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

E. Kavvousanos*, V. Sakellariou†, I. Kouretas*, V. Paliouras* and T. Stouraitis†
*Department of Electrical and Computer Engineering, University of Patras, Greece
†Department of Electrical Engineering and Computer Science, Khalifa University, UAE

***Abstract*—Residue Number System (RNS) has recently attracted interest for the hardware implementation of inference in machine-learning systems as it provides promising trade-offs in the area, time, and power dissipation space. In this paper we introduce a technique that utilizes regularization during training, and increases the percentage of residues which are zero, when the parameters of an artificial neural network (ANN) are expressed in an RNS. The proposed technique can also be used as a post-processing stage, allowing the optimization of pre-trained models for RNS implementation. By increasing the number of residues being zero, i.e., residue-level sparsity, the proposed technique facilitates new hardware architectures for RNS-based inference, allowing new trade-offs and improving performance over prior art without practically compromising accuracy. The introduced method increases residue sparsity by a factor of $4\times$ to $6\times$ in certain cases.**

1. Introduction

Artificial intelligence (AI) and machine-learning (ML) applications proliferate in diverse fields of human activity and have already demonstrated extraordinary achievements. AI/ML solutions require substantial computational effort for both inference and training as well as large memory cost for model parameter storage. These requirements are translated to excessive needs for energy, hardware complexity and computational time. Dedicated hardware accelerators target to efficiently provide the required computational power and storage, by exploiting area, time, and energy trade-offs.

Recently, several authors have introduced RNS-based hardware accelerators for ML applications. Due to its attractive arithmetic properties, mainly its advantage in the implementation of multiply-accumulate (MAC) operations, RNS has been utilized in the design of deep neural network (DNN) accelerators, where MACs are dominant. A common approach in RNS-based DNN implementations is to perform all MAC operations of a single convolutional or dense layer in the RNS representation and then use a converter, to obtain a partial result in normal binary representation [1], [2], [3]. With this intermediate result, the non-linear activation functions (ReLU, $\tanh$, softmax) can be computed and the results can be converted back to RNS format to be fed to the next layer. Methods that implement activation functions in the RNS domain have also been proposed [4]. While these methods manage to offer performance gains in the implementation of a single convolutional layer, they result in a significant amount of additional hardware to perform the conversions. More recently, end-to-end RNS architectures have been developed that mitigate this overhead and perform all computations in the RNS domain [5], with up to 61% reduction in energy consumption when compared to conventional positional binary systems. RNS has been used in hardware accelerators for Long Short Time Memory (LSTM) [4], where RNS-conscious approximations are also introduced. Furthermore, hardware optimizations enabled by the use of RNS, which reduce the number of multiplications required in a CNN, are introduced in [6]. Optimization from a different perspective is sought by regularization-based techniques that have been exploited for introducing structure to models as well as for the quantization of parameters [7] [8]; however, to our knowledge, they have not yet been applied to optimize model parameters for RNS-based hardware implementation.

In this paper we introduce a technique that imposes certain properties on the parameters of a model, in order to achieve desirable hardware implementation characteristics, such as reduced energy, increased performance, lower complexity without compromising accuracy. The proposed technique, departing from prior RNS-related literature on ML hardware accelerators, induces properties in the weights that render their processing amenable for RNS implementation. Specifically, it relies on an introduced regularization of the loss function during training, here shown to increase the number of neural-network parameters that are multiples of the elements of the RNS base, thus increasing the number of zeros in the RNS channels; we refer to the percentage of zeros in an RNS channel, as *residue sparsity*. The proposed technique can be applied to pre-trained models as well, as a post-processing step. The introduced method is here shown to increase residue sparsity by a factor of $4\times$ to $6\times$ in certain cases. This is translated to some energy savings in RNS-based hardware accelerators. No practical degradation of accuracy occurs. The choice of the RNS base is also found to be important in the context of the proposed method.

The remainder of the paper is organized as follows: Section 2 revisits the basics of RNS. Section 3 introduces the proposed training method that imposes zero-value residues in the weights. Section 4 discusses the impact of the proposed induced residue-level sparsity on architectures for hardware accelerators, both related to processing and storage. Finally conclusions are discussed in Section 5.

2. RNS basics

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

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

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, $1 \leq i, j \leq N$ [9]. The dynamic range of the representation is determined by $\mathcal{B}$, as

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

Let $\circ$ denote the operation of addition or multiplication among two RNS tuples X and Y. RNS is of interest because the operations of addition and multiplication of two integers x and y, mapped onto X and Y, can be performed as

$$Z = X \circ Y, \tag{5}$$

where $\circ$ is implemented point-wise, as

$$z_i = (x_i \circ y_i) \bmod m_i, \tag{6}$$

in a carry-free manner. The integer result z can be obtained from its image Z by means of the Chinese Remainder Theorem (CRT) [9]

$$Z \xrightarrow{\text{CRT}} z. \tag{7}$$

By appropriately selecting the elements of $\mathcal{B}$, benefits are expected for hardware systems that perform a substantial amount of multiplications and additions due to the parallelism imposed by (6) and the short word lengths involved in the residue channel processing.

3. Proposed RNS-conscious regularization

Assume a subset $\mathcal{B}_{\text{weight}} \subset \mathcal{B}$ of the base $\mathcal{B}$ that 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

$$M_{\text{weight}} = \prod_{\forall m \in \mathcal{B}_{\text{weight}}} m. \tag{8}$$

Commonly, training in ML systems uses back propagation to derive model parameters w that minimize a loss function [10],

$$E_{\text{loss}} = E(w; I, O), \tag{9}$$

where I are training inputs and O is an expression of the expected output. Regularization is a technique that can be used to impose additional properties on the model parameters [11] by suitably modifying the loss function, usually through the introduction of an additive term $R(w)$; i.e.,

$$E_{\text{loss}} = E(w; I, O) + R(w). \tag{10}$$

It is here proposed that a regularization term can be defined as

$$R(w; \mathcal{B}, \mathcal{B}_w) = \lambda \prod_{i=1}^{N} \prod_{k=-K_i}^{K_i} \sigma_i \cdot (w \cdot M_{\text{weight}} - k \cdot m_i)^2, \tag{11}$$

where are λ, σ_i are chosen hyperparameters and

$$K_i = \left\lfloor \frac{\frac{1}{2} M_{\text{weight}}}{m_i} \right\rfloor. \tag{12}$$

The weights w are assumed to lie in the interval $(-0.5, 0.5)$. The values of σ_is are chosen such that the consecutive multiplications of (11) derive products that neither decay nor explode. During training, the regularization term (11) drives a weight w, expressed in floating-point format, 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.,

$$\widehat{w}_i = Q(w M_{\text{weight}}) \bmod m_i = 0, \tag{13}$$

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

For the case of a network with N_w weights, the regularization term (11) can be extended as

$$R(w; \mathcal{B}, \mathcal{B}_w) = \lambda \sum_{n=1}^{N_w} \prod_{i=1}^{N} \prod_{k=-K_i}^{K_i} \sigma_i \cdot (w_n \cdot M_{\text{weight}} - k \cdot m_i)^2, \tag{14}$$

The proposed method can be used to optimize the parameters of a neural network, specifically for RNS processing, by increasing residue sparsity. Furthermore it can be applied to pre-trained networks as a post-processing procedure to modify already available model parameters.

3.1. An illustrative example

As an illustrative example of the application of the proposed method, assume that

$$\mathcal{B} = \{5, 7, 31, 32, 33\} \tag{15}$$

$$M = 5 \cdot 7 \cdot 31 \cdot 32 \cdot 33 \tag{16}$$

$$\mathcal{B}_{\text{weight}} = \{7, 32\} \tag{17}$$

$$M_{\text{weight}} = 7 \cdot 32 = 224. \tag{18}$$

Furthermore, as a test case, assume a CNN composed of three 2D-convolutional (Conv2D) layers followed by two Dense layers, trained for the CIFAR-10 benchmark. Table 1 details the structure of the test CNN. The obtained results are shown in Fig. 1, which reveals that the proposed method increases the number of integer weights which are a multiple of 7, by a factor of $6.35\times$. These weights have a residue mod 7 equal to zero. Similarly, the number of integer weights which are a multiple of 32, increases by a factor of $5.31\times$. A second choice for $\mathcal{B}_{\text{weight}}$, i.e., $\mathcal{B}_{\text{weight}} = \{7, 33\}$, is also quantitatively evaluated in Fig. 2. It is shown that the multiples of 7 increase by a factor of $6.63\times$, while the multiples of 33 increase by a factor of $4.45\times$. Furthermore, to evaluate the impact of the proposed method on bases featuring smaller moduli, we replace 33 with 3 and 11 in $\mathcal{B}$ of (15), resulting in $\mathcal{B}_{\text{weight}} = \{3, 7, 11\}$, without affecting the dynamic range. The results are shown in Fig. 3. It can be seen that the number of weights which are not multiples of any of the moduli in $\mathcal{B}_{\text{weight}}$, is practically nullified.

The accuracy of the employed CNN on the CIFAR-10 test set is 72%. Marginal loss (0.2%) was observed in all of the executed tests, attributed to the application of the proposed regularization schemes that increase residue sparsity.

Furthermore, VGG-16 [12] is employed on the ImageNet dataset, in order to assess the proposed regularization methodology with a state-of-the-art model. Because of the relatively high complexity of VGG-16, regularization

TABLE 1. 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

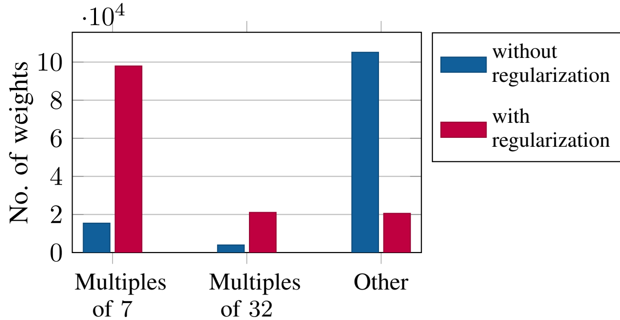

Figure 1. Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ before (blue) and after (red) regularization. Base $\mathcal{B}_{\text{weight}} = \{7, 32\}$ is assumed. Following regularization, the number of weights that are not multiples of either 7 or 32 is substantially reduced (other).

via (14) is only applied to the last Fully-Connected (FC) layer, which is parameterized by a 4096×1000 weight matrix. The results are shown in Figs. 4 and 5 for $\mathcal{B}_{\text{weight}} = \{7, 33\}$ and $\mathcal{B}_{\text{weight}} = \{3, 7, 11\}$, respectively. The bar plots show qualitatively similar results with Figs. 2 and 3 that refer to the CIFAR-10 CNN. Similarly, the accuracy of VGG-16 remains intact following the utilization of the proposed regularization method. However, applying the technique to the entire model may lead to accuracy degradation.

In general, careful tuning of the parameters λ, σ_i of (14) is required. In our experiments, we used $\lambda \in \{10^{-2}, 10^{-1}, 1\}$ and $\sigma_i \in [10^{-4}, 10^{-3}]$ depending on the employed RNS base. The values delivered by (14) for all the weights of the network and all the desirable multiples can get extremely small or huge, i.e., they require substantial dynamic range for their expression. During the training phase of the model in this experiment, the computation of (14) utilizes double-precision floating-point representation. Subsequently, the regularization term is converted into single-precision floating-point representation and incorporated into the loss function. During model evaluation, calculating the regularization term is unnecessary as only the accuracy metric is considered for assessment.

3.2. Computational complexity

As discussed in Section 3.1, the regularization term (14) can assume extremely large values, especially when i and k take many values, i.e., for a large number N of moduli of interest, and for large K_i. The double product of (14) is composed of P terms,

$$P = \sum_{i=1}^{N} (2K_i + 1). \tag{19}$$

For example, for $\mathcal{B}_{\text{weight}} = \{7, 32\}$, $N = 2$, and $M_{\text{weight}} = 7 \cdot 32 = 224$. From (12), it follows that $K_1 = 16$, $K_2 = 3$,

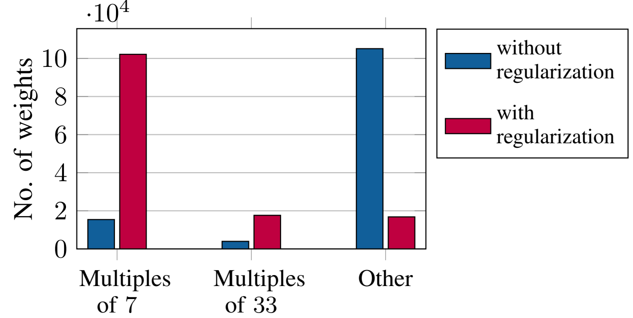

Figure 2. Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ before (blue) and after (red) regularization. Base $\mathcal{B}_{\text{weight}} = \{7, 33\}$ is assumed. Following regularization, the number of weights that are not multiples of either 7 or 33 is substantially reduced (other).

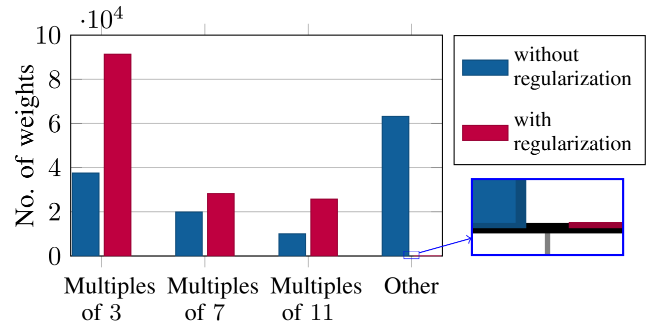

Figure 3. Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ before (blue) and after (red) regularization. Base $\mathcal{B}_{\text{weight}} = \{3, 7, 11\}$ is assumed. Following regularization, very few weights remain that are not multiples of any of the elements of $\mathcal{B}_{\text{weight}}$ (other).

leading to $P = (2 \cdot 16 + 1) + (2 \cdot 3 + 1) = 40$ terms per weight.

As the difference $(w_n \cdot M_{\text{weight}} - k \cdot m_i)^2$ may grow large for some terms, the overall regularization loss takes very large values because, even if $w_n \cdot M_{\text{weight}}$ may be close to a $k \cdot m_i$ value, the difference will not be exactly zero to nullify the loss.

To mitigate this issue, only $k \cdot m_i$ values close to $w_n \cdot M_{\text{weight}}$ are taken into consideration. This essentially applies a window such that a subset of $k \in [-K_i, K_i]$ is selected. The size of the window can be tuned. In our experiments the window is selected such that

$$|w_n \cdot M_{\text{weight}} - k \cdot m_i| < T, \tag{20}$$

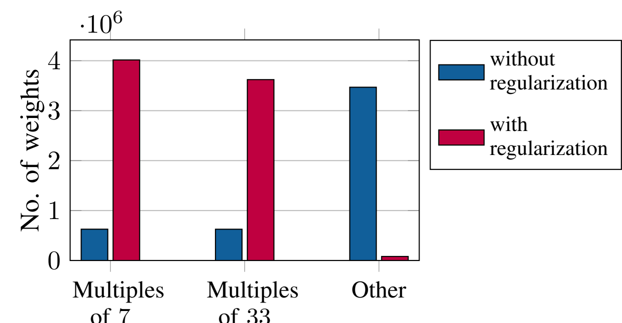

Figure 4. Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ before (blue) and after (red) regularization, for the last FC layer of VGG16 on ImageNet. Base $\mathcal{B}_{\text{weight}} = \{7, 33\}$ is assumed. Following regularization, the number of weights that are not multiples of either 7 or 33 is substantially reduced (other).

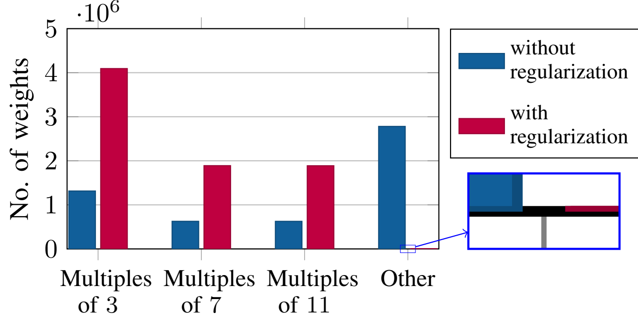

Figure 5. Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ before (blue) and after (red) regularization, for the last FC layer of VGG16 on ImageNet. Base $\mathcal{B}_{\text{weight}} = \{3, 7, 11\}$ is assumed. Following regularization, very few weights remain that are not multiples of any of the elements of $\mathcal{B}_{\text{weight}}$ (other).

TABLE 2. TRAINING SLOWDOWN

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

where T is a selected threshold. In our simulations $T \in \{6, 7, 8\}$ was used.

Table 2 shows the measured average slowdown of the training step for various scenarios with respect to the non-regularized training. All the simulations were carried out on an NVIDIA RTX A6000 Graphics Processing Unit [13].

4. Hardware architectures exploiting residue sparsity

The proposed regularization-based methodology can be exploited in a number of ways in RNS-based hardware accelerators, as detailed in the following. Various techniques and architectures exploiting residue-level sparsity are analyzed in terms of power dissipation.

4.1. RNS base selection

For this analysis, three RNS bases, $\mathcal{B}$, $\mathcal{B}'$ and $\mathcal{B}''$ are considered, consisting of five or six channels, for performing the convolution computations. A two- or three-channel subset of these bases, to which the proposed regularization method is applied, is assumed for the storage of weights. The resulting residue sparsity factors (ratio of zero residue values to the total number of weights) for the i-th residue channel are denoted as α_i.

The RNS bases $\mathcal{B}$ and $\mathcal{B}'$, are restricted to moduli of the form 2^k and $2^k \pm 1$ in order to simplify the implementation of the RNS arithmetic circuits, while $\mathcal{B}''$ also includes a modulo-11 channel. Modulo 2^k addition and multiplication are trivial, as the mod operator simply translates into performing all operations on the k least-significant bits (LSBs) only; thus simplified conventional multiplier and adder design practices apply. For modulo-(2^k-1) arithmetic, the end-around-carry technique can be used. Similarly, for channels of moduli of the form 2^k+1, diminished-1 addition can be used [14] [15], where all operands are decreased by one (zero values are handled separately) and an inverted end-

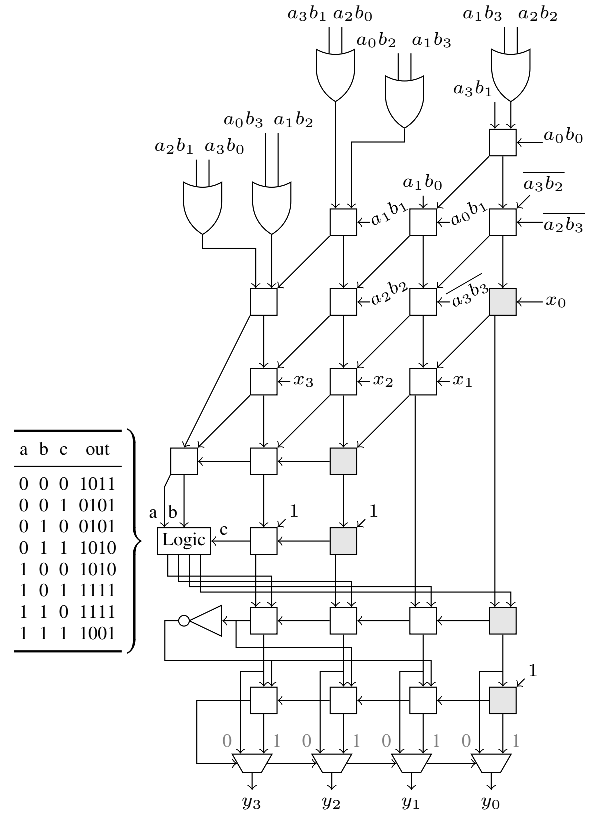

Figure 6. A modulo-11 MAC unit, which performs the operation $y = (ab + x) \bmod 11$, where y, a, b, x are residues mod-11 and a_i, b_i, x_i, y_i are bits of weight 2^i. White and gray boxes denote 1-bit full adders and 1-bit half adders, respectively. Trapeziums are 1-bit multiplexers. Block labeled 'Logic' is defined by the truth table in the figure. Logic synthesis further simplifies the unit.

around-carry logic is utilized. The implementation of the modulo-11 MAC generalizes techniques discussed in [16] for multiplication, and it is shown in Fig. 6.

Table 3 presents the area complexity and power dissipation of the MACs required for the moduli composing base $\mathcal{B}$, referring to synthesis results with a 0.5 V supply and a clock frequency of 1 GHz, using a 22-nm GlobalFoundries standard-cell library and Synopsys tools. For each modulo, absolute area and power dissipation are reported, as well as the percentage of the total five-modulo MAC area and power dissipation, which corresponds to the specific modulo. All modulo MAC designs are pipelined with one intermediate level of flip-flops (FFs) to support the desired clock frequency (1 GHz).

A second base is considered for comparative evaluation, defined as $\mathcal{B}' = \{5, 7, 9, 16, 17, 31\}$. Base $\mathcal{B}'$ comprises six moduli the maximum of which is 31. The complexities of the MACs for each of the moduli in $\mathcal{B}'$, are shown in Table 4. Two alternative choices are considered for the subsets of the base that offer sufficient dynamic range for the representation of the weights, namely,

$$\mathcal{B}^{(1)}_{\text{weight}} = \{7, 9, 17\} \tag{21}$$

$$\mathcal{B}^{(2)}_{\text{weight}} = \{5, 7, 17\}. \tag{22}$$

TABLE 3. MAC CHANNEL COMPLEXITY FOR BASE $\mathcal{B}$

MAC	Area		Power	
	(μm^2)	(%)[1]	(μW)	(%)[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

[1]Percentage of the total five-moduli MAC area (power), which corresponds to each modulo.

TABLE 4. MAC CHANNEL COMPLEXITY FOR BASE $\mathcal{B}'$

MAC	Area		Power	
	(μm^2)	(%)[1]	(μW)	(%)[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

[1]Percentage of the total six-moduli MAC area (power), which corresponds to each modulo.

We investigate whether the use of smaller moduli is beneficial in terms of increased sparsity, since the number of their integral multiples in the available dynamic range is larger, therefore more potential choices are available the regularization process, also taking into account that the MAC complexity for smaller moduli is less.

Figs. 7 and 8 depict the number of multiples of the elements of $\mathcal{B}^{(1)}_{\text{weight}}$ and $\mathcal{B}^{(2)}_{\text{weight}}$, before and following the application of the proposed regularization method. It can be observed that the method substantially reduces the number of weights that are not multiples of any of the moduli of the base (column Other) in both cases. When focusing on regularizing for multiples of a specific modulo (i.e., 17 in Figs. 7 and 8), it can be observed that the number of the multiples of the specific modulo increases. This can be of interest when a specific modulo imposes relatively high cost in terms of power dissipation and/or latency. Furthermore, it can be seen that when regularizing for multiples of all moduli, a lower number of moduli that are not multiples of any moduli, is achieved for the base $\{5, 7, 17\}$, compared to $\{7, 9, 17\}$.

TABLE 5. MAC CHANNEL COMPLEXITY FOR BASE $\mathcal{B}''$

MAC	Area		Power	
	(μm^2)	(%)[1]	(μW)	(%)[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

[1]Percentage of the total six-moduli MAC area (power), which corresponds to each modulo.

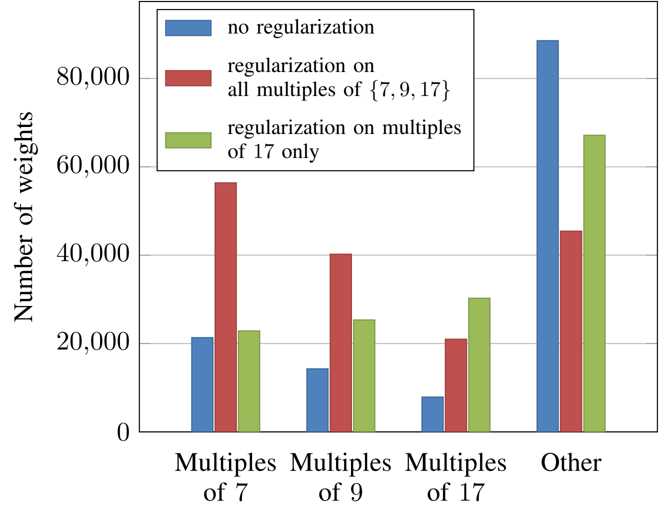

Figure 7. Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ without regularization (blue), with regularization on all multiples of $\{7, 9, 17\}$ (red) and regularization on multiples of 17 only (green).

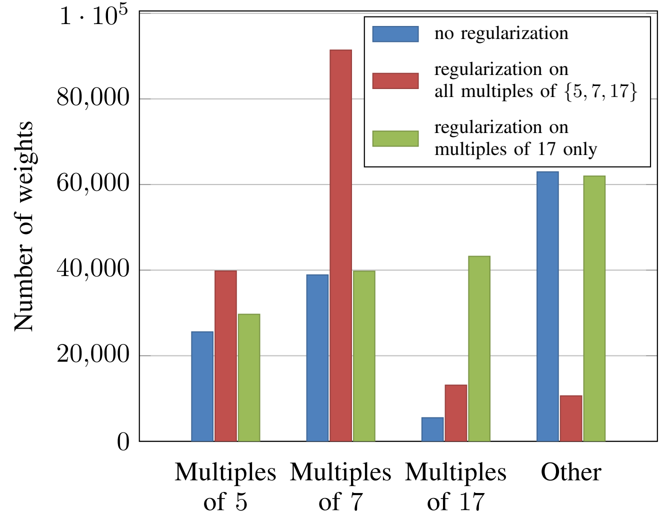

Figure 8. Histogram of $\lfloor w \cdot M_{\text{weight}} \rfloor$ without regularization (blue), with regularization on all multiples of $\{5, 7, 17\}$ (red) and regularization only on multiples of 17 (green).

4.2. Exploiting residue sparsity to reduce memory cost

Sparse DNN architectures typically rely on storing values (weights, biases and/or inputs) in some compressed vector representation, such as compressed sparse column or row (CSC or CSR) [17], [18]. These methods use a non-zero value vector and an index vector, to encode the coordinates of the non-zero elements. However, index-based compression schemes are not suitable for encoding sparse residue vectors. This is because the non-zero values do not occur at the same positions among the different residue channels, hence each channel must have its own index vector. This fact, combined with the small size of the residue channels (3 to 6 bits) makes the index storage overhead significant and effectively no compression can be achieved.

To mitigate this problem, a simple variable-length encoding is proposed in the following.

In the proposed scheme, a zero value is encoded using one bit and a non-zero value is encoded as a $(n+1)$-bit word, where n is the channel word length. Assuming a residue value $d = d_{n-1}d_n \dots d_0$, the proposed encoding G is defined as follows

$$G(d) = \begin{cases} 0, & \text{if } d = 0 \\ 1d_{n-1}d_n \dots d_0, & \text{otherwise.} \end{cases} \quad (23)$$

The compression achieved by this method depends on the sparsity factors α_i. The average size, $\widehat{n}$, of the encoded word assuming base $\mathcal{B}$ and using residue channels modulo-7 and modulo-33, is given by

$$\widehat{n} = \alpha_0 + (3+1)(1-\alpha_0) + \alpha_1 + (5+1)(1-\alpha_0) \quad (24)$$

$$= 10 - 3\alpha_0 - 5\alpha_1. \quad (25)$$

Using the sparsity factors obtained from the test-case of section 3.1, $\alpha_0 = 0.8$ and $\alpha_1 = 0.14$, the average size of the encoded word is $\widehat{n} = 6.9$ bits, translating to a 13.75% compression rate.

4.3. Exploiting residue sparsity to reduce power consumption

4.3.1. Clock-gating for smaller moduli channels. The simplest technique to exploit residue sparsity in order reduce power consumption is to freeze the sparse residue channels by clock gating. This means that the clock signal of arithmetic circuits and input registers of the sparse channels is masked with an enable bit, indicating a zero-value. This technique reduces dynamic power dissipation and results in an overall decrease of RNS MAC unit power consumption which depends on the sparsity factors and the contribution of the sparse channels to the total power consumption.

4.3.2. Zero-skipping per moduli channel. Zero-skipping techniques are common among sparse DNN accelerators. Instead of simply gating zero inputs (weights or activations), zero-skipping methods completely avoid loading any zero values in the processing elements (PEs). This means that instead of deactivating the PEs to save computation power, in case of zero values, thus wasting idle clock cycles, zero-skipping performs the next computation involving non-zero values. This results both in reduction of total energy consumption but also processing time. In a residue-level sparse DNN scenario, zero-skipping must be performed in the residue level. The workloads of the different residue channels become unbalanced, since the regularization results in different sparsity levels. This means that each channel may complete its computation at potentially different times, with channels with higher sparsity finishing first. One way to exploit this, is to completely deactivate (power gating) a residue channel when it completes the processing of its current input vectors.

Using the sparsity factors obtained from the test-case CNN of section 3.1, and the relative energy cost of each PE, as reported in Tables 3–5, the expected power savings of such a scheme are estimated in the following. Different test cases referring to various selections for $\mathcal{B}_{\text{weight}}$ and $\mathcal{B}$ are used to evaluate the proposed method's gains:

TABLE 6. 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

1) **Test case 1:** $\mathcal{B}_{\text{weight}} = \{7, 33\}$
In this case $\mathcal{B} = \{5, 7, 31, 32, 33\}$ is used as the full RNS base for accumulation. Since the modulo-7 and modulo-33 MAC units contribute to 11.7% and 40.4% of the total power consumption of a single MAC, the energy savings G resulting for sparsity factors α_i can be calculated as
$$G = 0.117\alpha_0 + 0.42\alpha_1. \quad (26)$$
Setting α_0=0.8 and α_1=0.14, the total power savings, assuming completely deactivating the arithmetic units of a residue channel when it finishes its computation, can reach up to 15.01%.
2) **Test case 2:** $\mathcal{B}_{\text{weight}} = \{7, 9, 17\}$
In this case, where $\mathcal{B}' = \{5, 7, 9, 16, 17, 31\}$, is used as the full RNS base for the accumulation, the total gains are
$$G = 0.117\alpha_0 + 0.183\alpha_1 + 0.258\alpha_2. \quad (27)$$
Replacing the resulting sparsity factors, $\alpha_0 = 0.45$, $\alpha_1 = 0.31$ and $\alpha_2 = 0.16$ (Fig. 7, regularization for all multiples of elements in $\{7, 9, 17\}$), G can reach up to 15.1% for this base selection.
3) **Test case 3:** $\mathcal{B}_{\text{weight}} = \{5, 7, 17\}$
Using this base the total gains are
$$G = 0.041\alpha_0 + 0.117\alpha_1 + 0.258\alpha_2. \quad (28)$$
By replacing the sparsity factors, $\alpha_0 = 0.23$, $\alpha_1 = 0.31$ and $\alpha_2 = 0.35$ (Fig. 8, regularization for multiples of 17 only), G can reach up to 13.8%.
4) **Test case 4:** $\mathcal{B}_{\text{weight}} = \{3, 7, 11\}$
Finally, by replacing modulo-33 channel with one modulo-11 and one modulo-3 channel, according to the power contribution of each channel we have:
$$G = 0.038\alpha_0 + 0.133\alpha_1 + 0.276\alpha_2. \quad (29)$$
Using the resulting sparsity factors $a_0 = 0.75$, $a_1 = 0.23$ and $a_2 = 0.21$ (Fig. 3), G is evaluated as $G = 11.7\%$

It can be seen that it is generally preferable to select the channels that have the highest hardware complexity, i.e., channels of the form 2^k+1, as the target of the regularization process, since their corresponding energy savings can be higher. Selecting $\mathcal{B}'$ with target channels $\mathcal{B}_{\text{weight}} = \{7, 9, 17\}$ for regularization results in the most significant saving percentage. However, in absolute values, $\mathcal{B}''$ performs better in terms of power consumption, as shown in Table 6.

4.4. Overall architecture implementation details

The weight compression mechanism and the zero-skipping scheme presented in subsections 4.2 and 4.3.2, respectively, introduce challenges in the implementation of a complete CNN architecture. In this subsection, a general

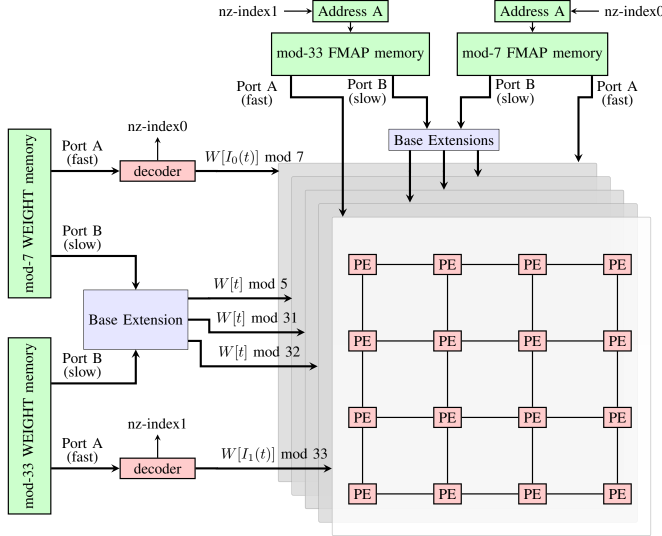

Figure 9. Overview of an RNS accelerator exploiting residue-level sparsity. It consists of n independent PE arrays of size $M \times M$, corresponding to the n different residue channels. A non-zero detector module reads a window of weight values (stored in a weight buffer) encoded with respect to $\mathcal{B}_{\text{weight}}$ 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.

architecture is proposed, which aims to minimize the hardware overhead required to address these challenges.

A general architecture is shown in Fig. 9. It consists of N independent processing arrays, corresponding to the N different residue channels, and two hierarchical memory components: a weight memory and a feature-map memory. A weight decoder module reads a window of weight values, fetched from the weight memory and encoded with respect to $\mathcal{B}_{\text{weight}}$ and provides the next non-zero weight value and index. 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. By exploiting temporal reuse of weights, which is possible due to the weight sharing scheme in CNNs, the decoder can be shared among a large number of MAC units thus amortizing its overhead. This is quantified in the following subsection.

4.4.1. Weight decoder. The weight decoder performs the decompression of the weight values, according to the proposed variable-length compression scheme of subsection 4.2. For each channel in $\mathcal{B}_{\text{weight}}$ 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. The maximum shift amount is limited to 4–8 positions depending on the channels (small channels such as modulo-7 have large sparsity, thus longer continuous strings of zeros are more probable than in larger channels such as modulo-33).

TABLE 7. 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

This index is also used to calculate the next address of the corresponding feature-map bank.

The weight decoder complexity is shown in Fig. 7. The decoder for $\mathcal{B}_{\text{weight}}$ ($\mathcal{B}'_{\text{weight}}$) consumes 37% (53%) of the power of a single MAC unit, with 28% (45%) of its area. The larger cost of $\mathcal{B}'_{\text{weight}}$ compared to $\mathcal{B}_{\text{weight}}$ is because it consists of three channels instead of two. In both cases, however, the decoder only adds a small overhead to the total power consumption of the system, since it is amortized over a large number of MAC PEs. Assuming a 4×4 processing array the total decoder power consumption overhead is 2.3% and 3.3% for $\mathcal{B}_{\text{weight}}$ and $\mathcal{B}'_{\text{weight}}$, respectively, since one decoder is shared among 16 MAC processing elements.

4.4.2. Memory organization. The different sparsity levels within the residue channels translate into different processing rates for each channel. This means that each channel requires a weight residue value from a different index of the weight vector at each timestep. Let $w = (w_0^t, w_1^t, \ldots, w_{N-1}^t)$ be the tuple of weights required by the processing elements at timestep t, where N is the size of $\mathcal{B}$ (the extended base used for the convolution), then $w_k^t = W[I_k(t)] \bmod m_k$, 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 and m_k is the channel modulus. The values w_k^t are provided by the weight decoder of subsection 4.4.1. Assuming that weights are stored in $\mathcal{B}_{\text{weight}}$, which consists of N_w channels, and then N_{be} channels are added to obtain the weight representation in $\mathcal{B}$ $(N = N_w + N_{be})$, then, due to the zero-skipping processing, the indices $I_k(t)$ are different for all the first N_w channels, while $I_k(t) = t$ for $k \geq N_w$, since no considerable sparsity is assumed for the channels obtained after base extension and thus no zero-skipping takes place for these channels. Regularization can target these channels as well, but this case is not considered here. Hence, we need to decouple the access to the weights of each residue channel. This can be accomplished by using separate memory banks for each channel. Moreover, 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. This can be achieved by utilizing dual-port RAM macros, which allow simultaneous access to two independent addresses. Port A, denoted as fast port, provides access to the weight vector according to the processing rate of each individual channel, while Port B, denoted as slow port, provides access to the weight vector at a rate of one value per cycle. The addresses of the fast ports are incremented at every cycle, unless the weight bit buffers are full, while the addresses of the slow ports (corresponding to weight values needed for base extension), are incremented only when a weight

word fetched from these ports is fully processed (both zeros and non-zero values are used for the base extension). This memory organization is thus compatible with the proposed compression and zero-skipping schemes, without requiring additional buffers or complex synchronization circuitry. Alternatively, the above functionality could be achieved using single-port RAM macros and a FIFO buffer for each weight channel, where the values read from the main weight memory, are temporarily stored, before being read by the base extension unit. For every zero value, the number of elements in the buffer would increase by one, since the difference between the required channel indices and base extension indices $(I_k(t) - t)$ will increase by one. The relative area and power efficiency of the proposed solutions depends on the actual memory sizes, the maximum number of zeros during a complete iteration of the a weight vector W, the complexity overhead of a dual-port RAM, as well as the target hardware platform. For example, dual-port RAMs are readily available in most FPGAs. In both cases, the memory overhead (either a second port or a FIFO buffer) can be compensated by the reduced memory requirements and corresponding data movements due to the achieved compression (Section 4.2).

5. Conclusion

A regularization formulation is introduced in this paper, which modifies ANN training in order to induce to the weights the desirable property of increased residue-level sparsity. When expressed in an RNS, the obtained weights exhibit an increased percentage of residues that are zero. The introduced method increases residue sparsity by a factor of $4\times$ to $6\times$ in certain cases without practical degradation of ANN accuracy. By comparatively evaluating RNS bases in the context of the proposed method, it follows that the choice of the RNS base is important for the exploitation of residue sparsity.

Building on the residue-level sparsity, it is subsequently shown that the particular property can be exploited to further improve RNS-based hardware accelerators, especially decreasing energy requirements. The proposed method has shown promising results for the final fully-connected layer of the VGG16 model. The application of the introduced regularization technique may lead to new RNS architectures that exploit residue sparsity and may render the RNS an interesting candidate for hardware accelerators in edge devices.

Acknowledgment

The research work by E. Kavvousanos was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI PhD Fellowships (Fellowship Number: 5458). The research work by V. Sakellariou and T. Stouraitis was supported by SRC project 2020-AH-2983 and Khalifa University's SOC center. Finally, the authors acknowledge the anonymous referees' suggestions that have improved the presentation of the introduced methods.

References

[1] E. B. Olsen, "RNS Hardware Matrix Multiplier for High Precision Neural Network Acceleration: "RNS TPU"," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[2] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–6.

[3] M. Abdelhamid and S. Koppula, "Applying the residue number system to network inference," *arXiv preprint arXiv:1712.04614*, 2017.

[4] V. Sakellariou, V. Paliouras, I. Kouretas, H. Saleh, and T. Stouraitis, "A high-performance RNS LSTM block," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 1264–1268.

[5] 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.

[6] V. Sakellariou, V. Paliouras, I. Kouretas, H. Saleh, and T. Stouraitis, "On reducing the number of multiplications in RNS-based CNN accelerators," in *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1–6.

[7] M. Wess, S. M. P. Dinakarrao, and A. Jantsch, "Weighted quantization-regularization in DNNs for weight memory minimization toward HW implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2929–2939, 2018.

[8] X. Long, Z. Ben, X. Zeng, Y. Liu, M. Zhang, and D. Zhou, "Learning sparse convolutional neural network via quantization with low rank regularization," *IEEE Access*, vol. 7, pp. 51 866–51 876, 2019.

[9] N. S. Szabó and R. I. Tanaka, *Residue arithmetic and its applications to computer technology*. McGraw-Hill, 1967.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[12] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *3rd International Conference on Learning Representations (ICLR 2015)*, pp. 1–14, 2015.

[13] NVIDIA, "NVIDIA RTX A6000 graphics card." [Online]. Available: https://www.nvidia.com/en-us/design-visualization/rtx-a6000/

[14] L. Leibowitz, "A simplified binary arithmetic for the Fermat number transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, no. 5, pp. 356–359, 1976.

[15] H. Vergos, C. Efstathiou, and D. Nikolos, "Diminished-one modulo $2^n + 1$ adder design," *IEEE Transactions on Computers - TC*, vol. 51, pp. 1389–1399, 01 2002.

[16] G. Dimitrakopoulos and V. Paliouras, "A novel architecture and a systematic graph-based optimization methodology for modulo multiplication," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 2, pp. 354–370, 2004.

[17] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 27–40. [Online]. Available: https://doi.org/10.1145/3079856.3080254

[18] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 243–254. [Online]. Available: https://doi.org/10.1109/ISCA.2016.30