A multiplier-free RNS-based CNN accelerator exploiting bit-level sparsity

V. Sakellariou[1], V. Paliouras[2], I. Kouretas[2], H. Saleh[1], T. Stouraitis[1]

[1]Department of Electrical Engineering and Computer Science
Khalifa University
[2]Department of Electrical and Computer Engineering
University of Patras

30th IEEE International Symposium on Computer Arithmetic, Portland, Oregon, USA. September 4-6, 2023.

Table of Contents

1 Introduction

2 Proposed PE (exploiting bit-level sparsity)

3 Overall CNN Architecture

4 Results

Presentation Outline

1 Introduction

2 Proposed PE (exploiting bit-level sparsity)

3 Overall CNN Architecture

4 Results

RNS Basics

- In RNS, integers are represented by their residues with respect to a modulus set: $\mathcal{B} = \{m_1, m_2, \ldots, m_K\}$

RNS Basics

- In RNS, integers are represented by their residues with respect to a modulus set: $\mathcal{B} = \{m_1, m_2, \ldots, m_K\}$
$$X \mapsto (x_1, x_2, \ldots, x_K),\ x_i = \langle X \rangle_{m_i} \tag{1}$$
- Multiplication and addition are done independently and in parallel in each channel
$a \oplus b = \left(\langle a_1 \oplus b_1 \rangle_{m_1}, \langle a_2 \oplus b_2 \rangle_{m_2}, \ldots, \langle a_N \oplus b_N \rangle_{m_K}\right)$

RNS Basics

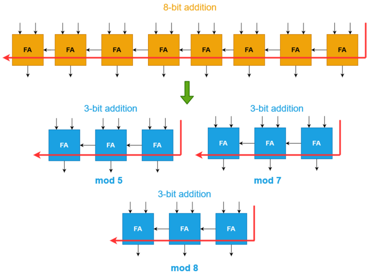
- In RNS, integers are represented by their residues with respect to a modulus set: $\mathcal{B} = \{m_1, m_2, \ldots, m_K\}$
$$X \mapsto (x_1, x_2, \ldots, x_K),\ \ x_i = \langle X \rangle_{m_i} \tag{1}$$
- Multiplication and addition are done independently and in parallel in each channel
$a \oplus b = \left(\langle a_1 \oplus b_1 \rangle_{m_1}, \langle a_2 \oplus b_2 \rangle_{m_2}, \ldots, \langle a_N \oplus b_N \rangle_{m_K}\right)$

- Large number computations are decomposed into smaller
- Smaller critical path $\rightarrow$ higher frequencies or reduced power dissipation
- Efficient implementation of MAC $\rightarrow$ good candidate for use in CNNs

RNS Basics

- In RNS, integers are represented by their residues with respect to a modulus set: $\mathcal{B} = \{m_1, m_2, \ldots, m_K\}$
$$X \mapsto (x_1, x_2, \ldots, x_K), \;\; x_i = \langle X \rangle_{m_i} \qquad (1)$$
- Multiplication and addition are done independently and in parallel in each channel
$a \oplus b = (\langle a_1 \oplus b_1 \rangle_{m_1}, \langle a_2 \oplus b_2 \rangle_{m_2}, \ldots, \langle a_N \oplus b_N \rangle_{m_K})$

- Large number computations are decomposed into smaller
- Smaller critical path $\rightarrow$ higher frequencies or reduced power dissipation
- Efficient implementation of MAC $\rightarrow$ good candidate for use in CNNs



Motivation: Multiplier-free PE

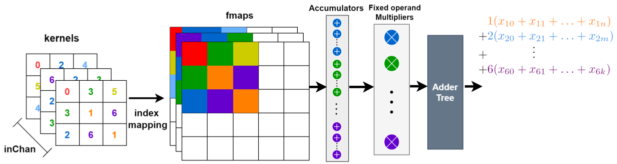

kernels
inChan
index mapping
fmaps
Accumulators
Fixed operand Multipliers
Adder Tree
$1(x_{10} + x_{11} + \ldots + x_{1n})$
$+2(x_{20} + x_{21} + \ldots + x_{2m})$
$+ \vdots$
$+6(x_{60} + x_{61} + \ldots + x_{6k})$

Figure: Modulo-7 Convolution

Motivation: Multiplier-free PE

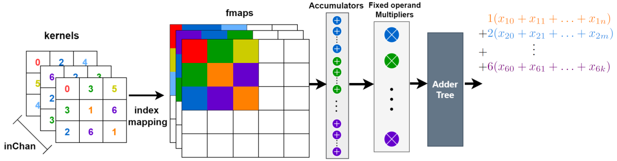

kernels
inChan
index mapping
fmaps
Accumulators
Fixed operand Multipliers
Adder Tree
$1(x_{10}+x_{11}+\ldots+x_{1n})$
$+2(x_{20}+x_{21}+\ldots+x_{2m})$
$+$
$+6(x_{60}+x_{61}+\ldots+x_{6k})$

Figure: Modulo-7 Convolution

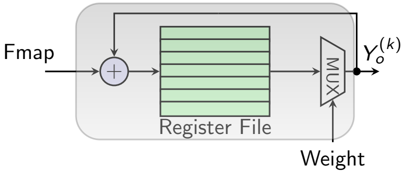


Figure: Direct implementation is the Multiplier-free (MF) PE → 1 $\log_2 C$-bit adder and C registers

Motivation: Multiplier-free PE

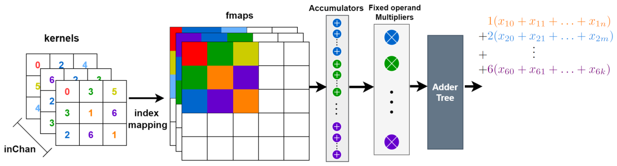

kernels
inChan
index mapping
fmaps
Accumulators
Fixed operand Multipliers
Adder Tree
$1(x_{10}+x_{11}+\ldots+x_{1n})$
$+2(x_{20}+x_{21}+\ldots+x_{2m})$
$+ \quad \vdots$
$+6(x_{60}+x_{61}+\ldots+x_{6k})$

Figure: Modulo-7 Convolution

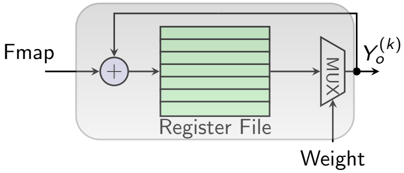


Figure: Direct implementation is the Multiplier-free (MF) PE → 1 $\log_2 C$-bit adder and C registers

Use many small word-length RNS channels

- small set of uniformly distributed weights
- increased number of common factors
- perform the additions first and then multiplications

Motivation: Multiplier-free PE

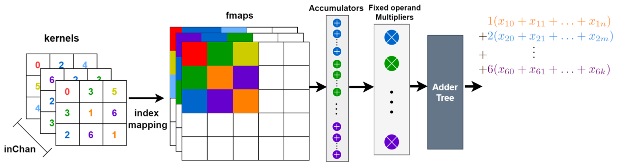

kernels
inChan
index mapping
fmaps
Accumulators
Fixed operand Multipliers
Adder Tree
$1(x_{10}+x_{11}+\ldots+x_{1n})$
$+2(x_{20}+x_{21}+\ldots+x_{2m})$
$+ \quad \vdots$
$+6(x_{60}+x_{61}+\ldots+x_{6k})$

Figure: Modulo-7 Convolution

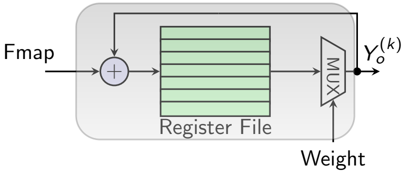


Figure: Direct implementation is the Multiplier-free (MF) PE $\rightarrow$ 1 $\log_2 C$-bit adder and C registers

Use many small word-length RNS channels

- small set of uniformly distributed weights
- increased number of common factors
- perform the additions first and then multiplications
- trivial/fixed operand mult. $\rightarrow$ simplified implementation

Motivation: Multiplier-free Distributed PE

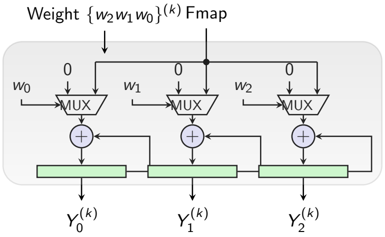


Figure: Distributed Multiplier-free (MF-D) PE $\rightarrow$ $\log_2 C$ adders and $\log_2 C$ registers.

Motivation: Multiplier-free Distributed PE

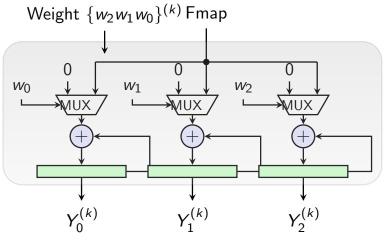


Figure: Distributed Multiplier-free (MF-D) PE $\rightarrow$ $\log_2 C$ adders and $\log_2 C$ registers.

Examples

W $= 5 = 101 \rightarrow$
Fmap is added to Y0,Y2

Motivation: Multiplier-free Distributed PE

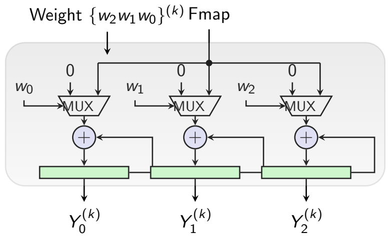


Figure: Distributed Multiplier-free (MF-D) PE $\rightarrow$ $\log_2 C$ adders and $\log_2 C$ registers.

Examples

W = 5 = 101 $\rightarrow$
Fmap is added to Y0,Y2

Problem

Under-utilization of adders in case of 0's
Throughput: **1 Input / cycle**

Presentation Outline

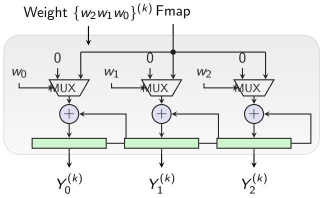
1 Introduction

2 Proposed PE (exploiting bit-level sparsity)

3 Overall CNN Architecture

4 Results

Increasing effective throughput

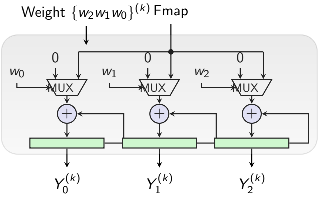


- To avoid under-utilization, **load two inputs per cycle**

Examples

W1 = 100, W2=010 $\rightarrow$
Fmap1 is added to Y2, Fmap2 is added to Y1

Increasing effective throughput



- To avoid under-utilization, **load two inputs per cycle**
- Conflict if two 1's in the same digital position

Examples

W1 = 100, W2=010 $\rightarrow$ Fmap1 is added to Y2, Fmap2 is added to Y1

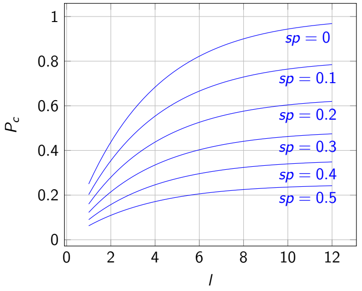


Figure: P_c of an MF-D PE channel for various sp vs l.

$$sp = \frac{\text{num. of zero weights}}{\text{total num. of weights}}$$

- Conflict probability P_c quickly increase as channel word-length l increases

Stack-based distributed PE (MF-D-S PE)

- Use a stack-like storage element to store conflicted inputs and **minimize stalls**

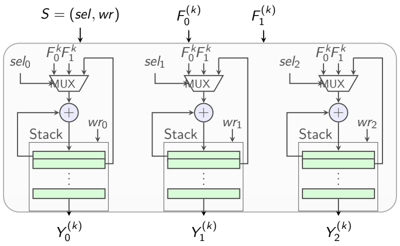


Figure: MF-D-S PE . The PE receives two FMAPs $F_0^{(k)}$, $F_1^{(k)}$, one select signal (sel) for each of the adders and a write-enable signal for each of the stacks.

Stack-based distributed PE (MF-D-S PE)

- Use a stack-like storage element to store conflicted inputs and **minimize stalls**
- In case of 2 ‘0’s, adders **process previously conflicted inputs**

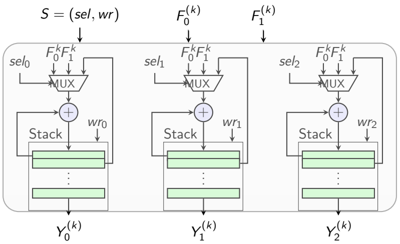


Figure: MF-D-S PE . The PE receives two FMAPs $F_0^{(k)}$, $F_1^{(k)}$, one select signal (*sel*) for each of the adders and a write-enable signal for each of the stacks.

Stack-based distributed PE (MF-D-S PE)

- Use a stack-like storage element to store conflicted inputs and **minimize stalls**
- In case of 2 ‘0’s, adders **process previously conflicted inputs**
- Higher sparsity translates into higher throughput

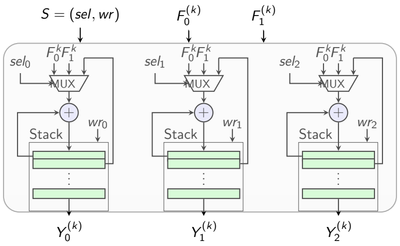


Figure: MF-D-S PE . The PE receives two FMAPs $F_0^{(k)}$, $F_1^{(k)}$, one select signal (*sel*) for each of the adders and a write-enable signal for each of the stacks.

Stack-based distributed PE (MF-D-S PE)

- Use a stack-like storage element to store conflicted inputs and **minimize stalls**
- In case of 2 ‘0’s, adders **process previously conflicted inputs**
- Higher sparsity translates into higher throughput
- Channel size also affects P_c $\rightarrow$ RNS is naturally more suitable representation: lower stall probability and less hardware resources

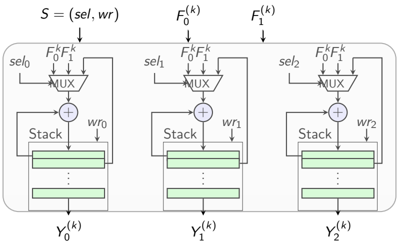


Figure: MF-D-S PE . The PE receives two FMAPs $F_0^{(k)}$, $F_1^{(k)}$, one select signal (sel) for each of the adders and a write-enable signal for each of the stacks.

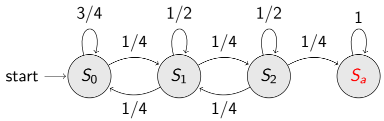
Stack-based distributed PE (MF-D-S PE)

Stack-based distributed PE (MF-D-S PE)

- Model as Markov chain to calculate probability of stalls P_c
- State represents num. of elements in stack

Stack-based distributed PE (MF-D-S PE)

- Model as Markov chain to calculate probability of stalls P_c
- State represents num. of elements in stack



$$\begin{tikzcd}
\text{start} \arrow[r] & S_0 \arrow[loop above, "3/4"] \arrow[r, bend left, "1/4"] & S_1 \arrow[loop above, "1/2"] \arrow[l, bend left, "1/4"] \arrow[r, bend left, "1/4"] & S_2 \arrow[loop above, "1/2"] \arrow[l, bend left, "1/4"] \arrow[r, bend left, "1/4"] & S_a \arrow[loop above, "1"]
\end{tikzcd}$$

Stack-based distributed PE (MF-D-S PE)

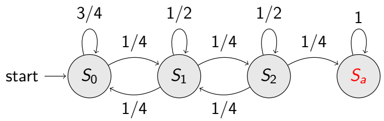
- Model as Markov chain to calculate probability of stalls P_c
- State represents num. of elements in stack

$$\begin{tikzcd}
\text{start} \arrow[r] & S_0 \arrow[loop above, "3/4"] \arrow[r, bend left, "1/4"] & S_1 \arrow[loop above, "1/2"] \arrow[l, bend left, "1/4"] \arrow[r, bend left, "1/4"] & S_2 \arrow[loop above, "1/2"] \arrow[l, bend left, "1/4"] \arrow[r, bend left, "1/4"] & {\color{red}S_a} \arrow[loop above, "1"]
\end{tikzcd}$$

- $\boldsymbol{N} = \sum_{k=0}^{\infty} \boldsymbol{Q}^k = (\boldsymbol{I} - \boldsymbol{Q})^{-1}$, where Q is the transition matrix
- stalls $N_s = \frac{(S+1)N}{T_a}$, $T_a = \boldsymbol{N1}$
- throughput $T = \frac{2N}{N+N_s}$

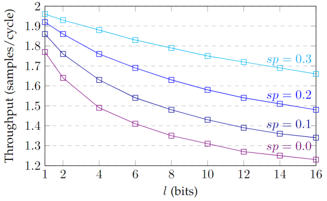
Stack-based distributed PE (MF-D-S PE)

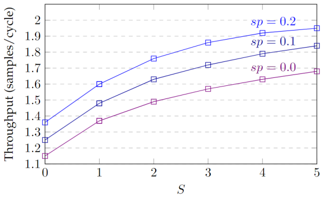
- Model as Markov chain to calculate probability of stalls P_c
- State represents num. of elements in stack


$$\begin{tikzcd}
\text{start} \arrow[r] & S_0 \arrow[loop above, "3/4"] \arrow[r, bend left, "1/4"] & S_1 \arrow[loop above, "1/2"] \arrow[l, bend left, "1/4"] \arrow[r, bend left, "1/4"] & S_2 \arrow[loop above, "1/2"] \arrow[l, bend left, "1/4"] \arrow[r, bend left, "1/4"] & S_a \arrow[loop above, "1"]
\end{tikzcd}$$

- $\boldsymbol{N} = \sum_{k=0}^{\infty} \boldsymbol{Q}^k = (\boldsymbol{I} - \boldsymbol{Q})^{-1}$, where Q is the transition matrix
- stalls $N_s = \frac{(S+1)N}{T_a}$, $T_a = \boldsymbol{N1}$
- throughput $T = \frac{2N}{N+N_s}$

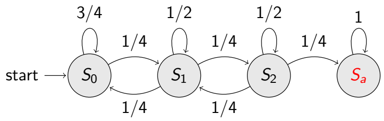
- P_c and thus throughput depend on channel size l, sparsity sp and stack size S.





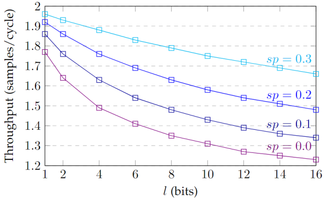
Stack-based distributed PE (MF-D-S PE)

- Model as Markov chain to calculate probability of stalls P_c
- State represents num. of elements in stack


$$\begin{tikzcd}
\text{start} \arrow[r] & S_0 \arrow[loop above, "3/4"] \arrow[r, bend left, "1/4"] & S_1 \arrow[loop above, "1/2"] \arrow[l, bend left, "1/4"] \arrow[r, bend left, "1/4"] & S_2 \arrow[loop above, "1/2"] \arrow[l, bend left, "1/4"] \arrow[r, bend left, "1/4"] & S_a \arrow[loop above, "1"]
\end{tikzcd}$$

- $\boldsymbol{N} = \sum_{k=0}^{\infty} \boldsymbol{Q}^k = (\boldsymbol{I} - \boldsymbol{Q})^{-1}$, where Q is the transition matrix
- stalls $N_s = \frac{(S+1)N}{T_a}$, $T_a = \boldsymbol{N1}$
- throughput $T = \frac{2N}{N+N_s}$

- P_c and thus throughput depend on channel size l, sparsity sp and stack size S.



Examples

For a 5-bit channel with $sp = 0$:
T=1.12 (MF-D) $\rightarrow$
T=1.32 (MF-D-S, S=1)

CSD encoding for reduced chance of stalls

- P_c depends on the distribution of ‘1’s and ‘0’s in the input weight vector.

CSD encoding for reduced chance of stalls

- P_c depends on the distribution of '1's and '0's in the input weight vector.
- An encoding that reduces the probability of conflicts (two '1's at the same position) would increase its throughput.

CSD encoding for reduced chance of stalls

- P_c depends on the distribution of ‘1’s and ‘0’s in the input weight vector.
- An encoding that reduces the probability of conflicts (two ‘1’s at the same position) would increase its throughput.
- The number of the non-zero elements of the weight representation can be minimized through the use of Canonical Signed Digit (CSD).

CSD encoding for reduced chance of stalls

- P_c depends on the distribution of ‘1’s and ‘0’s in the input weight vector.
- An encoding that reduces the probability of conflicts (two ‘1’s at the same position) would increase its throughput.
- The number of the non-zero elements of the weight representation can be minimized through the use of Canonical Signed Digit (CSD).
- In CSD, the value of each digit can be either 0, 1, or -1

CSD encoding for reduced chance of stalls

- P_c depends on the distribution of ‘1’s and ‘0’s in the input weight vector.
- An encoding that reduces the probability of conflicts (two ‘1’s at the same position) would increase its throughput.
- The number of the non-zero elements of the weight representation can be minimized through the use of Canonical Signed Digit (CSD).
- In CSD, the value of each digit can be either 0, 1, or -1
- Overhead: XOR gates needed to support subtractions

Examples

01110 → 1 0 0 (-1) 0
01101 → 0 1 1 0 1 stall is avoided

CSD encoding for reduced chance of stalls

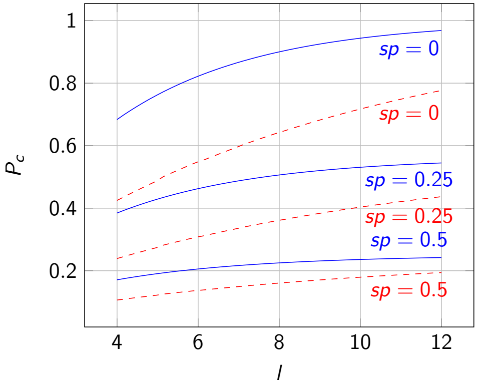


Figure: Conflict probability P_c of an MF-D PE channel for various sparsity levels sp and word lengths l. Blue color denotes binary encoding while red denotes CSD encoding.

Optimal encoding of pairs of weights

- CSD doesn't consider relative digit positions in weight pairs

Optimal encoding of pairs of weights

- CSD doesn't consider relative digit positions in weight pairs
- Jointly encode pairs of weights to minimize conflicts/stalls

$T(W)$ denotes the position of the trailing non-zero digit of W

Optimal encoding of pairs of weights

- CSD doesn't consider relative digit positions in weight pairs
- Jointly encode pairs of weights to minimize conflicts/stalls

$T(W)$ denotes the position of the trailing non-zero digit of W

Lemma

A signed-digit encoding (not necessarily canonical) $E(W_a, W_b) : (W_a, W_b) \mapsto (\hat{W}_a, \hat{W}_b)$*, such that* $C(\hat{W}_a, \hat{W}_b) = 0$*, for two non-zero weights, exists, if and only if:*

$$T(W_a) \neq T(W_b). \tag{2}$$

Optimal encoding of pairs of weights

- CSD doesn't consider relative digit positions in weight pairs
- Jointly encode pairs of weights to minimize conflicts/stalls

$T(W)$ denotes the position of the trailing non-zero digit of W

Lemma

A signed-digit encoding (not necessarily canonical) $E(W_a, W_b) : (W_a, W_b) \mapsto (\hat{W}_a, \hat{W}_b)$, such that $C(\hat{W}_a, \hat{W}_b) = 0$, for two non-zero weights, exists, if and only if:

$$T(W_a) \neq T(W_b). \tag{2}$$

Lemma

The probability P_c that no zero-conflict encoding for two weights W_a, W_b exists, i.e., $T(W_a) = T(W_b)$, is given by

$$P_c = \frac{1}{3} - \frac{1}{3 \cdot 4^n}. \tag{3}$$

Encoder scheme comparison

Table: Probability of a conflict P_c for mod 2^n, mod$(2^n - 1)$ and mod$(2^n + 1)$ and channel sizes $n = 4$ and $n = 5$

n	mod 2^n					mod $(2^n - 1)$					mod $(2^n + 1)^{‡}$				
	no enc	CSD	CSD/bin	combined	Opt.	no enc	CSD†	CSD†/bin	combined†	Opt.†	no enc	CSD†	CSD†/bin	combined†	Opt.‡
4	0.683	0.457	0.425	0.425	0.332	0.648	0.515	0.462	0.328	0.328	0.605	0.439	0.401	0.335	0.308
5	0.762	0.516	0.476	0.476	0.333	0.749	0.578	0.521	0.413	0.339	0.717	0.529	0.482	0.413	0.327

no enc: both weights in binary
CSD/bin: a, CSD; b, either binary or CSD
CSD: independent CSD encoding of weights
combined: a, CSD, b, binary, b_{CSD}, or b'_{CSD}
†: no re-encoding after EAC Opt.: Optimal
‡: diminished-1 representation

Encoder scheme comparison

Table: Probability of a conflict P_c for mod 2^n, mod(2^n-1) and mod(2^n+1) and channel sizes $n=4$ and $n=5$

n	mod 2^n					mod (2^n-1)					mod (2^n+1)‡				
	no enc	CSD	CSD/bin	combined	Opt.	no enc	CSD†	CSD†/bin	combined†	Opt.†	no enc	CSD†	CSD†/bin	combined†	Opt.‡
4	0.683	0.457	0.425	0.425	0.332	0.648	0.515	0.462	0.328	0.328	0.605	0.439	0.401	0.335	0.308
5	0.762	0.516	0.476	0.476	0.333	0.749	0.578	0.521	0.413	0.339	0.717	0.529	0.482	0.413	0.327

no enc: both weights in binary
CSD/bin: a, CSD; b, either binary or CSD
CSD: independent CSD encoding of weights
combined: a, CSD, b, binary, b_{CSD}, or b'_{CSD}
†: no re-encoding after EAC Opt.: Optimal
‡: diminished-1 representation

Table: Speedup ($\times$) for a mod-32 channel vs stack size S and sp

sp	$S=0$			$S=1$			$S=2$	ZS†
	Bin.	**CSD**	**Opt.**	**Bin.**	**CSD**	**Opt.**	**Bin.**	
0	1.13	1.32	1.50	1.32	1.65	**1.74**	1.37	1
0.1	1.23	1.43	1.57	1.43	1.75	**1.84**	1.53	1.11
0.2	1.34	1.52	1.64	1.55	1.83	**1.91**	1.72	1.25
0.3	1.45	1.61	1.71	1.68	1.89	**1.95**	1.85	1.42
0.4	1.57	1.70	1.78	1.79	1.94	**1.98**	1.94	1.66
0.5	1.67	1.78	1.84	1.89	1.97	**1.99**	1.98	2

†: Zero skipping

Encoder scheme comparison

Table: Probability of a conflict P_c for mod 2^n, mod$(2^n - 1)$ and mod$(2^n + 1)$ and channel sizes $n = 4$ and $n = 5$

n	mod 2^n					mod $(2^n - 1)$					mod $(2^n + 1)$‡				
	no enc	CSD	CSD/bin	combined	Opt.	no enc	CSD†	CSD†/bin	combined†	Opt.†	no enc	CSD†	CSD†/bin	combined†	Opt.‡
4	0.683	0.457	0.425	0.425	0.332	0.648	0.515	0.462	0.328	0.328	0.605	0.439	0.401	0.335	0.308
5	0.762	0.516	0.476	0.476	0.333	0.749	0.578	0.521	0.413	0.339	0.717	0.529	0.482	0.413	0.327

no enc: both weights in binary
CSD/bin: a, CSD; b, either binary or CSD
CSD: independent CSD encoding of weights
combined: a, CSD, b, binary, b_{CSD}, or b'_{CSD}
†: no re-encoding after EAC Opt.: Optimal
‡: diminished-1 representation

Table: Speedup ($\times$) for a mod-32 channel vs stack size S and sp

sp	$S = 0$			$S = 1$			$S = 2$	ZS†
	Bin.	**CSD**	**Opt.**	**Bin.**	**CSD**	**Opt.**	**Bin.**	
0	1.13	1.32	1.50	1.32	1.65	**1.74**	1.37	1
0.1	1.23	1.43	1.57	1.43	1.75	**1.84**	1.53	1.11
0.2	1.34	1.52	1.64	1.55	1.83	**1.91**	1.72	1.25
0.3	1.45	1.61	1.71	1.68	1.89	**1.95**	1.85	1.42
0.4	1.57	1.70	1.78	1.79	1.94	**1.98**	1.94	1.66
0.5	1.67	1.78	1.84	1.89	1.97	**1.99**	1.98	2

†: Zero skipping

Table: Hardware complexity (number of gates) of the different encoders

Encoding	mod 2^n	mod $(2^n - 1)$
CSD	21	29
CSD/bin	50	71
Combined	80	112
Optimal	495	515

Presentation Outline

1 Introduction

2 Proposed PE (exploiting bit-level sparsity)

3 Overall CNN Architecture

4 Results

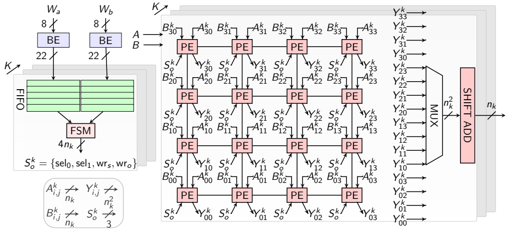
Processing Core

- PEs that share the same weight are grouped together

Processing Core

- PEs that share the same weight are grouped together
- Since the weight determines their operation, all PEs work in a synchronized manner

1. No additional storage elements/scheduling to support the different processing rates of PEs
2. Control logic is amortized over 16 PEs

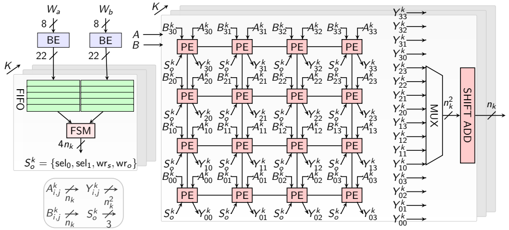


W_a
W_b
8
BE
22
K
FIFO
FSM
$4n_k$
$S_o^k = \{\mathrm{sel}_0, \mathrm{sel}_1, \mathrm{wr}_s, \mathrm{wr}_o\}$
$A_{i,j}^k$ n_k $Y_{i,j}^k$ n_k^2
$B_{i,j}^k$ n_k S_o^k 3
A
B
PE
B_{30}^k A_{30}^k B_{31}^k A_{31}^k B_{32}^k A_{32}^k B_{33}^k A_{33}^k
S_o^k Y_{30}^k Y_{31}^k Y_{32}^k Y_{33}^k
Y_{00}^k … Y_{33}^k
MUX
n_k^2
SHIFT ADD
n_k

Processing Core

- PEs that share the same weight are grouped together
- Since the weight determines their operation, all PEs work in a synchronized manner

1. No additional storage elements/scheduling to support the different processing rates of PEs
2. Control logic is amortized over 16 PEs
3. The CSD/optimal encoder is also amortized; its overhead becomes negligible

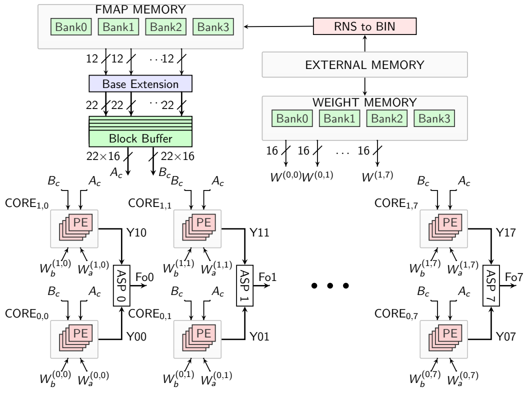


W_a
W_b
8
BE
22
K
FIFO
FSM
$4n_k$
$S_o^k = \{\mathrm{sel}_0, \mathrm{sel}_1, \mathrm{wr}_s, \mathrm{wr}_o\}$
$A_{i,j}^k$ n_k $Y_{i,j}^k$ n_k^2
$B_{i,j}^k$ n_k S_o^k 3
A
B
PE
MUX
SHIFT ADD
n_k^2
n_k
Y_{33}^k Y_{32}^k Y_{31}^k Y_{30}^k Y_{23}^k Y_{22}^k Y_{21}^k Y_{20}^k Y_{13}^k Y_{12}^k Y_{11}^k Y_{10}^k Y_{03}^k Y_{02}^k Y_{01}^k Y_{00}^k

Core overhead:

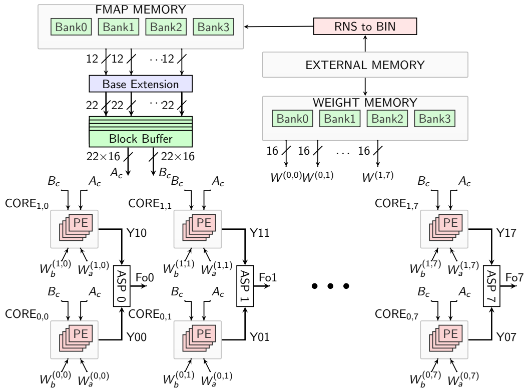
- shift-add units
- one additional base extension unit
- larger shift register array
- weight FIFO buffers

System-level Architecture and Dataflow



- Data is represented in a reduced RNS base of 8 or 12 bits
- 16 RNS Cores

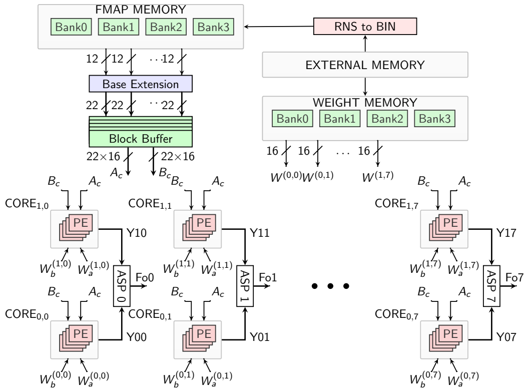
System-level Architecture and Dataflow


FMAP MEMORY
Bank0 Bank1 Bank2 Bank3
RNS to BIN
EXTERNAL MEMORY
WEIGHT MEMORY
Bank0 Bank1 Bank2 Bank3
12 12 ··12
Base Extension
22 22 ··22
Block Buffer
22×16 22×16
16 16 ... 16
$W^{(0,0)}$ $W^{(0,1)}$ $W^{(1,7)}$
B_c A_c
$\text{CORE}_{1,0}$ $\text{CORE}_{1,1}$ $\text{CORE}_{1,7}$
$\text{CORE}_{0,0}$ $\text{CORE}_{0,1}$ $\text{CORE}_{0,7}$
PE
Y10 Y11 Y17 Y00 Y01 Y07
ASP 0 ASP 1 ASP 7
Fo0 Fo1 Fo7
$W_b^{(1,0)}$ $W_a^{(1,0)}$ $W_b^{(1,1)}$ $W_a^{(1,1)}$ $W_b^{(1,7)}$ $W_a^{(1,7)}$
$W_b^{(0,0)}$ $W_a^{(0,0)}$ $W_b^{(0,1)}$ $W_a^{(0,1)}$ $W_b^{(0,7)}$ $W_a^{(0,7)}$
· · ·

- Data is represented in a reduced RNS base of 8 or 12 bits
- FMEM: stores intermediate layer results in the reduced RNS base, WMEM: stores weights

- 16 RNS Cores
- $\mathcal{B} = (5, 7, 31, 32, 33)$

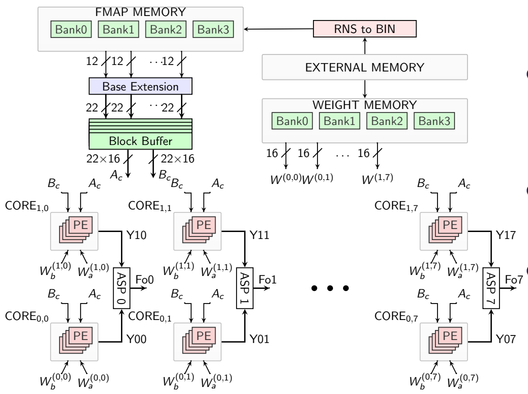
System-level Architecture and Dataflow



- Data is represented in a reduced RNS base of 8 or 12 bits
- FMEM: stores intermediate layer results in the reduced RNS base, WMEM: stores weights
- Base Extension: extend data to the full RNS base (22 bits)

- 16 RNS Cores
- $\mathcal{B} = (5, 7, 31, 32, 33)$

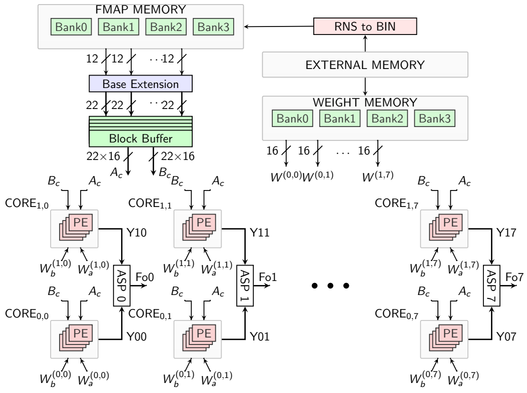
System-level Architecture and Dataflow


FMAP MEMORY
Bank0 Bank1 Bank2 Bank3
RNS to BIN
EXTERNAL MEMORY
12 12 ··· 12
Base Extension
22 22 ··· 22
Block Buffer
22×16 22×16
WEIGHT MEMORY
Bank0 Bank1 Bank2 Bank3
16 16 ··· 16
$W^{(0,0)}$ $W^{(0,1)}$ $W^{(1,7)}$
A_c B_c
CORE$_{1,0}$ CORE$_{1,1}$ CORE$_{1,7}$
CORE$_{0,0}$ CORE$_{0,1}$ CORE$_{0,7}$
PE
Y10 Y11 Y17 Y00 Y01 Y07
ASP 0 ASP 1 ASP 7
Fo0 Fo1 Fo7
$W_b^{(1,0)}$ $W_a^{(1,0)}$ $W_b^{(1,1)}$ $W_a^{(1,1)}$ $W_b^{(1,7)}$ $W_a^{(1,7)}$
$W_b^{(0,0)}$ $W_a^{(0,0)}$ $W_b^{(0,1)}$ $W_a^{(0,1)}$ $W_b^{(0,7)}$ $W_a^{(0,7)}$

- Data is represented in a reduced RNS base of 8 or 12 bits
- FMEM: stores intermediate layer results in the reduced RNS base, WMEM: stores weights
- Base Extension: extend data to the full RNS base (22 bits)
- A 2-D input block is fetched and broadcast to all cores

- 16 RNS Cores
- $\mathcal{B} = (5, 7, 31, 32, 33)$

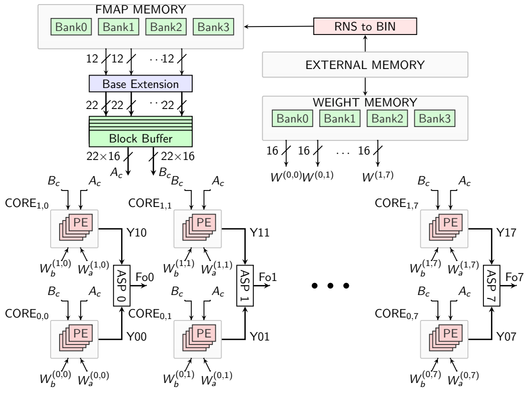
System-level Architecture and Dataflow


FMAP MEMORY
Bank0 Bank1 Bank2 Bank3
RNS to BIN
EXTERNAL MEMORY
12 12 ··· 12
Base Extension
22 22 ··· 22
Block Buffer
22×16
22×16
WEIGHT MEMORY
Bank0 Bank1 Bank2 Bank3
16 16 ··· 16
$W^{(0,0)}$ $W^{(0,1)}$ $W^{(1,7)}$
A_c B_c
CORE$_{1,0}$ CORE$_{1,1}$ CORE$_{1,7}$
CORE$_{0,0}$ CORE$_{0,1}$ CORE$_{0,7}$
PE
Y10 Y11 Y17 Y00 Y01 Y07
ASP 0 ASP 1 ASP 7
Fo0 Fo1 Fo7
$W_b^{(1,0)}$ $W_a^{(1,0)}$ $W_b^{(1,1)}$ $W_a^{(1,1)}$ $W_b^{(1,7)}$ $W_a^{(1,7)}$
$W_b^{(0,0)}$ $W_a^{(0,0)}$ $W_b^{(0,1)}$ $W_a^{(0,1)}$ $W_b^{(0,7)}$ $W_a^{(0,7)}$

- Data is represented in a reduced RNS base of 8 or 12 bits
- FMEM: stores intermediate layer results in the reduced RNS base, WMEM: stores weights
- Base Extension: extend data to the full RNS base (22 bits)
- A 2-D input block is fetched and broadcast to all cores
- One Activation/Scaling Unit per two cores

- 16 RNS Cores
- $\mathcal{B} = (5, 7, 31, 32, 33)$

System-level Architecture and Dataflow



- 16 RNS Cores
- $\mathcal{B} = (5, 7, 31, 32, 33)$

- Data is represented in a reduced RNS base of 8 or 12 bits
- FMEM: stores intermediate layer results in the reduced RNS base, WMEM: stores weights
- Base Extension: extend data to the full RNS base (22 bits)
- A 2-D input block is fetched and broadcast to all cores
- One Activation/Scaling Unit per two cores
- Block Buffer to decouple access to FMEM

Presentation Outline

1 Introduction

2 Proposed PE (exploiting bit-level sparsity)

3 Overall CNN Architecture

4 Results

Sparsity and throughput exploration on CNN benchmarks

Table: Sparsity of CNN benchmarks and expected speedup

Network (8-bit quant.)	**Weight Sparsity (%)**	**MF-D-S Theoretical Speedup***	**MF-D-S Simulation Speedup***	**ZS**†
VGG19	42	1.82×/1.95×	1.79×/1.88×	1.72 ×
ResNet50	12	1.42×/1.77×	1.47×/1.76×	1.13 ×
Yolo3	40	1.80×/1.94×	1.80×/1.88×	1.67 ×
InceptionV3	8	1.40×/1.73×	1.43×/1.73×	1.08 ×
MobileNet	7	1.39×/1.73×	1.41×/1.73×	1.03 ×

*: In entries of the form x/y, x refers to binary encoding and y refers to CSD encoding ($S = 1$)

Sparsity and throughput exploration on CNN benchmarks

Table: Sparsity of CNN benchmarks and expected speedup

Network (8-bit quant.)	Weight Sparsity (%)	MF-D-S Theoretical Speedup*	MF-D-S Simulation Speedup*	ZS[†]
VGG19	42	$1.82\times/1.95\times$	$1.79\times/1.88\times$	$1.72\times$
ResNet50	12	$1.42\times/1.77\times$	$1.47\times/1.76\times$	$1.13\times$
Yolo3	40	$1.80\times/1.94\times$	$1.80\times/1.88\times$	$1.67\times$
InceptionV3	8	$1.40\times/1.73\times$	$1.43\times/1.73\times$	$1.08\times$
MobileNet	7	$1.39\times/1.73\times$	$1.41\times/1.73\times$	$1.03\times$

*: In entries of the form x/y, x refers to binary encoding and y refers to CSD encoding ($S = 1$)

- Proposed method results in $1.73\times$ to $1.88\times$ speedup

Sparsity and throughput exploration on CNN benchmarks

Table: Sparsity of CNN benchmarks and expected speedup

Network (8-bit quant.)	Weight Sparsity (%)	MF-D-S Theoretical Speedup*	MF-D-S Simulation Speedup*	ZS†
VGG19	42	1.82×/1.95×	1.79×/1.88×	1.72 ×
ResNet50	12	1.42×/1.77×	1.47×/1.76×	1.13 ×
Yolo3	40	1.80×/1.94×	1.80×/1.88×	1.67 ×
InceptionV3	8	1.40×/1.73×	1.43×/1.73×	1.08 ×
MobileNet	7	1.39×/1.73×	1.41×/1.73×	1.03 ×

*: In entries of the form x/y, x refers to binary encoding and y refers to CSD encoding ($S = 1$)

- Proposed method results in 1.73× to 1.88× speedup

Takeaway

Unlike other sparse processing CNN architectures that rely on zero-skipping and require high sparsity levels to become efficient, the proposed method achieves gains with zero word-level sparsity (exploits bit-level sparsity)

Single PE comparisons

Table: PE comparison for various target clock periods (0.5 V)

T_{clk}	**Area (μm^2)**				**Power (μW)**			
(ns)	BNS	RNS	MF-D-S[†]		BNS	RNS	MF-D-S[†]	
			$S=0$	$S=1$			$S=0$	$S=1$
0.8	-	-	420	592/625	-	-	137	153/158
0.9	-	340	400	580/610	-	107	125	131/140
1.0	-	334	391	571/585	-	93	110	116/130
1.1	350	329	384	556/581	128	84	102	106/112
1.2	336	308	379	542/581	110	74	83	99/103

Single PE comparisons

Table: PE comparison for various target clock periods (0.5 V)

T_{clk}	Area (μm^2)				Power (μW)			
(ns)	BNS	RNS	MF-D-S†		BNS	RNS	MF-D-S†	
			$S = 0$	$S = 1$			$S = 0$	$S = 1$
0.8	-	-	420	592/625	-	-	137	153/158
0.9	-	340	400	580/610	-	107	125	131/140
1.0	-	334	391	571/585	-	93	110	116/130
1.1	350	329	384	556/581	128	84	102	106/112
1.2	336	308	379	542/581	110	74	83	99/103

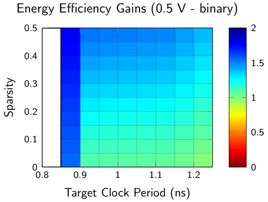
- 1.85× and 1.54× more energy efficient processing compared to binary and conventional RNS
- The MF-D-S (S=1) PE achieves higher energy efficiency gains $\frac{P_{RNS}}{P_{MF-D-S}} \times$ *speedup* as the clock period becomes smaller and sparsity increases.

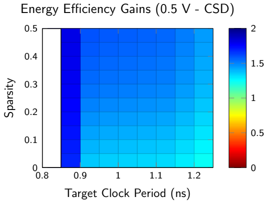
Single PE comparisons

Table: PE comparison for various target clock periods (0.5 V)

T_{clk}	Area (μm^2)				Power (μW)			
			MF-D-S†				MF-D-S†	
(ns)	BNS	RNS	$S=0$	$S=1$	BNS	RNS	$S=0$	$S=1$
0.8	-	-	420	592/625	-	-	137	153/158
0.9	-	340	400	580/610	-	107	125	131/140
1.0	-	334	391	571/585	-	93	110	116/130
1.1	350	329	384	556/581	128	84	102	106/112
1.2	336	308	379	542/581	110	74	83	99/103

- 1.85× and 1.54× more energy efficient processing compared to binary and conventional RNS
- The MF-D-S (S=1) PE achieves higher energy efficiency gains $\frac{P_{RNS}}{P_{MF-D-S}} \times$ *speedup* as the clock period becomes smaller and sparsity increases.





- CSD encoding can further increase energy efficiency.

Core-level comparisons

Table: Area and power breakdown of the various components

Units	Components	Area[†] (μm^2)			Power[†] (μW)		
		BNS	**RNS**	**MF-D-S**	**BNS**	**RNS**	**MF-D-S**
PE		350/277	329/279	556/529	128/220	93/140	120/229
Shift-add Unit		-	-	160/110	-	-	20/25
FMAP Base Ext.		-	250/190	250/190	-	61/86	61/86
Weight Base Ext.		-	107/86	107/86	-	25/36	25/36
ASP Unit	Scaling + ReLU + Pooling	622/600	5761/5450	5761/5450	118/202	384/615	384/615
Core	4×4 PE array + BE (+shift-add unit)	**6.27/5.3**$\times 10^3$	**6.30/5.43**$\times 10^3$	**10.35/9.23**$\times 10^3$	**1.98/3.50**$\times 10^3$	**1.63/2.73**$\times 10^3$	**2.04/3.36**$\times 10^3$

[†]: In entries of the form x/y, x refers to a supply voltage of 0.5 V and y refers to 0.65 V

Core-level comparisons

Table: Area and power breakdown of the various components

Units	Components	Area[†] (μm^2)			Power[†] (μW)		
		BNS	RNS	MF-D-S	BNS	RNS	MF-D-S
PE		350/277	329/279	556/529	128/220	93/140	120/229
Shift-add Unit		-	-	160/110	-	-	20/25
FMAP Base Ext.		-	250/190	250/190	-	61/86	61/86
Weight Base Ext.		-	107/86	107/86	-	25/36	25/36
ASP Unit	Scaling + ReLU + Pooling	622/600	5761/5450	5761/5450	118/202	384/615	384/615
Core	4×4 PE array + BE (+shift-add unit)	**6.27/5.3**$\times 10^3$	**6.30/5.43**$\times 10^3$	**10.35/9.23**$\times 10^3$	**1.98/3.50**$\times 10^3$	**1.63/2.73**$\times 10^3$	**2.04/3.36**$\times 10^3$

[†]: In entries of the form x/y, x refers to a supply voltage of 0.5 V and y refers to 0.65 V

- Area overhead is completely compensated by the increased processing rate
- Power efficiency can be increased by 31%–54% (57%–85%) depending on the sparsity of the weights, compared to conventional RNS (BNS)

System-level comparisons

Table: Comparisons to state-of-the-art implementations

	Eyerissv2[1]	ISSCC'20[2]	ISSCC'22[3]	RNSDNN[4]	This work RNS	**This work MF-D-S**
Process	65 nm	7 nm	65 nm	45 nm	22 nm	**22 nm**
Supply voltage (V)	N/G	0.575–0.825	1	1	0.65	**0.65**
Frequency	200 MHz	290–880 MHz	400 MHz	1.2 GHz	1 GHz	**1 GHz**
On-chip Memory (KB)	246	2176	150 KB	N/G	448	**448**
Bit Precision (FMAP,wgt)	8	8	8	16,8	12,8	**12,8**
Network	AlexNet	MobileNet-v1	VGG16	VGG16	VGG16	**VGG16**
Performance (GOPS†)	153.6	3604	N/G	134	220	**364**
Area (10^6 Gates)	2.69	N/G	N/G	2.18	4.74	**5.25**
Area (mm^2)	N/G	3.04	4.47	N/G	0.94	**1.04**
Area Eff.(GOPS†/10^6 Gates)	57.1	N/G	N/G	62	46.4	**69.3**
Power Eff. (TOPS†/W)	0.253 - 0.962‡	3.28 - 6.66‡	1.82	0.223	1.74/2.36*	**1.98/3.12***

*: full system/on-chip power cons. †1 OP = 1 MAC ‡: for dense - sparse network N/G: not given

System-level comparisons

Table: Comparisons to state-of-the-art implementations

	Eyerissv2[1]	ISSCC'20[2]	ISSCC'22[3]	RNSDNN[4]	This work RNS	**This work MF-D-S**
Process	65 nm	7 nm	65 nm	45 nm	22 nm	**22 nm**
Supply voltage (V)	N/G	0.575–0.825	1	1	0.65	**0.65**
Frequency	200 MHz	290–880 MHz	400 MHz	1.2 GHz	1 GHz	**1 GHz**
On-chip Memory (KB)	246	2176	150 KB	N/G	448	**448**
Bit Precision (FMAP,wgt)	8	8	8	16,8	12,8	**12,8**
Network	AlexNet	MobileNet-v1	VGG16	VGG16	VGG16	**VGG16**
Performance (GOPS†)	153.6	3604	N/G	134	220	**364**
Area (10^6 Gates)	2.69	N/G	N/G	2.18	4.74	**5.25**
Area (mm^2)	N/G	3.04	4.47	N/G	0.94	**1.04**
Area Eff.(GOPS†/10^6 Gates)	57.1	N/G	N/G	62	46.4	**69.3**
Power Eff. (TOPS†/W)	0.253 - 0.962‡	3.28 - 6.66‡	1.82	0.223	1.74/2.36*	**1.98/3.12***

*: full system/on-chip power cons. †1 OP = 1 MAC ‡: for dense - sparse network N/G: not given

- 32% energy efficiency increase compared to RNS Counterpart
- 8.87× more energy efficient that state-of-the-art RNS CNN accelerator

System-level comparisons

Table: Comparisons to state-of-the-art implementations

	Eyerissv2[1]	ISSCC'20[2]	ISSCC'22[3]	RNSDNN[4]	This work RNS	**This work MF-D-S**
Process	65 nm	7 nm	65 nm	45 nm	22 nm	**22 nm**
Supply voltage (V)	N/G	0.575–0.825	1	1	0.65	**0.65**
Frequency	200 MHz	290–880 MHz	400 MHz	1.2 GHz	1 GHz	**1 GHz**
On-chip Memory (KB)	246	2176	150 KB	N/G	448	**448**
Bit Precision (FMAP,wgt)	8	8	8	16,8	12,8	**12,8**
Network	AlexNet	MobileNet-v1	VGG16	VGG16	VGG16	**VGG16**
Performance (GOPS†)	153.6	3604	N/G	134	220	**364**
Area (10^6 Gates)	2.69	N/G	N/G	2.18	4.74	**5.25**
Area (mm^2)	N/G	3.04	4.47	N/G	0.94	**1.04**
Area Eff.(GOPS†/10^6 Gates)	57.1	N/G	N/G	62	46.4	**69.3**
Power Eff. (TOPS†/W)	0.253 - 0.962‡	3.28 - 6.66‡	1.82	0.223	1.74/2.36*	**1.98/3.12***

*: full system/on-chip power cons. †1 OP = 1 MAC ‡: for dense - sparse network N/G: not given

- 32% energy efficiency increase compared to RNS Counterpart
- 8.87× more energy efficient that state-of-the-art RNS CNN accelerator
- 2.05× more energy efficient that sparse version of Eyeriss

References

Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, “Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

C.-H. Lin, C.-C. Cheng, Y.-M. Tsai, S.-J. Hung, Y.-T. Kuo, P. H. Wang, P.-K. Tsung, J.-Y. Hsu, W.-C. Lai, C.-H. Liu, S.-Y. Wang, C.-H. Kuo, C.-Y. Chang, M.-H. Lee, T.-Y. Lin, and C.-C. Chen, “7.1 a 3.4-to-13.3tops/w 3.6tops dual-core deep-learning accelerator for versatile ai applications in 7nm 5g smartphone soc,” in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pp. 134–136.

Y. Ju and J. Gu, “A 65nm systolic neural cpu processor for combined deep learning and general-purpose computing with 95locality and enhanced end-to-end performance,” in *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.

N. Samimi, M. Kamal, A. Afzali-Kusha, and M. Pedram, “Res-DNN: A Residue Number System-Based DNN Accelerator Unit,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 2, pp. 658–671, 2020.

Q & A

Thank you!