Improving Residue-Level Sparsity in RNS-based Neural Network Hardware Accelerators via Regularization

E. Kavvousanos [1] V. Sakellariou [2] I. Kouretas [1] V. Paliouras [1] T. Stouraitis [2]

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

30th IEEE International Symposium on Computer Arithmetic
ARITH 2023
Portland, September 5, 2023

Overview

Table of Contents

RNS basics

An RNS maps an integer x to a tuple X of N residues

$$x \to X = (x_1, x_2, \ldots, x_N), \tag{1}$$

where $x_i = x \bmod m_i$ and m_i, $i = 1, 2, \ldots, N$, form a set called base $\mathcal{B}$,

$$\mathcal{B} = \{m_1, m_2, \ldots, m_N\}. \tag{2}$$

Moduli m_i of $\mathcal{B}$ are relatively co-prime; i.e.,

$$\gcd_{i \neq j}(m_i, m_j) = 1 \tag{3}$$

for all i, j, $1 \leq i, j \leq N$. The dynamic range of the representation is determined by $\mathcal{B}$, as

$$M = \prod_{i=1}^{N} m_i. \tag{4}$$

Neural Network Regularization

- Regularization is a set of strategies used in Machine Learning to reduce the generalization error, *i.e.*, overfitting.
- Modify the loss function: add regularization terms:

$$J'(\theta; X, y) = J(\theta; X, y) + a \cdot \boxed{R(\theta)} \tag{5}$$

- L1 regularization: $R(\theta) = \sum_i |\theta_i|$
- L2 regularization: $R(\theta) = \sum_i \theta_i^2$
- L1 and L2 regularization penalize large weights.

Table of Contents

Proposed RNS-conscious regularization

- Assume a subset $\mathcal{B}_{\text{weight}} \subset \mathcal{B}$.
- $\mathcal{B}_{\text{weight}}$ suffices to provide the dynamic range required for the representation of neural-network weights.
- The dynamic range provided by $\mathcal{B}_{\text{weight}}$ for the representation of the weights w, is

$$M_{\text{weight}} = \prod_{\forall m \in \mathcal{B}_{\text{weight}}} m. \tag{6}$$

- We propose the regularization function

$$R(w; \mathcal{B}, \mathcal{B}_w) = \lambda \prod_{i=1}^{N} \prod_{k=-K_i}^{K_i} \sigma_i \cdot (w \cdot M_{\text{weight}} - k \cdot m_i)^2, \tag{7}$$

where λ, σ_i are chosen hyperparameters and $K_i = \left\lfloor \frac{\frac{1}{2} M_{\text{weight}}}{m_i} \right\rfloor$.

Proposed RNS-conscious regularization

- The values of σ_i are chosen such that the product of (7) neither decays nor explodes.
- During training, the regularization term (7) drives a weight w, to assume a value which when converted to an integer $\widehat{w}$, is an integral multiple of m_i; therefore, the corresponding residue $\widehat{w}_i$ is zero, i.e.,

$$\widehat{w}_i = Q(wM_{\text{weight}}) \bmod m_i = 0,$$

  where $Q(x)$ rounds its argument to the nearest integer.
- In this way, the proposed regularization term increases the residue sparsity.

For the case of a network with N_w weights, the regularization term (7) can be extended as

$$R(w; \mathcal{B}, \mathcal{B}_w) = \lambda \sum_{n=1}^{N_w} \prod_{i=1}^{N} \prod_{k=-K_i}^{K_i} \sigma_i \cdot (w_n \cdot M_{\text{weight}} - k \cdot m_i)^2, \tag{8}$$

Training Results

- Test case: CNN on CIFAR-10 benchmark.

Table: CNN architecture

Layer	**Kernel Dimensions**
Convolutional 2D	$3 \times 3 \times 3 \times 32$
Max Pooling (2×2)	–
Convolutional 2D	$3 \times 3 \times 32 \times 64$
Max Pooling (2×2)	–
Convolutional 2D	$3 \times 3 \times 64 \times 64$
Fully Connected	1024×64
Fully Connected	64×10

- Assuming an RNS:

$$\mathcal{B} = \{5, 7, 31, 32, 33\}$$
$$M = 5 \cdot 7 \cdot 31 \cdot 32 \cdot 33$$

CIFAR-10 CNN Example

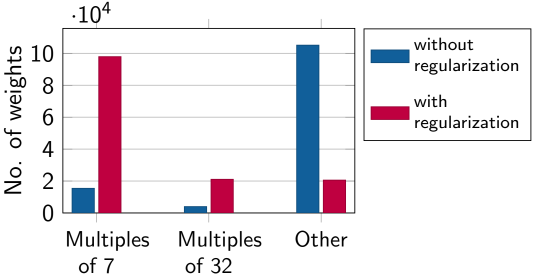


Figure: Histogram of $\lfloor w \cdot M_{\text{weight}} \rceil$ before (blue) and after (red) regularization. Base $\mathcal{B}_{\text{weight}} = \{7, 32\}$ is assumed.

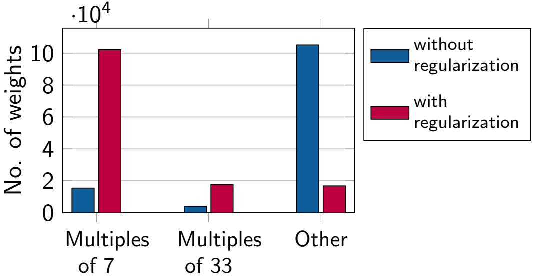


Figure: Histogram of $\lfloor w \cdot M_{\text{weight}} \rceil$ before (blue) and after (red) regularization. Base $\mathcal{B}_{\text{weight}} = \{7, 33\}$ is assumed.

CIFAR-10 CNN Example

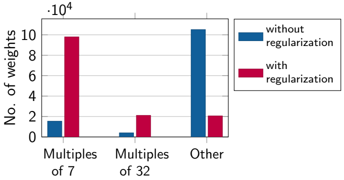


Figure: Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ before (blue) and after (red) regularization. Base $\mathcal{B}_{\text{weight}} = \{7, 32\}$ is assumed.

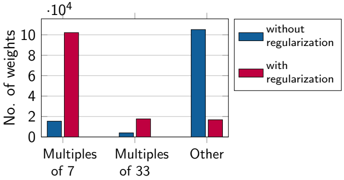


Figure: Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ before (blue) and after (red) regularization. Base $\mathcal{B}_{\text{weight}} = \{7, 33\}$ is assumed.

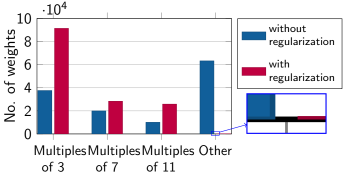


Figure: Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ before (blue) and after (red) regularization. Base $\mathcal{B}_{\text{weight}} = \{3, 7, 11\}$ is assumed.

VGG-16 Example

- Test case: VGG-16 on ImageNet benchmark.
- Last Fully-Connected layer (4096×1000 weights).

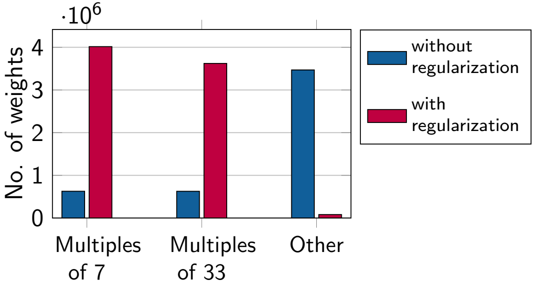


Figure: Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ before (blue) and after (red) regularization, for the last FC layer of VGG16 on ImageNet. Base $\mathcal{B}_{\text{weight}} = \{7, 33\}$ is assumed.

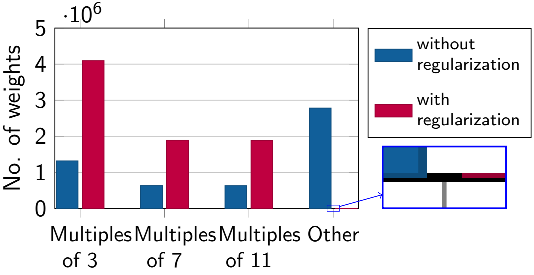


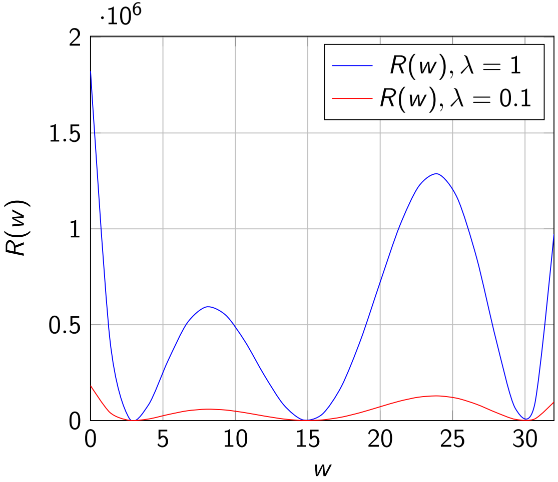
Figure: Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ before (blue) and after (red) regularization, for the last FC layer of VGG16 on ImageNet. Base $\mathcal{B}_{\text{weight}} = \{3, 7, 11\}$ is assumed.

Limiting the regularization loss

- Regularization loss (8) can assume extremely large values.
- Assume the regularization function:

$$R(w) = \lambda(w-3)^2(w-15)^2(w-30)^2.$$

- Parameters λ, σ_i of (8) are tuned to limit $R(w)$.
- In our experiments, $\lambda \in \{10^{-2}, 10^{-1}, 1\}$ and $\sigma_i \in [10^{-4}, 10^{-3}]$ depending on the employed RNS base.



Computational complexity

- Application of regularization loss (8) can deteriorate training speed significantly.
- For a large number N of moduli and for large K_i.
- The double product of (8) is composed of P terms,

$$P = \sum_{i=1}^{N} (2K_i + 1). \tag{9}$$

- For $\mathcal{B}_{\text{weight}} = \{7, 32\}$, $N = 2$, and $M_{\text{weight}} = 7 \cdot 32 = 224$. $K_1 = 16$, $K_2 = 3$, leading to $P = 40$ terms <u>per weight</u>.

Table: Training slowdown

$\mathcal{B}_{\textbf{weight}}$	**Slowdown**
Without regularization	$1\times$
$\{7, 32\}$	$9\times$
$\{7, 33\}$	$8.4\times$
$\{7, 9, 17\}$	$15.8\times$
$\{5, 7, 17\}$	$18.6\times$

Table of Contents

Hardware architectures exploiting residue sparsity

Exploit residue-level sparsity to:

- Reduce memory cost
- Reduce power consumption

Utilize low-cost moduli for the RNS bases

- $\mathcal{B} = \{5, 7, 31, 32, 33\}$
- $\mathcal{B}' = \{5, 7, 9, 16, 17, 31\}$
- $\mathcal{B}'' = \{3, 5, 7, 11, 31, 32\}$

Table: MAC Complexity for $\mathcal{B}$

MAC	Area (μm^2)	Area (%)[1]	Power (μW)	Power (%)[1]
modulo-5	14	4.6	5	4.1
modulo-7	28	9.3	14	11.6
modulo-31	70	23.3	37	30.6
modulo-32	33	11	16	13.2
modulo-33	156	51.8	49	40.4

Table: MAC Complexity for $\mathcal{B}'$

MAC	Area (μm^2)	Area (%)[1]	Power (μW)	Power (%)[1]
modulo-5	14	5.1	5	4.2
modulo-7	28	10.3	14	11.7
modulo-9	50	18.5	22	18.3
modulo-16	24	8.8	11	9.2
modulo-17	84	31.1	31	25.8
modulo-31	70	25.9	37	30.8

Table: MAC Complexity for $\mathcal{B}''$

MAC	Area (μm^2)	Area (%)[1]	Power (μW)	Power (%)[1]
modulo-3	10	4	4	3.8
modulo-5	14	5.6	5	4.7
modulo-7	28	11.3	14	13.3
modulo-11	93	37.5	29	27.7
modulo-31	70	28.2	37	35.2
modulo-32	33	13.3	16	15.2

Exploiting residue sparsity to reduce memory cost

- Commonly used compression schemes in sparse CNN architectures (CSC, CSR) are not suitable for the residue-sparse scenario
- Non-zero values occur at different indexes within the residue channels $\rightarrow$ multiple index vectors
- Utilize a variable-length encoding
- Compression ratio depends on the sparsity factors α_i.

$$G(d) = \begin{cases} 0, & \text{if } d = 0 \\ 1 d_{n-1} d_n \dots d_0, & \text{otherwise.} \end{cases} \tag{10}$$

The average size, $\widehat{n}$, of the encoded word assuming base $\mathcal{B}$ is given by

$$\begin{aligned} \widehat{n} &= \alpha_0 + (3+1)(1-\alpha_0) + \alpha_1 + (5+1)(1-\alpha_0) && (11) \\ &= 10 - 3\alpha_0 - 5\alpha_1. && (12) \end{aligned}$$

Using the sparsity factors $\alpha_0 = 0.8$ and $\alpha_1 = 0.14$, the average size of the encoded word is $\widehat{n} = 6.9$ bits, translating to a 13.75% compression ratio.

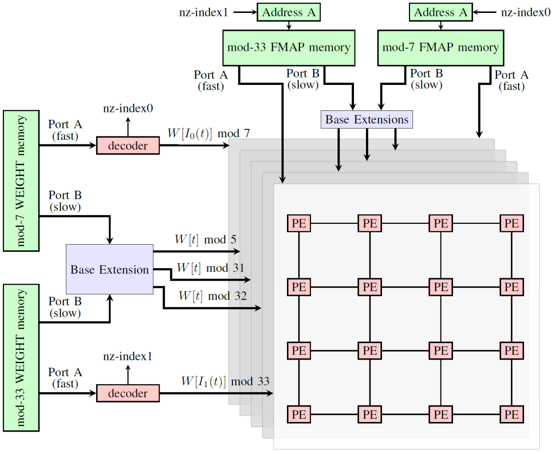
Exploiting residue sparsity to reduce power consumption

- Zero-skipping per moduli channel.
- The workloads of the different residue channels become unbalanced, since the regularization results in different sparsity levels
- Each channel may complete its computation at potentially different times
- Deactivate (power gating) a residue channel when it completes the processing
- Power savings depend on the achieved sparsity of each channel and its contribution to the total power consumption

Table: RNS base comparison

Base	$\mathcal{B}$	$\mathcal{B}'$	$\mathcal{B}''$
Power before regularization (μW)	121	120	105
Power after regularization (μW)	102.8	101.9	92.7
Savings (%)	15	15.1	11.7

Overall CNN architecture



- *n* (number of moduli) independent PE arrays of size $M \times M$
- A non-zero detector module reads a window of weight values (encoded with respect to $\mathcal{B}_{\text{weight}}$) and provides the next non-zero weight.
- The index of the next non-zero weight is used to select the corresponding input feature-map
- Base extension units are used to obtain the rest of the channels required for the convolution

Memory Organization

- Different sparsity levels $\rightarrow$ different processing rates for each channel
- Each channel requires a weight residue value from a different index of the weight vector at each timestep w_k^t
- $w_k^t = W[I_k(t)] \bmod m_k$, where W is the weight vector, $I_k(t)$ denotes the index of the weight required by the k-th channel $(0 \leq k < N)$ at time t.
- Base extension adds N_{be} channel to $\mathcal{B}_{weight}$ to obtain $\mathcal{B}$ $(N = N_w + N_{be})$.
- $I_k(t)$ are different for all the first N_w channels, while $I_k(t) = t$ for $k \geq N_w$
- Need to decouple the access to the weights of each residue channel $\rightarrow$ separate memory banks for each channel.
- At each timestep t, two weights from each channel are needed: one with an index of $I_k(t) \geq t$ and one with an index t required for the base extension $\rightarrow$ dual-port RAM macros.

Weight Decoder

- Performs the decompression of the weight values, according to the proposed variable-length compression scheme.
- For each channel in $\mathcal{B}_{weight}$ the weight decoder consists of a weight bit buffer, implemented with a programmable barrel shifter and a leading-one detector.
- The encoded weights are read from the corresponding weight memory bank and stored to the weight-bit buffer.
- The leading-one detector calculates the index of the next '1', corresponding to the next non-zero weight value, which determines the number of shift positions of the buffer.

Table: MAC and Decoder Complexity

Unit	Area (μm^2)	Power (μW)
Decoder $\mathcal{B}_{\text{weight}}$	85	45
MAC $\mathcal{B}$	301	121
Decoder $\mathcal{B}'_{\text{weight}}$	122	64
MAC $\mathcal{B}'$	270	120

- Decoder cost is small compared to MAC unit and amortized over a number if PEs
- For a 4×4 processing array the total decoder power consumption overhead is 2.3% and 3.3% for $\mathcal{B}_{weight}$ and $\mathcal{B}'_{weight}$

Table of Contents

Conclusions

- Modification of ANN training to induce increased residue-level sparsity in weights by regularization.
- $4\times$ to $6\times$ increase in residue sparsity in certain cases with minimal accuracy drop.
- Comparative evaluation of RNS bases in the context of the proposed method.
- Importance of the choice of RNS base for the exploitation of residue sparsity.
- Exploitation of residue-level sparsity to improve RNS-based hardware accelerators.
- Focus on decreasing energy requirements in hardware accelerators.
- Promising results for the final fully-connected layer of the VGG16 model.
- Potential for the proposed regularization technique to lead to new RNS architectures.
- Possibility of RNS becoming a candidate for hardware accelerators in edge devices.
- Future work: Optimize the method to scale efficiently with large models.

Thank you!