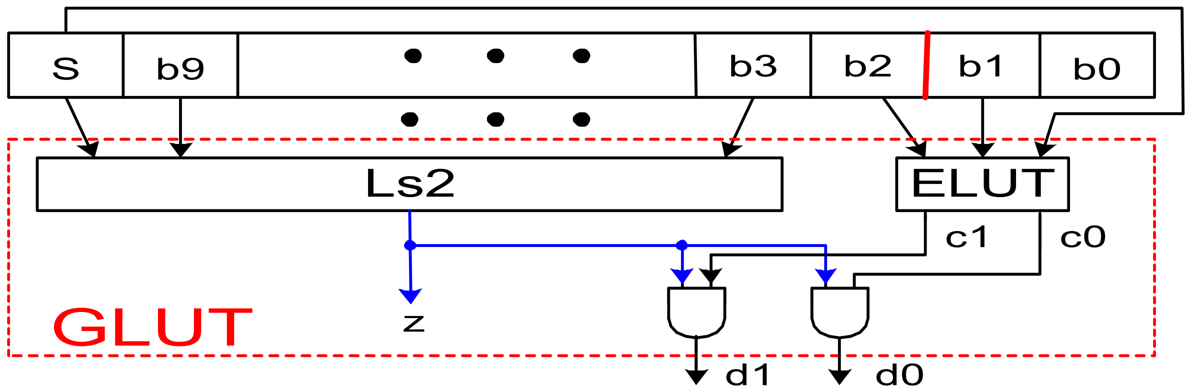
NANJING UNIVERSITY

An Efficient ACSU: Arch C

- Avoid the computation of absolute value with GLUT, retime the final addition operation to reduce the loop latency.



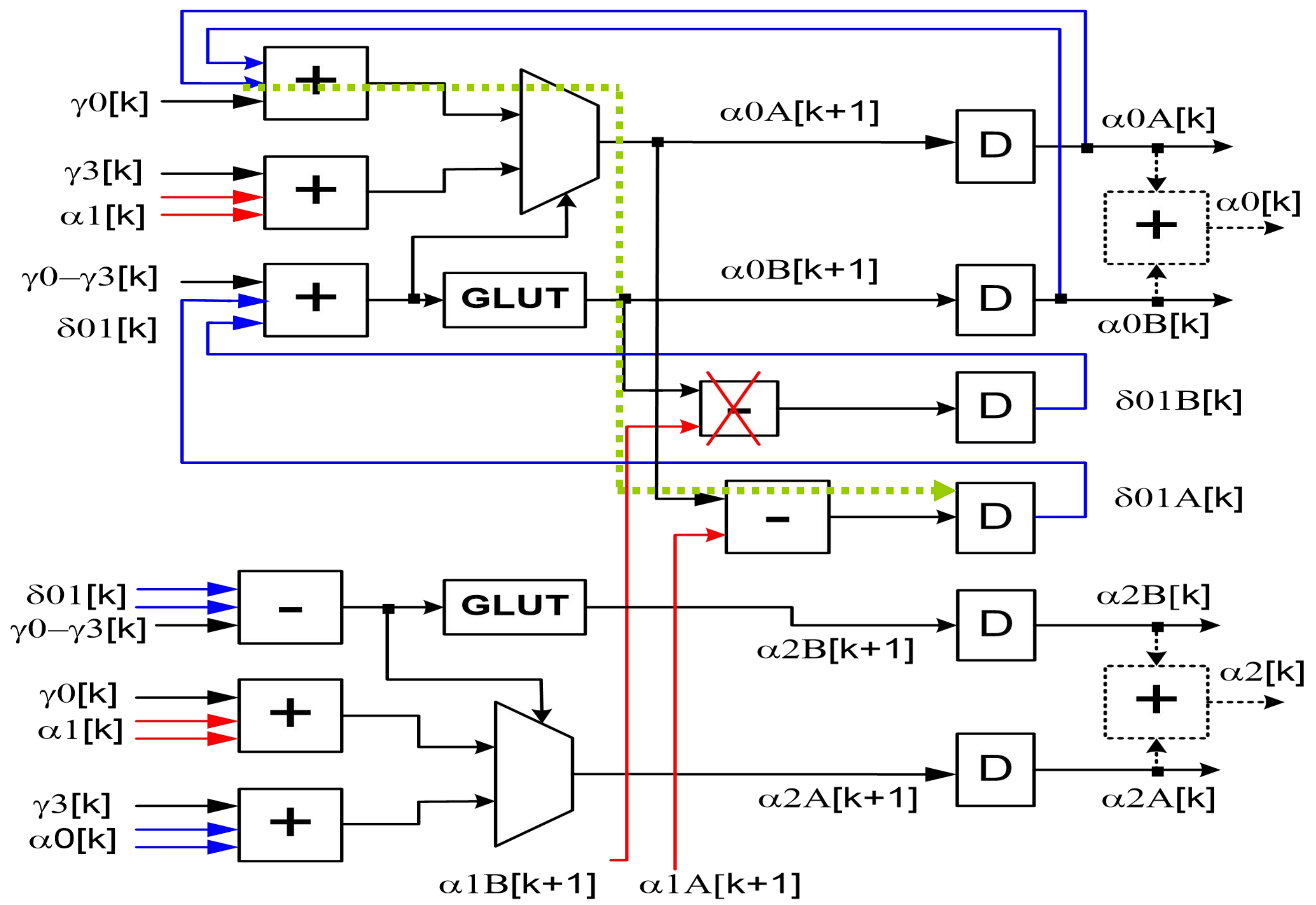
(a) efficient recursion, Arch-C



(b) function details of GLUT

Advanced Radix-2: Arch-D

$$(\alpha_0 + \gamma_0) - (\alpha_1 + \gamma_3) = (\gamma_0 - \gamma_3) + \delta_{01A} + \delta_{01B} \approx \gamma_{03} + \delta_{01A}$$

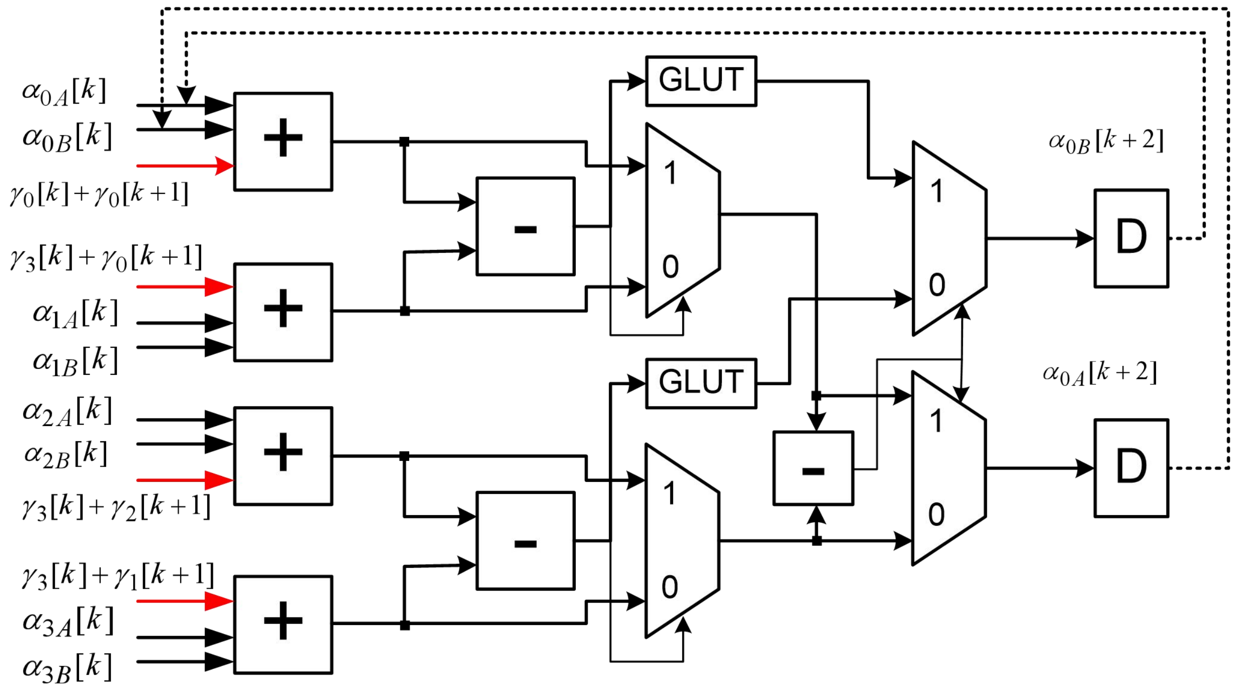


Improved Radix-4: Arch-F

- The throughput can be increased further if performing two iterations in one cycle.

$$\alpha_0[k+2] \approx \max\{\ \max^*\ (\alpha_0[k]+\gamma_0[k],\ \ \alpha_1[k]+\gamma_3[k]) + \gamma_0[k+1],$$

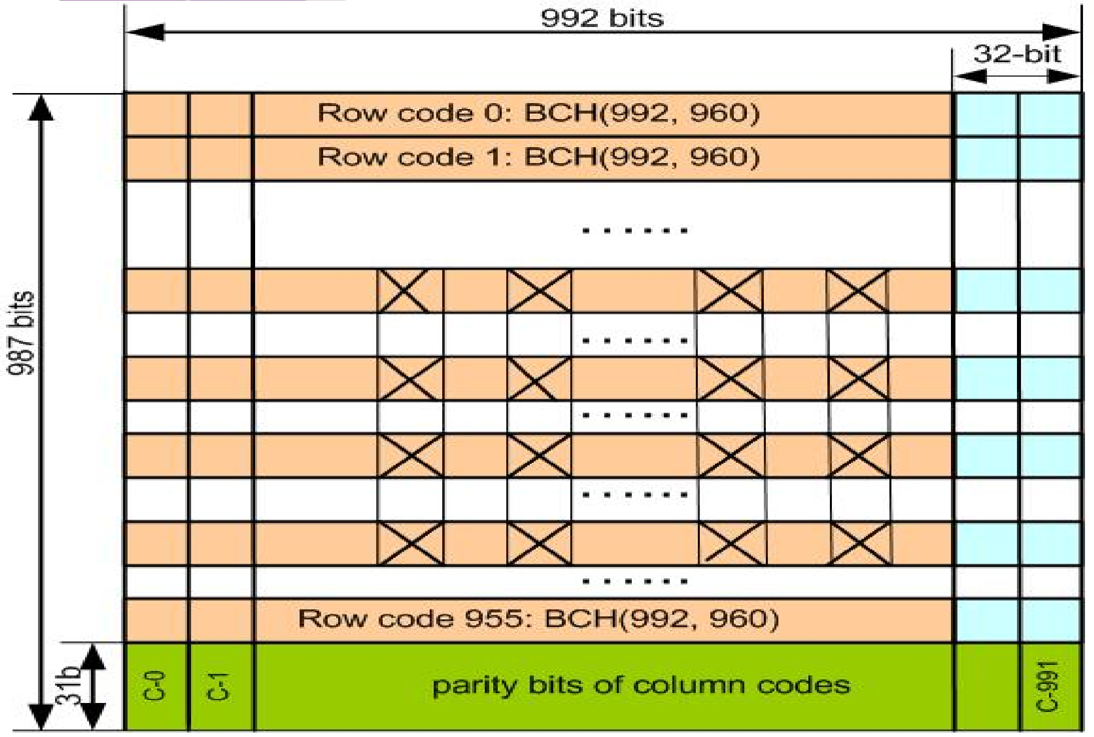
$$\max^*\ (\alpha_2 k]+\gamma_2[k],\ \ \alpha_3[k]+\gamma_1[k]) + \gamma_3[k+1])\}.$$



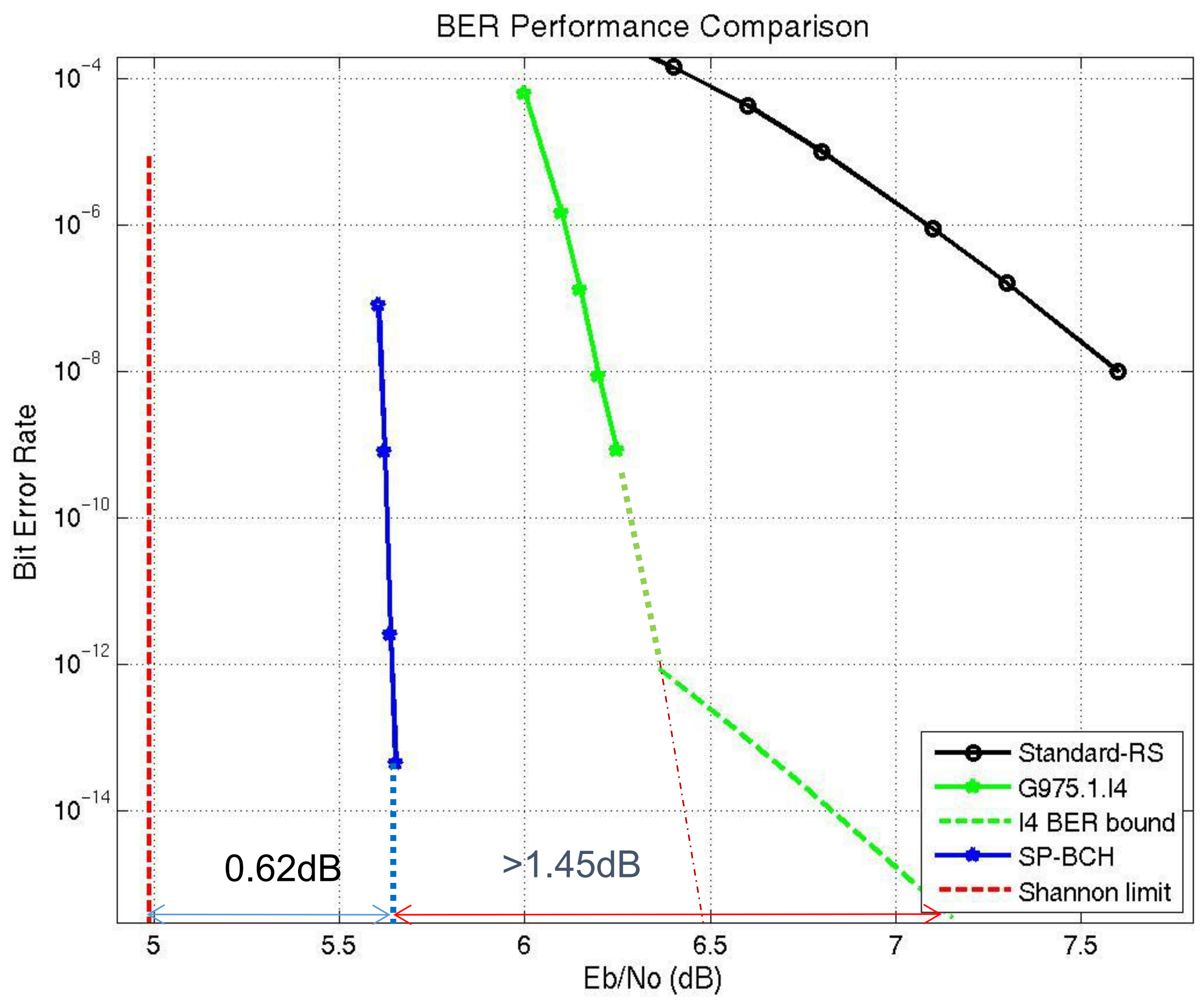
Performance Comparisons

	Max Clock Freq. (Mhz)	Relative Area	Relative Processing Speed
Arch-A	241	1.0	1.0
Arch-C	333	0.87	1.38
Arch-R	335	1.14	1.39
Arch-E	182	1.82	1.51
Arch-D	370	1.03	1.54
Arch-F	241	1.99	2.0

100Gbps+ Product Code Decoder Design



992 bits
32-bit
987 bits
Row code 0: BCH(992, 960)
Row code 1: BCH(992, 960)
Row code 955: BCH(992, 960)
31b
C-0
C-1
parity bits of column codes
C-991



- Product codes can achieve near-limit performance with iterative decoding, which would cause memory access conflicts in parallel processing and decrease the throughput linearly.

Implementation Results of Super-FEC

- Decoder (FPGA)
  - ◆Overall utilization: 62%
  - ◆Max clock frequency: 196 Mhz
  - ◆Max number of iterations: 40Gb/s at 15 iterations
- Encoder (FPGA)
  - ◆Overall utilization: 6%
  - ◆Max freq.: 238 Mhz
  - ◆Max rate: > 40Gb/s
- ASIC
  - ◆Under 40nm CMOS, can achieve over 100Gbps throughput

Part III: Introduction to Deep Learning

Applications of Deep Learning

- Deep Learning Has Achieved Breakthrough in Various Applications

Medicine

Fintech

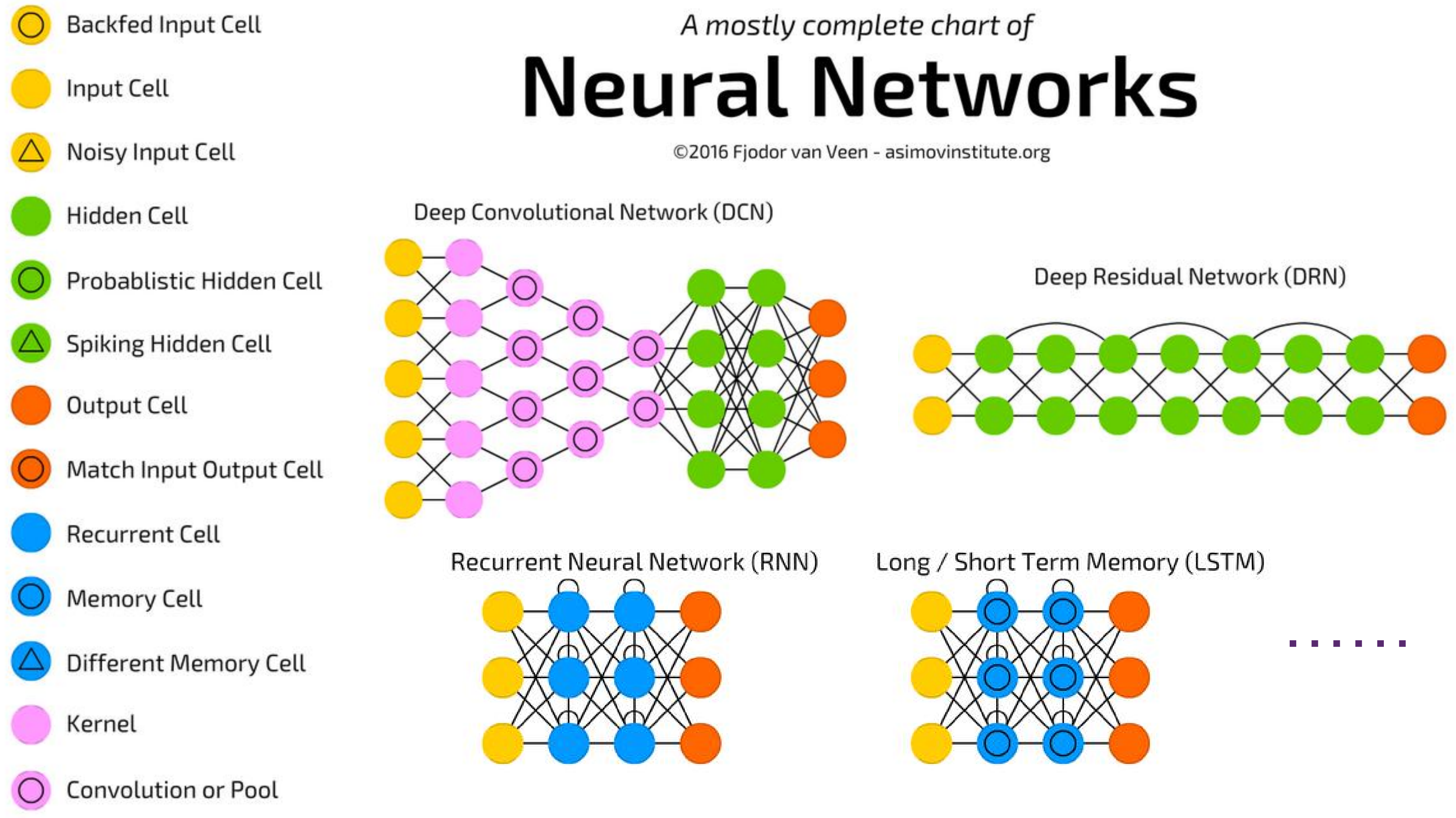
……

Speech

Self-driving Cars

Deep Neural Networks

- Fundamental for Deep Learning

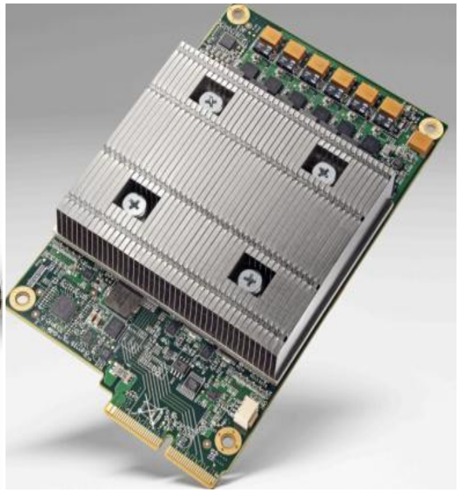


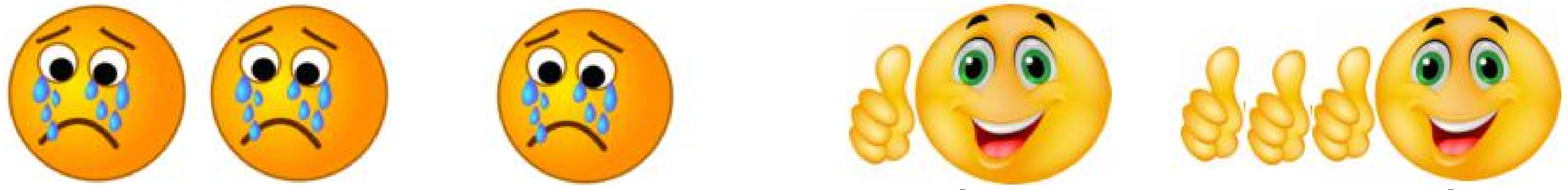
......

http://www.asimovinstitute.org/neural-network-zoo/

Deep Learning Inference Platform

- When we talk about energy efficiency……

CPU	GPU	FPGA	ASIC
			

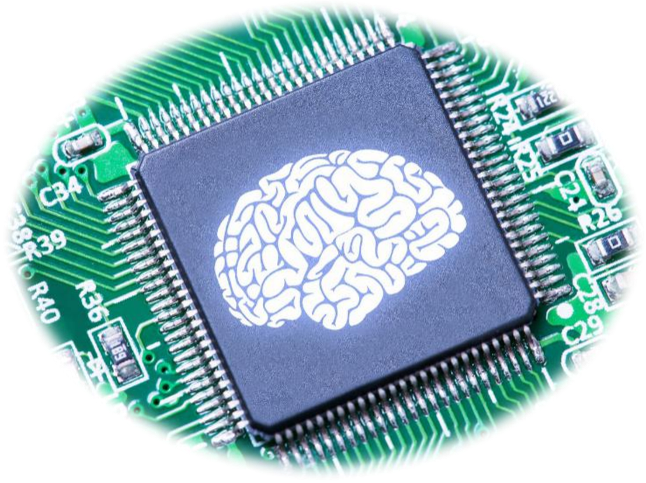


Our focus

- Efficient DNN processors are desired

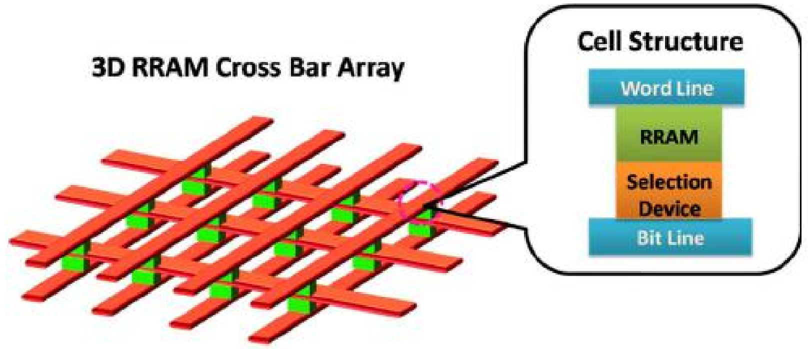
Academic Research in AI Processors

- Challenges of Deep Neural Network Processors
  - High-dimension design space
    - Tons of variables, even for basic designs
  - Complex parameter interactions
    - DNNs are notoriously difficult to tune
  - High memory storage requirement
    - DNNs are over parameterized
  - Ultra-High computation requirements
    - Hundred millions of MACs per image



Academic Research in AI Processors

- Dataflow Optimizations
  - ❑ Input/Output/Row Stationary Dataflow
  - ❑ Reconfigurable Data Reuse Pattern
  - ❑ Reconfigurable Layer Tiling
- Dedicated ISA for AI Processors
  - ❑ Example: Cambricon(寒武纪)
    - ➢ 9.86x higher code density than x86
    - ➢ Only 4.5% slower than hard-wired DNN accelerator
- Design with Emerging Memory Devices
  - ❑ 8T-6T hybrid SRAM(P↓, S↑)
  - ❑ RRAM-based accelerator
  - ❑ 3D Memory (BW↑)
  - ❑ Extremely low power
  - ❑ Compute in memory



Part IV: VLSI Optimizations for DNNs

Optimization for CNNs

- Optimization Schemes
  - Computation optimization：
    - Reduce 3D Conv to 1D Conv, optimize 1D Conv through fast convolution algorithms, then restore 1D Conv to 3D Conv
    - Fast FIR algorithm for convolution
  - Storage optimization：
    - Inter-layer partial storage and intra-layer ping-pong reuse
  - Bandwidth optimization：
    - Resource partition and pipeline process

NANJING UNIVERSITY

Efficient Hardware Architectures for DCNN

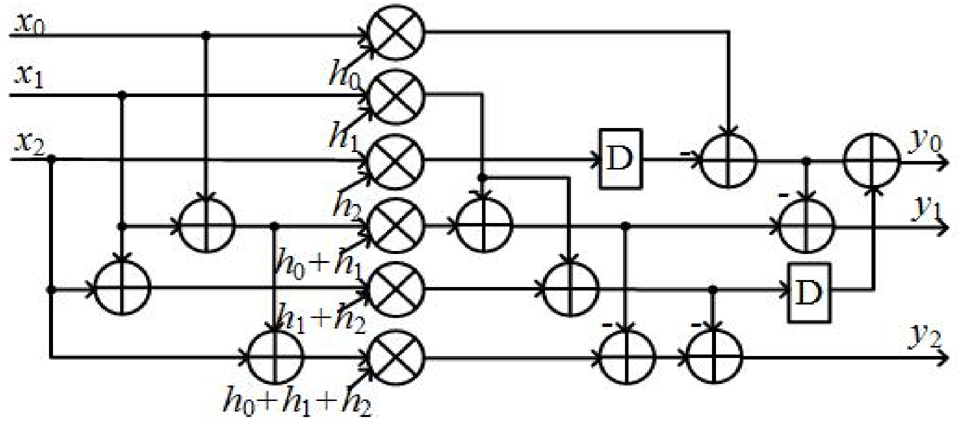
- Convolutions are basic and complex computations in CNNs
- We have derived N-parallel Fast FIR Algorithms (FFA) for 1D convolutions, e.g., 3-parallel and 5-parallel FFAs
- Fast Convolution Units (FCUs) for efficient hardware implementation of CNN
- 3-Parallel Fast FIR and 3-Parallel FCU
  - 3-parallel FIR with algorithm strength reduction

$$Y_0 = H_0X_0 - z^{-3}H_2X_2 + z^{-3}[(H_1+H_2)(X_1+X_2) - H_1X_1]$$

$$Y_1 = [(H_0+H_1)(X_0+X_1) - H_1X_1] - [H_0X_0 - z^{-3}H_2X_2]$$

$$Y_2 = [(H_0+H_1+H_2)(X_0+X_1+X_2)] \\ -[(H_0+H_1)(X_0+X_1) - H_1X_1] \\ -[(H_1+H_2)(X_1+X_2) - H_1X_1]$$

  - Save 33% multiplications compared to regular convolutions



NANJING UNIVERSITY

Efficient Hardware Architectures for DCNN

- Fast FIR Algorithm
  - 5-parallel Fast FIR Algorithm is derived for the first time
  - Save 40% multiplications

$$\mathbf{Y_5} = \mathbf{Q_5 H_5 P_5 X_5}$$

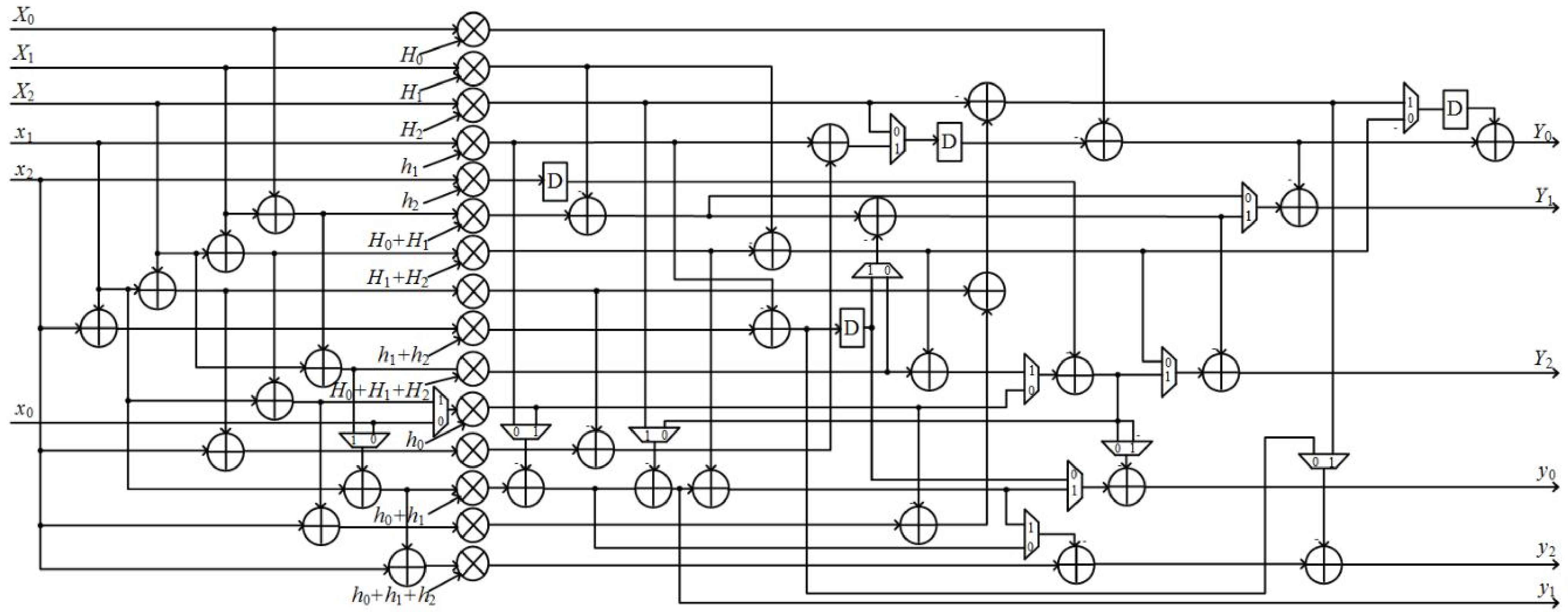
$$\mathbf{Y_5} = \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix}, \mathbf{X_5} = \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}, \mathbf{H_5} = diag \begin{bmatrix} H_0 \\ H_1 \\ H_2 \\ H_3 \\ H_4 \\ H_0 + H_1 \\ H_1 + H_2 \\ H_2 + H_3 \\ H_3 + H_4 \\ H_0 + H_1 + H_2 \\ H_1 + H_2 + H_3 \\ H_2 + H_3 + H_4 \\ H_0 + H_1 + H_2 + H_3 \\ H_1 + H_2 + H_3 + H_4 \\ H_0 + H_1 + H_2 + H_3 + H_4 \end{bmatrix}, \mathbf{P_5} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{Q_5} = \begin{bmatrix} 1 & 0 & 0 & 0 & z^{-5} & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -z^{-5} & 0 & 0 & 0 & z^{-5} & 0 & 0 & 0 & -z^{-5} & 0 & 0 & 0 \\ 0 & -1 & 0 & z^{-5} & 0 & 1 & 0 & 0 & -z^{-5} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -z^{-5} & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

NANJING UNIVERSITY

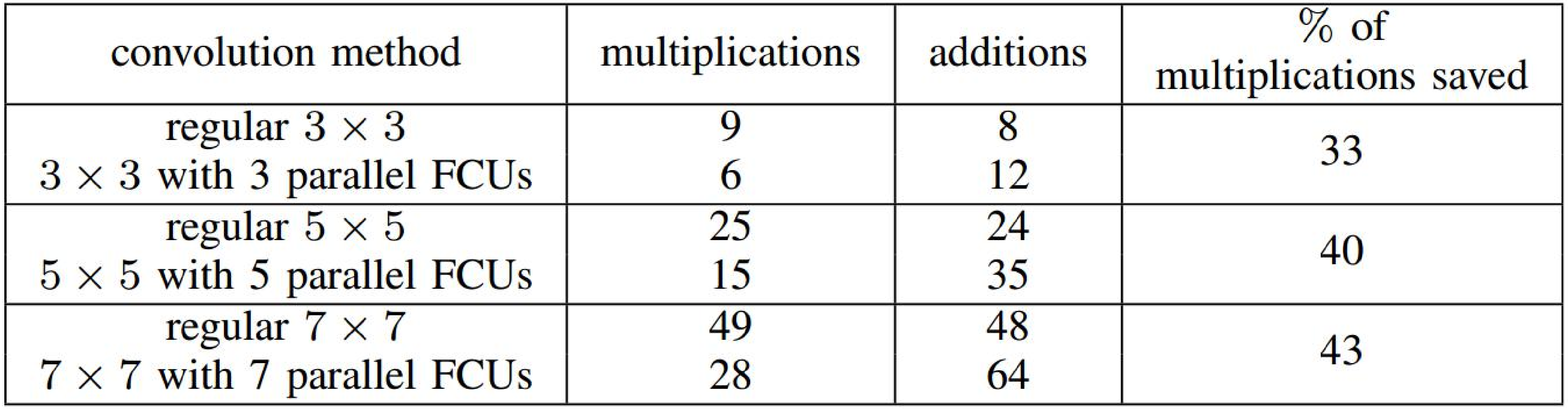
Efficient Hardware Architectures for DCNN

- Fast FIR Algorithm
  - Reconfigurable Fast Convolution Unit (FCU)
  - Operate as two 3-parallel FCUs or one 5-parallel FCU



Efficient Hardware Architectures for DCNN

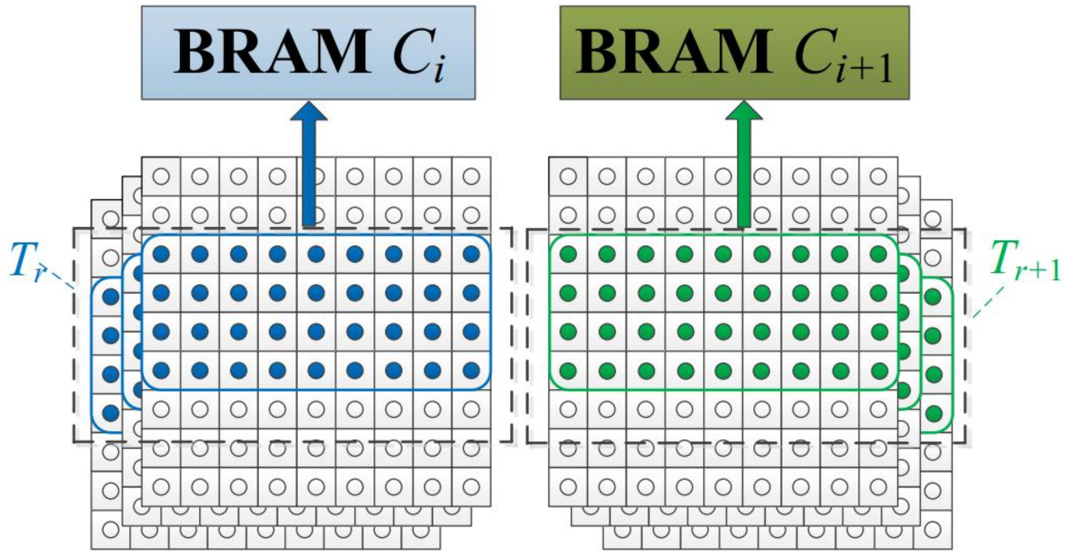
- Fast FIR Algorithm
  - ◆Comparison between regular convolutions and FCUs

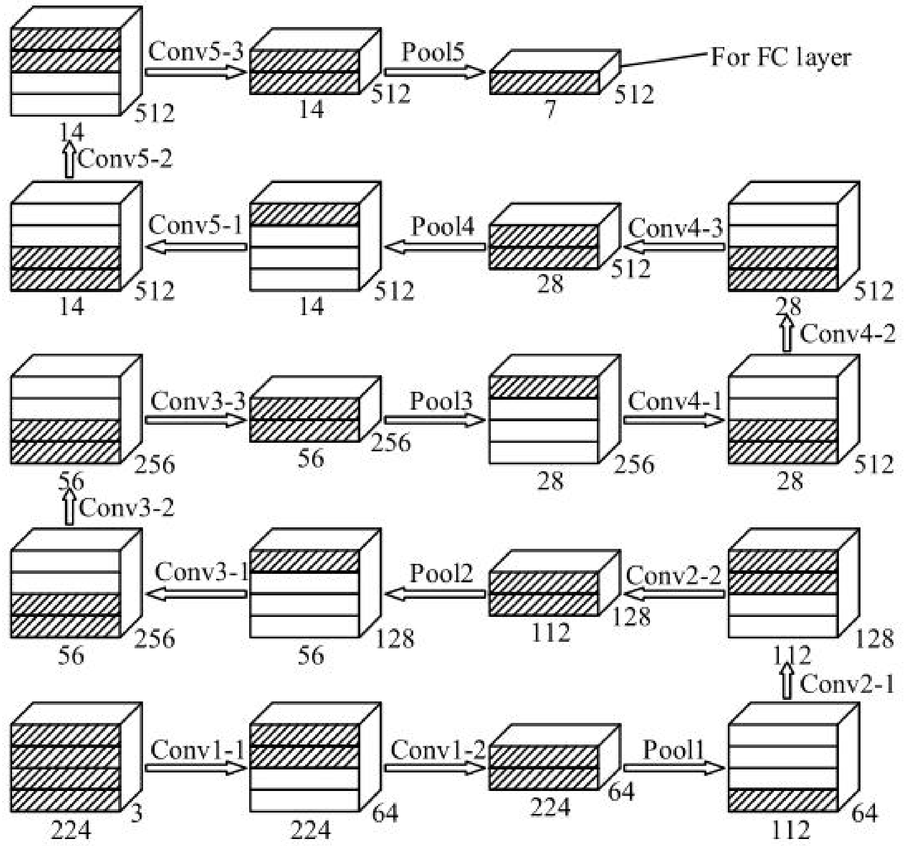


convolution method	multiplications	additions	% of multiplications saved
regular 3×3	9	8	33
3×3 with 3 parallel FCUs	6	12	
regular 5×5	25	24	40
5×5 with 5 parallel FCUs	15	35	
regular 7×7	49	48	43
7×7 with 7 parallel FCUs	28	64	

Efficient Hardware Architectures for DCNN

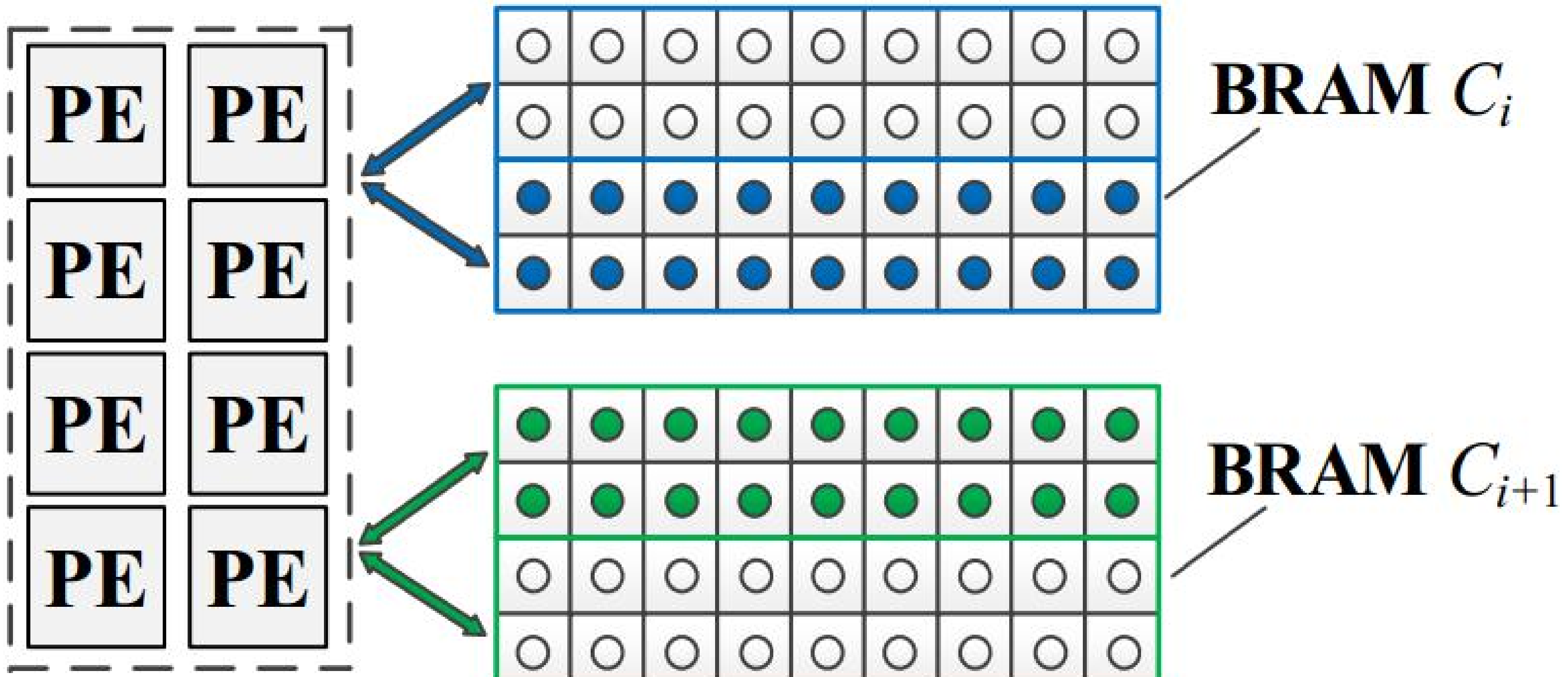
- Memory Efficient Storage and Computation Flow
  - Inter-layer partial storage
    - Each layer is tiled by factor Tr
    - The tiled intermediate data of each layer are stored in a specific BRAM
    - 14x reduction on storage requirement compared to layer-wise scheme





Efficient Hardware Architectures for DCNN

- Memory Efficient Storage and Computation Flow
  - Intra-layer ping-pong reuse
    - Each specific BRAM is split as a dual buffer
    - The two segments in a specific BRAM are reused in a ping-pong manner

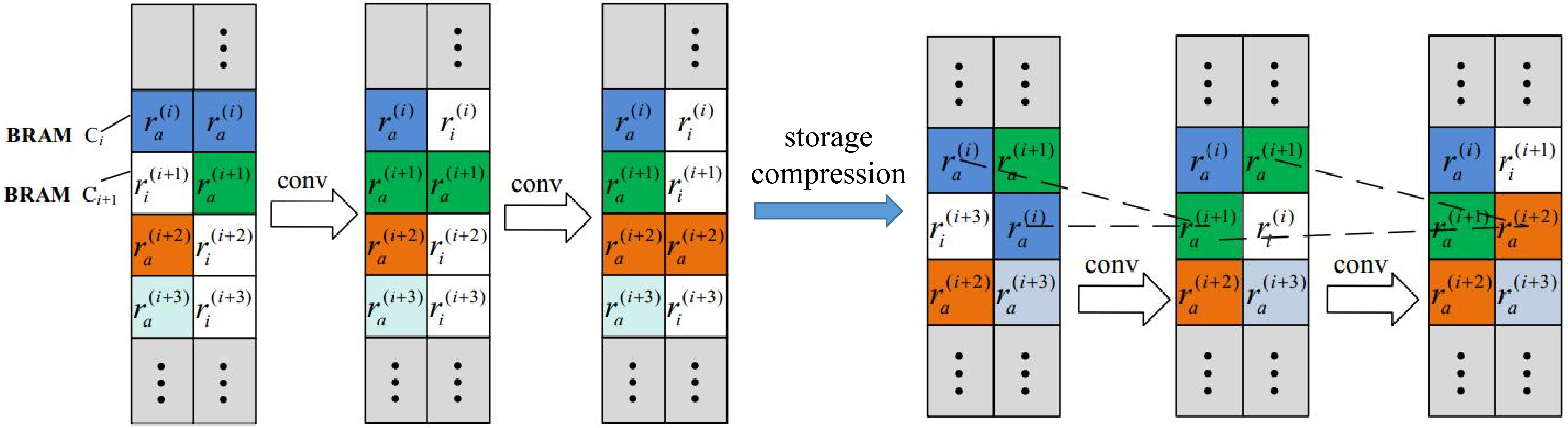


Efficient Hardware Architectures for DCNN

- Memory Efficient Storage and Computation Flow
  - Storage compression
    - Intelligently reuse idle-state BRAM
    - Save further 20% on-chip memory

$r_a^{(i)}$ $r_a^{(i+1)}$ active row data of layer i, i+1

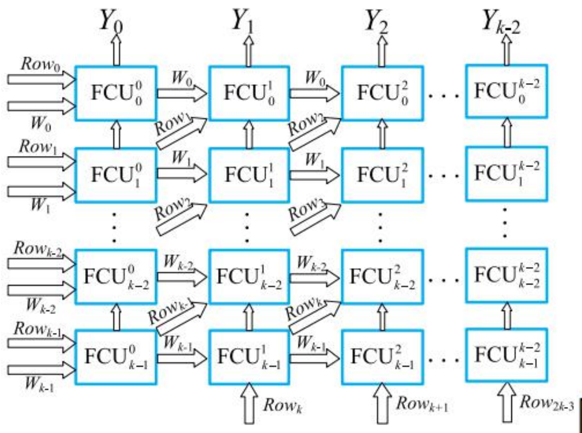
$r_i^{(i)}$ inactive row data of layer i



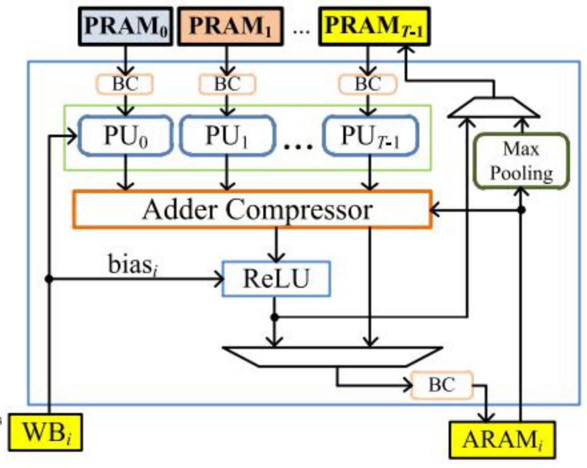
Efficient Hardware Architectures for DCNN

- Micro Architecture
  - PU is a 2D FCU array
  - CP mainly consists of PUs along with some other computation units
  - The overall architecture is composed of CPs and memory footprints

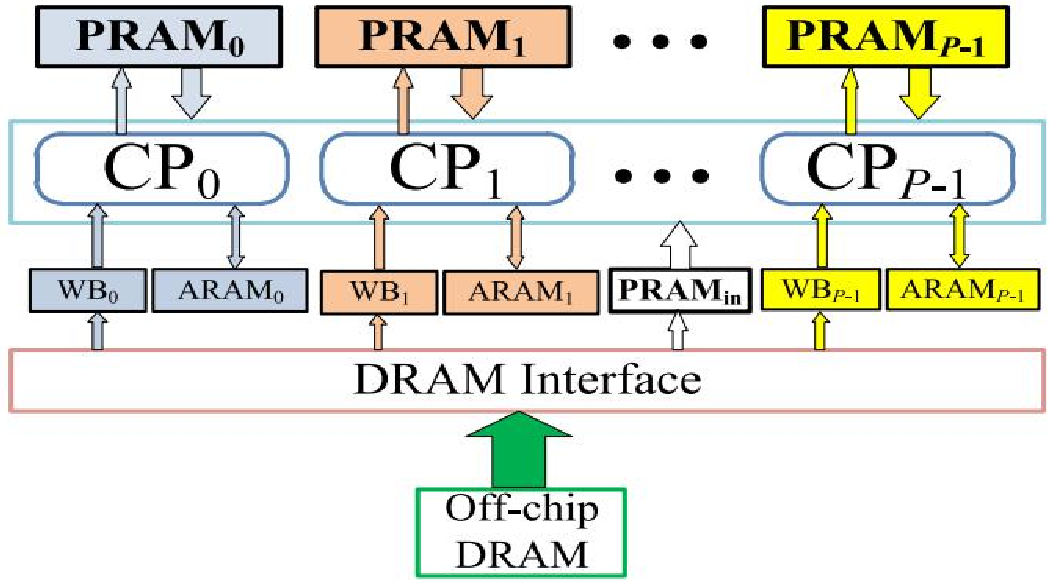
PU



CP

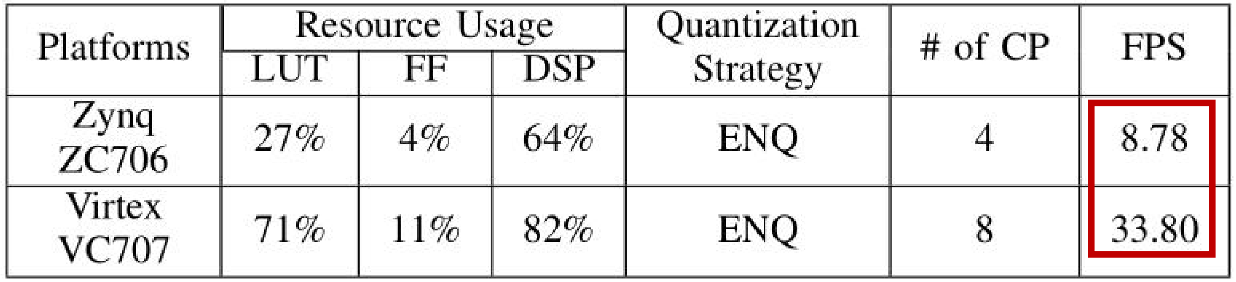


Overall Architecture

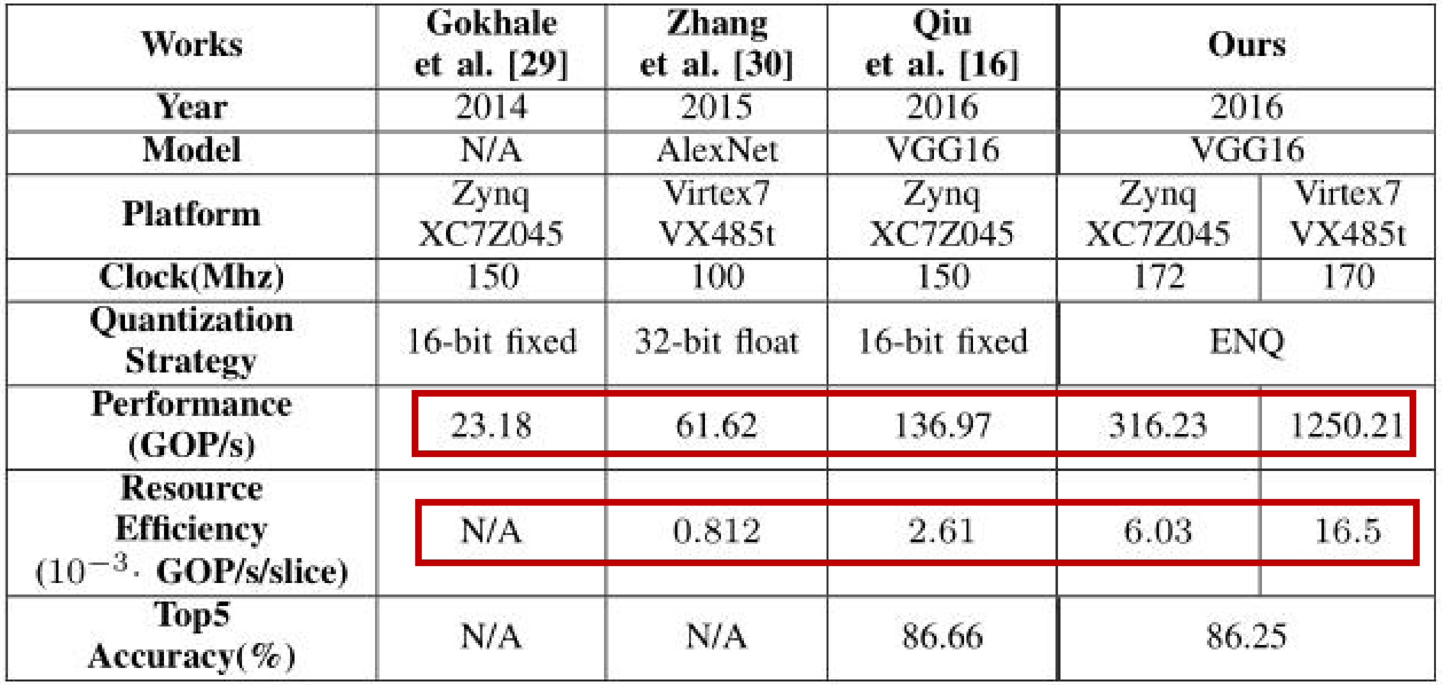


Efficient Hardware Architectures for DCNN

- Results and Comparison
  - Implementation results of VGG16 on two FPGA platforms
  - Comparison with previous works



Platforms	Resource Usage			Quantization Strategy	# of CP	FPS
	LUT	FF	DSP			
Zynq ZC706	27%	4%	64%	ENQ	4	8.78
Virtex VC707	71%	11%	82%	ENQ	8	33.80



Works	**Gokhale et al. [29]**	**Zhang et al. [30]**	**Qiu et al. [16]**	**Ours**	
Year	2014	2015	2016	2016	
Model	N/A	AlexNet	VGG16	VGG16	
Platform	Zynq XC7Z045	Virtex7 VX485t	Zynq XC7Z045	Zynq XC7Z045	Virtex7 VX485t
Clock(Mhz)	150	100	150	172	170
Quantization Strategy	16-bit fixed	32-bit float	16-bit fixed	ENQ	
Performance (GOP/s)	23.18	61.62	136.97	316.23	1250.21
Resource Efficiency ($10^{-3}\cdot$ **GOP/s/slice**)	N/A	0.812	2.61	6.03	16.5
Top5 Accuracy(%)	N/A	N/A	86.66	86.25	

FPAP: Energy-Quality Scalable CNNs

- Problems to Solve: I
  - Different data precisions requirements in various layers/models
    - Range from 1b to 16b for weights/activations
    - Traditional architectures: unified computational components of long data width
    - Not efficient when precisions vary

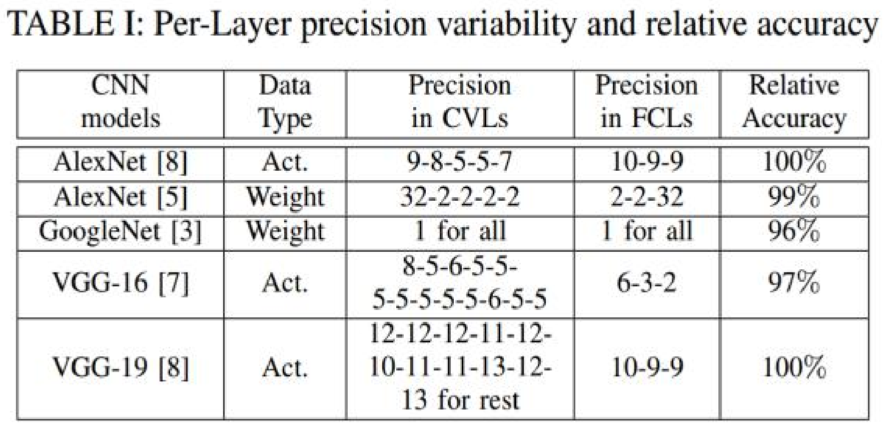


TABLE I: Per-Layer precision variability and relative accuracy

CNN models	Data Type	Precision in CVLs	Precision in FCLs	Relative Accuracy
AlexNet [8]	Act.	9-8-5-5-7	10-9-9	100%
AlexNet [5]	Weight	32-2-2-2-2	2-2-32	99%
GoogleNet [3]	Weight	1 for all	1 for all	96%
VGG-16 [7]	Act.	8-5-6-5-5-5-5-5-5-5-6-5-5	6-3-2	97%
VGG-19 [8]	Act.	12-12-12-11-12-10-11-11-13-12-13 for rest	10-9-9	100%

* Wang, Y., Lin, J., & Wang, Z. (2018, July). FPAP: A Folded Architecture for Efficient Computing of Convolutional Neural Networks. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (pp. 503-508). IEEE.

FPAP: Energy-Quality Scalable CNNs

- Problems to Solve: II
  - CNNs can have sparse representations
    - Both weights and activations
    - Irregular sparsity brings significant load-imbalance issue in hardware
  - A flexible but efficient architecture is required
    - Eliminates all computational redundancies
      - Adapt to different data precisions
      - Exploit both weight sparsity and activation sparsity
    - Enables energy-quality scaling through dynamic precision adjustment

Layer in AlexNet	Weights after Pruning
Conv1	81%
Conv2	20%
Conv3	19%
Conv4	20%
Conv5	20%

* Zhang, T., Ye, S., Zhang, K., Tang, J., Wen, W., Fardad, M., & Wang, Y. (2018). A systematic DNN weight pruning framework using alternating direction method of multipliers. *arXiv preprint arXiv:1804.03294*.

FPAP: Energy-Quality Scalable CNNs

- Fold Computations in CNNs
  - Precision-Adjustable Multiply-Add (PAMAC)
    - MAC operation can be decomposed into multiple shift-and-add ops:

$$Y = AW + T = \sum_{i=0}^{n-1} 2^i (W_i \times A) + T$$

    - The decomposed input can be either weight (WD) or activation (AD)
    - Fold this MAC into a single adder and sum those terms over multiple cycles
    - Can adapt to different data precisions n, only accumulate necessary terms
    - Throughput/energy scale with data precision
    - Smaller area and shorter critical path
    - Side effect: lower speed

FPAP: Energy-Quality Scalable CNNs

- Fold Computations in CNNs(cont'd)
  - Reduce computation of PAMAC
    - Bit-pair Encoding (BPE) algorithm reduce adds by half

$$V_i = (-2W_{2i+1} + W_{2i} + W_{2i-1}) \qquad V_i \in \{-2,-1,0,1,2\}$$

$$AW + T = AV_0 + 2^2 AV_1 + \cdots + 2^{n-2} AV_{\frac{n}{2}} + T$$

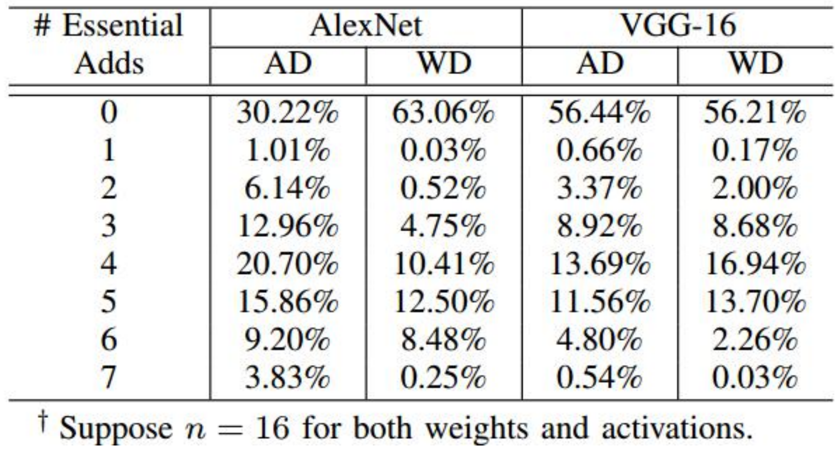
    - Possibly 25% of the add ops are still redundant
      - V_i may be zero
      - Only sum ``essential add terms"!

$$V_1 = 0\,, V_2 = 0$$

$$AW + T = AV_0 + 2^2 \cancel{AV_1} + 2^4 \cancel{AV_2} + \cdots + 2^{n-2} AV_{\frac{n}{2}} + T$$

$BPEB_i$	V_i	$BPEB_i$	V_i
3'b000	0	3'b001	1
3'b100	-2	3'b101	-1
3'b010	1	3'b011	2
3'b110	-1	3'b111	-0

TABLE II
DISTRIBUTION OF ESSENTIAL ADD TERMS



# Essential Adds	AlexNet		VGG-16	
	AD	WD	AD	WD
0	30.22%	63.06%	56.44%	56.21%
1	1.01%	0.03%	0.66%	0.17%
2	6.14%	0.52%	3.37%	2.00%
3	12.96%	4.75%	8.92%	8.68%
4	20.70%	10.41%	13.69%	16.94%
5	15.86%	12.50%	11.56%	13.70%
6	9.20%	8.48%	4.80%	2.26%
7	3.83%	0.25%	0.54%	0.03%

† Suppose $n = 16$ for both weights and activations.

* Wang, Y., Lin, J., & Wang, Z. (2018). FPAP: A Folded Architecture for Energy-Quality Scalable Convolutional Neural Networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, (99), 1-14.

FPAP: Energy-Quality Scalable CNNs

- Fold Computations in CNNs(cont'd)
  - Reduce computation of PAMAC
    - Fine-grained Weight-Decomposition (WD) and Activation-Decomposition (AD) decision
      - WD needs less adds in some layers while AD is better in others
      - Further: Per-MAC granularity
      - Ensure least number of adds
    - Dynamic approximate computing
      - Enable tradeoff between accuracy & throughput
      - Kind of "dynamic quantization" of CNNs

$$AW + T \approx 2^{n-2} A V_{\lceil \frac{n}{2} \rceil} + T$$

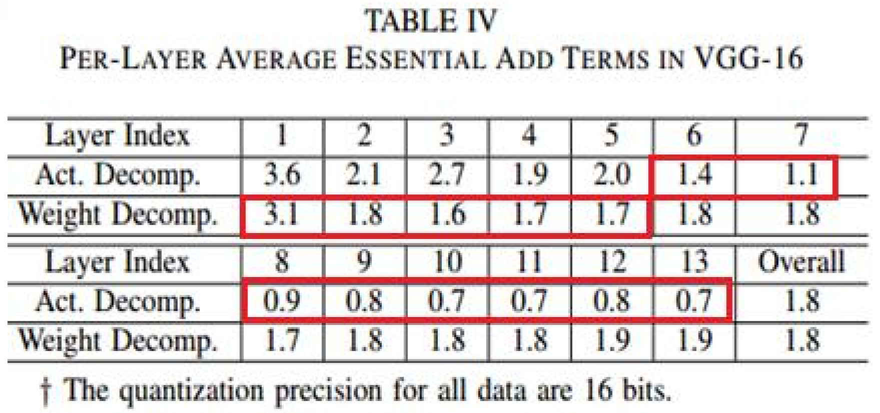
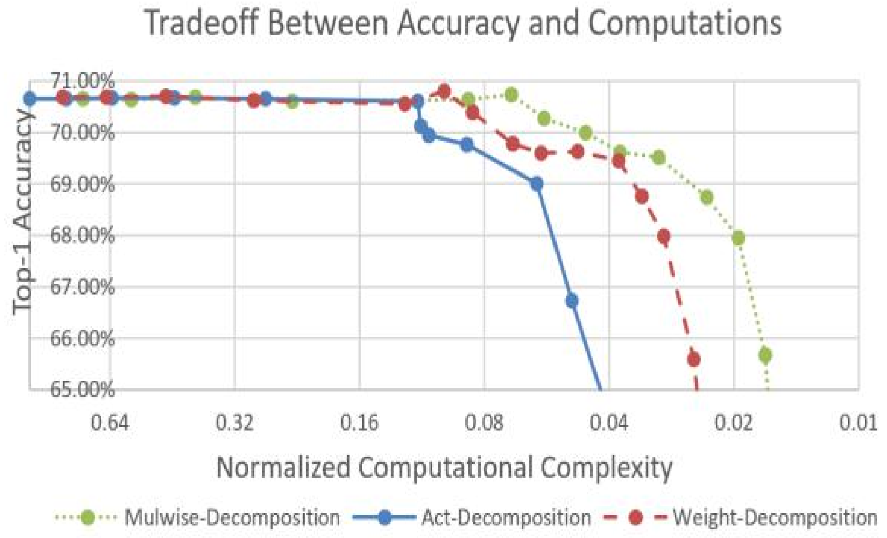


TABLE IV
PER-LAYER AVERAGE ESSENTIAL ADD TERMS IN VGG-16

Layer Index	1	2	3	4	5	6	7
Act. Decomp.	3.6	2.1	2.7	1.9	2.0	1.4	1.1
Weight Decomp.	3.1	1.8	1.6	1.7	1.7	1.8	1.8
Layer Index	**8**	**9**	**10**	**11**	**12**	**13**	**Overall**
Act. Decomp.	0.9	0.8	0.7	0.7	0.8	0.7	1.8
Weight Decomp.	1.7	1.8	1.8	1.8	1.9	1.9	1.8

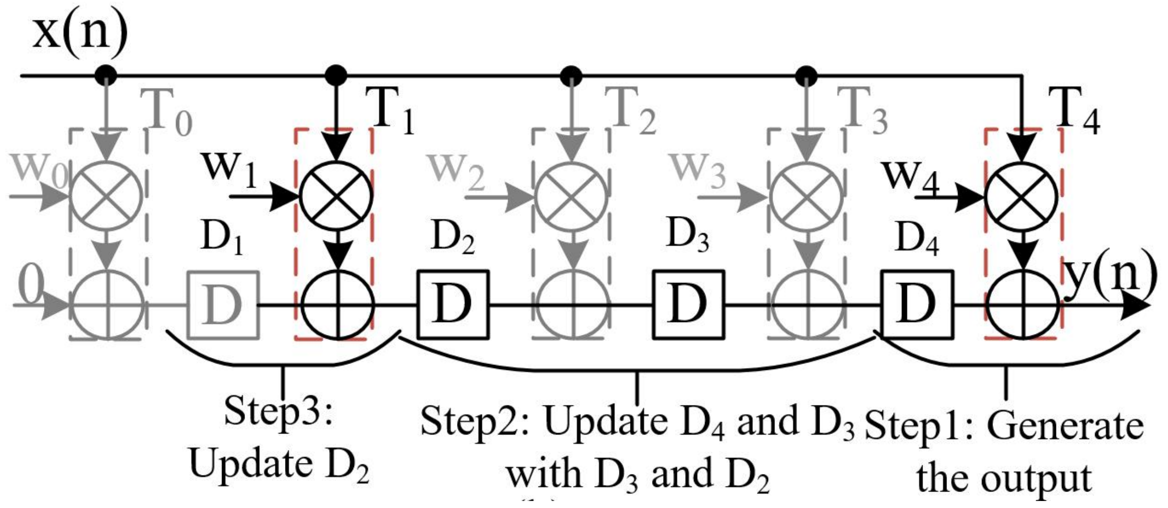
† The quantization precision for all data are 16 bits.



VGG-16

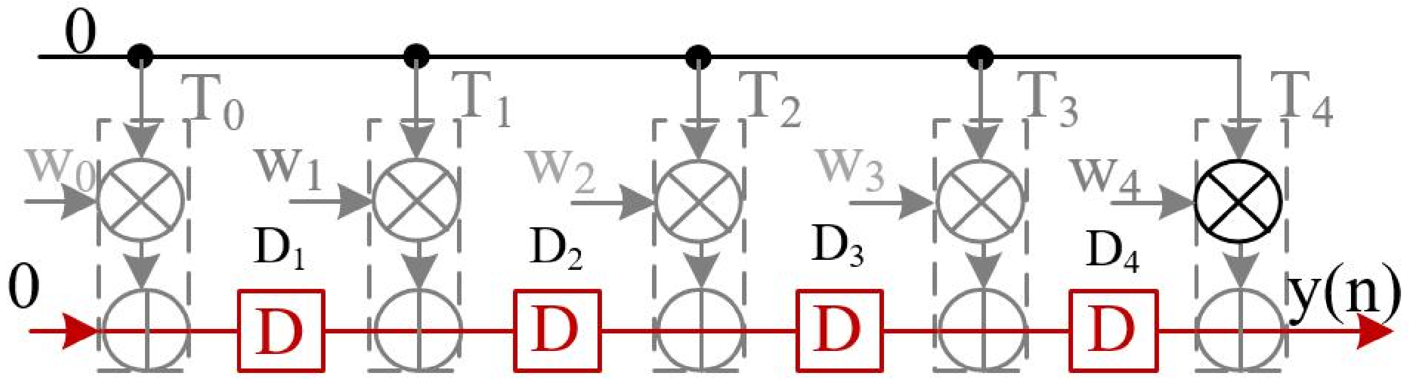
FPAP: Energy-Quality Scalable CNNs

- Fold Computations in CNNs(cont'd)
  - Exploit sparsity of weights/activations to reduce number of MAC Ops
  - Approach to exploit weight sparsity
    - Fold transposed-FIR filter into one MAC
    - Dynamically skip zero-coefficient taps


x(n)
T0
T1
T2
T3
T4
W0
W1
W2
W3
W4
0
D1
D2
D3
D4
D
y(n)
Step3: Update D2
Step2: Update D4 and D3 with D3 and D2
Step1: Generate the output

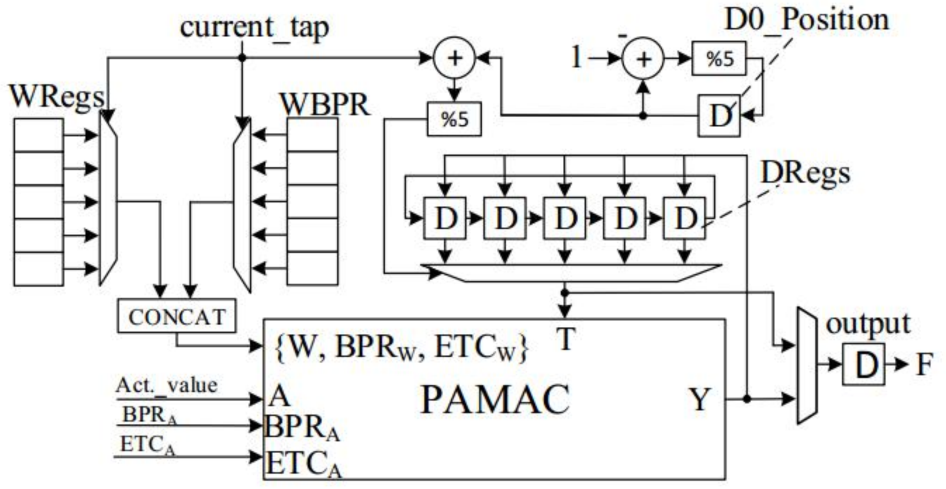
FPAP: Energy-Quality Scalable CNNs

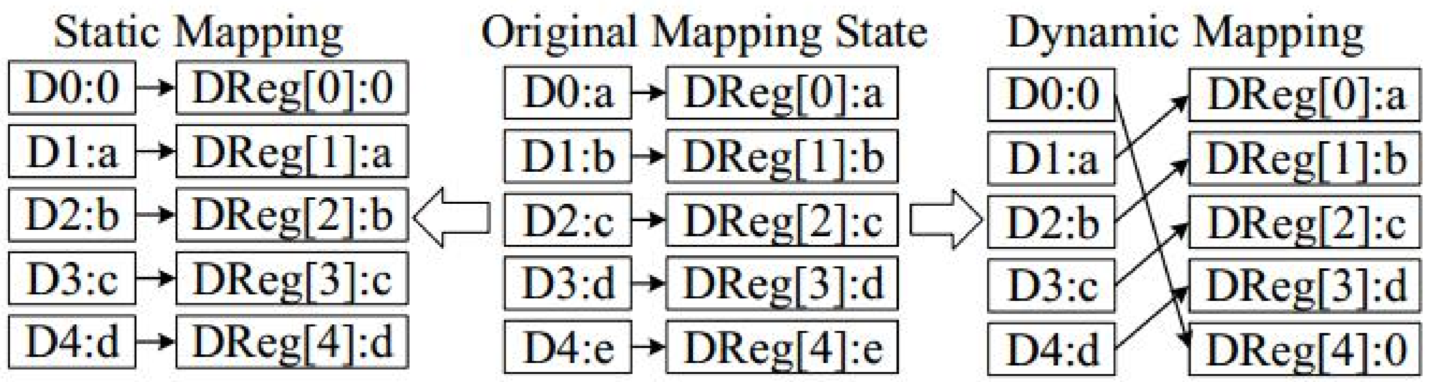
- Fold Computations in CNNs(cont'd)
  - Approach to exploit activation sparsity
    - Only shifting of delay elements is required when meeting zero input
      - No extra cycle compared to non-folded FIR filter
      - The number of cycles can be less than non-folded FIR filter when meeting more continuous zero activations



FPAP: Energy-Quality Scalable CNNs

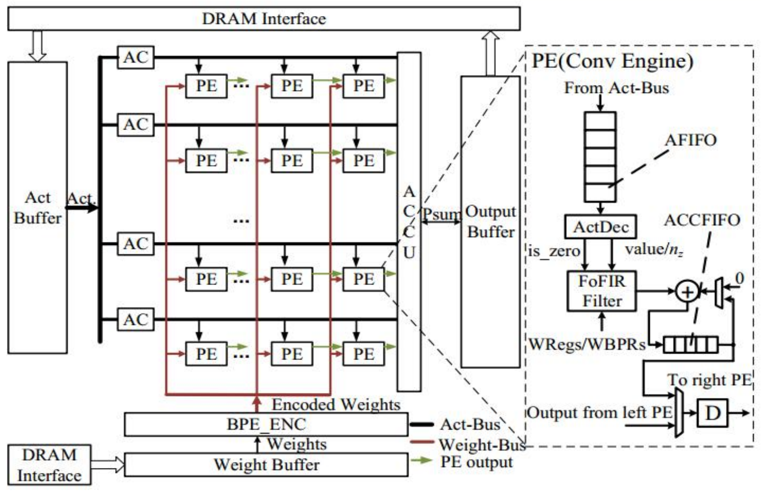
- Fold Computations in CNNs(cont’d)
  - VLSI implementation of folded FIR filter (FoFIR)
    - At least 60% of area saving for FIR filters with over 3 taps
    - Dynamic mapping from DRegs to delay elements
      - Eliminate unnecessary register-shifting operation/power for zero input with slight overhead

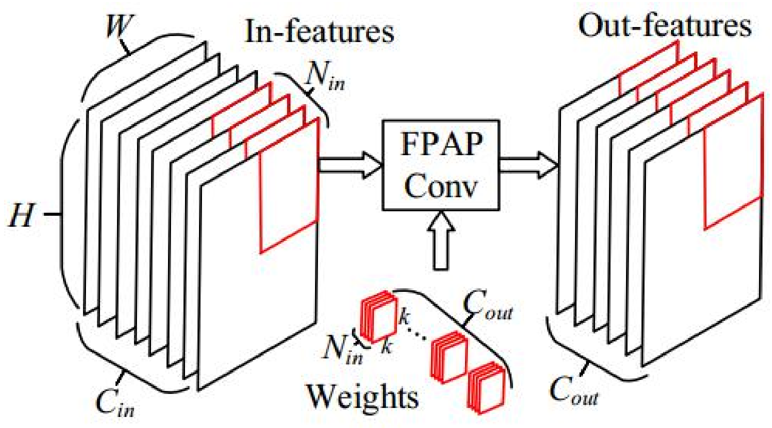




FPAP: Energy-Quality Scalable CNNs

- Mitigate the Load-Imbalance Issue
  - Top architecture: a scalable processing element (PE) array
    - Each column of the array work for one out channel
    - The input features are broadcast along each row of the PE array
    - Input channels are tiled into multiple groups (e.g., 16 channels as one group)
    - Each PE contains a folded FIR and convolves every input feature group in a row-by-row channel-by-channel way
    - Array synchronize when all PEs finish processing of an input channel group

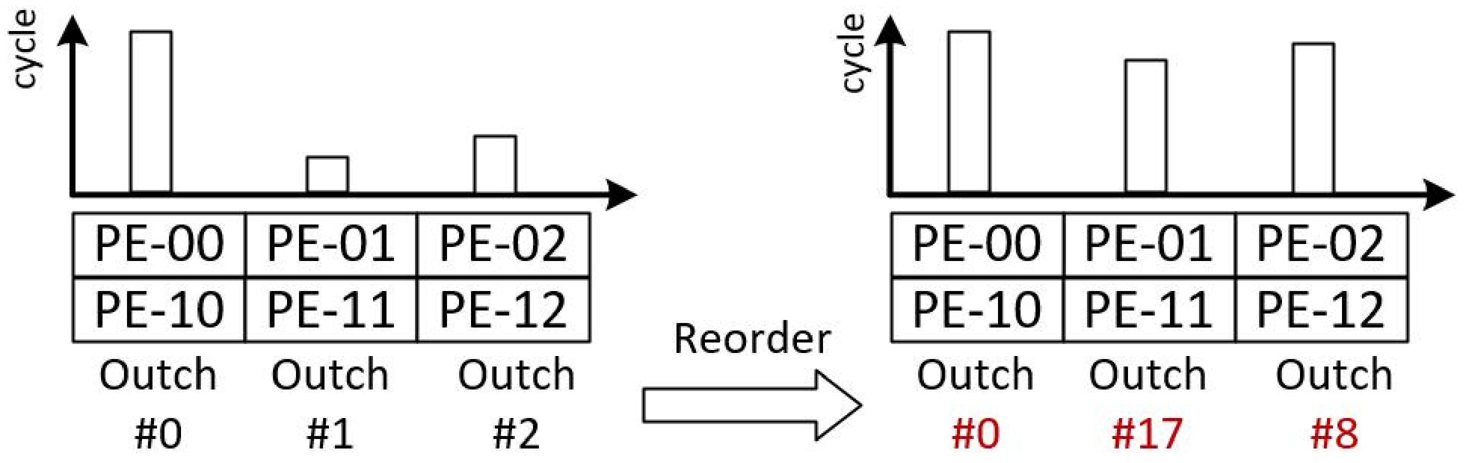




(b)

FPAP: Energy-Quality Scalable CNNs

- Mitigate the Load-Imbalance Issue
  - Different PEs take different cycles for different output channels
  - Solution: Find a good process order of output channels for each input channel group
    - Formulize an estimation of cycles (cost)
      - According to dataflow mapping and essential add terms in weights
    - Use genetic algorithm to minimize the cost
      - Obtain fairly balanced PEs under certain output channel order



FPAP: Energy-Quality Scalable CNNs

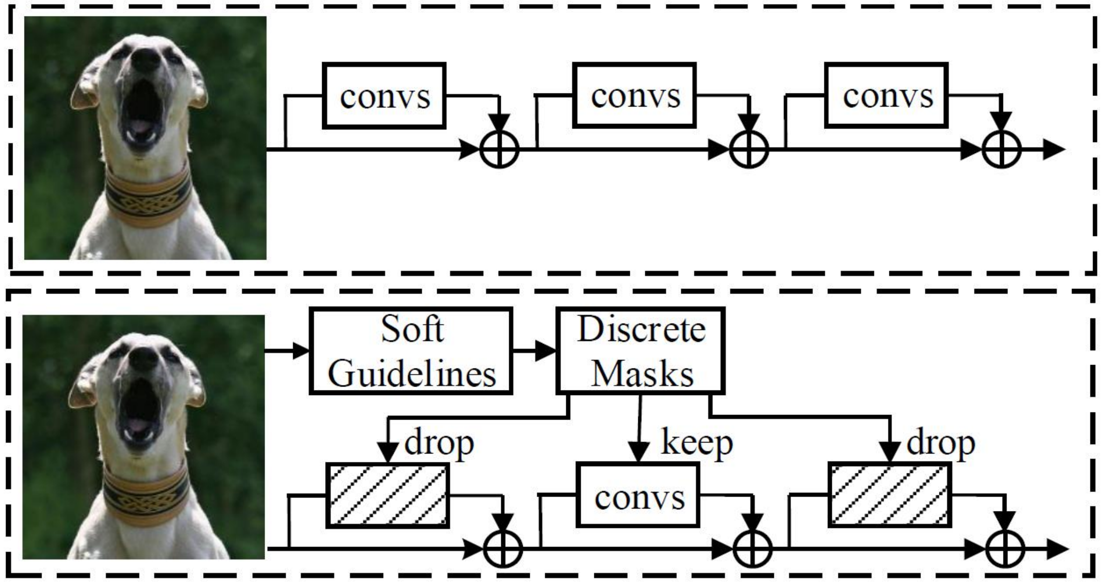
- Experimental Results
  - Folded architecture achieves low power and small area (<30mW/2.13mm^2 under 28nm)
  - The throughput/energy efficiency scales up as data precision or computational precision decreases
  - The equivalent throughput of FPAP is much higher than its peak throughput because all redundancies are eliminated

Characteristics	Results
Array Size/#MAC	8x32/256
Technology	TSMC 28nm
Area	2.13mm^2
Max. Frequency	1GHz
Peak Throughput	32GOP/s(16b)-256GOP/s(2b)
Power	<30mW
Benchmark	VGG-16
Equivalent Throughput	108.86GOP/s-349.6GOP/s
Energy Efficiency	7.68TOP/s/W-23.63TOP/s/W

NANJING UNIVERSITY

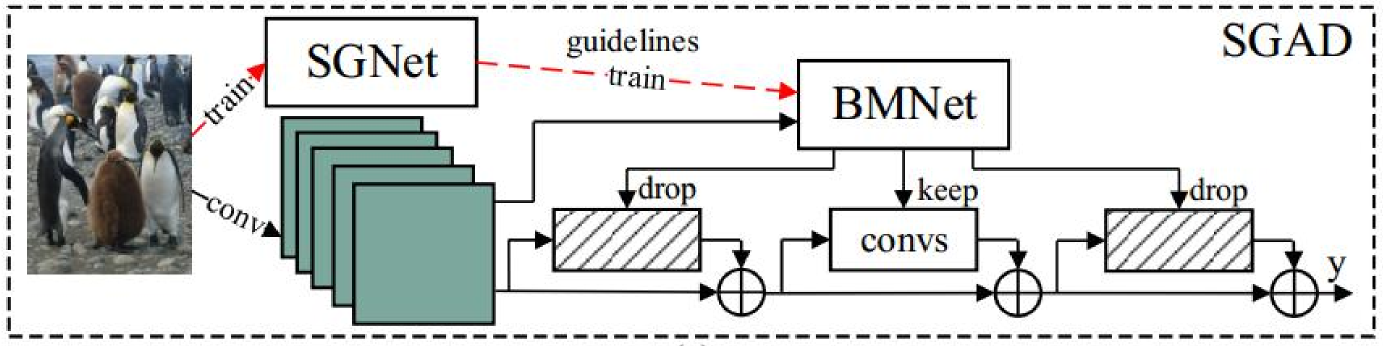
Soft-Guided Adaptively-Dropped NNs

- Background
  - The remarkable accuracy of DNNs comes at the expense of huge computational cost.
  - A novel soft-guided adaptively-dropped neural network is proposed to reduce the input-specific redundant computations.
- Overview of the Adaptive Dropping Mechanism:



Soft-Guided Adaptively-Dropped NNs

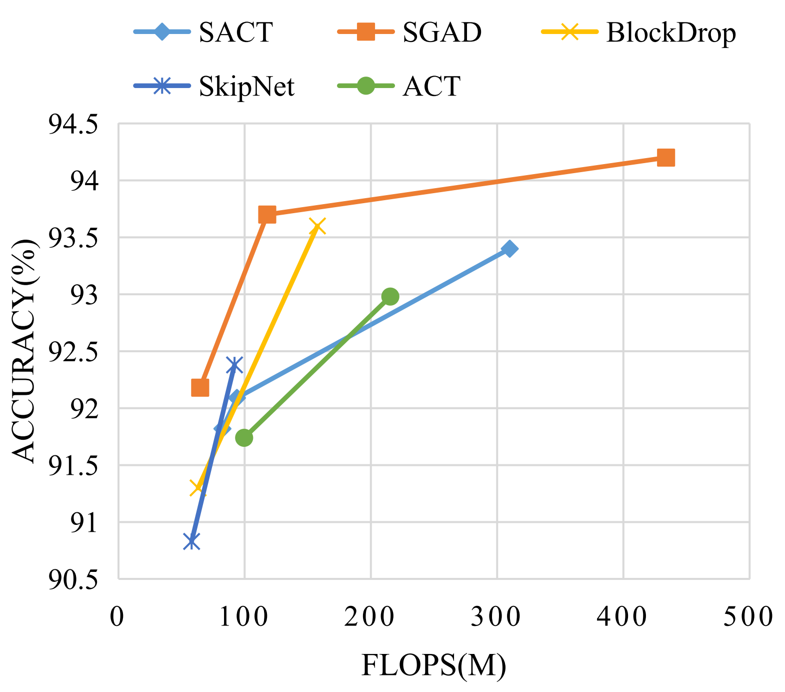
- Network Structure Description

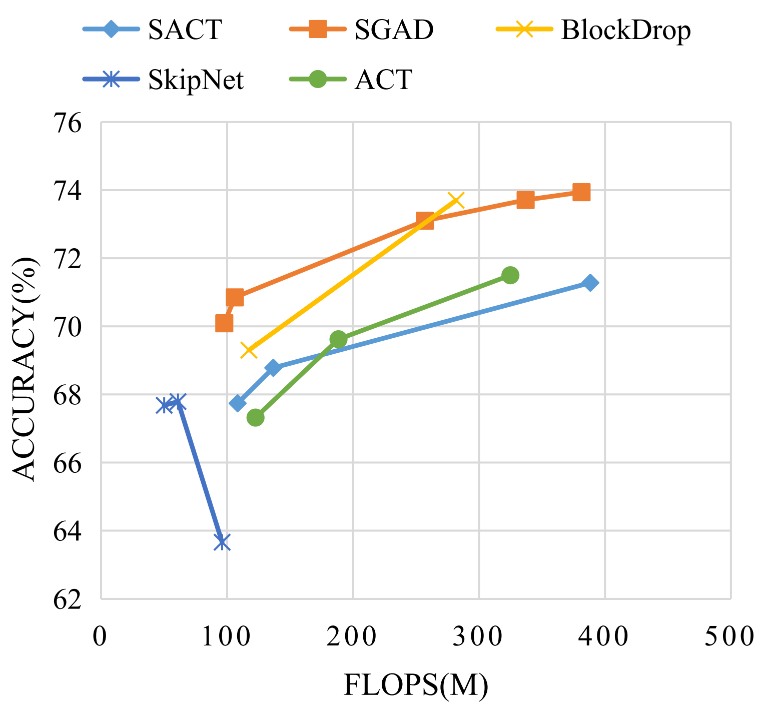


- Contains normal ResNet, BMNet, and SGNet
- BMNet - binary mask network
  - Decide which blocks of ResNet should be used for a specific input
  - Introduce less than 1% computation overhead
- SGNet – soft guideline network
  - Guide the BMNet to learn the adaptive dropping behavior during the training phase
  - Can be removed during the inference phase

Soft-Guided Adaptively-Dropped NNs

- Experimental Results
  - ◆ Reduced 76.6% FLOPs with ~0.8% accuracy loss on CIFAR-10

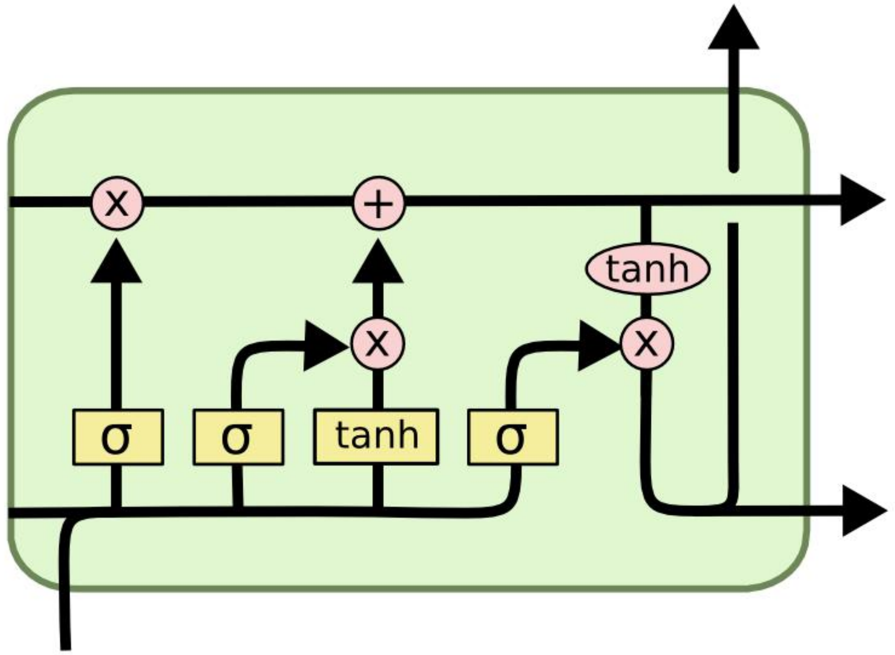




  - ◆ SGAD outperforms prior works on CIFAR-10/CIFAR100
  - ◆ 1.5-3.0X speedup on CPU during inference

Efficient Hardware Architecture for LSTM

- Long-Short Term Memory (LSTM) Model
  - LSTM is a powerful modeling method for sequential tasks
  - Widely used in speech recognition/translation/video analysis etc.
  - LSTM inference is computation intensive and requires lots of DRAM access
  - Require dedicated hardware architectures for embedded applications



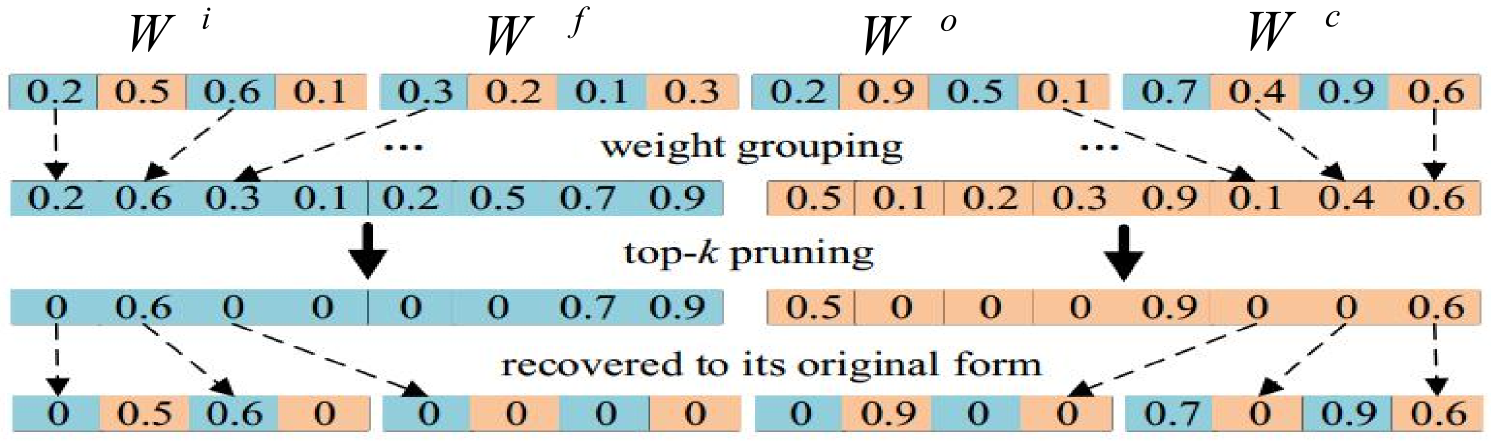
Efficient Hardware Architecture for LSTM

- Hardware-Oriented Compression for LSTM
- Efficient Architecture Design for LSTM

Hardware-Oriented Compression for LSTM

- Reduce Memory Footprint with Hardware-Efficient Schemes
  - Structured weight pruning (grouping + top-k pruning)
    - Eliminate the load-imbalance problem during hardware design
    - Explore sparsity diversity in different weight matrices
      - (8,3) top-k pruning example:

$$
\begin{aligned}
i_t &= \sigma(W_x^i x_t + W_h^i h_{t-1} + b^i),\\
f_t &= \sigma(W_x^f x_t + W_h^f h_{t-1} + b^f),\\
o_t &= \sigma(W_x^o x_t + W_h^o h_{t-1} + b^o),\\
\tilde{c}_t &= \tanh(W_x^c x_t + W_h^c h_{t-1} + b^c),\\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t,\\
h_t &= o_t \odot \tanh(c_t).
\end{aligned}
$$



Hardware-Oriented Compression for LSTM

- Reduce Memory Footprint with Hardware-Efficient Schemes
  - Clipped gating: enable sparse activations
    - Clipped sigmoid function in output gate for zero activations

$$o_t = \sigma(W_x^o x_t + W_h^o h_{t-1} + b^o), \qquad \text{output gate}$$

$$\sigma_{clip}(x) = \begin{cases} \sigma(x) & \text{if } \sigma(x) > T_\sigma, \ T_\sigma \in [0,1) \\ 0 & \text{otherwise.} \end{cases}$$

$$h_t = \hat{o}_t \odot \tanh(c_t). \qquad \text{hidden state}$$

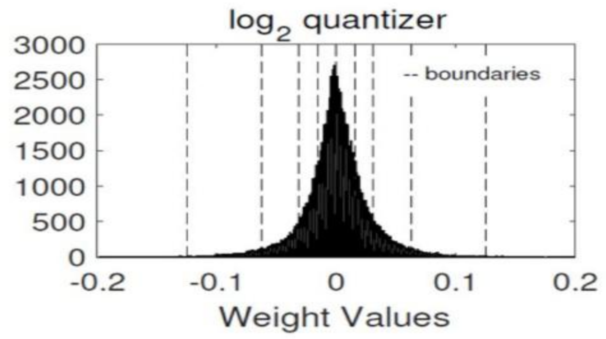
    - Extra regularizer on loss function to control the expected activation sparsity

$$Rs = \lambda_s \cdot \max\{0, S_e - S_m\}, \qquad S_e \in [0,1)$$

Hardware-Oriented Compression for LSTM

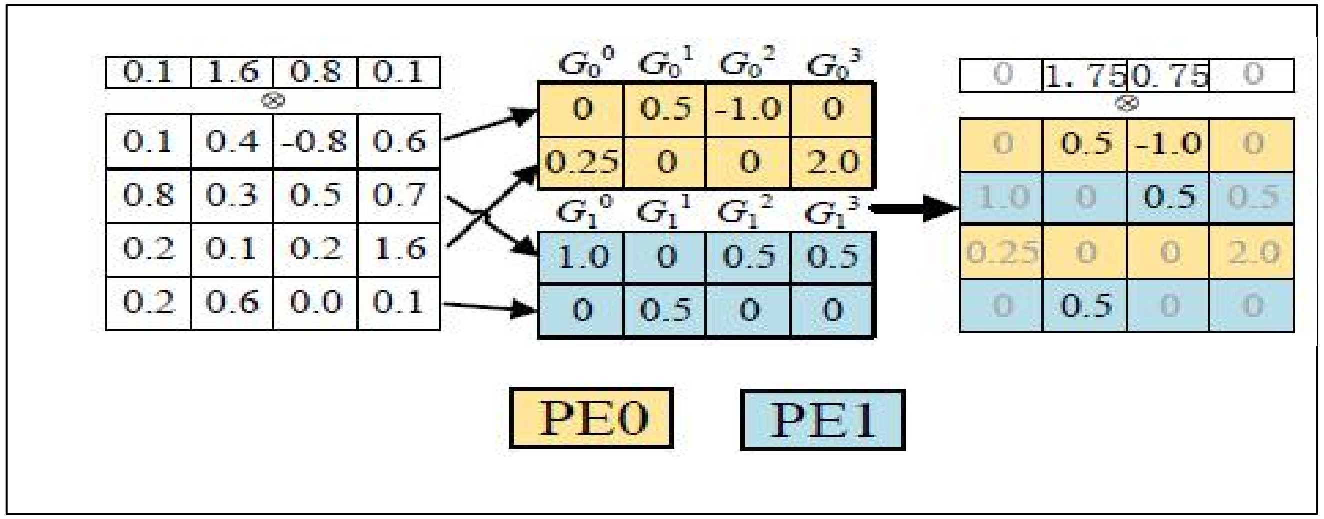
- Reduce Memory Footprint with Hardware-Efficient Schemes
  - (Multiply-free) Log-domain quantization for weight matrices

$$\mathrm{Log}Q_{m,-f}(W) \in \{\pm 2^{m}, \cdots, \pm 2^{0}, \pm 2^{-1}, \cdots, \pm 2^{-f}, 0\}$$



E. H. Lee et al., "LogNet: Energy-efficient neural networks using logarithmic computation," *2017 ICASSP*

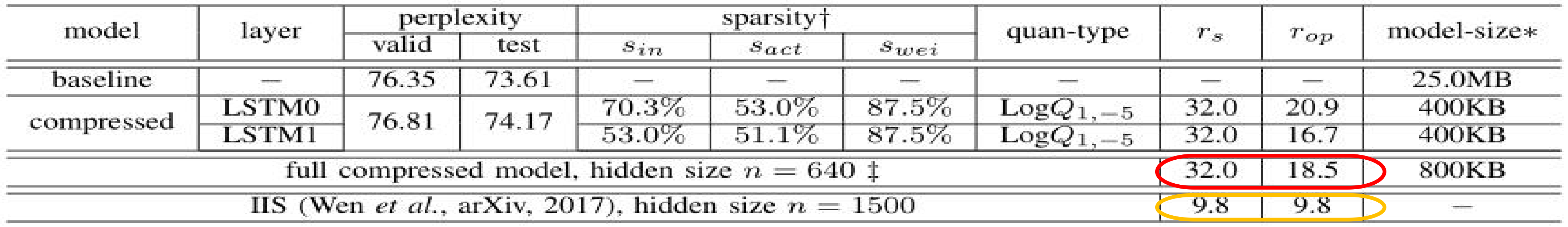
  - Linear quantization for activations



A Simple Case Study

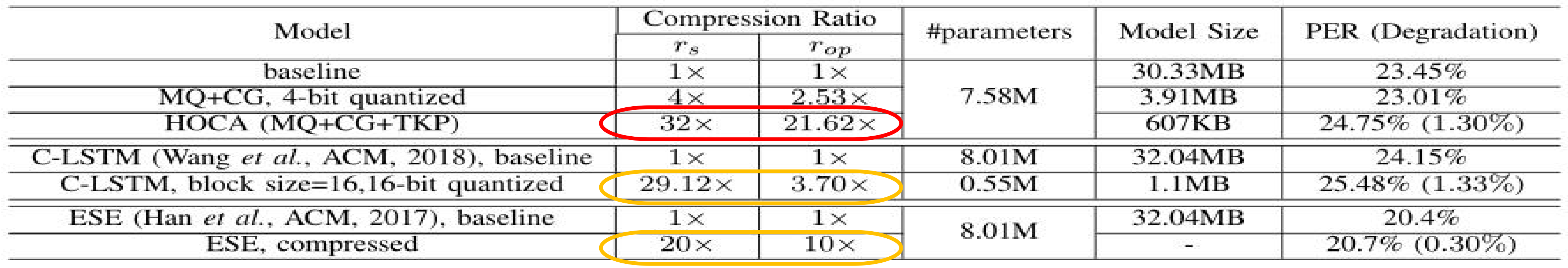
Hardware-Oriented Compression for LSTM

- **Experimental Results**: (16,2) pruning + 4-bit non-zero wei&idx
  - Language Modeling on Penn Treebank (PTB) Corpus



model	layer	perplexity		sparsity†			quan-type	r_s	r_{op}	model-size*
		valid	test	s_{in}	s_{act}	s_{wei}				
baseline	–	76.35	73.61	–	–	–	–	–	–	25.0MB
compressed	LSTM0	76.81	74.17	70.3%	53.0%	87.5%	$LogQ_{1,-5}$	32.0	20.9	400KB
	LSTM1			53.0%	51.1%	87.5%	$LogQ_{1,-5}$	32.0	16.7	400KB
full compressed model, hidden size $n = 640$ ‡								32.0	18.5	800KB
IIS (Wen *et al.*, arXiv, 2017), hidden size $n = 1500$								9.8	9.8	–

  - Phoneme speech recognition on TIMIT Dataset



Model	Compression Ratio		#parameters	Model Size	PER (Degradation)
	r_s	r_{op}			
baseline	1×	1×	7.58M	30.33MB	23.45%
MQ+CG, 4-bit quantized	4×	2.53×		3.91MB	23.01%
HOCA (MQ+CG+TKP)	32×	21.62×		607KB	24.75% (1.30%)
C-LSTM (Wang *et al.*, ACM, 2018), baseline	1×	1×	8.01M	32.04MB	24.15%
C-LSTM, block size=16,16-bit quantized	29.12×	3.70×	0.55M	1.1MB	25.48% (1.33%)
ESE (Han *et al.*, ACM, 2017), baseline	1×	1×	8.01M	32.04MB	20.4%
ESE, compressed	20×	10×		-	20.7% (0.30%)

MQ:mixed quantization, CG: clipped gating, TKP: top-k pruning

E-LSTM： An Efficient Hw Arch for LSTM

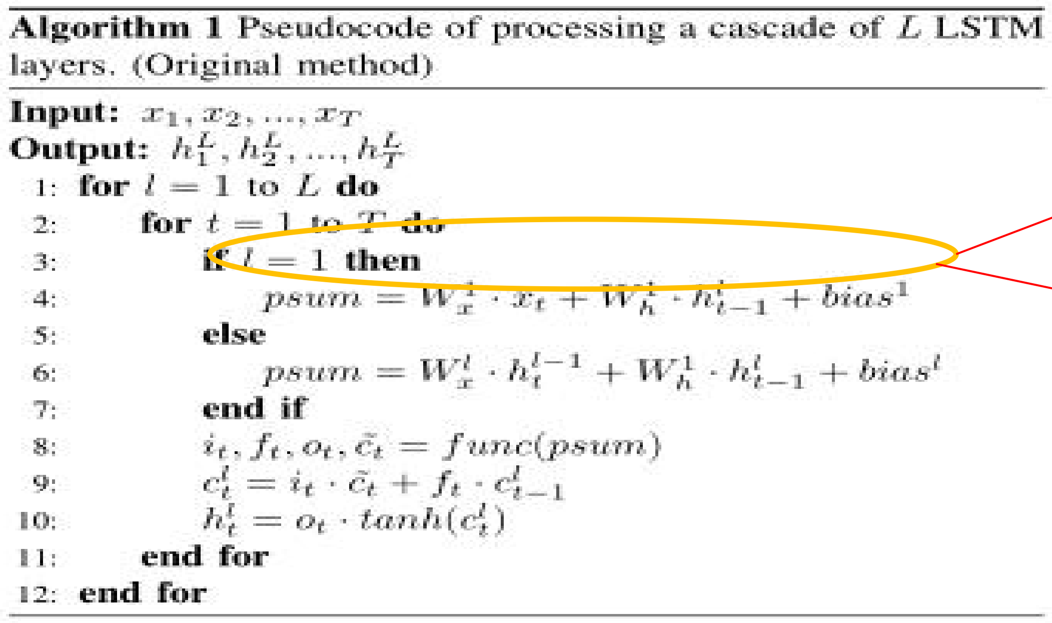
- Hardware-Oriented Compression for LSTM
- Efficient Architecture Design for LSTM

Efficient Hardware Architecture for LSTM

- Rearrangement of the Computation Process
  - Decouple recurrent/non-recurrent part

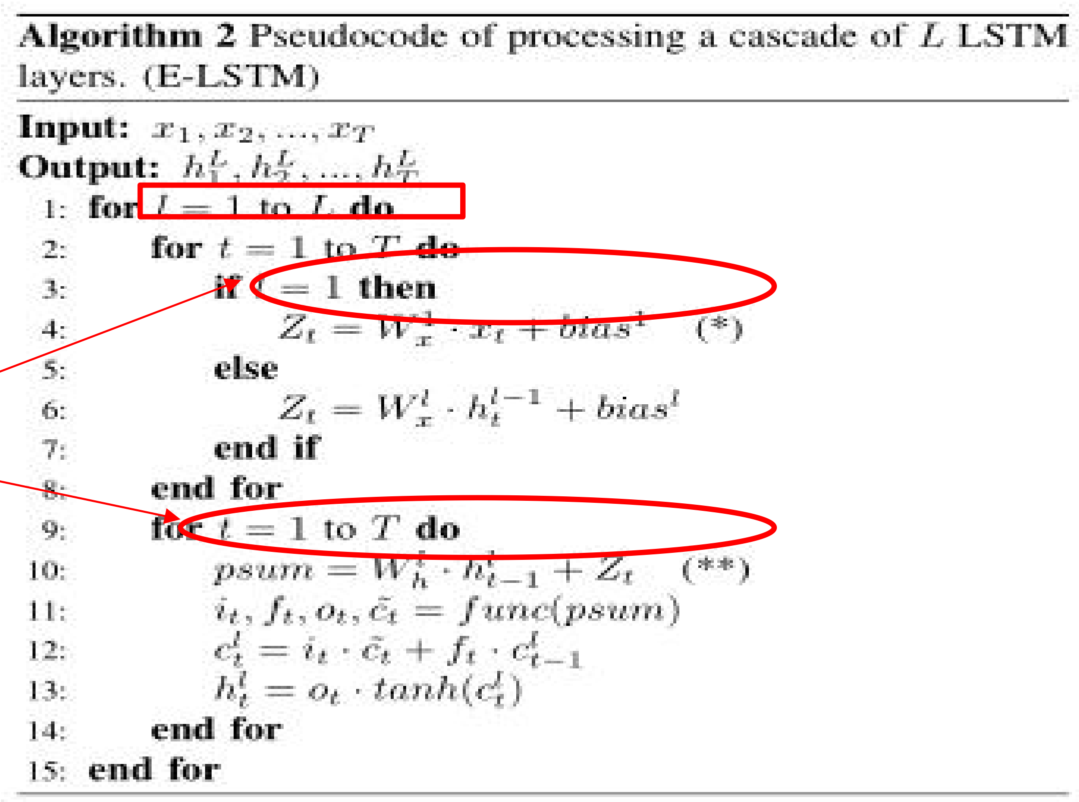
    → explore weight reuse
  - Same computation pattern

    → hardware reuse



Algorithm 1 Pseudocode of processing a cascade of L LSTM layers. (Original method)

```
Input: $x_1, x_2, ..., x_T$
Output: $h_1^L, h_2^L, ..., h_T^L$
for $l = 1$ to $L$ do
    for $t = 1$ to $T$ do
        if $l = 1$ then
            $psum = W_x^1 \cdot x_t + W_h^1 \cdot h_{t-1}^1 + bias^1$
        else
            $psum = W_x^l \cdot h_t^{l-1} + W_h^1 \cdot h_{t-1}^l + bias^l$
        end if
        $i_t, f_t, o_t, \tilde{c}_t = func(psum)$
        $c_t^l = i_t \cdot \tilde{c}_t + f_t \cdot c_{t-1}^l$
        $h_t^l = o_t \cdot tanh(c_t^l)$
    end for
end for
```

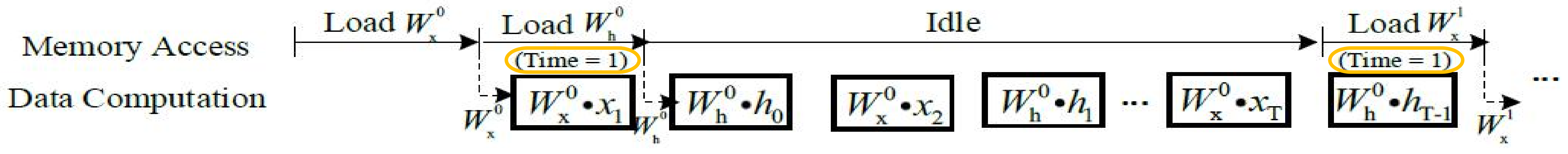


Algorithm 2 Pseudocode of processing a cascade of L LSTM layers. (E-LSTM)

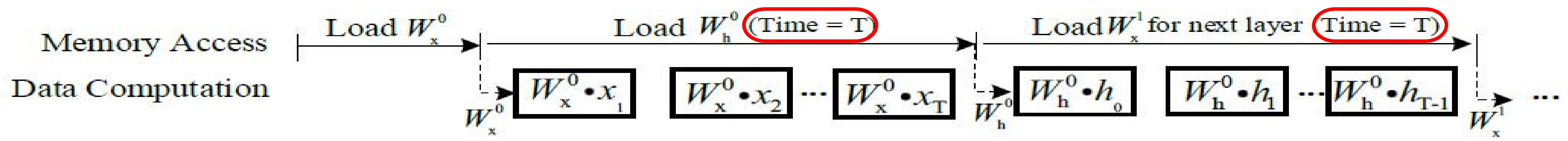
```
Input: $x_1, x_2, ..., x_T$
Output: $h_1^L, h_2^L, ..., h_T^L$
for $l = 1$ to $L$ do
    for $t = 1$ to $T$ do
        if $l = 1$ then
            $Z_t = W_x^1 \cdot x_t + bias^1$    (*)
        else
            $Z_t = W_x^l \cdot h_t^{l-1} + bias^l$
        end if
    end for
    for $t = 1$ to $T$ do
        $psum = W_h^l \cdot h_{t-1}^l + Z_t$    (**)
        $i_t, f_t, o_t, \tilde{c}_t = func(psum)$
        $c_t^l = i_t \cdot \tilde{c}_t + f_t \cdot c_{t-1}^l$
        $h_t^l = o_t \cdot tanh(c_t^l)$
    end for
end for
```

Efficient Hardware Architecture for LSTM

- Rearrangement of the Computation Process
  - Alleviate DDR transfer burden



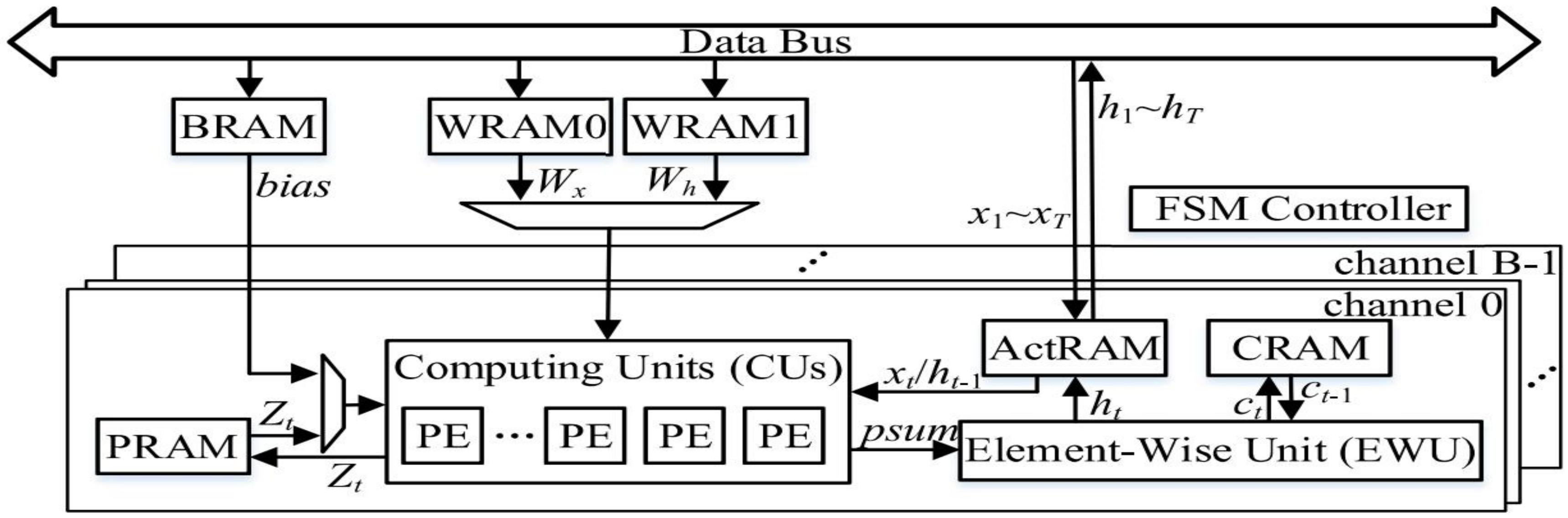
Original process



Our process

Efficient Hardware Architecture for LSTM

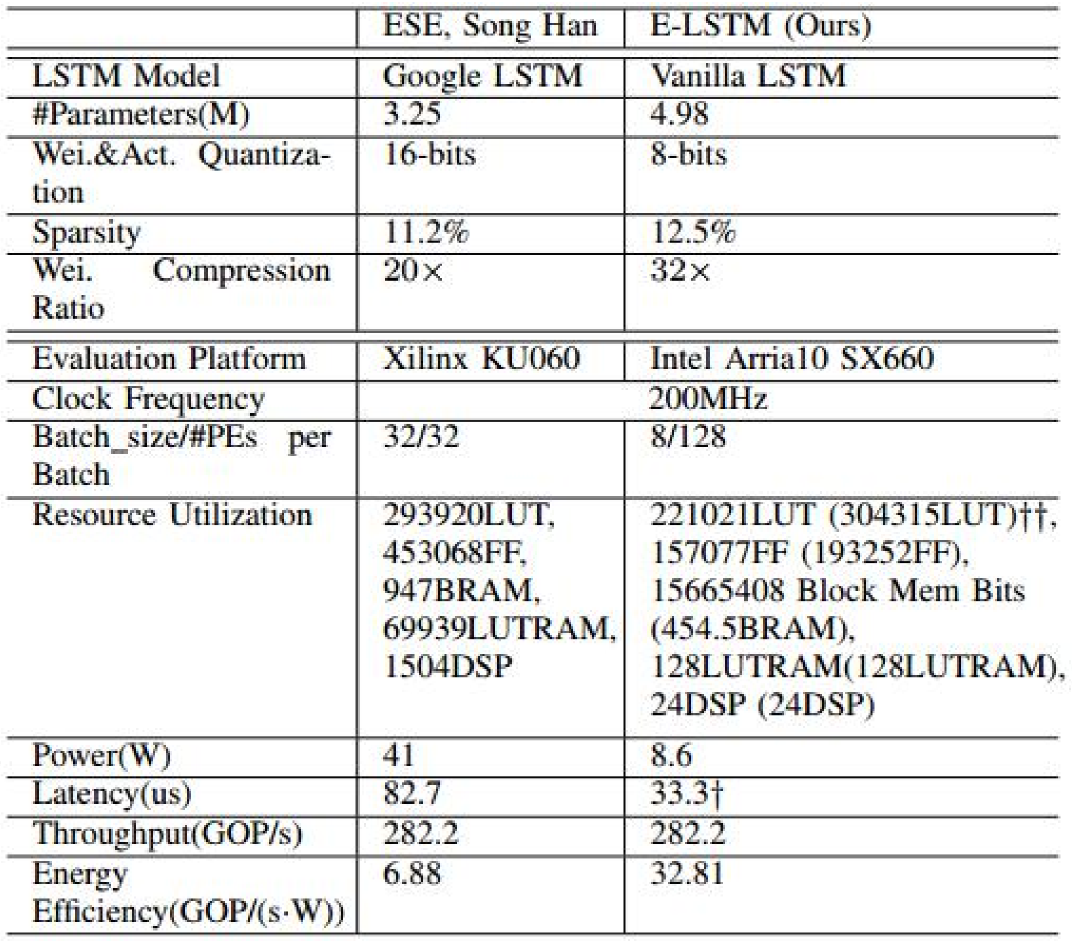
● Overall Architecture for Batch Processing



◆ The bandwidth requirement can be reduced by T times due to the increased time budget for memory fetching in the rearranged computation process.

Efficient Hardware Architecture for LSTM

- Efficient LSTM Hardware Architecture
  - Efficient model compression algorithm => 32X compression ration
  - Compared to ESE (深鉴科技)
    - 62X reduced DSP slices
    - 546X reduced LUTRAM
    - 2X reduced BRAM
    - 5X energy efficiency



	ESE, Song Han	E-LSTM (Ours)
LSTM Model	Google LSTM	Vanilla LSTM
#Parameters(M)	3.25	4.98
Wei.&Act. Quantization	16-bits	8-bits
Sparsity	11.2%	12.5%
Wei. Compression Ratio	20×	32×
Evaluation Platform	Xilinx KU060	Intel Arria10 SX660
Clock Frequency		200MHz
Batch_size/#PEs per Batch	32/32	8/128
Resource Utilization	293920LUT, 453068FF, 947BRAM, 69939LUTRAM, 1504DSP	221021LUT (304315LUT)††, 157077FF (193252FF), 15665408 Block Mem Bits (454.5BRAM), 128LUTRAM(128LUTRAM), 24DSP (24DSP)
Power(W)	41	8.6
Latency(us)	82.7	33.3†
Throughput(GOP/s)	282.2	282.2
Energy Efficiency(GOP/(s·W))	6.88	32.81

Part V: Conclusions

Conclusions

- VLSI optimization for signal processing systems can bring drastic improvement on power or speed in modern IC design
- DNNs are fundamental for deep learning
- Efficient implementation of DNNs is highly desired for practical applications

感谢您的聆听！

南京大学集成电路与智能系统实验室

Email：zfwang@gmail.com